Ordering number 2007-ISAL-0107 Year 2007

Thesis

Grid Caching: Specification and Implementation of Collaborative Cache Services for Grid Computing

Submitted to the

National Institute of Applied Sciences of Lyon

In fulfillment of the requirements for a
Doctoral Degree

Affiliated Area: Computer Science

Doctoral school of Computer Science and Informatics (EDIIS-EDA 335)

Prepared by

Yonny CARDENAS BARON

Defended at 10^{th} December 2007 in front of the Examination Committee.

Committee Members :

Pr.	**Norman PATON**	University of Manchester	Reviewer
Dr.	**Jean-Marc NICOD**	Université de Franche-Comté	Reviewer
Pr.	**Lionel BRUNIE**	INSA de Lyon	Supervisor
Pr.	**Jean-Marc PIERSON**	Université Paul Sabatier	Co-supervisor
Pr.	**Abdelkader HAMEURLAIN**	Université Paul Sabatier	Examiner
Dr.	**Claudia RONCANCIO**	Université Joseph Fourier	Examiner

LIRIS
Lyon Research Center for Images and Intelligent Information Systems

Numéro d'ordre : 2007-ISAL-0107 Année 2007

Thèse

Grid Caching: Spécification et Mise en Oeuvre de Services de Caches Collaboratifs pour les Grilles de Calcul

présentée devant

L'Institut National des Sciences Appliquées de Lyon

pour obtenir
Le grade de docteur

Spécialité : Informatique

École doctorale : Informatique et Information pour la Société (EDIIS-EDA 335)

Par

Yonny CARDENAS BARON

Soutenue le 10 Décembre 2007 devant la Commission d'examen

Jury :

Pr.	**Norman PATON**	University of Manchester	Rapporteur
Dr.	**Jean-Marc NICOD**	Université de Franche-Comté	Rapporteur
Pr.	**Lionel BRUNIE**	INSA de Lyon	Directeur de thèse
Pr.	**Jean-Marc PIERSON**	Université Paul Sabatier	Co-directeur de thèse
Pr.	**Abdelkader HAMEURLAIN**	Université Paul Sabatier	Examinateur
Dr.	**Claudia RONCANCIO**	Université Joseph Fourier	Examinatrice

LIRIS
Laboratoire d'Informatique en Image et Systèmes d'information

Abstract

Grids support multiple models of distributed computation and need to operate on large data entities in a distributed way. A significant quantity of these data are used only for a limited period of time. Furthermore, grids are characterized by a very high dynamism both in terms of available resources and terms of effective data access patterns. Consequently, temporary data management in grids is highly critical and global coordination of the storage network resources is mandatory.

In this context, caching is recognized as one of the most effective techniques to manage temporary data and a collaborative cache is traditionally proposed to scale cache capabilities in distributed environments.

Sharing distributed storage resources is an important issue that deals with strategies for placement of temporary data in a particular location. These decisions are essential for the function and the performance of the application and system. Finding and selecting the storage locations to hold temporary data in a dynamic way is a complex process that requires detailed and accurate information about temporary data activity in the whole system.

Current data management mechanisms for grid environments only provide basic capabilities for supporting storage, access and transfer of large amounts of data. However, they do not provide in the context of diverse access patterns, the efficient mechanisms to obtain and manage enough temporary storage capacity to handle large datasets in a dynamic way from heterogeneous, distributed and autonomous resources.

The idea that we defend in this thesis is that the interactions between caches determine the capabilities that can be provided by the grid. Therefore, expanding the scope of the cache interactions creates new possibilities for the management of temporary data. These interactions must be flexible, organized and monitored to take advantage of individual and collective cache capacities and can then be used to improve grid data management functionalities and services.

The thesis proposes an approach for the design and implementation of collaborative cache systems in grids that supports capabilities for monitoring and controlling cache interactions. Our approach permits compostion and evaluation of high-level

collaborative cache functions in a flexible way.

Our proposal is based on a multilayer model that defines the main functions of a collaborative grid cache system. These capabilities are implemented by a set of common and standard operations that support the data access, the monitoring and the configuration of a cache. This model and the provided specification are used to build a flexible and generic software infrastructure for the operation and control of collaborative caches.

The infrastructure is composed of a group of autonomous cache elements called Grid Cache Services (GCS). The GCS is a local administrator of temporary storage and data which is implemented as a grid service that provides the cache capabilities defined by the model. It implements interfaces and associated cache operations. We implemented a prototype of the GCS with Globus Toolkit 4 middleware and we have made wide area performance measurements for our GCS prototype.

In the manuscript, we study a possible configuration for a group of GCS that constitutes a basic management system of temporary data called Temporal Storage Service (TSS). TSS uses a group of caches that collectively provide storage capabilities to store temporary data. The results of a simulation experiment demonstrate that group caches (GCS) operating according to their activity can provide supplementary storage space with minimal cost.

We conclude the manuscript by summarizing the main contributions of this thesis: 1) a definition and specification of cache service as a distinct grid component; 2) a definition of an architectural infrastructure to construct and operate caches in a wide area Grid; and 3) a design of mechanisms for management of distributed and temporary data based on collaborative cache techniques.

Acknowledgments

First and foremost I would like to thank my supervisors Lionel Brunie and Jean-Marc Pierson. Thanks to them, I have discovered the interesting world of research. Through all the years their advice and expertise have been very valuable for me in professional and personal dimensions.

I would like to thank Norman Paton and Jean-Marc Nicod who accepted the hard task of reviewing my manuscript and being members of my examination committee.

My thanks to Abdelkader Hameurlain and Claudia Roncancio who graciously accepted to act as members of my examination committee.

I specially thanks Serge Miguet who organized and directed the Region Rhône-Alpes biomedical grid project (Ragtime) which funded this thesis.

Thanks to Pedro Giffuni and Sophie Barre for the generous help with the grammar of the manuscript.

Thanks to my girlfriend Annelyse, my parents, brothers and friends for their constantly support and encouragement.

And finally I wish to thank my colleagues from the LIRIS laboratory with whom I have discussed some of the interesting details and implications of my thesis: Hector Duque, Ludwig Seitz, Rami Rifaieh, Julien Gossa, David Coquil, Girma Berhe, Ny-Haingo Andrianarisoa, Dejene Ejigu, Omar Hasan, Marian Scuturici and Rachid Saadi.

Contents

List of Figures

List of Tables

Chapter 1

Résumé Français

Ce résumé est destiné aux lecteurs francophones. Il a pour but de leur donner une idée précise du contenu de cette thèse. Néanmoins, faute d'espace, les détails et explications relatifs à de nombreux points ne peuvent être abordés dans ce résumé. Nous prions le lecteur intéressé de consulter la partie anglaise de ce document.

1.1 Introduction

Le partage de ressources est une problématique importante depuis le début de l'informatique. Les systèmes de temps partagé ont été aussi développés pour gérer les machines de telle sorte que les utilisateurs partagent un ordinateur avec l'illusion de l'avoir pour eux tout seuls. Depuis lors l'évolution technologique a rendu possible la création d'Internet. Les ressources partagées les plus importantes sont les données. Les données représentent l'information utile pour les utilisateurs et les applications. Internet a été créé pour permettre le partage de toutes sortes de données. Le succès principal d'Internet réside dans sa capacité à rendre possible le partage des données sur un plan mondial. La conséquence en est la demande toujours croissante des capacités de stockage et d'échange des données.

Les technologies de grille permetent l'intégration et le partage des ressources hétérogènes et distribuées entre différents domaines administratifs, avec pour objectif de fournir un accès dynamique à ces équipements chaque fois que nécessaire. Cette technologie permet la définition "d'organisations virtuelles" plaquées sur un ensemble d'institutions désirant partager l'ensemble de leurs infrastructures de calcul. Les utilisateurs associés au sein de ces organisations virtuelles et leurs applications disposent de la puissance de cette informatique répartie, et de ses espaces de stockage de données disponibles à la demande.

Les applications de grille essayent de résoudre des problèmes complexes. L'objectif

principal de ces applications est souvent relié avec l'analyse et le traitement intensifs de grandes quantités de données. Il s'agit d'une tâche très difficile pour deux raisons. Les volumes de données se développent rapidement et constamment, exigeant de grandes capacités de stockage. En second lieu, ces ensembles de données sont produits, stockés, et employés d'une manière distribuée, devant ainsi être nécessairement déplacés.

Ce transfert de données consomme énormément de bande passante et d'espace de stockage, sans coordination globale les ressources sont employées de manière inefficace et non rentable. Une quantité significative de données est employée pendant une période limitée, par example lors de phases d'analyse de données, autrement dit une proportion importante de données disséminées dans un système réparti, comme une grille, est constituée de données temporaires. Un système de gestion optimisé est donc nécessaire pour gérer la diffusion de ces données temporaires dans les systèmes répartis.

L'intérêt pour des données temporaires est plus économique que technologique : la gestion de l'infrastructure de stockage permet d'atteindre une rentabilité plus grande pour des données temporaires que pour des données persistantes. Il y a deux raisons à cela. En premier lieu, l'infrastructure nécessaire aux données temporaires croît moins rapidement. Ensuite, l'infrastructure est réutilisée constamment. Le point crucial ici est la manière dont ces ressources sont gérées pour améliorer la disponibilité, les temps de réponse, les débits et la facilité d'utilisation.

L'utilisation de caches est reconnue comme une des manières les plus efficaces de gérer des données temporaires. Un système de cache gère et héberge des données pour une période limitée dans un espace de stockage dont il dispose. Ces données peuvent être réutilisées et partagées entre différents clients, permettant ainsi une meilleure utilisation des ressources de stockage.

Des systèmes de caches coopératifs ont été imaginés pour permettre aux systèmes distribués de s'adapter à des besoins de stockage croissants. Ces systèmes apparaissent comme une solution naturelle au stockage de données temporaires dans le cadre dans le cadre des grilles de calcul. Il est nécessaire d'établir les spécifications de ce "cache de grille" et d'approfondir les différents aspects reliés à son déploiement structuré.

Cette thèse fournit les spécifications des composants de base nécessaires à la construction du cache de grille, infrastructure assurant le fonctionnement du système de données temporaires distribuées. Ce travail de recherche est structuré par le développement d'un système de collaboration de cache pour des grilles et les conditions spécifiques des applications de grille. Dans ce contexte, où les utilisateurs s'attendent à l'accès infini aux ressources et où les ressources sont limitées et partagées, la gestion des ressources de données surgit comme question centrale.

1.2 État de l'art

1.2.1 Systèmes de caches

Ces systèmes permettent la réutilisation et le partage de données temporaires hébergées dans des espaces de stockage qu'on appelle caches. Ces systèmes présentent de grands avantages en termes de temps d'accès, d'utilisation des réseaux et des ressources de stockage. La localisation et durée de vie des données sont enregistrées par le système et permettent la gestion des espaces octroyés. En cas de besoin le système a les informations nécessaires pour retrouver les données ou libérer de l'espace pour de nouveaux besoins. Le système régule l'utilisation de l'espace de stockage en fonction de la demande et de l'activité des données présentes optimisant ainsi les transferts entre les différents éléments de stockage et tirant partie de la variété de leurs performances. Le système de cache est un mécanisme intermédiaire entre fournisseur et consommateur de données, de multiples consommateurs se partageant le service offert.

Applications du mécanisme de cache

Architecture d'ordinateur au cœur même des ordinateurs entre processeur et mémoire principale, le cache, espace de stockage rapide et réduit, réduit les temps d'exécution en anticipant le recours prévisible des applications à certaines données et certaines instructions.

Architectures multiprocesseurs ici le problème est beaucoup plus complexe, la cohérence globale doit être maîtrisée. Des techniques récentes traitent l'intégralité de la mémoire principale comme un cache.

Systèmes de fichiers répartis souvent côté client, les caches stockent les fichiers les plus récemment utilisés pour éviter le recours aux serveurs distants, permettant la montée en charge et augmentant l'insensibilité aux problèmes logistiques (eg. AFS).

Bases de données stockage de données fréquemment consultées, ainsi que des objets spécifiques: déclencheurs, procédures stockées, vues partielles fréquemment sélectionnées.

Applications Web et client-serveur peuvent impliquer d'importants mouvements de données ou des accès fréquents à des données distantes. C'est le cas de certaines applications de streaming multimédia. Les caches peuvent être implémentés côté serveur pour l'accélérer, en cache local côté client, ou à un niveau intermédiaire.

Applications réparties dans les intergiciels, tels que Globe, basés sur les modèles objet pour les applications réparties, le système de caches utilise la réplication

des objets. A l'inverse CORBA laisse entièrement la lourde tâche de gestion du cache à la charge du développeur.

Avantages et inconvénients des caches

Au nombre des avantages on peut citer:

- Réduction des volumes transférés et donc de la sollicitation des réseaux
- Réduction des temps de latence
- Réduction et répartition de la charge des serveurs
- Robustesse augmentée par la réduction de l'accès à des serveurs potentiellement défaillants

Fourniture d'informations d'activité des données permettant la description de modèles de traitements en relation avec les activités d'accès aux données, très utile pour analyser l'utilisation des caches partagés par un grand nombre d'utilisateurs.

Et pour les inconvénients, d'abord la difficulté à garantir la cohérence des données entre les différents caches et le lieu d'origine, suite, par exemple à une mise à jour locale sur l'un des caches. Ce problème a fait l'objet de nombreux travaux. La latence croît en cas d'absence dans le cache local des données désirées.

L'efficacité du système dépend de nombreux paramètres qui peuvent se contrarier mutuellement et dont l'ajustement réclame du savoir faire. De même il faut choisir entre rechercher les données à la demande ou de manière prévisionnelle, sachant que l'apport de ces techniques sophistiquées reste marginal. De nombreuses stratégies ont été proposées. Pour de meilleurs résultats, un panachage adapté de plusieurs d'entre elles doit être mis en œuvre.

Dans un environnement de grille de calcul, il est intéressant de savoir si la stratégie doit être choisie dynamiquement en fonction de la charge de travail. De plus les décisions de remplacement affectent l'état d'un cache particulier. Dans un cache collaboratif de meilleures performances pourraient être atteintes grâce à une coordination des décisions au niveau global. Une telle infrastructure est nécessaire.

Métrique des performances

Pour juger des performances des systèmes de cache, on mesure principalement :

- La fréquence du nombre de fois où les données ont été retrouvées en cache par rapport au nombre total de recherches (Hit Ratio)
- Le gain en volume ainsi obtenu (Byte Hit Ratio)
- Le gain en terme de latence ainsi obtenu (Delay Saving Ratio)

Notez, par exemple, que les deux premiers évoluent différemment quand on joue sur la taille des objets. S'il est préférable de favoriser les objets les plus demandés, il y aura un compromis à trouver pour les objets les plus volumineux. On dispose d'autres observables que sont l'utilisation de CPU ou de disque et le taux d'entrées-sorties.

1.2.2 Cache Web

Dans sa forme la plus simple, le système de caches Web est similaire à un cache traditionnel : les données sont stockées et gérées pour favoriser le partage et la réutilisation, les documents stockés en cache sont restitués à la demande minimisant ainsi les sollicitations des serveurs, le trafic réseau et les temps de réponse.

Cache intermédiaire (Proxy Cache) : cache interposé entre client et serveur, il intercepte les requêtes HTTP et fournit aux clients les données demandées après les avoir, le cas échéant, obtenues des serveurs.

Cache coopératif : un groupe de caches élémentaires se servent les uns les autres, et coopérent pour former un système global puissant. Leur organisation générale est importante. La littérature en distingue quatre types : hiérarchique, réparti, multidiffusion (multicast) et pair à pair. Aucun, spécifique par son architecture et ses protocoles de communication, ne satisfait la totalité des besoins d'une grille de calcul qui varient très rapidement et où la question d'une architecture évoluant dynamiquement entre les différents modèles de base est une question qui se révèle pertinente.

Discussion

Le cache collaboratif présente cependant des complications : complexité accrue des processus de gestion, le dynamisme recherché implique un choix intelligent et donc une connaissance fine du fonctionnement instantané de chaque cache ou bloc de donnée élémentaires. Cette thèse propose la spécification d'une infrastructure de base de cache collaboratif à même de faire face à ces défis. D'autres travaux de recherche pourraient ensuite être entrepris qui devraient se concentrer sur la conception et l'évaluation des procédures intelligentes de gestion et la vérification de l'applicabilité de ces procédures aux grilles en environnement réel.

1.3 Gestion des données dans une grille

Une grille fournit une infrastructure et des services aux applications nécessitant de très grandes quantités de données réparties dans de multiples lieux de stockage. Ses fonctions essentiellessont le transfert fiable et la gestion de réplicas, au-delà, la grille fournit des services de gestion de cohérence des réplicas, catalogues de métadonnées. Ces opérations sont contrôlées par des mécanismes d'autorisation et d'authentification. Les données sont partagées et distribuées entre plusieurs domaines administratifs, indépendamment du support de stockage en s'appuyant sur des informations relatives aux données, à leur activité, aux droits d'accès, versions et métadonnées.

Les grilles dites "de données" manipulent des volumes de données jusqu'au pétaoctet, sont munies d'un système de dénomination uniforme faisant correspondre un nom de fichier logique à des données largement réparties et supportent différents niveaux de confidentialité.

1.3.1 Organisation et standardisation

Les ressources accessibles à travers une grille sont partagées dans le cadre d'organisations virtuelles (VO) [53]. La VO représente différents agrégats organisationnels de ressources et d'utilisateurs qui collaborent, elle fixe les règles d'accès, souvent via des autorités de certifications. Des protocoles standards encadrent les échanges d'informations nécessaires à ces collaborations. Les interfaces des services sont indépendants des implémentations particulières et satisfont à des standards promus par des organismes tels que l'OGF (Open Grid Forum) [62] qui a adopté l'OGSA (Open Grid Services Architecture) [55] basé sur les services Web.

La boîte à outils Globus [63] permet la construction de services de grille au standard OGSA. Une grille est alors composée d'un ensemble de services que l'utilisateur peut intégrer pour gérer ses données distribuées (stockage, transfert, localisation des réplicas et bases de données). D'autres services peuvent compléter ces services de bas: catalogue des métadonnées, services de gestion des autorisations et authentification. Les utilisateurs devant implémenter leur propre mécanisme de maintient de cohérence.

1.3.2 Projets actuels dans le domaine des grilles de données

Dans le domaine des hautes énergies, les grilles de calcul sont mises en œuvre pour permettre l'analyse des énormes masses des données qui seront collectées CERN par le LHC (Large Hadron Collider), voir par exemple le projet EGEE [47]; en bioinfor-

matique : modélisation et simulation de processus biologiques, projet eDiaMoND[120]; l'astronomie avec dont le IVOA (International Virtual Observatory) [74] coordonne les projets; les sciences de la terre avec par exemple le projet NESgrid [92] pour la sismique.

1.3.3 Transfert des données

Problématique essentielle, le transfert des données concerne la gestion du transfert, le transport effectif des données d'un point à un autre et le contrôle d'accès. Il y a deux types de mécanismes de transports de données : le transport proprement dit des données sur le réseau, le protocole le plus utilisé étant sans doute GridFTP [4]; ensuite les fonctions d'entrées-sorties qui cachent aux applications les complexités des protocoles réseaux et leur permettent d'accéder aux fichiers distants comme s'ils étaient locaux.

1.3.4 Gestion du stockage de données

SRM (Storage Resource Manager) du Lawrence Berkeley Laboratory est une spécification l'interface de pilotage définie comme des services Web [108] [114]. SRM offre un moyen unique de dialoguer avec tout type de ressource de stockage. Divers systèmes de stockage vont finalement offrir une interface SRM qui va cacher la complexité des mécanismes de base. SRM ne sauvegarde pas lui-même les données mais intervient comme une interface à des systèmes de stockage de masse.

SRB (Storage Resource Broker) [12] [103] est un intergiciel qui offre un accès uniforme a un ensemble hétérogène de ressources de stockage : systèmes de fichiers, systèmes d'archivage, base de données. SRB fournit une vue unifiée des données stockées sur cet ensemble disparate en les organisant en collections virtuelles indépendantes de leur localisation ou organisation réelle. SRB permet la création de collections partagées.

Discussion

SRM et SRB sont des mécanismes pour gérer la charge des équipements élémentaires de stockage, ils emploient des techniques de cache pour réduire les temps de latence, ils ne fournissent pas d'espace distribué à la demande, ils ne fédèrent pas non plus des données collectives temporaires. Notre position est que la fédération d'espaces de stockages distribués et hétérogènes pour les données temporaires peut être utilisée pour implémenter des mécanismes qui fournissent de l'espace de stockage à la demande.

SRM et SRB offrent une vue logique des ressources physiques de stockage permettant

la construction de systèmes de stockage virtuels. Ceci peut permettre de supporter la collaboration de caches individuels. Dans le cadre d'une grille, il faut ajouter des possibilités de surveillance et de configuration pour supporter la fédération de caches.

1.3.5 Accès aux bases de données

Le problème se complexifie avec le besoin d'intégrer d'autres types de stockage de masse comme les bases de données nécessitées par la biologie ou l'astrophysique.

OGSA-DAI (Open Grid Services Architecture Data Access and Integration) est un intergiciel qui permet l'accès aux bases de données en tant que services Web [7] [38]. Dans le respect d'OGSA, OGSA-DAI permet l'accès aux ressources externes de données comme les fichiers, les bases de données relationnelles ou XML.

Dans le cas d'une base de données distribuée, la problématique de cache est très différente. Le support du cache doit être assez flexible et générique pour s'adapter à d'autres possibilités comme l'éclatement ou la fusion de requêtes pour profiter de la disponibilité en cache d'une certaine partie des données [3].

1.3.6 Réplication

Dans les environnements de grilles de données à forte activité, la duplication ou la réplication sont un moyen d'améliorer les performances et de minimiser la pression sur les réseaux en offrant aux ressources de calcul un accès local aux données en cache. La fiabilité du système ainsi que sa faculté de répondre aux montées en charge en dépend.

RLS (Replica Location Service) est un cadre architectural qui permet de répertorier et rechercher les réplicas [29] [39]. Partie de Globus, RLS dispose d'une table de correspondance entre les informations logiques et physiques de localisation des ressources, la redondance et l'équilibrage de charge passe par de multiples copies disséminées de cette table.

GDMP (Grid Data Mirroring Package) est un gestionnaire de réplication qui offre des services sûrs et rapides de transfert [113] [48]. Il supporte la réplication point à point.

Les mécanismes de réplication ne disposent pas d'outils spécifiques pour gérer les données temporaires distribuées. Une forte limitation des possibilités de réplication dans un environnement dynamique est le manque de mécanisme d'accès à du stockage à la demande entre des localisations différentes de la grille. Le mécanisme de

réplication ne fournit pas des informations sur l'état des services de stockage ou de l'activité des données utilisées pour supporter une gestion de haut niveau. Des mécanismes flexibles et sophistiqués sont nécessaires pour supporter dynamiquement différents modèles de calculs avec différents types d'accès aux données.

1.4 Spécifications du Cache Collaboratif pour les Grilles

Dans cette section, nous décrivons les principaux aspects du cache collaboratif de grille. Le but principal est l'automatisation de l'exploitation de grands volumes de données temporaires dans un contexte de grille alors même que, contrairement au web, il n'existe pas pour les grilles d'infrastructure de caches étendus. Nous définissons cette infrastructure de base [22] [24].

Nous fournissons les spécifications des composants de base sur lesquels construire un tel système, générique et ouvert. Notre approche s'appuie sur les interactions essentielles entre caches élémentaires. Nous développons le précepte que n'importe quel cache doit être accessible et contrôlable par n'importe quel utilisateur, application ou service dans la grille.

Ces spécifications comportent les trois éléments principaux suivants :

- Modèle multicouches : stockage (storage), commande (control), collaboration et coordination.
- Opérations de cache : ensemble d'opérations standards véhiculant des informations normalisées.
- Services de Cache (GCS) : sont les unités fonctionnelles implémentant le modèle et les opérations.

Ces éléments permettent de construire et d'évaluer des fonctions de collaboration de haut niveau de manière flexible. Les GCS, agrégeant leurs fonctionnalités, peuvent être combinés de différentes façons.

La dynamique des ressources de grille et des données temporaires impose un contrôle et un pilotage continus. L'infrastructure est basée sur l'enregistrement, l'accessibilité et l'analyse d'informations sur l'activité des différentes entités de données et de cache.

L'approche, principalement :

- Fournit une infrastructure agissant sur de grandes quantités de données temporaires

- Définit et sépare des principales fonctions du cache collaboratif
- Appuie le pilotage sur l'enregistrement, l'échange et l'analyse de l'activité des données et du cache
- Fournit un support étendu des interactions de caches élémentaires
- Offre le nécessaire pour surveiller et configurer le fonctionnement et interactions des caches
- Ainsi que pour combiner et configurer les diverses formes d'organisations de collaborations
- Fournit les interfaces standards aux fonctions de cache en environnement de grille
- Permet l'accès aux opérations de cache à une grande variété d'applications.

1.4.1 Modèle en couches

Nous présentons un modèle de référence pour la composition et le pilotage des caches collaboratifs de grilles. Les différents niveaux séparent les fonctions conceptuelles du système. Ce modèle de cache est employé comme référence conceptuelle pour implémenter les éléments d'un système générique de cache collaboratif.

Le modèle comporte quatre couches :

- Couche de stockage (storage layer)
- Couche de contrôle (control layer)
- Couche de collaboration (collaboration layer)
- Couche de coordination (coordination layer)

La figure 1.1 représente le modèle proposé.

Couche de stockage

La couche de stockage représente les différentes ressources de stockage utilisées pour réaliser le système de cache. Le modèle définit un cache comme un mécanisme qui gère des données temporaires contenues dans une ressource de stockage. Ainsi le but central de la couche de stockage est de fournir les possibilités de stockage et d'accès aux éléments de données dans une ressource de stockage spécifique.

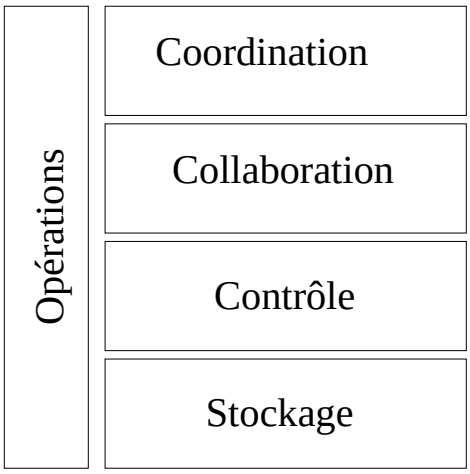

Figure 1.1: Modèle de référence des caches collaboratifs

Couche de contrôle

La couche de contrôle ou pilotage régule et enregistre les actions dans la couche de stockage. Elle vérifie l'activité d'accès aux données dans les ressources de stockage. Son but principal est de fournir les capacités de configurer et surveiller les actions des éléments de cache. Ceci permet d'intervenir sur le fonctionnement du cache pour adapter son comportement selon les buts recherchés. Par exemple, différentes méthodes de remplacement des données en cache (cf. annexe A) peuvent être supportées avec des paramètres de pilotage représentés de manière standard.

Couche de collaboration

La couche de collaboration définit les interactions entre caches élémentaires. Elle expose et étend les possibilités des caches pour qu'ils puissent travailler conjointement. Notre idée est que ces interactions déterminent les possibilités qui peuvent être fournies par la grille. Par conséquent, l'extension de la portée des interactions de cache crée de nouvelles possibilités pour la gestion des données temporaires.

Couche de Coordination

La couche de coordination organise les interactions entre caches selon différents types de collaborations. La coordination concerne surtout le processus de décision qui implique des opérations collectives entre caches pour supporter les fonctions de haut niveau de gestion des données.

1.4.2 Opérations Fonctionnelles

Les spécifications définissent les opérations fonctionnelles essentielles d'un cache collaboratif pour permettre l'implémentation du modèle présenté dans la section précédente. L'approche proposée dans cette thèse nécessite la définition des interfaces standards d'opération pour l'interaction des différents composants dans un environnement de grille.

Nous proposons que ces opérations soient disponibles pour l'interaction entre caches telle que définie par la couche de collaboration. Un cache élémentaire peut faire appel à des opérations sur d'autres caches élémentaires disséminés dans la grille. Pour construire le cache collaboratif, les caches élémentaires sont implémentés comme des services Web. Ceux-ci offrent un moyen standard d'interopération entre différentes applications fonctionnant sur des plateformes variées. Ils sont utilisés pour construire une infrastructure de cache orientée service. L'infrastructure est construite via la combinaison de composants de cache définis par des interfaces de service désignées dans ce travail sous le nom d'opérations de cache. À cet égard chaque cache expose ses opérations au système réparti comme service grille de cache. Nous classifions les opérations de cache dans trois types principaux :

- Opérations d'accès
- Opérations de surveillance
- Opérations de configuration

Opérations d'accès

Les opérations d'accès définissent les interfaces d'accès aux données stockées dans les ressources de stockage de cache. Elles permettent le pilotage et l'enregistrement des actions liées aux manipulations des données gérées par les caches élémentaires. Les opérations d'accès sont exposées au niveau de la couche de collaboration. On peut ainsi les invoquer pour les caches élémentaires distants. Les clients peuvent ainsi rechercher des données dans n'importe quel cache élémentaire dans le système. Réciproquement, un cache peut être sollicité pour stocker un objet de données au profit de n'importe quel autre cache, service ou application. Pour plus de détails concernant la définition des opérations d'accès nous renvoyons le lecteur à la section 4.4.2.

Opérations de surveillance

Les opérations de contrôle sont des interfaces d'échange d'informations sur les caches élémentaires et leur activité. L'information obtenue permet la gestion de la collab-

oration des caches (aide aux décisions) et l'évaluation des performances.

Nous proposons d'imposer aux mécanismes de cache dans la grille la fourniture des informations de leur activité en temps réel qui sont nécessaires à la gestion collective. Le but des opérations de surveillance est de fournir les informations appropriées pour des décisions appropriées. Pour plus de détails concernant la définition des opérations de surveillance nous renvoyons le lecteur à la section 4.4.3 .

Opérations de configuration

Les opérations de configuration définissent les interfaces de base pour paramétrer les caches. Ces opérations établissent les conventions standards qui permettent l'identification des caches élémentaires et définissent les fonctions et les interactions de base de ces éléments. Les opérations de configuration fournissent la possibilité de changer les paramètres fonctionnels d'un cache élémentaire pour un but particulier. Il peut s'agir des opérations d'initialisation, de maintenance, d'ajout et de modification des rapports entre composants et des caractéristiques de fonctionnement des composants eux-mêmes en période d'activité.

Les opérations de configuration fournissent le support de base permettant la réalisation de fonctions de coopération de cache de niveau élevé. L'arrangement des possibilités fonctionnelles est destiné à fournir le support des différentes combinaisons et stratégies qui constituent le système de cache collaboratif. Pour plus de détails concernant la définition des opérations de configuration nous renvoyons le lecteur à la section 4.4.4.

1.4.3 Informations de cache

L'information relative à l'état et au comportement du système est la ressource fondamentale du pilotage et de la gestion des données temporaires du cache collaboratif de grille. Le système peut grâce à elle offrir le support au pilotage et à la configuration du travail de collaboration entre composants.

Nous avons établi une structure d'information qui comprend l'information statique de base relative aux entités de données et de caches et l'information sur l'activité qui inclut des informations sur l'état et le comportement des entités de données et de cache. D'autre part, l'information statistique constituée des mesures de performances, est calculée en utilisant l'information d'activité et n'est pas définie dans la structure d'information de cache. L'information de cache peut être gérée par des systèmes conçus pour gérer des données structurées telles que les bases de données relationnelles.

L'information de cache est typiquement produite par le cache élémentaire impliqué.

Ainsi, un cache élémentaire maintient sa propre information de configuration. L'information de cache est rassemblée et stockée par le cache élémentaire qui exécute les actions sur les données. Cette information est exposée directement aux caches élémentaires qui utilisent les interfaces standards d'opération de cache. L'information statistique peut être calculée en interne. Elle peut également être calculée par un système externe qui recueille l'information d'activité nécessaire en appelant les opérations des caches élémentaires distants.

L'information de cache est structurée dans un schéma XML extensible où les différents éléments contiennent les valeurs qui décrivent les dispositifs, paramètres ou attributs des principaux composants de cache. Pour plus de détails concernant la définition des éléments contiennent des informations de cache nous renvoyons le lecteur à la section 4.6.

1.5 Service de Cache pour les Grilles (GCS)

Dans cette section nous décrivons les concepts et l'implémentation du composant de base pour les grilles de caches collaboratifs: le service de cache pour la grille Grid Cache Service (GCS). Ce service suit le modèle et les spécifications proposés dans le précédent chapitre.

Notre principal objectif est de travailler sur des données temporaries dans des environnements de grille et de manière dynamique. Aussi avons-nous décidé d'utiliser des groupes de caches distribués dans la grille et qui fournissent de manière collective des capacités de stockage de données temporaires [20] [24]. La figure 1.2 illustre un scénario d'utilisation de groupe de GCS pour une manipulation dynamique de données temporaires dans des environnements de grille. Un client invoque un GCS pour stocker des données temporaires; si le GCS ne dispose pas de ressource de stockage suffisante, il peut invoquer les opérations d'accès des caches distants pour stocker les données en des sites de grille multiples.

Dans ce scenario, chaque cache est un GCS qui gère de manière individuelle les ressources locales de stockage afin de fournir les capacités de stockage temporaire du système. Dans ce chapitre nous décrivons le composant propre au système: le service de cache pour la grille (GCS).

Le service de cache pour la grille est l'élément actif qui soutient l'infrastructure. Il implémente les couches de stockage, de contrôle et de collaboration du modèle de référence. Nous avons développé un prototype du GCS qui est utilisé comme démonstrateur de l'approche de cache pour la grille. Ce prototype fournit un support pour les opérations spécifiées et pour les informations de cache.

Le concept de service de cache pour la grille

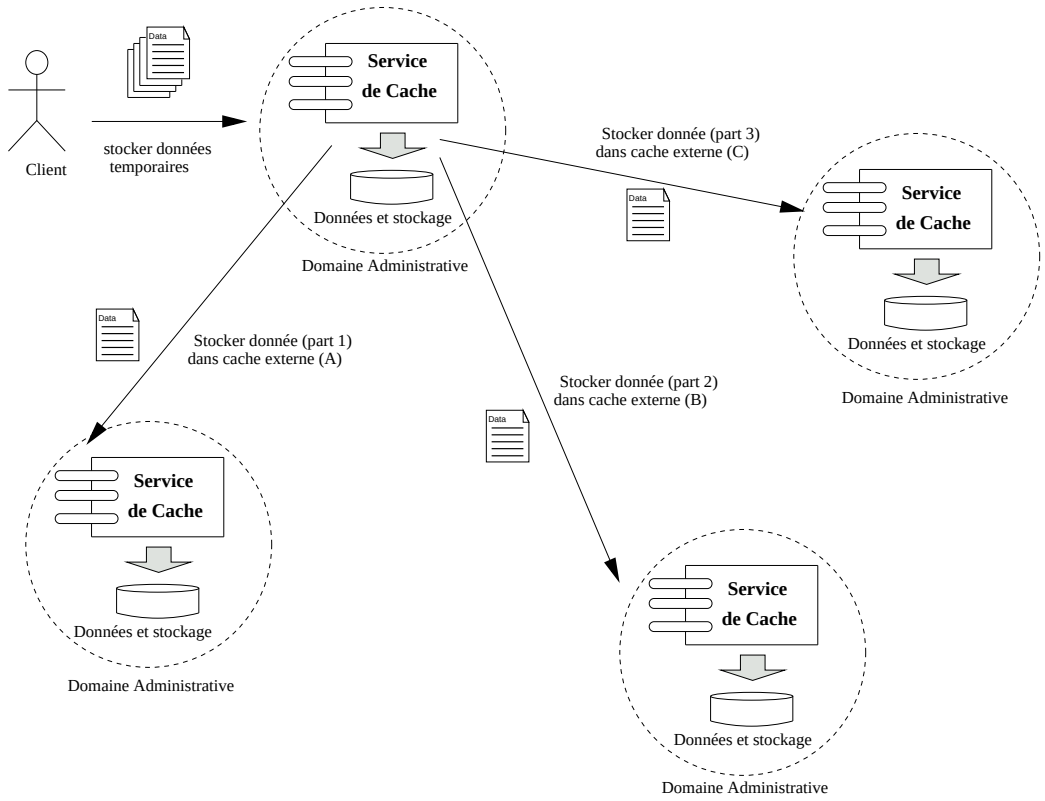

Figure 1.2: Utilisation du Service de Cache

Le GCS est un administrateur local du stockage et des données temporaries. Il est implémenté comme un service de grille qui fournit des capacités de cache de base suivant le modèle de référence. Ainsi le GCS présente des interfaces et des méthodes associées aux opérations de cache : accès, surveillance et configuration pour les opérations de données temporaires.

Les fonctions du GCS

La principale function du GCS est d'assurer qu'une entité spécifiée de donnée existe dans la resource de stockage pour une période limitée de temps et d'enregistrer les informations détaillées sur les données et les actions du cache.

Dans ce context, le GCS offre ces principales fonctions :

- Gestion des ressources de stockage sous-jacentes pour les données temporaires basées sur des techniques de cache;
- Fourniture d'espace de stockage temporaire à la demande pour stocker des entités de données (fichiers) via des opérations définies d'accès;
- Fourniture d'informations détaillées et à jour sur l'activité en données et du cache par une représentation définie des informations et par des opérations;

- Fourniture de capacités de cache de base pour des actions de surveillance des entités de données stockées dans les ressources de stockage;
- Support des opérations de caches collaboratifs pour une grande palette de clients (dont d'autres caches).

1.5.1 Implémentation d'un prototype de GCS

Nous avons développé le prototype pour la plate-forme Globus Toolkit 4. Globus Toolkit est une suite d'outils pour développer et déployer des systèmes et des applications de grille. Il respecte le standard OGSA (Open Grid Services Architecture) pour la conception de systèmes de grille. OGSA définit un standard d'architecture ouverte pour les applications basées sur la grille. Il permet de gérer des calculs distribués orientés service avec des services web. Le Web Services Resource Framework (WSRF) est une spécification destinée au développement d'applications orientées service à partir de services web. Le couple OGSA/WSRF a été standardisé par le consortium international Open Grid Forum (OGF).

L'utilisation de services de grille repose, d'une part, sur le Web Service Description Language (WSDL),c'est un langage XML qui permet la description des interfaces de services web de manière standardisée; et, d'autre part, sur un protocole d'échanges de requêtes et de réponses entre services web. Le protocole le plus utilisé pour la communication entre services web est SOAP; c'est un protocole qui permet d'échanger au format XML des messages encodés en utilisant le protocole de communication http. Le prototype a été développé dans le langage de programmation Java afin de pouvoir être exécuté dans un conteneur de web services au sein de Globus.

1.5.2 Architecture du prototype

L'implémentation du prototype du GCS est divisée en cinq principaux modules. La figure 1.3 présente l'architecture générale du prototype. Le client invoque l'opération de cache de l'interface du service en envoyant l'élément XML *OperationRequest* comme paramètre. La requête est encapsulée dans un message SOAP. L'interface de service reçoit les invocations d'opérations de cache. Ce module est implémenté comme un service de grille qui utilise les outils et les bibliothèques fournis par le Globus Toolkit. Un fichier WSDF décrit les opérations de cache comme des portTypes en utilisant l'opération comme paramètres.

L'interface de service est déployée dans le conteneur Globus WS Java Core. Le conteneur se charge de la gestion de la logistique sous-jacente liée à la communication, aux messages, aux sessions et à la sécurité. Le module gestionnaire de cache (Cache Manager) effectue les opérations de cache en se basant sur les éléments d'information de cache comme structures de données. Ce module est principalement supporté

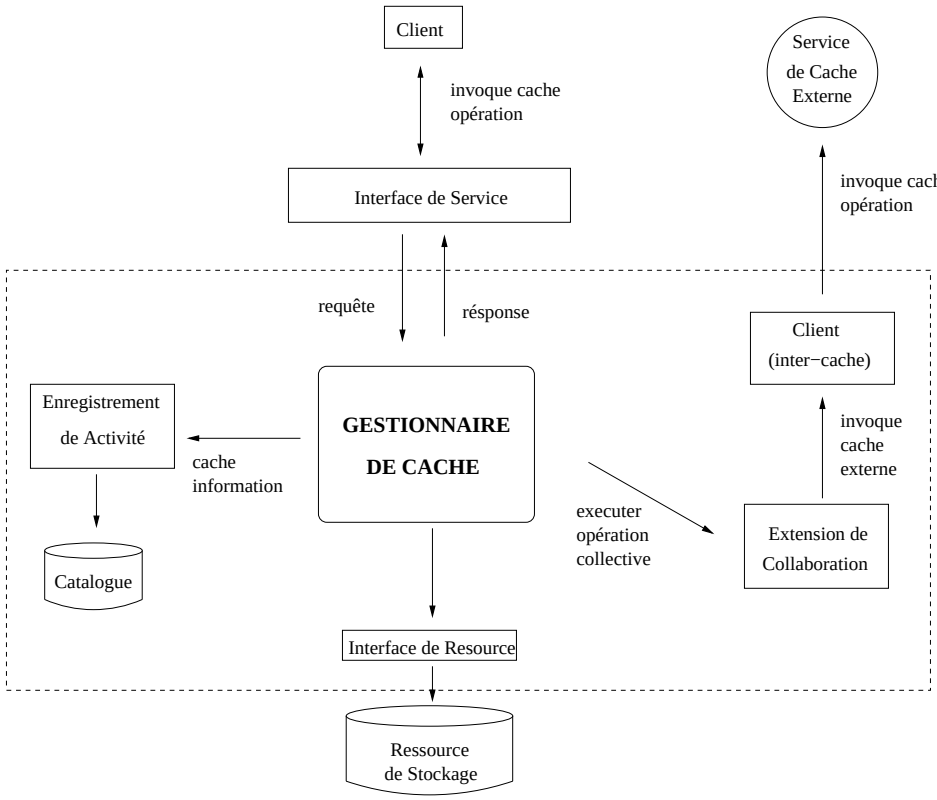

Figure 1.3: Architecture du prototype du GCS

par la classe appelée *CacheImpl*. Cette classe implémente les mécanismes de remplacement de cache et une méthode pour chaque opération de cache. Un exemple d'exécution de l'opération SetData est présenté plus loin dans la section 5.5.1.

Le module gestionnaire de cache implémente trois méthodes de remplacement de base : LRU, LFU et SIZE. Il utilise les informations d'activité enregistrées dans les éléments d'information de l'activité et des entités pour supporter les fonctions de remplacement. Il repose sur le module d'enregistrement de l'activité en données et en cache pour récupérer et mettre à jour les informations de cache sur les actions opérées.

Le module d'enregistrement de l'activité en données et en cache gère le catalogue d'activité en données et en cache. Il utilise les éléments d'information d'activité pour administrer les informations d'activité. L'extension de collaboration représente des modules d'extension qui exécute des procédures de collaboration (implémentées par des opérations et spécifiques aux besoins en collaboration). Actuellement le prototype implémente des extensions simples pour les opérations *GetData* et *SetData* qui transfèrent l'opération de requête originale aux caches distants.

Une interface de programmation (API) Java est fournie pour utiliser l'implémentation du prototype du GCS. Elle est divisée de deux groupes ou paquetages de classes : le paquetage d'information de cache, qui est composé de classes pour les définitions

des éléments d'informations de cache et d'opérations de cache, et le paquetage de l'implémentation de la logique métier du cache.

1.5.3 Publication de données avec GCS

Les clients GCS créent une requête explicite pour stocker des entités de données. Cette requête est exécutée par le GCS comme une demande pour disposer d'un espace de stockage temporaire. Nous désignons cette capacité par le terme de publication car elle permet d'exposer et de partager des données pour une durée limitée entre des clients qui utilisent le GCS. Cette capacité est principalement supportée par l'opération d'accès SetData().

La figure 1.4 montre un exemple d'en-tête de requête pour l'opération SetData. Les champs contiennent les informations utiles pour l'API cliente à l'invocation de l'opération du GCS. Le numéro de requête permet d'identifier chaque requête d'opération de manière individuelle. La destination de cache contient la référence à l'instance de service de cache pour la grille. Le champ de version est utilisé pour distinguer les futures versions du service. L'élément de requête contient aussi l'élément d'entité de donnée à publier et un élément optionnel de métadonnée qui est discutée plus loin.

```
<SET_DATA_REQUEST>
  <REQUEST_HEADER>
    <OPERATION> set_data </OPERATION>
    <TYPE_REQUEST> client </TYPE_REQUEST>
    <REQUEST_NUMBER> 454799 </REQUEST_NUMBER>
    <CACHE_ID_SOURCE></CACHE_ID_SOURCE>
    <CACHE_ID_DESTINATION>
       http://liris-7080.insa-lyon.fr:8080/wsrf/services/gridcaching/LCS
    </CACHE_ID_DESTINATION>
    <DATETIME_REQUEST>
       2007-07-26T16:00:00
    </DATETIME_REQUEST>
    <VERSION>0.1</VERSION>
  </REQUEST_HEADER>
</SET_DATA_REQUEST>
```

Figure 1.4: Un exemple d'en-tête de requête de l'opération SetData

1.5.4 Performance du Grid Cache Service

Nous présentons des mesures de performance pour notre prototype de service de cache de grille (GCS). Les tests incluent les trois sites qui participent au projet

GGM. Le site local, lequel invoque les opérations de cache, se trouve à Lyon. Les sites distants hébergent un conteneur GT4 and déploient un GCS aussi bien qu'un serveur GridFTP; ils sont situés dans les villes de Toulouse et Lille, en France. Nous avons réalisé 1000 opérations initiées depuis le site local. Pour chaque opération, un simple client GCS est créé; il invoque une requête d'opération de cache et obtient la réponse par le service de cache distant. Pour chaque opération d'accès (publication ou récupération d'entité de donnée), une connexion à GridFTP est établie. Puis le transfert de donnée est démarré entre les sites. Nous avons effectué des opérations significatives de surveillance et de gestion : les opérations de récupération de contenu de cache, de ressources de stockage utilisées et une méthode de remplacement. De plus, nous avons vérifié les opérations de configuration de la méthode de remplacement et de configuration totale du cache. Les résultats démontrent aussi que le temps nécessaire pour la gestion du cache est marginal par rapport aux transferts de données. Ces tests fournissent une vision des performances des opérations typiques et individuelles du GCS. Pour plus de détails concernant des mesures de performance de GCS nous renvoyons le lecteur à la section 5.6.

1.6 Espace de Stockage Temporaire (TSS)

Nous présentons le concept général de l'Espace de Stockage Temporaire (TSS pour Temporal Storage Space). C'est un système de gestion des données qui manipule des données temporaires de manière automatique. Le concept de TSS vise à fournir un aperçu des mécanismes de la couche de coordination. De façon similaire, il aspire à donner une notion des capacités de caches collaboratifs à exploiter avec GCS.

L'objectif de ce chapitre n'est pas d'implémenter un système TSS opérationnel mais de vérifier que le service GCS est fonctionnel, que toutes les opérations (API) sont fonctionnelles et que les procédures de coordination peuvent être implémentées de telle sorte qu'elles ouvrent d'intéressantes perspectives. En d'autres termes, ce chapitre propose une validation fonctionnelle du service et une preuve du concept de la couche de coordination. Cela explique que des paramètres soient fixés et que des choix ne soient pas discutés.

La fonction de TSS est de stocker un ensemble déterminé de données temporaires (fichiers) sur des sites distribués où des GCS sont déployés. TSS relève un espace de stockage temporaire sur chaque site pour gérer des données de grande taille. Puis chaque GCS libère automatiquement l'espace inutilisé selon les techniques de cache.

Les fonctions de TSS incluent : la découverte de l'espace, identification d'un espace de stockage disponible pour stocker les données spécifiées sur la grille; le transfert de données, déplacement efficace de données entre des ressources de stockage distantes; l'enregistrement et la mise à jour des informations de description du contenu et l'échange d'informations sur l'état d'activité des données, de manière à ce que

d'autres sites puissent découvrir des contenus de données disponibles et obtenir des informations sur l'activité d'accès à ces données. Par les opérations de cache, le service expose des informations sur l'état de chaque de la entité de donnée (fichier), dont les actions d'accès au fichier qui ont été réalisées.

Les aspects qui ont guidé les décisions fondamentales prises pour la conception de TSS sont les suivants :

- Conception d'un système à base de composants reposant sur des services GCS
- Opération basée sur des services de grille de bas niveau et réutilisables
- Utilisation des opérations de cache et des informations d'activité sur les données et le cache pour agir avec le système
- Utilisation d'un module souple de coordination pour organiser des interactions spécifiques entre les caches
- Utilisation de l'architecture OGSA et des ressources sous forme de services web
- Framework WSRF pour l'interopérabilité
- Utilisation du Globus Toolkit 4.1 comme plate-forme intergicielle

Suivant les spécifications OGSA/WSRF, la structure générale de TSS consiste en un groupe d'instances GCS qui sont chacune exposées comme un service de grille déployé dans un conteneur Globus. Le conteneur prend en charge plusieurs aspects de logistique sous-jacents liés à la communication, aux messages, aux sessions et à la sécurité. Les capacités de services de cache sont exposées à travers l'interface de service pour les utilisateurs et les applications de la grille.

Dans TSS, un module de coordination organise les interactions entre les services GCS pour traiter les requêtes adressées à un ensemble ou un sous-ensemble des caches. Il contrôle l'état et le comportement du groupe de caches. Dans le cadre de TSS, nous nommons Local Cache Service (LCS) une instance de service de cache individuelle. Chaque LCS expose des informations de cache qui sont accessibles à travers des interfaces standard de grille pour les opérations.

Pour utiliser les capacités de TSS, les clients invoquent les opérations de cache pour accéder au contenu des données de cache, obtenir des informations de surveillance sur l'activité des données et des caches et ajuster les valeurs et paramètres qui servent à la configuration de la fonction de base de l'installation du cache. Pour effectuer les opérations d'accès au cache, TSS repose sur GridFTP,qui n'est pas un service web, et sur deux autres services WSRF : RFT et Delegation.

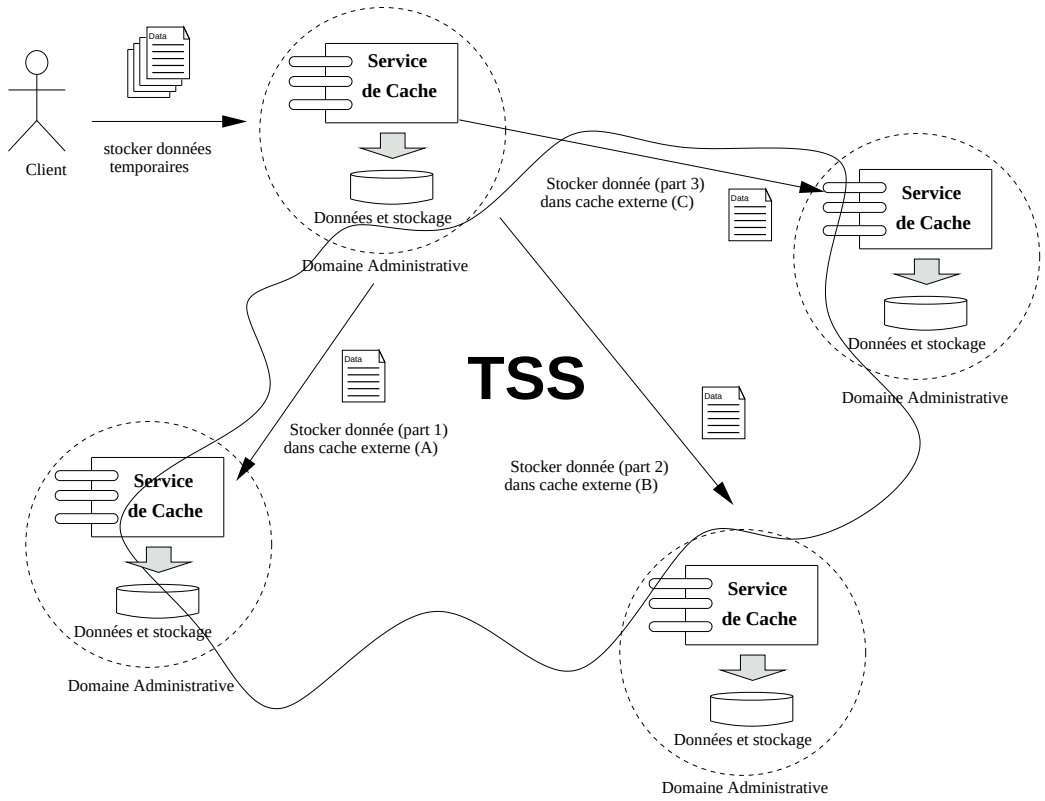

Figure 1.5: Espace de Stockage Temporaire (TSS)

TSS est composé d'un groupe de services de caches locaux (LCS), lesquels sont déployés dans de multiples domaines d'organisation, et d'un service de cache collectif (CCS pour Collective Cache Service). Les LCS travaillent ensemble pour fournir un support de base à TSS. La figure 1.5 illustre les composants TSS déployés dans une grille.

Le Service de Cache Local (LCS pour Local Cache Service)

Le service de cache local (LCS) est une instance de service de cache pour la grille qui est déployée sur chaque site de l'organisation virtuelle. Le LCS fournit des fonctions de cache pour des applications internes et pour les utilisateurs clients de l'organisation. Lorsqu'il passe en mode de travail collectif, il étend les capacités de cache avec le support des services de cache des autres grilles qui composent le TSS.

D'un point de vue interne, le LCS, à l'instar d'une passerelle, prend en charge les opérations soumises par les clients de l'organisation. D'un point de vue externe, il assure les opérations sollicitées par d'autres LCS. Les LCS traduisent les opérations d'accès aux données internes en opérations inter-caches en invoquant des opérations semblables depuis le LCS déployé dans d'autres organisations de la communauté virtuelle.

Le Service de Cache Collectif (CCS pour Collective Cache Service)

Le service de cache collectif (CCS) est un service qui permet aux LCS de coordonner leurs interactions. Le CSS accepte les mêmes interfaces d'opérations de un GCS mais le traitement est différent. Les invocations reçues aux interfaces du CCS sont traduites en interactions de pair à pair en incluant deux LCS ou davantage. Le CCS est un service spécial pour un groupe de GCS. Il accepte les opérations d'accès de la part du LCS et des opérations de surveillance et de configuration de la part d'utilisateurs ou d'administrateurs de ressources de grille.

Le CCS exécute un module spécialisé appelé coordinateur; ce dernier implémente les procédures spécifiques aux décisions collectives. Le coordinateur invoque des classes implémentées par des utilisateurs et qui sont spécifiques aux besoins des applications. Il transforme sa tâche en invocation d'opérations de cache auprès de plusieurs LCS pour organiser les interactions. Il implémente la logique qui incorpore les règles de réalisation des actions de caches collectifs basées sur l'utilisation des opérations de cache.

Les LCS et CSS apportent les capacités principales de l'espace de stockage temporaire (TSS). Ces capacités sont : la dissémination des données, la collecte des données, la prolifération des copies, la surveillance et la configuration.

1.6.1 Expériences

Nous présentons une expérience qu'illustre la capacité d'extensibilité offerte par le GCS. Notre but dans cette expérience est d'utiliser le système TSS comme une preuve de concept de la faisabilité et de l'efficacité de l'intégration d'instances GCS génériques dans un réseau collaboratif de caches. Pour cela, cette expérience a été conçue pour montrer, sur un exemple simple, que les GCS peuvent être organisés en un système collaboratif TSS, que les outils de surveillance founis peuvent être utilisés pour analyser le comportement de ce système, comportement individuel d'une instance de cache spécifique et comportement global du système comme un tout, et que les décisions de re-configuration peuvent être prises pendant la phase d'exécution pour optimiser les performances du système.

Nous simulons un environnement de grille constitué par un groupe de dix GCS. Cette expérience illustre la capacité d'extensibilité offerte par le GCS. Une série de simulations montre comment on peut implémenter une gestion collaborative d'un groupe d'instances de GCS qui forme un système TSS. Elle vise à illustrer plus précisément comment on peut surveiller l'activité des instances de cache, implémenter les décisions de reconfiguration et changer dynamiquement les paramètres opérationnels de cache. Nous avons implémenté et évalué trois scénarios de base liés à la capacité de GCS à stocker des données distantes, également appelées espace de stockage partagé.

Pour plus de détails concernant la simulation de TSS nous renvoyons le lecteur à la section 6.3.

En somme, ces simulations montrent que :

- Un ensemble d'instances de GCS peut être organisé pour bâtir un système TSS opérationnel
- Le coordinateur TSS a la capacité de surveiller de manière constante les instances de GCS en utilisant des méthodes de surveillance fournies
- Le coordinateur a la capacité de modifier les paramètres opérationnels des instances de GCS
- De manière spécifique, le coordinateur peut anticiper (pré-emption) l'espace de stockage affecté aux services GCS les moins actifs pour le stockage distant
- Sous des motifs classiques d'accès aux données, le coût de cette pré-emption pour les GCS est en moyenne inférieur à 1%

La série de simulations présentées dans cette section illustre comment un système collaboratif composé d'un ensemble de GCS peut fournir un stockage temporel de données à la demande.

Elle permet de valider les caractères opérationnel et effectif des API de surveillance fournies par le GCS. Elle montre aussi comment des opérations collaboratives peuvent être implémentées en invoquant des opérations qui simplifient la reconfiguration.

En particulier, ces simulations montrent comment on peut anticiper l'espace de stockage sur les GCS les moins actifs. Cet "espace de stockage pré-empté" peut être utilisé pour optimiser et équilibrer le comportement global du système en stockant les données de manière distante, en dupliquant les données les plus sollicitées ou en migrant les données vers des instances de cache moins actives.

Cette optimisation devrait être basée sur la surveillance des motifs d'accès aux données et des conditions opérationnelles de la grille. Des expériences sont en cours pour évaluer le bénéfice, en termes de taux de succès, de différentes heuristiques d'optimisation sous des conditions variées de motifs d'accès, de trafic du réseau ou d'utilisation du processeur.

1.7 Conclusion

Au cours de la thèse, nous avons examiné l'utilisation des caches collaboratifs dans les grilles de calcul. Les grilles permettent l'exécution de multiples modèles de calcul distribué qui manipulent grandes entités de données d'une façon distribuée.

Une quantité significative de ces données est utilisée seulement pour une période limitée. Par conséquent, l'administration de données temporaires dans les grilles est critique et rendent la coordination globale des ressources de stockage nécessaire. Dans ce contexte, les systèmes de caches sont reconnus comme les techniques les plus efficaces pour la gestion des données temporaires et le cache collaboratif est traditionnellement proposé pour ajuster les capacités de cache aux besoins croissants dans les environnements distribués.

Cette thèse propose une approche de la conception et de l'implémentation de tels systèmes de cache collaboratif dans les grilles de données, systèmes offrant les fonctionnalités de contrôle et de gestion des interactions entre éléments de la collaboration. Notre approche permet la composition et l'évaluation des fonctions dun système de cache collaboratif de haut niveau de façon flexible. Notre proposition est basée sur un modèle multicouche qui définit les fonctions principales d'un système de cache collaboratif pour les grilles. Ces capacités sont exécutées par un ensemble d'opérations communes et standardisées qui permettent l'access aux données, le contrôle et la configuration d'un cache. Ce modèle et la spécification fournie sont utilisés pour construire une infrastructure logicielle flexible et générique pour l'opération et le contrôle du cache collaboratif.

Dans notre première contribution, la spécification du Cache Collaboratif pour les Grilles, nous décrivons les principaux aspects du cache collaboratif de grille. Nous fournissons les spécifications des composants de base sur lesquels construire un tel infrastructure, générique et ouverte. Ces spécifications comportent les trois éléments principaux suivants : Modèle multicouches composé par les couches de stockage (storage), contrôle (control), collaboration et coordination. Opérations de cache : ensemble d'opérations standards véhiculant des informations normalisées.

Notre deuxième contribution, le Service de Cache pour les Grilles (GCS), est l'élément actif qui soutient l'infrastructure. Il implémente les couches de stockage, de contrôle et de collaboration du modèle de référence. Nous avons développé un prototype du GCS qui est utilisé comme démonstrateur de l'approche de cache pour la grille. Ce prototype fournit un support pour les opérations spécifiées et pour les informations de cache. Nous avons présenté des mesures de performance pour notre prototype de service de cache de grille au sein du projet GGM. Les résultats démontrent que le temps nécessaire pour la gestion du cache est marginal par rapport aux transferts de données.

Notre troisième contribution,l'Espace de Stockage Temporaire (TSS), c'est un système de gestion des données qui manipule des données temporaires de manière automatique. Le concept de TSS vise à fournir un aperçu des mécanismes de la couche de coordination. Il présente une validation fonctionnelle du GCS en utilisant les opérations (API) et les procédures de coordination qui peuvent être implémentées. Nous avons présenté une expérience qui illustre la capacité d'extensibilité offerte par le GCS. Elle vise à illustrer plus précisément comment on peut surveiller l'activité des instances de cache, implémenter les décisions de reconfiguration et changer

dynamiquement les paramètres opérationnels de cache. La série de simulations présentées illustre comment un système collaboratif composé d'un ensemble de GCS peut fournir un stockage temporel de données à la demande.

Les perspectives de ce travail sont en direction de la gestion de la coordination, de l'extensibilité, de la tolérance aux pannes des GCS et de la construction d'organisations de caches collaboratifs. Nous prévoyons de poursuivre le développement de l'intergiciel TSS.

Chapter 2

Introduction

Resource sharing has been a major topic from the beginning of the computer era. At first, system users would book time on the central machine and have the computer for themselves while they were using it. With batch systems, users would submit their tasks to the system which executes them one at time. Timesharing systems have been developed to manage the machines so the users could share a computer with the illusion that they had it to themselves. This was the first system to manage resource sharing; since then the technological evolution has made it possible to create the Internet.

The most important shared resources are data. Data represent useful information for users and applications. Data are used to share knowledge. The Internet was created to share all kinds of data. The main success of the Internet is to make possible the sharing for data in world-wide form. As a consequence, the requirement for data storage and movement constantly grows.

Although the main goal of the Internet is to share data, Internet technologies can be used to share other resources like computing processing, storage space, and devices for signal detection or measurement. The main problem for sharing resources on the Internet is that systems and mechanisms that control these resources are different, so they are not able to cooperate. A considerable effort is required to make these systems and resources interoperate in coordinated way.

Grid technology enables the integration and sharing of heterogeneous and distributed resources between different administrative domains; its objective is to provide dynamic access to these facilities when needed. Grid technology is a result of the evolution of distributed system and the Internet. It makes it possible for several institutions to establish "virtual organisations" supported by their computing infrastructure. It provides access to distributed computing power and data repositories on demand, and allows partners to share computing resources between users and applications.

Grid development has been motivated by the need to gather and coordinate enough computer resources to solve challenging problems such as: physical and chemical processes, weather and climate models, biological and human genome structures, geological and seismic activity.

Grid applications attempt to solve these complex problems. The main objective of these applications is often related with intensive analysis and processing of large amounts of data. There are two reasons that make this task substantially critical. The first one is that data grow quickly and constantly; this requires more and more storage capacity. Second, these datasets are produced, stored, and used in a distributed way, so they necessarily have to be moved.

Data movement requires consumption of important network bandwidth and storage capacity; without global coordination the resources are used in an inefficient and non cost-effective way. A significant quantity of these data are used only for a limited period of time, especially when a portion of data is analysed or processed, thus an important proportion of data disseminated in a distributed system, like a grid, is temporary. In this respect, an advanced system is necessary to manage temporary data dissemination in distributed systems.

Sharing of grid resource capacities creates enormous expectations for collaboration: Users dream of having infinite computing power for applications and unlimited storage space to put their data whenever they need it. As timesharing designers, grid architects have to provide this perception with limited and shared resources.

Data movement across distributed systems represents active data; therefore data processing and movement gives additional value to data. Interesting data have a lot of activity: This increases the probability for their sharing and reuse.

Grid power is based on the capacity to gather resources from different origins which are geographically distributed; data is naturally moved between different places. Therefore, considerable capabilities of grid are invested to deal with data movement that is temporary. Managing data movement in a grid has become a main preoccupation as a consequence of data growth in relation with bandwidth and storage. In this respect, there are propositions to change the paradigm of distributed systems. For example a Datacentric grid [109] proposal suggests to make data immovable but make the code mobile in grids, with considerable technological and economic implications that make it too hard to achieve in the near or not so distant future.

Interest in temporary data is more economically than technologically biased: the storage management infrastructure for temporary data is more cost-effective than the infrastructure for persistent data. There are two reasons to explain this difference. The first is that the storage infrastructure for temporary data needs to be increased less quickly. Second, the infrastructure is reused constantly. The point here is the way that these resources are managed to help increase levels of availability, response time, throughput and utilisation. Management permits the greatest possible benefit

to be obtained from these shared and distributed resources.

Caching is recognised as one of the most effective techniques to manage temporary data. A cache is a system that automatically controls data objects contained for a limited period of time in a storage resource. A cache allows reuse and sharing of data between different clients, helping data resources to be used more efficiently. Collaborative cache systems, like web caching, have been proposed for distributed systems to scale cache capabilities. Collaborative cache systems, in this sense, appear as a natural solution to manage temporary data in a grid.

The specification of grid caching is necessary to develop the different aspects related with the deployment of collaborative cache systems into a structural model: such specification must describe the essential characteristics to be satisfied by the grid cache. It must also permit to exploit the grid and caching capabilities conjointly. The grid community has not really elaborated this specification yet.

This thesis provides a specification for the base components to build caching in grids. These components permit to compose an infrastructure to operate distributed temporary data. The structure for this research is provided by the development of a collaborative cache system for grids and the specific requirements of grid applications. In this context, where users expect infinite access to resources and where resources are limited and shared, the management of data resources arises as a central question.

2.1 Motivation

In this section, we examine the grid caching scenario presented in the introduction in relation to automatic operation of large volumes of temporary data. We present two use-cases within this scenario and utilise them to derive constraints and requirements for a generic and flexible approach to collaborative caching in a grid environment. The first use-case presents the same fundamental requirement from different perspectives.

2.1.1 Use cases

Gathering storage capacity from different locations

Perspective 1: Data Dissemination

A genome research centre develops several projects. These projects address large problems related with the analysis of genetic information of complex protein se-

quences. The processing data are stored in a single repository; they were produced from previous experiments and collected for posterior usage. Project researchers want to use the grid infrastructure to realise new experiments that need intensive data processing.

In order to save time, researchers want to execute multiple tasks at the same time; they want, similarly, to take advantage of non-local information using available computing resources in the grid when local computing resources are scarce. To do so, researchers use a grid interface to find and reserve available computing resources to perform the necessary operations. These processing resources will be available for a specific period of time.

Applications that compute large datasets cannot assume or control the data placement and must take of the account resources required to move data among locations, so data must be placed near the processing units. Likewise, users and applications want to transparently access local and remote data using the same tools and interfaces. When the computing resources are available, users need to find enough temporary storage in each location to hold large datasets that are fetched temporarily. Researchers want this procedure to be done with faster response times. The experiments execute diverse tasks in several locations using partner resources.

Perspective 2: Temporary Storage Space

An astrophysics research project studies the evolution of stars and galaxies. A particular astronomical phenomenon is rarely and sporadically produced. Astronomers decide to capture detailed information about this phenomenon during its occurrence. Different telescopes are used to capture the electromagnetic radiation produced during the same night. A huge amount of data is produced in a short time period and the project has a limit on the amount of storage.

Astronomers want to use the grid because the data produced cannot be collected in a single location and then processed using sequential process. Users want to delegate the task of gathering storage capacity from distributed locations. Similarly, they want to be able to use specialised storage mechanisms that are managed by heterogeneous systems.

Additionally, researchers want these data to be available a wide kind of users, applications, mechanisms and systems that need to access and reuse them for other purposes. Users require that the collection of distributed resources and data appear as a single and coherent system. This system permits to make operations on temporary data with a high degree of transparency in relation with location or individual resource technology. Users want to use the grid as a hypothetical infinite storage repository that provides huge capabilities with fast response times.

Grid Administration

This use-case is motivated by the use-case presented before. In each perspective, a resource administrator maintains the proper operation of resources and services on the grid. The administrator also supports the deployment of new applications and services on the grid required by new projects.

The resource administrator wants to observe and analyse the status and behaviour of the diverse resources used to operate temporary data: he gets a periodic report about actions in progress. During the processing of an experience a particular storage system goes down; at this point a report is generated to the resource administrator. Then a new request for getting more space capacity is launched to replace the missing capacity.

At the same time, the resource administrator focuses on response time in order to assess performance levels. He makes sure that the response time is increasing. He decides to gather information about the level of resource utilisation. To do so, the resource administrator uses a grid tool to collect detailed status and configuration information. After analysing the information, he determines locations with an excessive level of utilisation of the storage resource and also locations with low levels.

At this point, he decides to modify operation and control parameters of the group of components that operates temporary data. He adds the operation rate as a new criterion for the selection of location where future datasets are placed. With this corrective action in mind, he can change the distribution of work load through grid locations. He then supervises the operation of resources and data to evaluate the effect on performance.

2.2 Aspects of the Management of Temporary Data on a Grid

There are a lot of issues related with the management of temporary data with cooperative cache on Grids. We divide these aspects into three thematic groups: Caching, cooperative caching and management of collective operation. Several of the issues refered here are not dealt directly in this thesis. Our intention is to give first the largest overview of the problem so that the reader can see the complexity of it. In the Section 2.5 we present the particular topics that are addressed in this work.

2.2.1 Aspects Related with Caching

- *Placement of cache on grid*

  Placement deals with where to place a cache in order to achieve optimal functionality and performance. Cache placement includes localisation in relation with access interactions between applications and data sources. The placement of cache elements is a fundamental design decision: it is determined by application requirements and system objectives. Cache placement plays an essential role in interaction capabilities.

  The selection of the location where such temporary data can be accessed is capital for the cache system design; the main objective is to select an accessible and convenient location to store large datasets that are needed by applications. Cached data must then be easily and quickly available for shared use in all applications.

- *Integration of heterogeneous resources*

  This aspect deals with the incorporation of storage resources to keep temporary data; these resources are mutually provided by several partners. These resources belong to diverse technologies and they are controlled by distinct software systems. Usually, each storage mechanism has proprietary access interfaces available for specialised purposes or functions. The grid deals with access to these resources with the same interfaces, and specialised storage resources are designed to store temporary data. These resources have particular properties or technology to provide high throughput on data access.

- *Data selection*

  Data selection deals with the decision about which data entities are held by the system. The use of these data for current processing is a main interest for the system. Similarly, this includes the time period during which data is functional and potentially reusable.

  The decision process includes that data is held during a specific period of time over which it can be reused. A lot of strategies have been proposed in the literature for determining which data should be kept in temporary storage.

- *Regulation of performed actions*

  The regulation of actions deals with the usage regulation of the resource itself in a mutual environment. This includes access to content in the storage resource, who can use the temporary data, and what kinds of operations are allowed.

The nature of data is different and applications use these data in different ways. This aspect concerns operations that are allowed and users who are authorised to make these operations. In this context, control also includes the capacity of regulating storage resource usage.

- *Data coherency*

  Interaction with data between users often requires a strategy for representing a shared state. There are different ways of keeping a data state consistent in order to allow manipulations in that state. This aspect includes how to keep temporary data in coordination with changes to the data in the original data source; if the original data or a temporary copy changes, the copies must be updated to reflect those changes. The importance of keeping temporary data updated varies between applications.

  Data consistency is a specialised topic widely developed in general distributed data systems, it is not an issue for which the grid creates particular new challenges. In this work, we assume that data is accessed in read-only manner; therefore data coherency is given only limited consideration.

- *Effectiveness*

  Effectiveness deals with the degree to which the system responds to the requested operations and the level of resources used to provide these operations. The principal means of judging if the system meets the requirements is that specified levels are maintained to the satisfaction of application processing. Effectiveness indicators thus have the highest interest. Effectiveness includes issues like availability, response time, and success rate of access operations.

2.2.2 Aspects Related with Cooperative Caching

- *Interaction*

  Interaction deals with actions that can be done together by different system elements (what kinds of actions it is possible to make in conjunction, the coordination applied to these actions to operate and control temporary data between grid members). Temporary data management needs the execution of sophisticated operations between several components.

- *Organisation*

  Organisation refers to necessary arrangements of system elements to operate and control in a common way temporary data. The organisation is strongly related with expected interactions, functions and performance to be supported

by the system. There are diverse alternatives to organise the distributed system components; they depend on requirements and application objectives. The organisation is often represented as architectures and schemes conformed to by specialised components.

- *Distributed data placement*

  The possibility of sharing available storage resources that are distributed is an important issue to deal with when putting temporary data in a particular location. This decision is essential for the function and the performance of the system. Choosing the location for holding temporary data in a dynamic way is a complex process that requires detailed and accurate information about temporary data access and movement in the whole system.

- *Data localisation*

  In contrast with data placement, data localisation deals with the process to establish which location contains a requested piece of data. The process is complex because the resources are distributed and they are continuously changing. Data localisation includes the search process in function of data content or description. In practice, users want to get a specific dataset that contains particular information; strategies for temporary data content description are needed to provide this capacity.

- *Load balancing*

  Another important issue is the distribution of work load through grid locations. If a single location is bottlenecked, it causes a performance degradation of a portion of grid or even slow operation of the entire system. Allocating strategies are indispensable for sharing different task responsibilities. At the same time, this distribution must be in accordance with requirements and properties of grid applications.

- *Interoperability*

  Interoperability makes it possible to executed defined interactions between grid components that operate temporary data. This includes necessary information to be exchanged for effective and coordinated common operation. Similarly, the interoperability includes the definition and implementation of common interfaces that will be used by components.

2.2.3 Aspects Related with Collective Operation

- *Interactions*

Temporary data management can be achieved as a result of the realisation of the sequence of local actions and interactions between locations that must have an intended effect. Operation is an essential issue because we need to ensure that the series of events are done with particular conditions. The capacity for planning and following these actions, executed in an indirect way by different components, is a fundamental condition for system objectives.

- *Control and Registry*

  Control deals with the capacity to direct and regulate actions and interactions in order to operate temporary data. Control determines the behaviour of the entire system over time in a dynamic way. This issue is particularly important when components are heterogeneous and are administrated by different domains. Different levels of control must be established to respond to diverse requirements. Control enables the capacity to make adjustments to improve the performance.

- *Components Configuration*

  Configuration deals with the possibility of changing the operation and control parameters for a particular purpose. Configuration is an arrangement of functional units according to their nature and key characteristics. Configuration is concerned with maintaining, adding, and updating the relationships among components and the status of components themselves during operation. Configuration often permits to choose the schemes and strategies that affect the system function. Different states or configurations deal with particular or specific requirements.

- *Activity Monitoring*

  Monitoring deals with the capacity of supervising, observing, and testing activities and appropriately reporting these measurements to evaluate the effect in the functional system. Monitoring makes the periodic or continuous measurement of data actions and resource usage; this provides an ongoing verification of progress toward the achievement of goals. Monitoring includes several aspects such are the definition of monitored information and how to get this information from the specific resource in a distributed system like the grid, this includes how to obtain this information from a group of resources. In addition, monitoring includes how that information is used for system operation and control.

- *Resource Use Accounting*

  Accounting is the act of collecting information about resource usage for the purpose of capacity analysis, cost allocation and charging. Accounting processes require that resource consumption be measured, rated, assigned, and

registered between participants. Resource use tracing permits to establish access abuses or burdening infrastructure at the expense of other applications or users. Similarly, an inefficient use reports information that enables decisions to be made.

In a temporary data context, accounting is also the process of keeping track of data activity while accessing storage resources; including the amount of time spent in the network, the services accessed, and the amount of data transferred during the session. Accounting data activity information is used for trend analysis, capacity planning, billing, auditing and for cost allocation of data and storage resources.

- *Fault Tolerance*

  Temporary data management between distributed locations requires maintain the operation of the system as a whole. Considering the several resources working in parallel and the long execution time, in the case of the failure of one or more of the elements, the others can continue to operate with minimal impact. When a fault occurs it is important to determinate, as fast as possible, exactly where the fault is and to reconfigure or modify the components in such a way to minimise the impact of the operation. Fault tolerance is not an issue studied in this thesis. We assume that the proposed infrastructure allows develop many fault tolerance mechanisms in the future.

- *Security*

  Security is concerned with protecting access and controlling facilities. Applications and users can only perform actions that have been allowed. This involves specifying and implementing a security policy. The actions in question can be reduced to operations of access, modification and deletion. Passwords and other authorisation or access control mechanisms must be maintained. Security in the temporary data context is also concerned with the confidentiality and the integrity of data on medium storage. Confidentiality refers to the protection against accidental or malicious disclosure to third parties. As for integrity, it refers to the protection of data against unauthorised modification.

  Security is not an issue studied in this thesis. We assume that the users are authorised and authenticated by a service or system defined by the community that deploys the grid. Other thesis in our research team propose solutions for the protection of confidential data on Grids [105] [107] [106].

- *Performance and Effectiveness*

  Performance deals with the timelines with which the system responds to the requested operations and the level of resources used to perform these operations. Performance includes effectiveness of operations and efficiency aspects

such as the success rate of access data operations and the resource usage capacity percentage.

To deal with these concerns, components must be monitored to assess performance levels. This includes associating appropriate metrics and values with relevant indicators of different levels of performance. Performance needs to monitor many actions and interactions to provide information in determining operating level. By collecting this information, analysing it, and by using the resultant analysis as feedback to the old configuration and parameters, grid administrators can become more adept at recognising situations and improving performance.

2.3 Requirements

This section presents the general requirements for grid caching. They are motivated by the use-cases presented before. We divide these requirements into two groups: Local and Collective Operation requiremts. Several of the requirements refered here are not dealt directly in this thesis. In the Section 2.5 we mention the particular requirements that are addressed in this work.

2.3.1 Local Operation Requirements

The requirements for local operation of temporary data in grid enviroments are:

- *Delegation of temporary data operation*

  From users and applications' points of view, it should be possible to delegate the responsibility of the operation of temporary data to a specialised entity that applies sophisticated strategies for determining which data should be kept in the storage resource. At the same time, it must be possible to regulate the access to data and storage as shared resources. In the first use-case, applications and users delegate the complex task of operating temporary data to specialised grid components. These grid components control individual storage resources on behalf of users and applications. Furthermore, these components implement the mechanisms of interaction in order to work together.

- *Accessibility by different types of clients*

  Entities and capabilities related with temporary data access must be available for the grid environment. These capabilities must be available for different

kinds of users, applications and services. In the first use-case, different tasks are realised by diverse applications in multiple locations. In each location, applications require to execute access operations to manipulate the temporary data. The differences related with each particular technology must be transparent for clients.

- *The uniformity of operations and interfaces*

  The operations and interfaces necessary to handle temporary data must be uniform. The components that operate temporary data must support a standard set of operations and interfaces. In the first use-case, the same action to find enough storage space is launched on several locations. In each location, the request must be interpreted in the same form. In the second use-case the resource administrator gathers information about the level of capacity of resource utilisation. Here, monitoring information must be represented in a common form.

- *The capacity to gather resources on demand*

  Storage space for temporary data must be provided incrementally following the availability of shared resources. This need is also caused by the non-predictive character of temporary data. Supplying the requirements on demand gets more importance in grid environments because of the dynamic character of the resources. In the first use-case, users try to gather storage capacity from different and distributed sources and locations. Additionally, each location tries to get the maximum storage capacity. In this case different strategies can be applied to obtain the capacity desired from the shared asset.

- *Optimization for efficient use of resources*

  Users expect to get unlimited storage resources for storing their data. Resources however are finite and their capacity limited. Sharing strategies should provide enough resources for the enormous storage space requested. Within this context, it is necessary to use resources with maximal efficiency. In our second use-case, the resource administrator needs to optimise the utilisation of the group of available resources. The administrator needs to collect information that will help him to recognise situations where the resources are not used in the best way so the appropriate corrective actions can be taken.

2.3.2 Collective Operation Requirements

This section presents the requirements for a proposed system in relation with some general aspects of collective temporary data operation. These requirements are motivated by use-cases presented in Section 2.1.1.

- *Accounting data activity*

  It is necessary to collect information about resource usage and data actions performed by the system. Accounting processes require that the resource consumption be measured, rated, assigned, and registered between participants. Tracing permits to establish the degree of efficiency of resource use. Furthermore, it permits to establish the behaviour, state, and activity for individual components and evaluate the effect on the collective system.

  In the second use-case, it is necessary to establish the level of capacity throughout the progress of resource utilisation through a specific time period. In the same way, that information can be compared with other traced items of similar components. This permits to establish if there is a trend of unbalanced resource utilisation between locations. Data activity accounting is highly important to make a general analysis and evaluation of the system function. Finally, it is also necessary to establish use trends and access patterns.

- *Flexibility to choose schemes and strategies*

  Many questions must be raised in every situation related with the management of distributed temporary data. Each situation has particular characteristics and needs that make it difficult to propose a universal solution. A temporary data management system must be able to deal with a large number of strategies, parameters and options. Depending upon the choices made in each situation, a variety of schemes and strategies are available for the effective and efficient control and operation of temporary data.

  In the second use-case, in case of a modification of the relationships among components and locations in the distribution of workload, the administrator must change the operation of the system to get the expected effect. This capacity configuration permits a choice to be among the schemes and strategies that affect the system function.

- *Performance Monitoring*

  An absolute prerequisite for the management of temporary data is the ability to measure the performance of data operations, we cannot hope to manage and control a system or an activity unless we can monitor its performance. One of the difficulties for performance monitoring is to obtain and use the appropriate information that describes performance. This information must be fully specified.

  In the second use-case, the resource administrator requests appropriated information in order to determine if the main cause of a long response time is the

excessive level of utilisation of some resources in comparison to others. Detailed information permits to take the decision for the appropriate correction. In this way, performance monitoring must provide information about

  - availability, percentage of time that components are available
  - response time, time to execute an action
  - effectiveness, percentage of success operations
  - throughput, rate of operations processed
  - utilisation, percentage of capacity used

- *Enabling the control of storage resource*

  The infrastructure to operate on temporary data in the grid is gathered from diverse resources provided from multiple locations. Each location must permit the use of its resources for partners (in an automatic way to store data). This capacity must be available for the grid and is operated by the management system of temporary data. In the second use-case, the resource administrator modifies some operations and control parameters of the components of the system. This includes remote components that provide functions to command their individual operations. This functions must be available for the collective system.

- *Coordination for effective operation*

  The use of distributed resources and components requires a proper interaction relation to act together effectively. This requires that components support operations to make possible common organisational actions addressed to get a global effect on the functional system. In the second use-case, changing the distribution of work load through grid locations requires that coherent actions regulate work distribution between components.

- *Detection and faults*

  The coordination of temporary data between different locations requires maintain once of the proper operation of the system as a whole. Typically, several resources and components work in parallel; if a fault occurs it is important to determine, as rapidly as possible, where the fault is exactly; and to reconfigure or modify the components in such a way as to minimise the impact on operation. In the second use-case, the resource administrator needs to get opportune and detailed information about the failure. This information must help him determine the problem. With this information, the resource administrator must be able to recover the system to a functioning state.

2.4 Constraints

This section presents the general constraints for grid caching. Should be noted that they are general contrains of grid environments. They also are motivated by the use-cases presented before. Several of the constraints refered here are not dealt directly in this thesis. In the Section 2.5 we mention the particular constraints that are addressed in this work.

- *Autonomy of resource control*

  The systems that hold temporary data are managed by local software which applies particular strategies to control and operate resources and data. Each installation has particular characteristics, functions and capabilities that are operated locally. Originally, these installations were built to operate in an individual and autonomous form. Sharing mechanisms must deal with these self-control capacities. This constraint includes the configuration options that are dependent on particular solutions.

  In our second use-case, local installations operate in an autonomous way. They must support configurable parameters that permit to give coherency to collective interactions. In this context, they need a structure that permits an interaction level to be built from their autonomous capabilities.

- *Dynamicness of distributed environments*

  In grid environments, resources are provided in a dynamic way following their availability. Additionally, resource utilisation and grid activity change constantly; so grid activity can present fluctuations that affect specific operations such as data actions. The management of temporary data must adapt to the dynamic changes of application actions and grid environments. The adaptability involves several aspects: interactions between application requirements and components capabilities, and interactions between components in reaction to these requirements.

  In our first use-case, the research experience requires the provision of storage capabilities in a short time period; the system can then react following the available resources. Such ability must be supported by each component that responds applying strategies that modify its activity at that moment. The ability to deal with the dynamic character of grid environment is essential to achieve the expected performance.

- *Heterogeneity integration*

Grids aim at making resources and capabilities that are supported by different operating systems and software solutions work together with diverse configurations. Storage resources span a range of architectures and technologies that make the interaction between components complex. These components must be available throughout the grid in a transparent form; hiding the particularities of the underlying system. Therefore, the management of temporary data should reduce to a minimum the dependency of specific mechanisms and solutions, and should provide a maximum of accessibility. In our first use-case, research projects should be able to use different and specialised storage solutions even though they are managed by systems that are not compatible together at first sight.

- *Distribution transparency*

  Since data and resources are distributed through several locations in the grid, the access operations to temporary data should not depend on location. Perception from users and applications about available capabilities must be similar throughout the grid. The management system of temporary data must provide interactions in a unique and coherent way.

  In our first use-case, data is distributed through different locations; user applications require that the data stored in the grid appears as a single and coherent repository. This must permit access operations to be executed without having to manage the details of data distribution.

- *Coordination scalability*

  Grid potential is based on the capacity to provide a huge amount of resources for a large group of users. Solutions that work well in a small scale may not have success when they are deployed in a larger scale. Therefore, the temporary data management system must support intensive usage situations that involve large amounts of resources and users.

  Usually, grid solutions should not be based on centralised systems. In our first use-case, the system gathers storage capacity from distributed locations. This must permit the system to be expanded by adding distributed and autonomous components. In the first use-case, it is clear that the level of required scalability must be established in a dynamic way following temporal grid conditions.

- *Multiple administrative domains*

  Typically, the majority of system administrators would not consent to furnish storage resources on a grid that would imply to give up complete control over these devices and mechanisms. The temporary data management system must therefore permit organisations to delegate administrative powers over the

storage resource so that it can be automatically and collectively managed by a specialised system.

The first use-case illustrates this need: the temporary data management system permits used local resources under conditions clearly defined in relation with duration and authorised operations. The management system acts on behalf of the organisation and the system administrator.

2.5 Challenge and Positioning

Grids create enormous expectations based on sharing resources and collaboration. Users thus dream of having infinite computing power for applications and unlimited storage space to put their data whenever they need it. Users expect to use a grid as a hypothetical infinite storage system that allows to store large data entities. Grid architects have the challenge to provide this perception with limited and shared resources.

A Grid is a dynamic environment that manages different models of computation with various data access patterns. For this reason, grid environments need high flexibility to dynamically control the configuration and behaviour of individual components. Similarly, the relationships between components determine the capabilities that can be provided by the grid. Increasing the scope of the cache interactions creates new possibilities for the management of temporary data.

The main objective of grid is to make possible the sharing of resources in a wide scale. Sharing computing, communications and data resources originates new possibilities for sophisticated forms of cooperation based on grid technologies. These cooperation modes can be used to improve grid functions and services.

Increasing the possibilities of interactions between components, however, is not sufficient to support collaborative systems like cooperative caching. Interactions must be coordinated so as to take advantage of collective work capacities. The coordination consist of the ability to operate and control the main aspects related with the configuration and behaviour of collective interactions. In this way, a group of distributed and autonomous cache components can work as a whole.

The main challenge of grid caching is the dynamical management of collaboration between caches. It consists in a high-level procedures to operate and control the coordination of collaborative cache interactions. Grid caching management also includes the strategies to optimise and improve different cooperation mechanisms and schemes.

A flexible and generic infrastructure to develop grid caching is necessary. It must

describe the essential characteristics to be satisfied by grid caching. In this thesis, the main interest is centred on the specification of this infrastructure into a structural model for the management of collaborative caches in grids. We believe that this is the fundamental condition that permits to integrate most of the different aspects related with operation of temporary data in grid.

We propose a specification to build a grid caching infrastructure that defines the base components and their interactions, which are used to compose high-level collaborative cache functions in a flexible way.

The specification defines the grid caching capabilities within a multilayer *model* that defines and separates their main functions. The capabilities are implemented by a set of *operations* that support defined cache functions. The autonomous cache elements called *Grid Cache Services* are the functional units of the infrastructure that implements the model and operations. These concepts are developed in Chapters 4 and 5.

The majority of constrains mentioned in Section 2.4 are general grid topics addressed directly by grid technologies which our infrastructure is developed. However, infrastructure focus is put on the autonomy of resource control.

The infrastructure proposed focuses on the following elements in relation with requirements mentioned in Section 2.3: delegation of temporary data operation, accessibility by different types of clients, accounting and performance monitoring, flexibility to choose schemes and strategies and coordination for effective operation. In Chapter 6 we explore forms to deal the following requirements: the capacity to gather resources on demand, optimization for efficient use of resources. We do not deal the fault tolerance which is large topic that we propose as future research.

Chapter 3

Related Work

3.1 Cache Systems

3.1.1 Overview

In the context of this work, a cache is a mechanism that manages temporary data contained in a storage resource. Caches manage data that are expected to be reused and shared for a limited period of time. Caches provide significative benefit: when accessing data stored in the cache it is faster and less expensive to access as from the original source. The performance benefit is usually in terms of access time, the other benefit derives from less expensive network and storage resources usage.

As a mechanism that manages data, caches control the store where the data is available during the time that they can be used and reused. Thus the data access operations are realized by way of a cache. Every time a particular dataset is put in temporary storage the cache mechanism conserves that data copy for later use. Caches operate and register the access to all datasets. When temporary space be comes insufficient to accommodate new datasets, the cache mechanism selects which datasets are erased and which datasets are held in the temporary store in order to get necessary capacity to store the new data.

The cache mechanism regulates the use of storage resources based on data access demand. Caches select the dataset that are held in the temporary store in a dynamic way based on data activity. In this way caching is used to optimize data transfers between system components with different performance capabilities. Caching is an intermediary mechanism between data consumer and data provider; this position permits sharing of data and storage resources between several consumers.

3.1.2 Applications of the cache mechanism

Computer architecture

The cache mechanisms were first introduced in computer architectures by Wilkes [130]. The cache mechanism is implemented by a type of specialized high speed memory. This memory is often smaller and faster than the main storage. It is typically used to optimize the data transfers between a processor and main memory. The cache is used to hold a copy of instructions and data obtained from main storage and probably to be needed next by the processor [115]. Cache utilization is based on general tendency of the applications to access often the same set of instructions and data. This permits reductions in execution time and data access time to the main memory [110].

Multiprocessor architectures

Multiprocessor architectures use cache mechanisms to reduce bus traffic and increase the number of processors [82]. Multiprocessor cache design is much more complex than uniprocessor design primarily because of the consistency problem. In machines such as KSR-1 [100] and more recently SDAARC [46] main memory is composed of caches. In the architecture called COMA (Cache Only Memory Access) the data in memory does not have fixed locations so data is migrated around the system on demand. Using the main memory as a big cache increases significantly the hit rate, hence the performance.

Distributed file systems

Most distributed file systems support client side caching. The clients hold in local caches the recently used files to reduce expensive access to remote file servers. The design of distributed file system places caches in an intermediate position between clients and file servers [14]. Distributed file systems like AFS [104] and its descendant CODA [78] exploit caching on client's local disks to achieve scalability and high degree of fault tolerance, and makes the client less dependent on the availability of the server. Initial versions of the Network File System NFS [60] protocol included cache specifications, but since version 3 this is outside of the protocol; this approach permits the implementation of different caches policies [116].

Databases

Frequently accessed database objects can be stored in a special client area to reduce the time required to read and write data [64]. Database caching stores relevant database objects such as triggers, constraints and stored procedures [87]. In query result caching, the content is checked with back-end database queries and the cache stores only partial back-end table data. Frequently, query result caches use materialized view technology to store and match the cache content [6].

Web and client-server applications

Caches are used in client server environments that require frequent remote data access or data movement. This includes streaming multimedia applications which present dynamic access patterns [134]. In Web environments and more generally in the client-server applications, caching can be deployed on the server side, like in server accelerators, in the client side, like local caches, or in an intermediate level like proxy caches.

Distributed applications

In middleware based on object models for distributed applications like Globe [11] the cache facilities can be created using object replication capabilities. In Globe each object implements its own replication strategy, in this sense the interfaces associated to objects are standardized, this permits implementing different and various cache or replication strategies with objects [116]. Thus in Globe middleware a high degree the flexibility is obtained. In contrast, CORBA [90], an object model and specification for distributed applications, does not have special provisions for caching; the implementation of a cache subsystem is left completely to the application developer, this approach may involve a considerable effort.

3.1.3 Cache advantages and disadvantages

There are several advantages to using caching:

Reduction of amount of data transfers: Successful data retrieval from a cache eliminates the need to contact the original data source. Thus, caching represents an effective means for reducing data transfer demands and therefore network communication.

Data access latency reduction: Caching permits retrieval datasets from a position located near by, instead of remote locations that often need more time for data transmission. As a consequence of reduction of data transfers, the data movement without cache intervention can be done relatively faster due to less congestion and more available resources for transmission.

Reduction of the workload of the data sources: Caching reduces the data access operations that must be dealt with by a particular data source. In this way data access processing is distributed among several locations and components. A cache acts as a load balancer.

Enhanced robustness: If the remote data source is not available due to remote system failure or network breakdown, the consumer can obtain a data copy from the cache. Therefore, the robustness of the system is enhanced.

Provides information about data activity: A side effect of a cache is the possibility of providing information about data usage; this permits establishment of treatment patterns in connection with data access. It represents a very useful description of data activity that permits to analyze the behavior of data access. This concerns the cache utilization in distributed systems environments where cache content is shared by a wide user community.

It is important, however, to note that there are several disadvantages of using cache mechanisms.

The main disadvantage is the difficulty of guaranteing that the data copy in cache is always consistent with the original source. Data is often distributed over different caches to improve performance. With several copies distributed between caches, problems arise over time if copies of data become inconsistent, causing the local copy of the data to be out of synchronization with the contents of the original data source, or with the other cached copies. These inconsistencies may result from local updates on the cached data or due to an outdated version of the cached data compared to what is stored on other caches.

The consistency problem is the most complex aspect of cache design. Using several caches implies the existence of multiple copies of the same data and the requirement is to keep all the copies consistent. To overcome this problem, caches implement consistency models. A lot of work has been done in the field of cache consistency [59], [44], [80], [65], [124].

There is a possibility that the *access latency* may increase in the case of a cache miss; this is due to the additional time that is spent for extra cache processing. When an access retrieval is not successful from cache, extra work is necessary to fetch requested data from original source. To deal with this inconvenience, cache access delay should be minimized; this may be achieved in the contexts where time access are significative.

The effectiveness of a cache depends on a variety of (often conflicting) parameters that need to be skillfully balanced for a given requirement [110]. The first design consideration is the *fetch or admission policy*. Most system use fetch on demand because of its simplicity, but some use prefetching which can improve the miss-ratio. Several prefetching techniques can be used but Przybylski [98] notes that complicated fetch strategies perform only marginally better than simple fetch on demand due to limited memory resources and strong temporal succession of cache misses.

A cache mechanism has a fixed amount of storage for holding data objects. When this storage space fills up the cache must choose one or more objects to drop in order to make room for newly referenced objects. The cache's *replacement method* determines which objects should be removed from the cache. The goal of the replacement policy is to make the best use of available resources, including storage

space and network bandwidth. The choice of cache replacement policies can have a significant impact in the hit rate of the cache, as well as local resource utilization.

There exist a multitude of replacement strategies proposed in the literature: a unique strategy is not possible for all different workload situations. Although most of the works provide evidence that the proposed strategies are the best, it is often possible to find some strategies that give good results in different evaluations. An overview of main cache replacement methods is presented in Annexe A.

Depending on the workload, different replacement strategies can be useful. As workloads on the grid can change it is a valuable question how a cache can use different replacement strategies in a dynamic manner. Furthermore, replacement decisions affect the state of a cache instance. In collaborative caching, a coordination of replacement decisions at different caches could give superior performance; coordinated placement of data entities could improve the performance further. An infrastructure that permits coordinated and flexible replacement strategies in a collaborative cache is necessary.

3.1.4 Performance Measure Metrics

There are mainly three performance measures in web caching: Hit Ratio, Byte Hit Ratio and Delay Saving Ratio. They are defined as follows:

Hit rate (HR) Represents the number of hit references over total number of references. It is the most popular measurement of cache efficiency. A hit rate of 70% indicates that 7 of every 10 requests to the cache find the object being requested.

Byte Hit Ratio (BHR) This is the number of bytes returned directly from the cache as a fraction of the total bytes accessed. This measure is not often used in cache studies for computer system architectures because the objects (cache lines) are of constant size. Web objects, of course, vary greatly in size from a few to millions of bytes. Byte hit rate is of particular interest for network resources control. A byte hit rate of 30% indicates that 3 of 10 bytes requested by clients were returned from the cache; conversely 70% of all bytes returned to users were retrieved across the network.

Delay Saving Ratio (DSR) It is a measure that takes into account the latency of fetching a data object. It represents the reduced latency due to a cache hit over total latency when it is not present in the cache.

Note that object hit rate and byte hit rate trade off against each other. In order to maximize object hit rate it is better to keep many small but popular objects, however, to optimize the byte hit rate it is better to keep large popular objects. It is clearly preferable to keep objects that will be popular in the future and drop

unpopular ones, the trade-off is whether to have a prejudice against large objects or not.

Other measures of cache efficiency include the cache server utilization of CPU or disk storage system, which are driven by the particular cache server implementation. Throughput, cache responses per time unit, is a common metric in many kinds of benchmarks. Throughput is frequently an input parameter for benchmarks.

The response time measures how quickly a cache responds to requests. For a single request, it is the amount of time elapsed between sending the request and receiving at the end the response. Since the individual time response varies greatly, it is useful to establish average, median and other statistical measurements. The average response time or latency of object retrieval is a measure of interest for cache administrators end users.

3.1.5 Web caching

In its simple form, web caching is similar to memory caching: a web cache mechanism stores and manages web data to reuse and share. A web cache stores copies of the documents requested by Internet users, so that subsequent requests may be satisfied by the cache mechanism. Thus, a web cache holds data in anticipation of future requests. In the context of the client/server interaction on the Internet, a cache aims to eliminate the need to contact the original server. In this way, web caching reduces network data transfers, server workload and client data access latency.

Proxy Caching A Proxy cache is a mechanism designed to offer faster access to cached content on the Internet; a proxy cache acts as a gateway between web clients and web servers. A proxy cache intercepts the HTTP requests from clients and, if it finds the requested data object in its cache, it returns the data object to the user. If the object is not found, the cache goes to the original object server to get the data on behalf of the user, possibly deposits it in its cache, and finally returns the object to the user.

Cooperative Web Caching

A group of caches cooperating with each other in terms of serving each others requests and making storage decisions results in a powerful model to improve cache effectiveness. An important aspect of cooperative caching design is how the caches are organized at different locations of the distributed system. This aspect is essential to establish the possibilities of interactions and the relationships between the group of caches. A review of the literature suggests four main types of schemes or architectures: hierarchical, distributed, organization based on multicast and peer to peer approach.

There are a lot of collaborative cache architectures and cache communications protocols: a definite architecture and protocol that satisfies all different grid caching requirements is not possible. Different schemes and protocols can give good results under different collective workload and access patterns. A concise overview of cooperative cache architectures and cache communication protocols is presented in Appendix B.

As data movement application requirements on the grid can change frequently, it is a relevant question to consider how a collaborative cache can be organized under different architectures (hierarchical, multicast, distributed, peer to peer) in a dynamic manner. A coordination of cache interactions could give support for new data management functions. Coordinated movement of data entities could improve the performance further. We consider it necessary to have an infrastructure that permits dynamic and flexible deployment of different cache architectures in a grid environment.

Discussion

Although the support for flexible and dynamic features constitutes an interesting aspect for collaborative caches, we can not ignore possible complications: introducing flexibility and dynamicity adds additional complexity to the management process. Also the more flexible and dynamic procedures should be intelligent enough to exploit potential capabilities.

The selection between different parameters should be supported by accurate information in order to take correct decisions. This implies that this kind of flexibility adds another form of complexity to the management process: the collaborative cache system has to obtain and manage information about activity for all cache and data instances.

In this thesis we propose a specification to build a basic collaborative cache infrastructure that supports these features. Future research should therefore concentrate on the following two topics: design and evaluation of intelligent management procedures and verification of the applicability of these procedures to grid in real environments.

3.2 Grid Data Management

This section describes the main aspects, characteristics and works related with data management in grids. An overview of representative projects and developed solutions is presented.

A data grid [30] is a grid system that provides the infrastructure and services for data-intensive applications that need to access and transfer large amount of data stored in distributed storage resources. Venugopal proposes a taxonomy of data grids and makes a revision of the related concepts and technologies [123]. Data grid systems provide the following main capabilities:

- Search and discover the required data entity through all available and distributed datasets.
- Transfer large datasets between storage resources with a short time delay.
- Control and operate multiple data copies.
- Select computational resources and process data on them.
- Control and operate data access permissions.

Essentially a Data Grid supports two basic functionalities: a reliable data transfer mechanism and a replica discovery and management mechanism. Additionally, other services can be provided depending on the application requirements. These include mechanisms for replica consistency and metadata catalogs. All data operations are controlled by authentication and authorization mechanisms that ensure data access and manipulation.

In the same way, data grid infrastructure maintains shared datasets distributed across administrative domains. These data are preserved independent of the underlying storage systems. This makes it necessary to keep information associated with data and data activity like access control, versions and metadata.

The grid resources are heterogeneous and are under the control of their own local administrative domain. Thus, a grid deals in general with aspects such as sharing of resources, authentication and authorization of entities, management and scheduling of available resources. The data grids share this general aspects and have particular characteristics:

Large amounts of data: Data-intensive applications manipulate large datasets in the range of hundreds of Megabytes (MB) to Petabytes (PB) and beyond. Resource management in data grids, thus, tries to minimize latencies of bulk data transfers.

Share data and storage: Resource sharing is extended to share distributed data collections and repositories to store new datasets.

Uniform namespace: Datasets share the same logical namespace where every data element has a unique logical filename. This logical filename is mapped to one or more physical references on several storage resources across the grid.

Access control: Different levels of access, distribution and confidentiality can be necessary for data.

3.2.1 Organization and standardization

Grid computing is a technology that provides access to different types of resources. The resource sharing among different entities is based on the concept of Virtual Organizations (VO) [53]. A VO is formed when different organizations aggregate resources and collaborate in order to achieve a common goal. A VO establishes the resources available for the participants and the rules and conditions under which the resources are used. The VO also provides protocols and mechanisms for applications to determine the accessibility of these resources. Often, a VO uses mechanisms such as the certificate authorities (CA) and trust chains for security.

The VO seeks interoperability between the resources and the components that are provided by different organizations. This requires standard protocols and service interfaces for information exchange among VO participants. Service interfaces must be separated from a particular implementation and must be described in a language and platform with independent format.

The Grid computing research community proposes standards that deal with these requirements, for example through forums such as the Open Grid Forum (OGF) [62]. The OGF community adopted the Open Grid Services Architecture (OGSA) [55], which is based on the Web services. Web services are self-contained, stateless software components which are capable of being accessed via standard network protocols (for example SOAP over HTTP). Web services are described in XML (eXtensible Markup Language) [18] and use standard messages mechanisms for the exchange of data.

Grid services are standardized web service that support grid capabilities in a secure, reliable, and stateful manner. Grid services may also provide lifetime management and state notification. OGSA uses standard web service mechanisms for discovering and invoking Grid services.

An OGSA Data Service [56] implements interface and associated behavior for the manipulation of data visualizations for accessing and managing data resources in Grid environments. Data service interfaces describe the data and provide operations to manipulate it. A particular data entity can be represented in many ways by different data services that support different sets of operations and data attributes: The abstraction provided by a data service is called data virtualization. The Data Access and Integration Services Working Group (DAIS-WG) at GGF, have produced a set of standards to represent data through grid data services [8].

These standards provide the support for complex access data hiding grid mecha-

nisms. This infrastructure is necessary for transferring and managing the data using underlying or core mechanisms such as data storage, data transport, and resource management.

The Globus Toolkit [63] is a software platform under an open-source license to build and deploy grid services. The Globus Toolkit supports grid services that follow OGSA architectural principles and it also offers a development environment for producing grid services that follow OGSA principles.

In this platform, a grid is composed by a collection of differentiated services. Users can integrate services to manage distributed data. These services mainly include: a storage service, a data transfer mechanisms, services to access databases and a replica location service. Other globus components also can complement the above services: a metadata catalog service, and a community authentication and authorization service, for example. Users would need to implement their own consistency mechanisms for managing state information about files registered into the grid. This platform assumes a single name space for users, data and resources.

3.2.2 Data Grid Projects

In this section, some representative Data Grid projects, developed for different application domains, are presented.

High Energy Physics (HEP) Often cited examples for Data Grids are those being developed for analyzing the huge amounts of data that will be generated by the Large Hadron Collider (LHC) at CERN. There are various Grid projects around the world that are setting up the infrastructure for physicists to process data from HEP experiments. Some of these are the DataGrid [49] and EGEE projects [47] funded by European Union, the LHC Computing Grid (LCG) [27] directed by CERN, the Particle Physics Data Grid (PPDG) [97] and Grid Physics Network (GriPhyN) [69] in the United States, GridPP in the UK. These projects have common features such as a hierarchical or tiered model for distributing data, shared facilities for computing and storage, and human resources dedicated to manage the infrastructure. Some of them are entering production stage.

BioInformatics The modeling and simulation of biological processes, coupled with the need for accessing existing databases, has led to the adoption of Data Grid solutions by bioinformatics researchers worldwide. These projects include federating existing databases and defining common data formats for information exchange. Some examples are the Japanese project BioGrid for brain activity analysis and protein folding simulation, the eDiaMoND project [120] in the UK for breast cancer treatment, and the BioInformatics Research Network (BIRN) [13] for imaging of neurological disorders using data from diverse databases.

Astronomy The astrophysicists community around the world are linking observatories and infrastructure for accessing the data archives that have been gathered by telescopes and instruments around the earth. These include the National Virtual Observatory (NVO) in the US, Astrophysical Virtual Observatory in Europe, and AstroGrid in the UK. The International Virtual Observatory Alliance (IVOA) [74] is coordinating these efforts around the world to ensure interoperability. These projects provide uniform access to data repositories, along with access to software libraries and tools that might be required to analyze the data. Other services that are provided include access to high-performance computing facilities and visualization tools for users. Other astronomy grid projects include those under construction for the LIGO (Laser Interferometer Gravitational-wave Observatory) and SDSS (Sloan Digital Sky Survey) projects.

Earth Sciences The disciplines such as geology for earthquake and climate use grid infrastructures for modeling and simulation in scale. An example is the NESgrid project [92] which link earthquake researchers that use high performance computing and sensor equipment. They collaborate together designing and performing experiments. Earth disciplines aim to integrate high-performance computational and data resources to analyze large datasets produced by climate modeling and simulation.

3.3 Remote Data Transfer

The mechanism of data transport is an essential element in the grids that support data-intensive applications. In addition to transmission of bits between resources, the data transport includes other aspects such as access control and data transfer management.

There are two main types of data transport mechanisms in grids. The former is the transfer protocol that specifies the process to initiate and control data transfers. It controls bit movements between two network entities. The most widely-used transport protocol in grids is GridFTP [4]. The second type of transport mechanism provides application-specific operations such as I/O facilities. The I/O facilities allow an application to access remote files as if they were locally stored. This mechanism presents a transparent interface through an API that hides the complexity of the network protocols.

3.3.1 GridFTP

GridFTP is an extensible data transfer protocol which extends the standard FTP protocol to include features required by grid environments. GridFTP benefits from FTP architecture which is the protocol most commonly used for data transfer in wide area networks and is widely implemented.

The following features of GridFTP are extensions to FTP [4] :

Security Infrastructure GridFTP extends the FTP protocol by supporting GSI (Grid Security Infrastructure)[63] and Kerberos-based authentication.

Third-party control Allows a user or application at one location to initiate, monitor and control a data transfer operation between two other locations.

Parallel data transfer An extension to the FTP PASV command where in the server presents a list of ports to connect to, rather than just a single port. This allows for multiple connections to download the same file or for receiving multiple files in parallel.

Striped data transfer Striped data transfers are realized through a new transfer mode called the extended block mode. The sender notifies the receiver of the number of data streams by using the End of Data (eod) and End of Data Count (eodc) codes. The (eodc) code represents how many (eod) codes should be received to consider a transfer finished. The sender provide additional information to ensure that the receiver obtains the data correctly.

Partial file transfer The extended retrieve (eret) command supports partial file transfer. Thus GridFTP enables transfers of arbitrary subsets or portions of a file.

Automatic negotiation buffer sizes The set buffer (sbuf) and auto-negotiate buffer (abuf) extensions allow the resizing of TCP buffers. This extension permits negotiation the sizes of the TCP buffers and congestion windows to improve the transfer performance.

Restartable data transfer GridFTP also supports restart for stream mode transfers which is not provided in the format of the FTP protocol. GridFTP sends restart markers indicating a byte range that has been successfully written. In case of a failure, transmission is resumed from the point indicated by the last restart marker received by the sender.

The Globus Toolkit [54] provides a public implementation for the GridFTP server; this implementation supports most of features except the striped data transfer and automatic buffer size negotiation. It provides libraries and an API for clients to connect to GridFTP server. A client command-line tool implemented with these libraries is also supplied with globus distribution. The Reliable Transfer Service (RFT) [61] is grid service interface distributed with Globus Toolkit that manages file transfers and using GridFTP.

Data movement requires consumption of important network and storage capabilities. GridFTP and I/O mechanisms provide essential functions for remote data access, however specific system is necessary to coordinate collective data movement using resources in an efficient and cost-effective way. Collaborative caching requires

global supervision of grid data movement to support high-level data management services. Supervising of GridFTP and I/O mechanisms must permit to establish data movement activity in the grid.

3.4 Storage Management

3.4.1 SRM

The Storage Resource Manager (SRM) from Lawrence Berkeley Laboratory is a control interface specification defined as web services [108] [114]. SRM aims to provide a common way to interact with all storage resources. The SRM has been designed to be the universal interface (through a standard protocol) for the management of storage devices such as disks and tape storage systems. Diverse kinds of storage device will eventually offer an SRM interface that will hide the complexity of the mechanisms behind it. SRM does not store data itself but acts as interface to massive storage systems.

SRM implements storage resource virtualization for data management in grids. SRM assigns default space quotas, allocates space for files, invokes the external file transfer services to move files into the storage space, marks the files for a certain lifetime, and uses file replacement policies to optimize the use of the shared space. SRM performs automatic garbage collection of unused files by removing selected files whose lifetime has expired when the space is needed. SRM could be managing a disk cache (Disk Resource Manager - DRM), or managing a tape archiving system (Tape Resource Manager - TRM), or a combination of both called a Hierarchical Resource Manager (HRM).

The Disk Resource Manager (DRM) manages dynamically a single shared disk cache (total disk space that is managed with cache policies). This disk cache can be a single disk or a collection of disks. The disk cache is available to the client through the operating system that provides the usual file system capabilities to create and remove directories and files, and to open, read, write, and close files. However, the space is not pre-allocated to clients. In contrast, the amount of space allocated to each client is managed dynamically by the DRM. The function of a DRM is to manage the disk cache using some client resource management policy that can be set by the administrator of the disk cache. The policy may restrict the number of simultaneous requests by each client.

A Tape Resource Manager (TRM) is a middleware component that interfaces to systems that manage robotic tapes. Such systems usually have a disk cache that is used to stage files temporarily before transferring them to clients. The function of TRM is to process requests for file transfers, queue such requests in case the system is busy. A Hierarchical Storage Manager (HRM) is a combination of a DRM and a

TRM. Because a robotic tape systems are mechanical in nature, they have a latency of mounting a tape and seeking to the location of a file. Pre-staging can help mask this latency. It can use the disk cache for pre-staging files for clients, and for sharing those files between clients.

SRM is a storage management protocol which does not implement file access or file transfer capabilities. For these operations the applications must access or transfer directly the file using other facilities or services.

Some laboratories have implemented the SRM specifications as components that facilitate grid access to different mass storage systems (MSS). The Fermi National Accelerator Laboratory (Fermilab) developed an implementation of SRM as part of the Enstore MSS [52]. The Lawrence Berkeley National Laboratory (LBNL), built an SRM as an independent component in front of HPSS [70]. The CERN developed a prototype of SRM for the CASTOR system [26].

3.4.2 SRB

The Storage Resource Broker (SRB) [12] [103] is a middleware that provides a uniform access to heterogeneous storage resources including filesystems, archival storage systems and database systems. SRB aims to provide a unified view of the data stored in disparate storage devices by providing the capability to organize them into virtual collections independent of their physical location and organization. SRB enables the creation of shared collections through management of consistent state information, latency and load management, logical resource usage, and multiple access interfaces.

Filesystems and databases are managed as physical storage resources (PSR) which are then organized as logical storage resources (LSR). Data entities are organized within a hierarchy of collections and sub-collections. Data entities within collections have associated metadata which describe system attributes such as access information and size, and descriptive attributes which register information considered important by the users. The metadata is stored within a catalog (MCAT) which also includes attributes of the collections and the PSR. Attribute-based access to the data items is available by searching catalog.

The SRB middleware consists of the SRB Master daemon and the SRB Agent processes associated with each Master. The SRB Agent uses the metadata catalog service to obtain the essential system metadata required for processing client storage request. A federation of SRB servers can be created interconnecting the masters. In that configuration, a server acts as a client to another server. A client request is handed over to the appropriate server depending on the location determined by the MCAT service.

SRB supports transparency for data access and transfer describing the data inde-

pendently of the underlying storage system. The collection takes care of updating and managing consistency of the data along with other state information such as timestamps and audit trails. Consistency is managed with synchronization mechanisms that lock data against access and propagate updates throughout the system until global consistency is achieved.

SRB implements a UNIX-like file I/O interface that supports get and put operations on storage objects. SRB middleware provides a mapping from defined storage interfaces to the native interfaces supported by each underlying storage resource. SRB implements specific-resource drivers by each interface for each resource. SRB I/O can stage files from a tape or archival storage to disk storage for faster access.

The container concept was designed by SRB to aggregate many small files in the cache mechanism before storage in the archival storage system. The SRB includes a caching system designed to handle container I/O. All container writes are done to the cache copy first. Reading is done only on the cache copy. SRB is one of the most widely used data management products in various application domains including the UK eScience (eDiaMoND) [120], BaBar [112], BIRN [13], IVOA [74].

Discussion

SRM and SRB are mechanisms to manage the load on an individual resource storage. They employ cache techniques as an intermediary mechanism for reducing the access latency to massive storage systems. SRM and SRB do not implement collective capabilities to provide distributed storage space on-demand; they do not support collective temporary data federation. Our position is that the federation of distributed and heterogeneous storage resources for temporary data can be used to implement mechanisms that provide on-demand storage space.

The SRM principle, a control interface specification to interact with all storage resources, can be used to control and operate a generic cache mechanism for grids. However, that specification also must support capabilities for federation and coordination of collaborative caching.

SRM and SRB transform physical storage resources into logical storage resources to build virtual storage systems. This principle can be used to support collaborative caching from individual storage resources. In grid caching this virtualization must provide monitoring and configuration capabilities to support collective cache federation.

3.5 Databases Access

Initially, grid applications requirements focused on storage, replication and movement of file-based data. These requirements have become more complex in biological and astronomical communities which manipulate large amount of data using databases for storage and retrieval. In the same way, other domains such as medical research, health-care and engineering also aim to use a grid to access and integrate multiple and distributed collections of structured data. Similarly, these data need to be made available and accessible to distributed groups of users and their applications, which makes these communities good candidates to adopt grid infrastructures. Few grid infrastructures have been developed in this area, in the next section we present only the most representative middleware OGSA-DAI.

3.5.1 OGSA-DAI

OGSA-DAI (Open Grid Services Architecture Data Access and Integration) [7] [38] is a middleware platform that enables databases to be accessed as web services. The OGSA-DAI implementation has been designed with the aim of allowing external data resources, such as files, relational databases and XML databases to be incorporated within the OGSA framework through a standard interface based on grid services specification. The OGSA-DAI project has as its main goal to provide a uniform access to data resources, and integrating existing DBMS with OGSA grids.

OGSA-DAI has designed and developed a group of services for integrating database access and description with the core capabilities of OGSA. This middleware includes a collection of components for querying, transforming and delivering data in different ways, and a simple toolkit for developing client applications.

The Grid Data Services (GDS) are OGSA-DAI components that access databases using drivers and use additional components for data formatting, data delivery and request handling. Applications can use the core GDS components directly to access individual data stores or can use a distributed query processor, OGSA-DQP, to coordinate access to multiple database services.

Standard specifications define the functionalities with independence of the database paradigm or specific data system being accessed. In contrast with specific connectivity technologies like JDBC, these standardized interfaces permit database connectivity without consideration of the underlying technology. In this way, it supports access to multiple relational database systems, XML storage managers and filesystems.

OGSA-DAI services provide metadata about the DBMS. Metadata are used to expose DBMS capabilities to the grid through the service interfaces. The connection

mechanisms employed to connect to the databases are also exposed for clients capable of interpreting such information. For relational databases, the database schema may be retrieved from the service; this information is useful for high level services such as distributed query processing.

The infrastructure design supports a document-oriented interface for database requests, in which a single document may specify a collection of related activities. The activities are operations that a data service resource can perform, including data resource manipulation, data transformation and third party data delivery operations. In this way it is possible to compose data requests in one message round trip and compose activities so that redundant data movement is avoided.

The distribution of queries over multiple databases, executed at multiples locations can concern huge temporary data movement. This implies high access time for remote queries. Caching can permit the total or partial reutilization of previous query results.

Cache operation of a distributed database is considerably different from conventional caching schemes: the support of database caching in grids should be flexible and generic so it may deal with different capabilities like splitting and merging of client queries into sub-queries for available datasets in cache, in order to reduce remote data transfers.

In the case of complex queries, it can be useful to find partial results from the cache content and the remaining portion of the dataset may be fetched from remote locations [3]. After the complete execution of all the sub-queries, the requested data can be obtained by the union of the above multiple datasets.

3.6 Replication

Grids with high data-intensive requirements need to support geographically-distributed collaboration in which all participants require an access to the datasets produced within their VO. Duplication or replication can be used as means for improving performance allowing computing resources to have access to locally cached data. The duplication of the datasets is therefore a strategy to support the scalability and reliability of data access and to reduce the network traffic.

Caching and replication are closely related mechanisms: They share some basic objectives and may differ only in their policies. This makes it hard to establish clear differences between both mechanisms. However, the main difference is based on the kind of process that involves creation and maintenance of copies. Thus, caching is often initiated on consumers demand and it has more autonomy to determine how long to keep in cache depot. In contrast, replication is often initiated by providers

in pre-established form and the copies have a more persistent character.

The Info Dissemination Working Group of OGF [62] proposes the following definitions [91]:

Caching : is where portions of frequently accessed data are copied on demand and may be kept synchronized with a master. Rules such LRU are often adopted in caching systems to determine how to keep a piece of data cached.

Replication : is where portions of a body of data (and optionally its metadata) are copied (and optionally transformed), often in a pre-planned way, and sometimes synchronized with one or more masters over a long period.

Replication is limited by the size of storage available at different sites within the grid and the bandwidth between these sites. A replica management system therefore ensures the access to the required data while managing the underlying storage.

Typically, a replica management system consists of the storage resources which are linked to each other via high-performance data transport mechanisms. The replica manager directs the creation and control of replicas according to the established policies and the availability of storage resources. A catalog or a directory keeps the record of the replicas and their locations. The catalog is queried by applications and other services to discover available replicas of a particular data entity. Some implementations merge the manager and the catalog into one service.

Replica management systems generally provide client libraries that allow querying of the catalog to discover datasets and to request replication of a particular dataset. Data replication has been studied extensively in the literature of distributed systems such as distributed filesystems and distributed databases [68] [93] [129]. Most research has focused on data consistency and fault tolerance.

3.6.1 RLS

The Replica Location Service (RLS) [29] [39] is an architectural framework that allows registration and discovery of replicas. It is provided with the Globus Toolkit middleware [63]. RLS maps the logical identifier of a data entity to the physical locations of replicas for the data entity. The RLS consists of two main components: the Local Replica Catalog (LRC) and the Replica Location Index (RLI). The LRC maintains consistent information about logical to physical mappings on site or storage resource. The RLI indexes the catalog itself. RLS reliability and load balancing are obtained by deploying multiple, and sometimes redundant, LRCs in a distributed index.

In RLS, a data entity is represented by a logical file name (LFN) and keeps some information such as its size, its creation date, and any other metadata that might help users to identify the data entities that they seek. The physical location is represented by a unique physical file name (PFN) which is a URL (Uniform Resource Locator) to the data file on storage. Thus, the LRC provides the PFN corresponding to a LFN.

A data entity may be replicated across several geographical and administrative domains, and the information about its replicas may be present in different replica catalogs. A RLI creates an index of the replica catalogs as a group of logical file names and a reference to replica catalog entries. In this way, it is possible to establish diverse configurations of replica indexes, for example, a hierarchical, centralized, or partitioned index configuration.

The RLI information is periodically updated using soft-state messages that summarize the state of items in the LRC. RLS is suitable for replicating data that are write-once read-many. Data gathered from different sources that needs to be distributed geographically correspond with this category. Data are accessed in a read-only manner and, therefore, they do not require strict consistency. RLS is also an independent replication service that does not manage data transfer or data replication itself. It provides only an index for the replicated data.

Data Replication Service (DRS) [31] [39] is a grid service based on RLS provided with the Globus Toolkit. The primary functionality of the component is to ensure that a specified set of files exist on a storage site. It makes local replicas of data by transferring files from one or more source locations, and registering the new replicas in a RLS catalog.

3.6.2 GDMP

The Grid Data Mirroring Package (GDMP) [113] [48] is a replication manager that provides secure and high-speed data transfer services for replicating large data files and object databases. GDMP supports point-to-point replication capabilities. It uses the capabilities of other grid components, such as replica catalogs and GridFTP.

The design of GDMP is based on the publish-subscribe model. In this system, the server publishes the group of new files that are added to the replica catalog and the client can request a copy of these after establishing a secure connection to the server. GDMP uses the GSI authorization mechanism. Initially, the clients register with the server receiving notifications about new data that is available which is then requested for replication. Clients have the responsibility for handling failures during replication. For example, the client should reconnect with the server and request a re-transfer if the connection fails while replicating a data entity. GMP uses GridFTP for data transfers.

GDMP manages object databases created by HEP experiments. For data replication it is favorable to handle objects rather than files because a file can contain up to a billion objects. GDMP was originally designed for the CMS experiment at the LHC in which the data is generated at one point and has to be replicated geographically in the world. For this experiment consistency requirements are not strong. The data is organized as files containing objects where each object corresponds to a collision.

In this context, objects requested by a site are copied to a new file at the source. This file is then transferred to the receiver, and the database at the remote end is updated to add the new objects. The file is then removed at the origin. GDMP supports static replication in which the client site determines the duration and the volume of replication.

3.6.3 DTM

Data Tree Manager (DTM) [41] is a service for managing data replication in network-enabled server (NES) environments. DTM seeks to minimize computation times by decreasing data transfers between the clients and the platform. To avoid multiple transmissions of the same data from a client to a server, DTM allows to leave data inside the platform after computation; data will be further used by the client as reference. A client can choose whether a data will be persistent inside the platform or not. This property is called the data *persistence mode*; it permits clients to establish several modes of persistence for temporary data to be kept in the system. This enables it to be established if the temporary data can be stored or moved between locations.

In order to avoid interlacing between data messages and computation messages, DTM separates data from computation management: it implements a Replica Manager component which sends replication orders to data transfer mechanism. It allows the choice of the best replica to be transferred when a replication operation occurs. This choice is based on network forecast information provided by the NWS (Network Weather Service) [131].

DTM is implemented in the DIET platform a NES multi-agent platform using the GridRPC paradigm [25] [85]. DTM designers propose to integrate data management at the application level to coordinate data transfers between Grid locations or applications. A data management API for GridRPC platforms is proposed by the OGF community.

3.6.4 Discussion

Collective replication tasks in grids require user intervention for different operations. Replication mechanisms do not implement specific mechanisms to manage distributed and temporary data. A strong limitation for exploiting replication capabilities in dynamic environments is the fact that there are no mechanisms providing on-demand storage space across different grid locations.

Replication mechanisms do not incorporate information about storage services state or data activity used to support high-level data management. Sophisticated and flexible mechanisms are required to dynamically support different models of computation with different data access patterns.

3.7 Caching

Caching in grid environments has been a slightly studied subject. Few works have been proposed that analyse caching issues in grids. We present some publications in this area:

Semantic Caching [42] [43] propose a distributed semantic caching system designed to improve query data evaluation in grids. This system seeks to reduce data transfer and query processing time, by using a semantic caching approach. Two locality-based cache resolution techniques are mixed: geographic locality-based resolution for object caches and semantic locality-based resolution for query caches. These semantic locality assumptions are based on the notion of community which is used to group users having the same interests in grid virtual organisations. Like in web caching, this approach is mainly centered in collaborative cache data resolution.

Cache Data Resolution [76] studies a cache data resolution algorithm based on the routing paths of the requested URLs. The suggested algorithm categorises these paths to into a few groups based on the traversing path. It counts the number of requests per group to discover the groups that are heavily referred and implement load balancing protocols.

Caching for Grid Based Data Warehouses [40] proposes a cooperative caching strategy to speed up OLAP queries across an enterprise grid. Authors suggest combining and aggregating cached data for future related OLAP queries. They propose a cache admittance scheme which uses a decrease and refresh mechanism for controlling admission to and eviction from the cache, and a fast, aggregate-aware benefit metric for incoming OLAP view fragments.

Replacement Policies for SRM [45] deals with the problem of cache replacement policies for Storage Resource Managers (SRMs). A SRM manages a disk

storage of limited capacity that retains data objects. A replacement policy is applied to determine which object in the cache needs to be evicted when space is needed. It uses a utility function based on the solution of the fractional knapsack problem under the assumption that the cache capacity is sufficiently large and holds a significantly large number of files. It describes an algorithm for evaluating this function during cache replacements. It also defines measure for cache replacement policies that take into account the latency delays in retrieving, transferring and processing of the files on SRM systems called "average cost per reference" which is used to establish the minimum utility function.

These approaches deal with conventional and specific aspects of cache systems such as replacement policies, data resolution, cache communication, or query caching. Often these aspects are dealt in an isolated way. They also make a strong assumption that traditional cache techniques (such as those implemented in web cache systems) are directly applicable to grids.

3.8 Discussion

In this chapter we discussed representative technologies related with grid caching: web caching and grid data management mechanisms. Our focus of interest is grids composed by a collection of different services using the Globus middleware platform.

The management of data in grid environments needs particular capabilities which are due to the special characteristics of grid applications: they are fundamentally related with the support sharing access to massive data in distributed storage resources. Current data management systems provide basic facilities to store and transfer large datasets. In addition, data management seeks to support capabilities to discover, publish and control, multiple data copies through of the grid.

The utilization of the Internet as the base infrastructure has influenced the adoption of web technologies, like web services, for exposing data management capabilities. The grid community proposes the Open Grid Services Architecture (OGSA) and a set of specifications to build grid systems based on web services technology.

Data management in grids is possible by composition of differentiated services for accessing and manipulating data in a distributed environment. Users would integrate storage services, data transfer mechanisms, services to access databases, a replica location service, and other mechanisms like metadata catalog service, and the GSI authentication service or delegation service to manage distributed data. Users also must implement their own data consistency mechanisms.

Automatic management capabilities, however, are limited and therefore user in-

tervention is required for different operations: implementation of temporary data mechanisms based on dynamic mechanisms that adapt to changes of grid conditions is not supported. Finally, in these solutions, the consistency mechanisms must be implemented by end users.

There are several data management mechanisms available for grid environments: they mostly provide basic capabilities for supporting data intensive applications that require to store, access, and transfer large amounts of data. More integration and automatization is necessary, however, to support high levels of collaboration as required on a grid scale.

The study of the representative technologies related with web caching and grid data management mechanisms have permitted identification of the operations and information to be supported by a collaborative cache in grids.

Chapter 4

Specification of Collaborative Caches for Grids

4.1 Overview

In this chapter we describe the main aspects of the specification for Collaborative Cache Services in grid environments. Our main goal is to support the automatic operation of large volumes of temporary data in grids based on collaborative caching. However, grids do not currently provide wide area cache infrastructures comparable to that of cooperative web caching as described in the Section 3.1.5 and Appendix B.

We have decided to define a basic infrastructure of collaborative cache for grid environments [22] [24]. This infrastructure provides the necesary functions to manage temporary data in grids that are not supported by current data management solutions as was discussed in Section 3.8. We have opted for an approach that specifies the base components required to build a generic and open collaborative cache system in grids.

Our approach proposes a form of generic caching that exploits the grid features: it seeks to enhance the essential interactions between the caches to expand the capabilities of intercache collaboration. We develop the precept that any cache must be accessible and monitorable by any user, application or service in the grid. Therefore, three main elements are specified:

Reference Multilayer Model A collaborative cache system is built from the composition of four functional layers: *storage*, *control*, *collaboration* and *coordination*. The model is described later in Section 4.2.

Cache Operations The collaborative cache capabilities are supported by a set of

common or standard *operations* that implement the proposed model. The operations are defined in Section 4.3 and Appendix D. The information exchanged by these operations is denominated *cache information*. Cache information elements are defined in Section 4.5 and Appendix C.1.

Grid Cache Service The infrastructure is composed of a group of autonomous cache elements called *Grid Cache Services*. A Grid Cache Service is the functional unit of the system; it implements the model and operations to support collaborative cache capabilities. The design and implementation of the Grid Cache Service is discussed in Chapter 5.

The specifications define a generic infrastructure for building collaborative cache systems in grid environments. These elements permit composition and evaluation of high level collaborative cache functions in a flexible way. Different forms of organisation of the groups of *Grid Cache Services* permit aggregation of capabilities to operate and monitor temporary and distributed data in grids.

The objective is not implementation of a traditional cache system. The approach proposes a new service to users and applications to store temporarydata. This system is transparent. However, clients make explicit requests to this cache service to store or retrieve data.

This service offers functionalities which are not addressed by existing middlewares, for example on demand temporary storage. Our goal is to design a generic and flexible solution, these functionalities are designed as a new service. Existing middleware components are not affected by this service. However, existing applications must be slightly modified. In this perspective, this service is part of the programming environment.

The dynamic character of the grid resources and temporary data impose a requirement for continuous monitoring and control of the different actions related with data access in the distributed system. Thus, the infrastructure is based on the registry, accessibility and analysis of the information related with activity of the different data and cache entities: it provides and extends the support of the cache interactions. The principal aspects of the approach are:

- Provide an infrastructure for the operation of large amounts of temporary data
- Definition and separation of the principal collaborative cache functions
- Operation and control based on a registry, exchange and analysis of data and cache activity
- Provide and extend the support of diverse cache interactions
- Provide capabilities for monitoring and configuration of cache actions and interactions

- Support the composition and configuration of diverse forms of organisation for collaboration
- Provide standard interfaces to provide and request cache functions in grid environments
- Extend the access to cache operations to a wide variety of applications

4.2 Cache Model Layer

In the Section 2.3 we have established the main requirements for grid caching. Current cache systems and grid data management solutions do not permit simultaneously to satisfy that requirements. We have decided to develop an approch that specify, define and formalize the functions using a model that representa conceptualviewofthesystem, issuesraisedbythese4layersmustbeaddressed indesignandimplementation process.

We present a reference model for the composition and operation of collaborative caches in grids. This cache model defines and organises the main functions needed to operate and control a collaborative cache system. Different levels separate the conceptual functions of the system. This cache model is used as a conceptual reference to implement the components of a generic collaborative cache system. The cache model has a vertical definition (see Figure 4.1) through four layers:

- Storage Layer
- Control Layer
- Collaboration Layer
- Coordination Layer

An implementation of this model is discussed in Section 5.3.

4.2.1 Storage Layer

The storage layer represents different storage resources that can be used to implement a cache system. The model defines a cache as a mechanism that manages temporary data contained in a storage resource. Thus, the central purpose of the storage layer is to provide capabilities to store and access the data entities in a specific storage resource.

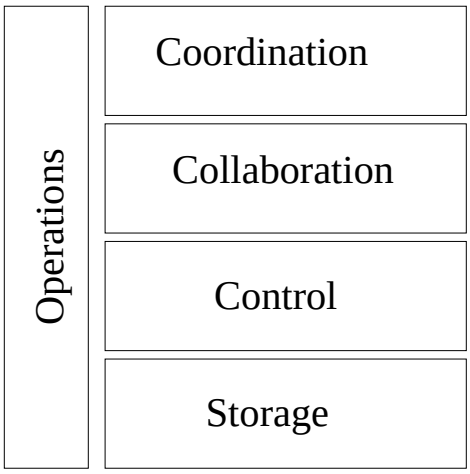

Figure 4.1: Cache Layer Model

In contrast to collaborative cache systems discussed in Section 3.1.5 and Appendix B the model proposes to build the infrastructure of collaborative caches from available storage resources. This is essential to provide storage space on-demand.

This layer includes diverse mechanisms used to keep large amounts of temporary data, such as file systems, disks, archival storage systems, tape storage systems and other massive storage systems. The storage resource also can include high level resource data managers like as SRM (Section 3.4.1) or SRB (Section 3.4.2): They are the underlying resources used by the cache mechanims.

The storage layer consists of the necessary interfaces to data manipulation of large volumes of temporary data in grid repositories. The I/O facilities of the underlying storage resources can be invoked to access data content. It uses the interfaces of the underlying system and exposes them to other layers of the system. This support is also applied to sophisticated temporary storage mechanisms such as DPSS [118] making possible a high-level interaction with conventional storage mechanisms like file systems. This characteristic seeks to make possible the inter-operation of those isolated mechanisms.

4.2.2 Control Layer

The control layer regulates and registers the actions in the storage layer. It verifies the activity related with data access in to storage resources. Its principal purpose is to provide the capacities to configure and monitor the actions of the cache entities. This makes it possible to modify and regulate the cache functions for adapting its behaviour for specific purposes. For example, different cache replacement methods (see Appendix A) can be supported with control information represented in a standard way.

Control layer registries contain information about the control and operation of the cache entities. This information includes the individual data access actions that permits to establish dynamic and historic use and evolution of temporary data; we call this information data activity. Data activity gives a notion of individual data item importance, and permits an estimation to be made of its use probability for caching management purposes.

In contrast to collaborative cache approaches discussed in Section 3.1.5 and Appendix B, the model defines that the information used to control the cache mechanism must be available for collective cache management process. This information is provided in real-time with system operation.

4.2.3 Collaboration Layer

The collaboration layer defines and supports the interactions between caches. Its goal is to expose and extend the capabilities of caches for working jointly. Our idea is that the interactions between caches determine the capabilities that can be provided by the grid. Therefore, expanding the scope of the cache interactions creates new possibilities for the management of temporary data.

In contrast with wide area caching discussed in Section 3.1.5 and Appendix B, the collaboration is often limited to data resolution between caches. In this layer, the model establishes that any cache is accessible for any user, application or service in the grid. Therefore, the collaboration layer defines the capacity to provide and request cache functions.

The collaboration layer extends cache interactions to aspects related to full data access, monitoring, operation and coordination of caches. This layer exposes cache capabilities as service oriented operations. Collective cache capabilities are composed from individual cache operations.

A cache instance translates the access operations requests to invocations of underlying resources. Each grid operation is regulated and registered by the implementation of the control layer. Similarly, the registered information is exposed through cache operations to support monitoring of service operation.

4.2.4 Coordination Layer

The coordination layer supports the organisation of interactions between caches. Its objective is to implement different schemes of cache collaboration. The coordination is mainly related to the decision process that implies collective operations between caches to support high-level data management functions.

The challenge is to support different schemes or architectures of collaborative caches, as described in Appendix B.1. This is possible if that cache implements the first three layers and the system makes decisions based on information about cache activity. An implementation of cache instance is described later in Section 5.4 and an example of coordination using that cache instances is presented in Section 6.1.

The model suggests that any collaborative cache system can be built from resources provided by the storage layer, the activity information provided by the control layer, and operations defined by collaboration layer. In this thesis we specify and implement these first layers. These form the basic bricks on which the coordination layer can be built. The specification and implementation of the coordination layer is proposed as future work.

4.3 Cache Operations

Cache layer model layersrepresenta conceptualviewofthesystem, issuesraisedbythesefourlayersmustbeaddressed whendesigningandimplementingsuchasystem. Cache Operationsarethefunctions offeredbythesystemtouse,monitor,parameterizeit. Mostofthemtraverseallthelayerslike application messages in network layers. The Chapter 5 proposes an implementationthatfollowsthereferencemodel, it also addresses the questions of operational usage and implementation. The Chapter 6 illustratestheuseofsomemonitoringandconfigurationoperations.

The specification defines the essential functional operations supported by collaborative caching. These functional capabilities consider standard actions that a cache grid component must support. Cache operations define the functions necessary to support the capabilities proposed by the model presented in the previous section. The approach proposed in this thesis requires the definition of standard operation interfaces for the interaction of the different components in the grid environment.

We propose that these operations be available for interaction between caches as defined by the collaboration layer. A cache instance can invoke operations on other caches disseminated in the grid for collaboration. Cache instances are implemented as Web services for building the collaborative cache system. Web services provide a standard means of interoperating between different applications running on a variety of platforms. They are used to build a cache service-oriented infrastructure [133]. This infrastructure is constructed via the composition of cache components defined by service interfaces referred to in this work as *Cache Operations*. In this respect each cache exposes its operations to the distributed system as a cache grid service [21]. An implementation of caches instances is described later in Chapter 5.

Some operations register and exchange information about data and cache activity, others change the operation or configuration of the cache entities. We classify cache

operations into three main types:

- Access operations
- Monitoring operations
- Configuration operations

The next sections make referencestoappendixes C and D whereoperationsaredescribed-inmoredetails. Some specific cordination items of the operations are not specified here because is the responsability of the designer of the coordination layer to specify them. They will be described in more detail in the context of the chapters 5 and 6.

4.3.1 Access operations

Access operations define the Cache Service interfaces to access data in the cache storage resources. These operations represent storage resource I/O access facilities. The access operations permit to control and register the actions related with the manipulation of data managed by the cache instance.

The implementation of access operations following the model described in Section 4.2 aims to achieve that any cache instance is visible and accessible from any user, application or service in the grid. This approach differs from architectures described in Appendix B.1 where caches and their resources are used exclusively for one kind of application.

Cache operations provide uniformity to request these cache access functions. Typically, access operations translate grid data access requests into I/O facilities of the underlying storage or transfer mechanisms (see Section 3.4).

Interfaces specify the different parameters for each type of access operation. The value of these parameters is registered in each operation invocation for monitoring data and cache activity. That information is registered and available for exchange using a standard XML Schema described in Section 4.5.

Expanding access for facilitating the collaboration

Access operations are exposed at the level of the collaboration layer. This makes it possible to invoke access operations of remote cache instances. Thus, clients can retrieve an object data from any cache in the system. Conversely, a cache can be asked to store a data object from any other cache, service or application.

Composable capabilities

Access operations provide basic capabilities which are available to compose and support elaborate functions: for example, the mechanisms for data discovery within the group of caches are composed using access and configuration operations (see Section 4.4.4). Access operations do not impose any method for the data resolution which can be implemented by diverse techniques (see Section B.2). In contrast, these access operations define interfaces to query the cache content catalogue. This information may be exchanged or used by any means between caches such as P2P, global catalogue, summary caches, CARP, polling, etc.

Sections 6.2.1 and 6.2.2 describe the utilisation of access operations to compose a system that operates temporary data in grids.

The access operations definitions are introduced in the Section 4.4.2, they are:

- Get or retrieve data entity (Section 4.4.2)
- Set or publish data entity (Appendix D.1.1)
- Get or request cache content (Appendix D.1.4)
- Set data description or metadata (Appendix D.1.5)
- Get data description or metadata (Appendix D.1.6)

4.3.2 Monitoring Operations

Monitoring operations are interfaces to exchange information about the cache elements and their activity. These permit activity information to be obtainedfrom distributed cache entities. Information provided by monitoring operations is used for collaborative cache management (support for decisions) and performance evaluation.

Traditional collaborative cache architectures (Appendix B.1) do not support facilities for real-time monitoring. Performance evaluation often is done with historical traces [35]: this is not adequate for dynamic grid environments which are characterised by a very high dynamism both in terms of available resources and in terms of effective data access patterns.

We propose that cache mechanisms in the grid must provide information about its activity in real-time to support collective management processes. The goal of the monitoring operations is to provide the suitable information for appropriate decisions. Suitable information implies the pertinent and detailed information concerning cache operation requirements. Appropriate condition implies the availability with the dynamic and synchronous cache operation.

Monitoring operations are defined in Appendix D.2. Monitoring operations provide information about cache entities and their activity using a standard XML Schema. The definition of elements that present this information is described in Appendix 4.6.

Further Sections 6.2.1 and 6.2.2 present a scenario of the utilisation of monitoring operations for implementing of collaborative cache system.

Monitoring operations are constituted by an ensemble of capabilities that capture, register, and expose information about the state and behaviour of the collaborative cache system. In the same way as access operations, monitoring operations seek to establish a minimal uniformity and standardisation for exchange, and to facilitate the common exploitation of this information.

Requesting cache instances for monitoring information provides measurements to evaluate the effect of a cooperative cache system. Examples of cache monitoring information include the state of an individual cache, the verification of tasks and transfers in progress, the number of requested and performed requests, the number of accesses to a data entity, the cache hit and miss ratio, disk free space, etc.

Monitoring operations makes possible the periodic or continuous measuring of different data and cache actions and resource usage. This provides an ongoing verification of progress toward the achievement of cooperative cache system goals.

Information provided by monitoring operations can be gathered, grouped and processed depending on the granularity needed: For instance, the information can be given for a group of individual caches, for one specific cache, for a given time interval, etc.

The monitoring operations definitions are introduced in the section 4.4.3, they are:

- Get processed requests (Section 4.4.3)
- Get cache entity description (Appendix D.2.1)
- Get replacement method (Appendix D.2.2)
- Get storage description (Appendix D.2.3)
- Get cache group (Appendix D.2.4)
- Get transfers (Appendix D.2.6)
- Get data actions (Appendix D.2.7)
- Get data replacements (Appendix D.2.8)
- Get storage capacity (Appendix D.2.9)

4.3.3 Configuration operations

Configuration operations define the basic interfaces to set attributes and values for cache entities. These operations establish the standard conventions that permit to arrange the cache components and define the basic functions and interactions of these components. Configuration operations provide the possibility of changing the functional parameters of a cache instance for a particular purpose. These aspects are concerned with initialisations, maintaining, adding, and updating the relationships among components and the status of components themselves during time operation.

The goal of the configuration operations is to provide the basic support to compose high-level cooperative cache functions. The arrangement of the functional capabilities is addressed to support different schemes and strategies that constitute a collaborative cache system (see Appendix B.1).

The feature of on-demand configuration of remote caches instances is a difference with current wide area collaborative cache systems.

Facilities for adaptation

Configuration operations provide the capability to change the state and relationship of cache components as the user or grid requirements change. They allow to specify initial or default values for attributes so that cache entities initiate operation in the desired states, setup the proper parameters values, and form the desired relationships with other cache entities. Grid administrators can use these operations to define and modify default attributes and load the predefined set of attributes into cache components.

While the cooperative cache is in operation, the reconfiguration of a cooperative cache system is desired for performance evaluation, support of resource upgrade, or fault recovery. This evaluation can be established using the information provided by the monitoring operations described in the previous section.

Modify parameter values

The configuration operations include capabilities to operate the specific cache arrangement and allows for the dynamic behaviour of the caches. The configuration operations set attributes to individual cache entities supported by its implementation such as replacement method, default time to live, group subscription, etc.

Define and modify relationships

Configuration operations permit to define and modify the relationships between cache entities. A relationship describes an association, interaction, or condition that exists between cache instances. They permit the organization and control of

interactions between caches: creation and modification of cache groups and schemes, creation and modification of hierarchies, dissemination of cache contents, methods of data resolution, etc.

The configuration operations definitions are introduced in Section 4.4.4, they are:

- Set replacement method (section 4.4.4)
- Set cache description (Appendix D.3.1)
- Set cache group (Appendix D.3.3)
- Set storage (Appendix D.3.4)
- Set default time to live (Appendix D.3.5)
- Set cache coordinator (Appendix D.3.6)
- Set collective work mode (Appendix D.3.7)

4.4 Cache Operation Definitions

In this section we present the interface definitions for the cache operations discussed in Section 4.3. These operations are supported by each cache instance to provide the functional capabilities for temporary data management based on collaborative caching.

In the remainder of this chapter we discuss various XML Schema elements for the defined operation interfaces. To facilitate comprehension, some non relevant details of element definition have been omitted.

4.4.1 Request and Response Elements

Each operation interface is composed of two elements: the *request* element and the *response* element. In turn, the *request* element always contains a *requestHeader* element that includes the necessary information to register and process the requested operation. Similarly, the *response* element always contains a *responseHeader* element that includes the information generated by the cache service in response to the requested operation.

The name notation of request and response elements is composed by the name of the operation and the word request or response: for example, for the *SetData()* operation these elements are *SetDataRequest* and *SetDataResponse* respectively.

The *requestHeader* and *responseHeader* elements are common to all operations: they are used by the requester entity to supply the information necessary to invoke the operation. In the service side, the headers are used to register information about received requests.

The utilisation of a header element like in the conventional communications protocols permits to support a wide range of the implementations based on different message passing technologies.

Request Header

The *requestHeader* element contains the information necessary to register and process each operation in the cache system, the parameters include: the *operation* name, the *type request* (client or intecache), and the *request number* that is a consecutive number that identifies the request. It is generated by a local software instance; the cache instance *source* and *destination* of the operation; the *date time* of the invocation the operation, and the *version* number to distinguish between versions of the specification or its software implementation. Figure 4.2 shows the XML Schema definition of the *requestHeader* element.

```
<xsd:element name="requestHeader">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="operation" type="operationsType"/>
      <xsd:element name="type_request" type="requestType"/>
      <xsd:element name="request_number" type="xsd:integer"/>
      <xsd:element name="cache_id_source" type="xsd:string"/>
      <xsd:element name="cache_id_destination" type="xsd:string"/>
      <xsd:element name="dateTime_request" type="xsd:dateTime"/>
      <xsd:element name="version" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.2: XML Schema element definition of requestHeader

Response Header

The *response header* element contains the necessary information to register and provide a response to each operation received by the cache system. The parameters include: the *operation* name, the *type of response* (successful, failed or not supported), the *request number* that is consecutive number that identify the original request, the instance *source* and *destination* of the operation, the *date time* of the creation of the operation response, the number of *version* to distinguish between versions of the

specification or its software implementation and optionally an extended description of the response in textual form. Figure 4.3 shows the XML Schema definition of the *response header* element.

```
<xsd:element name="responseHeader">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="operation" type="operationsType"/>
      <xsd:element name="type_response" type="responseType"/>
      <xsd:element name="request_number" type="xsd:integer"/>
      <xsd:element name="cache_id_source" type="xsd:string"/>
      <xsd:element name="cache_id_destination" type="xsd:string"/>
      <xsd:element name="dateTime_response" type="xsd:dateTime"/>
      <xsd:element name="version" type="xsd:string"/>
      <xsd:element name="description" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.3: XML Schema element definition of responseHeader

4.4.2 Access Operations

GetData() Operation

In this section we show an example of a definition of a cache operation element. For a complete description of all the elements see the Appendix D.1.

Definition of GetData() Operation

GetData() is used to retrieve from a cache service a *Data Entity* element (defined in Section 4.6.1) that corresponds to data object storage in the cache instance. The *GetDataRequest* needs as parameter a *data name* supplied as a unique identifier (example Logical File Name) with the *SetData()* operation (see Section D.1.1) or the *id* generated internally by the cache service.

In intercache working, the cache service starts the data transfer between cache services using a mechanism of data transport (see Section 3.3). The cache service can execute a mechanism for the collaborative resolution between several caches; this will be possible if the collaborative mechanism is implemented and configured by the cache group.

The *GetDataResponse* returns the *data entity* element if the data object is present in the cache system. If it is not present, the *type response* of the response header is

set to failed. Figure 4.4 shows the XML Schema element definition of the GetData() operation.

```
<xsd:element name="GetDataRequest">
  <xsd:complexType>
  <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="data_id" type="xsd:string"/>
      <xsd:element name="data_name" type="xsd:string"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="GetDataResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="data" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.4: XML elements definition of the GetData() operation

The rest of the access operations defined in the appendix D.1 are:

- Set or publish data entity (Appendix D.1.1)
- Get or request cache content (Appendix D.1.4)
- Set data description or metadata (Appendix D.1.5)
- Get data description or metadata (Appendix D.1.6)

4.4.3 Monitoring Operations

GetRequestProcessed()

GetRequestProcessed() returns a subset of the total *request* elements (section C.1.4) registered by the cache instance. The *getRequestProcessedRequest* takes as parameters the initial and final date of the time period required.

This operation permits to obtain the detailed information about the request processed by the cache service at specific time period. An extension of the operation can include other parameters such as the *type request*, the *type response*, the *status*,

the *source* or the*destination*. The operation implementation can be supported by queries to a database management system that registers the requests processed by the system.

The operation returns a *getRequestProcessedResponse* element with a list of the *request* elements. Figure 4.5 shows the XML Schema element definition of the GetRequestProcessed() operation.

```
<xsd:element name="getRequestProcessedRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="start_date_time" type="xsd:dateTime"/>
      <xsd:element name="finish_date_time" type="xsd:dateTime"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getRequestProcessedResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="request"  maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.5: XML Schema element definition of the GetRequestProcessed() operation

The rest of the monitoring operations defined in Appendix D.2 are:

- Get cache entity description (Appendix D.2.1)
- Get replacement method (Appendix D.2.2)
- Get storage description (Appendix D.2.3)
- Get cache group (Appendix D.2.4)
- Get transfers (Appendix D.2.6)
- Get data actions (Appendix D.2.7)
- Get data replacements (Appendix D.2.8)
- Get storage capacity (Appendix D.2.9)

4.4.4 Configuration Operations

SetReplacementMethod()

SetReplacementMethod() actives the *replacement method* in the cache instance. It also updates the value in the *replacement method* element of *cache* entity information (Section 4.6.1). The *setReplacementMethodRequest* element takes as parameters the name of the new *replacement method*.

This operation configures dynamically the *replacement method* applied by a cache service. It is used to modify the operation and possibly the performance of the cache service. In the cache implementation different replacement methods can be supported. The specified method is applied immediately the operation is processed. Cache instance implements the necessary procedures to update the parameters required by each configuration.

The operation returns a *setReplacementMethodResponse* element with *response header*. Thus the *type response* element can have successful values if the method was changed, or not supported if the cache mechanism does not implement the method required. Figure 4.6 shows the XML Schema definition element of the SetReplacementMethod() operation.

```
<xsd:element name="setReplacementMethodRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="replacement_method" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="setReplacementMethodResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.6: XML Schema element definition of the SetReplacementMethod() operation

The rest of the configuration operations defined in Appendix D.3 are:

- Set cache description (Appendix D.3.1)

- Set cache group (Appendix D.3.3)
- Set storage (Appendix D.3.4)
- Set default time to live (Appendix D.3.5)
- Set cache coordinator (Appendix D.3.6)
- Set collective work mode (Appendix D.3.7)

4.5 Cache Information

Fundamental to the operation and control of temporary data in grids with collaborative caching is the ability to gather information about the status and behaviour of the collective cache system, which is the function of monitoring operations. Thus, the cache system supports the capability to control and configure collaborative work between the components.

Cache information update only Such modification does not affect the behaviour of the cache entity. Example: Updating some cache instance description like the organisation name that deploys the cache service. The cache entity updates the appropriate value then returns an acknowledgement.

Cache information update plus resource modification In addition to updating values of the cache information this modification can affect the state of the underlying resource. For example, updating the storage capacity parameter modifies the available cache capacity to be shared in the system.

Cache information update plus action The modification of some parameters cause the cache instance to initiate certain action. For example, changing the replacement cache method or time to live, requires to execute a different algorithm to select the data to be evicted.

We establish three types of information necessary to constitute the infrastructure for operation and control of temporary data with the collaborative cache system:

Entity information This is the static information that describes the current configuration of the components that constitute the system. For example, the description of the cache service instance or of the storage resource. This information will not change frequently.

Activity information This dynamic information is related with actions or events in the cache system, such as data access requests, data transfer between caches or internal cache replacements. This information is called data and cache activity.

Statistics This is information that may be derived from activity information, such as the average number of access request processed per time unit by a specific cache instance.

We established a cache information structure for the operation and control of a collaborative cache system. This structure is constituted by *entity information* with static and basic information about data and cache entities and *activity information* that comprises information about the state and behaviour of data and cache entities. On the other hand, the *statistical information* that is constituted of performance measures, is calculated using the activity information and is not defined in the cache information structure. The cache information can be management by systems for managing structured data such as relational databases.

Cache information is typically generated by the cache instance involved. Thus, a cache instance maintains its own configuration information. Cache information is collected and stored by the cache instance that executes the underlying data actions. This information is exposed directly to cache instance using the standard caches operation interfaces. Statistical information can be calculated internally for the cache service instance. It can also be calculated by an external system that gathers the necessary activity information invoking the operations of the remote cache instances.

The entity configuration information describes the features and status of the entities that compose the collaborative cache management. The configuration information includes a specification of the storage resources under management of cache system.

Cache information is structured in an extensible XML schema where the individual elements contain values that reflect features, parameters or attributes of the main cache components (storage resources, data, cache and cache groups). The Section 4.6 describes the definition of cache information elements.

4.6 Cache Information Definitions

The management infrastructure of the temporary data grid based caching defines in a logical view the different elements that compose a typical collaborative cache system. These definitions permit the representation of essential element features. This definitions are established as common or standard representations for all system components.

This set of definitions have the character of intercache specification for exchange information related with the description of system elements and entities and their behaviour.

In the remainder of this section we discuss the purpose and definition of the relevant elements and entities used to build a logical view of the system.

The entity information elements defined are:

- Storage
- Data
- Metadata
- Cache
- Cache Group

The activity information elements defined are:

- Storage Usage
- Data Action
- Data Permission
- Data Transfer
- Request

4.6.1 Entity Information Elements

Storage

A storage entity contains the information related to individual storage resources: it includes essential features of the underlying mechanisms that supports the resource. A typical example is a filesystem supported by a set of disks: it contains the total available storage capacity assigned to be managed by the cache, this permits the system to establish the potential capacity of the system gathering the information from all available resources. Additional information is included that permits the establishment of operational characteristics for operation like the type of technology and operating system software. Figure 4.7 shows the XML Schema element definition of the storage entity.

```
<xsd:element name="storage">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="type" type="xsd:string"/>
      <xsd:element name="operating_system" type="xsd:string"/>
      <xsd:element name="filesystem" type="xsd:string"/>
      <xsd:element name="capacity" type="xsd:integer"/>
      <xsd:element name="capacity_unit" type="storageUnitsType"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:integer"/>
  </xsd:complexType>
</xsd:element>
```

Figure 4.7: XML Schema element definition of storage entity information

Data

Data entity information describes and identifies each individual data object handled by the cache mechanism. The data entity information is used by the cache installation to manage individual data objects in the system. It describes the essential features of the data content. This information is created when data is registered in the system.

Data entity information includes a *name* used for identification; the Logical File Name (LFN) can be used as a unique identifier for each data entity registered in the cache system. Typically, the virtual organisation or user community defines and manages the logical name space and assures a unique immutable LFN for each data entity within that organisation. Internally, the cache instance makes a mapping between LFN and the internal location represented by a Physical File Names (PFN) which is the location of data entity on cache storage resource. Cache entities use the LFN as unique identifier for all operations.

The *data id parent* registers the other identifier's data in a hierarchical relation. The *data id group* element registers the identifier of the collection structure where data is included. A data element always includes a reference to metadata elements that describe its content Section 4.6.1.

The cache system does not deal with the data object structure, it assumes that the data is materialised as a file and its logical structure is handled by the application. The elements of data entity are used to register a basic description. Figure 4.8 shows the XML Schema of the data entity declaration.

```
<xsd:element name="data">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="type" type="xsd:string" minOccurs="0"/>
      <xsd:element name="size" type="storageUnitsType" use="required"/>
      <xsd:element name="internal_path_location" type="xsd:string" />
      <xsd:element name="data_id_parent" type="xsd:integer" minOccurs="0"/>
      <xsd:element name="data_id_group" type="xsd:integer" minOccurs="0"/>
      <xsd:element name="owner_user" type="xsd:string" minOccurs="0"/>
      <xsd:element name="owner_organization" type="xsd:string" minOccurs="0"/>
      <xsd:element name="creation_date" type="xsd:dateTime" minOccurs="0"/>
      <xsd:element ref="data_metadata" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:integer" />
  </xsd:complexType>
</xsd:element>
```

Figure 4.8: XML Schema element definition of the data entity information

Metadata

The metadata entity is an optional description of the data object content. In the cache system the metadata is the minimal information about the semantic content of data entities. This information includes references to external specific, and detailed, metadata or metadata services. A cache instance registers essential metadata pieces such as key words for general semantic classification purpose. Metadata may register descriptions about target applications, data provenance, software version, algorithms, parameters and short annotations. The metadata is exposed by cache entities as additional support for retrieval operations. Metadata is supplied by the data provider who publishes or registers the data entity in the cache. In the life time of a data entity, metadata can be updated following the evolution of the data entity used. Figure 4.9 shows the XML Schema of the metadata entity declaration.

The element *software provenance* describes the application that produce the data content. The *software target* element indicates the application that can consume the data content. The *description* element records a textual description for data content addressed to users. The *annotation* element records commentaries added by users.

Cache

The cache entity information is a basic description of the cache instance. This includes essential configuration values and operational parameters: it represents the features of each cache instance in the system and is used to identify and build a

```
<xsd:element name="data_metadata">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="key_word_list" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="software_provenance" type="xsd:string"/>
      <xsd:element name="software_target" type="xsd:string" />
      <xsd:element name="external_metadata_id" type="xsd:string" />
      <xsd:element name="external_metadata_service" type="xsd:string"/>
      <xsd:element name="description" type="xsd:string"/>
      <xsd:element name="annotation" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:integer" />
  </xsd:complexType>
</xsd:element>
```

Figure 4.9: XML Schema element definition of the metadata entity information

logical view of components that compose the collaborative cache system.

Cache entity information comprises the *name service* that is the reference of the service instance which is used to invoke the grid cache service. Similarly, it includes the description information about the organisation and geographical location where the service is deployed.

Configuration values such as *replacement method* (LRU, LFU, SIZE , etc); *time to live*, the minimal time that a data object must be kept in cache before it can be evicted; and *storage resource* to be handled by the cache instance, are also included. Similarly, it comprises operation parameters such as *Cache group*, (see Section 4.6.1) that indicates the group or groups which the cache is member. The identification of an optional cache coordinator is included to support hierarchical structures. Figure 4.10 shows the XML Schema element definition of the cache entity element.

The *organisation* element records the name of the corporation or administrative entity that deploys the cache instance. The *organisation unit* element registers a group or department inside a main organisation.

Cache Group

A cache group is entity information that describes a set of caches considered together as an organisation. This information permits to represent groups of caches working together or sharing a common feature (cache protocol, membership, administrative organisation). This entity information is used to describe possible relationships between cache components: it allows to assign an arbitrary name and type to a group for description purposes. Figure 4.11 shows the XML Schema element definition of the cache group entity element.

```
<xsd:element name="cache">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name_service" type="xsd:string" />
      <xsd:element name="organization" type="xsd:string"/>
      <xsd:element name="organization_unit" type="xsd:string" minOccurs="0"/>
      <xsd:element name="location_city" type="xsd:string" />
      <xsd:element name="location_country" type="xsd:string" minOccurs="0"/>
      <xsd:element name="replacement_method" type="replacementMethodType" />
      <xsd:element name="default_ttl" type="timeUnitsType"/>
      <xsd:element name="cache_id_coordinator" type="xsd:string" minOccurs="0"/>
      <xsd:element ref="cache_group">
      <xsd:element name="work_on_collective_mode" type="xsd:boolean" default="false" />
      <xsd:element ref="storage_resource" type= type="storageUnitsType" use="required">
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:integer" />
  </xsd:complexType>
</xsd:element>
```

Figure 4.10: XML Schema element definition of cache entity information

```
<xsd:element name="cache_group">
  <xsd:complexType>
     <xsd:sequence>
        <xsd:element name="members" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="type" type="xsd:string" minOccurs="0"/>
        <xsd:element name="name" type="xsd:string" minOccurs="0"/>
     </xsd:sequence>
     <xsd:attribute name="id" type="xsd:integer"/>
</xsd:complexType>
</xsd:element>
```

Figure 4.11: XML Schema element definition of cache group entity information

The activity information elements are defined in Appendix C.1 are:

- Storage Usage (Appendix C.1.1)
- Data Action (Appendix C.1.2)
- Data Transfer (Appendix C.1.3)
- Request (Appendix C.1.4)

4.7 Discussion

This work proposes a collaborative cache system for operation and control of large volumes of temporary data in grid environments.

While other approaches exist, inspired by distributed filesystems or replication, that can provide the functionalities required to administrate this task, the design and implementation of a system based on collaborative caches represents an alternative to deal with the dynamic character of temporary data in distributed environments like a grid. The main feature is the automatic and autonomous character of a cache for data management. Additionally, grid environments are based on sharing and collaboration of resources; this is the same principle of the collaborative cache systems.

Extensibility and Scalability

The extensibility and scalability of the infrastructure is based on the ability to add distributed capabilities by using the individual service-oriented cache operations. This infrastructure allows a collective cache system to be built that distributes the work between many autonomous cache services.

This ability is focused on the collaboration for exchanging information about the data utilisation and information about cache operation. Thus the proposed approach is composed by a reference model, a common information structure, and a set of standard operations to implement the infrastructure.

Functional aggregation

The system is composed of four levels to allow a progressive integration of the elements and aspects of the system. It starts by establishing the state and configuration of the available storage resources, a control level supervises and registers the actions realised with data objects in the local installation; the next level implements organisation mechanisms for working together effectively (collaboration layer); and the final level builds a logical view of the data and system operation for an efficient global management.

Service oriented infrastructure

We propose a service-oriented infrastructure to support temporary data requests from a distributed community. The infrastructure confronts the related aspects of orchestrating and managing large temporary data with numerous distributed storage resources. The service-oriented properties that characterises the infrastructure are:

- The cache information is an abstraction that provides a logical view of the system's components and behaviour, defined in terms of operations providing

cache capabilities.

- Cache operations are defined in terms of the messages exchanged between cache capability providers and requester components. The internal structure of components including features such as its implementation technology, process structure and database structure are deliberately abstracted away. This permits incorporation of existing temporary storage mechanisms (legacy systems).
- The cache service is described by machine-processable metadata; the description exposes the details necessary for use the service (description orientation). The cache service tends to use a small number of operations with relatively complex messages. The messages and descriptions are defined in a platform-neutral and standardised format (XML).

Delegate operation of temporary data

The model defines an infrastructure responsible for the operation of temporary data based on specialised cache entities that individually applies sophisticated strategies for determining which data should be kept in a storage resource. At the same time, these entities supervise and register the access to data and storage resources. They control individual storage resources on behalf of users and applications; furthermore, those components implement the mechanisms of interaction in order to work together.

Wide accessibility

Cache reference model and cache operations seek to provide temporary storage capabilities to a wide range of grid applications, services and users. In each location of the distributed system, the applications require to execute access operations to manipulate the temporary data. The differences related with each particular technology are transparent for clients.

Security

The distribution of data over a grid makes data protection much more difficult than on closed systems. Data on grids may be stored in different locations but all storage sites are not accredited to receive data. Achieving a high security level can be mandatory for particular applications or virtual communities but security is always a trade off between inconvenience for the users and the desired level of protection.

In order to permit users to use specific security mechanisms, clients must integrate invocations to protection function and caches access operations. In the grid enviroment many security functions need to be provided such as:

- Reliable authentication of users.

- Secure transfer of data from one grid element to another.
- Secure storage of data on a storage resource.
- Access control for resources such as data, storage space or computing power.
- Anonymization of sensible data entities.
- Tamper-proof logging of operations performed on data entities.
- Traceability

The features that should protect data [106] while it is being stored on a grid cache, are access control and anonymization. Users need to trust the servers on which their data are going to be processed. Our research team proposes an access control mechanism that provides a decentralized permission storage and management system. It also include an encrypted storage mechanism. All permissions are encoded in certificates, which are stored by their owners and used when required. Permissions can be created on demand, by the owners of the resources or by administrators to whom this responsibility has been delegated, without the need to contact a central permission storage system [105] [107].

Uniform operations and interfaces

The operations and interfaces necessary to handle temporary data are uniform. The components that operate temporary data support a standard set of operations and predefined interfaces. Thus, in each location the requests are interpreted in the same manner; similarly the information for describing the state and behaviour of the data and system components is represented in a common form.

Enabling the control of resources

The infrastructure uses resources gathered from multiple locations. Following the model proposed, each location permits the use of its resources for partners in an automatic way. The infrastructure permits sharing these capabilities in the grid environment and permits to operate of the collaborative system for temporary data management.

Coordination for effective operation

The use of distributed resources and components is organised for efficient global utilisation. This requires that components implement mechanisms that organise collaborative actions to get a global effect on the system function. In this way coherent actions regulate the work distribution between components.

Performance Monitoring

The proposed infrastructure provides the ability to measure the performance of data operations and cache efficacy. In this context, it defines and provides the appropriate information to describe performance. This information is detailed and provided in an opportune way so it permits to take the decisions for operating the system in the most effictive way.

Chapter 5

Grid Cache Service (GCS)

In this chapter we describe the design and implementation of the base component for collaborative grid caches, the *Grid Cache Service (GCS)* that implements the model and specifications proposed in the last chapter.

We first present an overview of the GCS. Section 5.2 presents the design principles of the GCS, then the relation of the GCS design following the reference model described in Section 4.2, Section 5.4 describes the implementation of a prototype of the GCS, then section 5.5 presents the use of the GCS capabilities, then we present some performance tests on the GCS prototype. The chapter finishes with a general discussion.

5.1 Overview

As our main goal is to operate temporary data in grid environments in a dynamic way, we decided to use a group of caches distributed in the grid that collectively provide storage capabilities for temporary data [24]. Figure 5.1 illustrates a scenario for the utilisation of a group of GCSs to operate temporary data in grid environments in a dynamic way. A client invokes a GCS to store temporary data, if the GCS does not have available storage resources it can invoke the access operations of remote caches to store the data across multiple grid locations.

In this scenario each cache is a Grid Cache Service (GCS) which individually manages the local storage resources to provide the temporary storage capabilities of the system. In this chapter we describe the individual component of the system: the *Grid Cache Service*. The collective operation of the group is analysed in Chapter 6.

The *Grid Cache Service* is the active element that supports the infrastructure. It implements the storage, control and collaboration layers of the reference model (Sec-

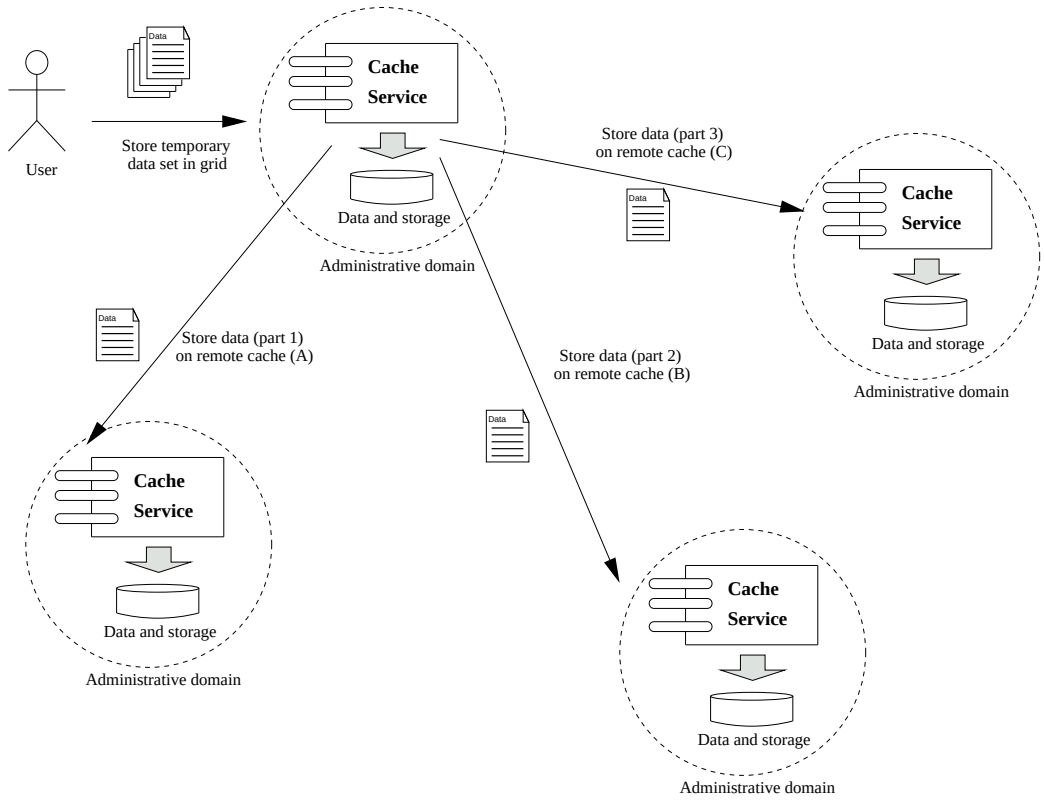

Figure 5.1: GCS cache extensibility

tion 4.2). The design of the system implementing the coordination layer is presented in Chapter 6.

We have implemented a prototype of the Grid Cache Service (GCS) which is used as a demonstration of the grid caching approach developed in Chapter 4. The prototype provides support for the specified operations and cache information.

The Grid Cache Service Concept

The *Grid Cache Service (GCS)* is a local administrator of temporary storage and data which is implemented as a grid service that provides basic cache capabilities following the reference model described in Section 4.2. Therefore, GCS implements interfaces and associated behaviour of the cache operations: access, monitoring and configuration (see Section 4.3) for the operation of temporary data.

The GCS Functions

The main function of the GCS is to ensure that a specified data entity exists in a storage resource for a limited period of time and to register detailed information about the data and cache actions.

In this context, the GCS offers these main functions:

- management of underlying storage resources for temporary data based on cache techniques.
- provides temporary storage space on demand to store data entities (files) through defined access operations.
- provides detailed and real-time information about data and cache activity through defined information representation and operations.
- provides basic cache capabilities to monitoring actions of the data entities stored in the storage resource.
- supports collaborative cache operations to a wide range of clients (including other caches).

5.2 Design Principles

In order to satisfy the requirements for operation of temporary data in grids we use the infrastructure proposed in Chapter 4. In this framework we have elaborated a set the design principles to implement the Grid Cache Service; these principles establish the main features of the implementation.

The design principles define a GCS as the basic unit of the system. These principles guide the development of our grid caching approach. They support the logical separation of the functions proposed by the reference model and implement the defined operations and information structures. Our purpose is to allow the composition of GCSs that working together to access and manger temporary data in grid environments.

5.2.1 Cache Virtualization

The virtualization is an abstract view of the behaviour of the data in a cache mechanism. This abstraction is supplied by cache operations plus cache information implemented by a cache service. The information and operations permit to get a description of the data and cache state and activity.

The cache virtualization consists of providing essential cache capabilities to other entities and caches. Clients delegate the operation of temporary data storing to GCS: to support this function, GCS also provides a detailed description of the data and cache activity.

The goal of the virtualization is to allow the integration of the resources and mechanisms that are not originally used for caching; this characteristic is essential to

aggregate resources dynamically. Virtualization seeks common functionality to access and share storage cache resources.

5.2.2 Autonomy

Autonomy establishes that a cache is a distinct and independent component in a distributed system. This principle changes the traditional notion of cache as an hidden mechanism: a cache is traditionally integrated into an application or solution as a internal component, for instance a cache in a database systems.

Since the cache is independent, external in fact, of the applications or particular solutions, it is distinguishable from other components of the system; this implies that the function of the cache mechanism is separated and recognised as specific in relationship with the rest of the components. Caches are also autonomous because they control their behaviour and resources (data and storage).

A cache service instance shares its resources via defined cache operations. The virtual organisation that deploys a Grid Cache Service establishes the shared degree of the cache resources and deploys the mechanisms to support them.

Finally, an individual cache instance does not depend on the cache group for its operation, and similarly operation of the cache group must not be compromised by the behaviour of an individual cache. An implication of this principle is that the basic cache functions of each individual cache instance are not affected when the cache works isolated or in a collective way.

5.2.3 Accessibility

The principle of accessibility establishes that the cache data content, resources and information are accessible and shareable for a wide variety of client applications, rather than a particular application or solution. Accessibility implies the visibility and awareness of the specific function of the cache mechanism which can be used for multiple applications or system components. This principle seeks to increase the re-utilisation level of the data and resources managed by the cache system.

Figure 5.2 illustrates the accessibility principle. The cache content and information description that describes its activity are available for a wide range of clients in order to increase the level of re-utilisation.

This principle implies the exposition and availability of the information that describes the configuration and activity of the data and the cache. The access to this information in a standardized way permits to build a detailed view of the behaviour

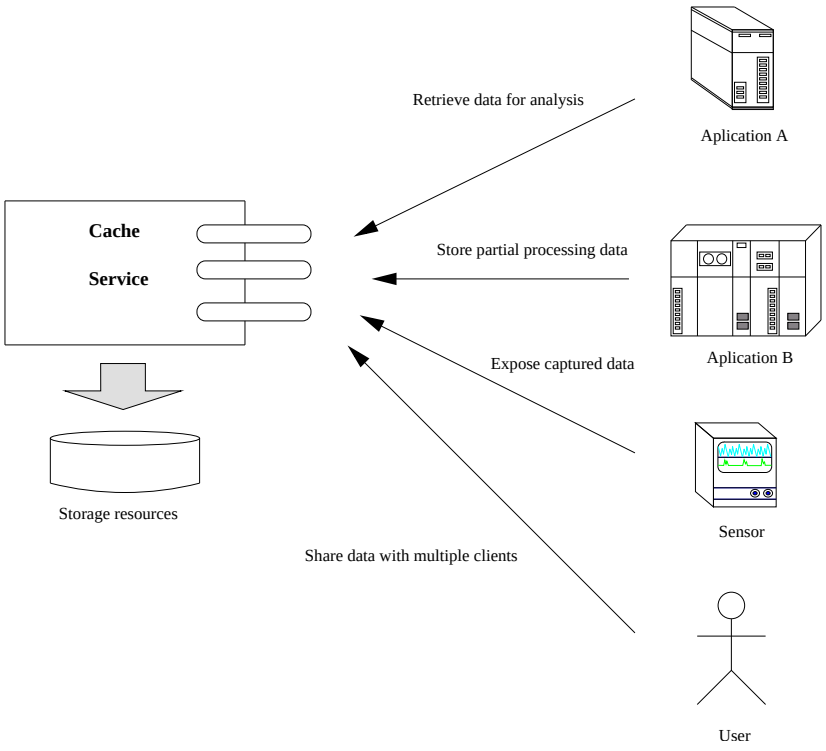

Figure 5.2: Cache accessibility

of the system. This same information permits to implement higher-level temporary data management mechanisms.

The principle of accessibility establishes that the cache operations are exposed by standard interfaces to a wide range of clients (applications, human users, devices, other grid services, and so forth). Similarly, the same interfaces are used for interaction with other caches. This last notion seeks to facilitate the collaboration between caches.

5.2.4 Uniformity

The principle of uniformity establishes that the cache capabilities are provided in a similar way in all the distributed systems. This implies a minimum standardisation of the cache operations and information supported by each GCS.

The principle of uniformity seeks to enable multiple caches to work together using the individual cache capabilities. The goal is to permit the interaction between caches invoking reciprocal and similar operations through groups of caches. Uniformity permits the aggregation of the cache capabilities provided by the virtualization of distributed and diverse resources.

While virtualization in principle provides abstraction of the individual operations

and control of a cache service instance, the principle of uniformity makes possible the collective operation of multiple GCSs that provides the same abstraction of data and storage resources.

Uniformity includes support for standard information about underlying resources; this information is exchangeable by any elements of the collaborative cache system.

With the notion of uniformity, diverse clients have a single perception of the cache. This seeks to provide ubiquitous accessibility and interaction to a wide range of grid applications, services and users.

5.2.5 Extensibility

This principle establishes that collaboration between caches is extensible to all possible cache operations. This includes the access, monitoring and configuration operations. Extended cache operations support the composition of complex capabilities from basic cache operations. This seeks new possibilities for exploiting the real potential of collaborative caching.

In contrast to collaborative web cache systems, the interactions between GCS must include all types of operations between caches. Extensibility thus permits a cache to invoke any operation of another cache service. For example, a cache can put or get data any other cache. In contrast, in other wide area distributed systems like the web, the interaction between caches of different domains is limited to data resolution.

This principle seeks to agree with the grid requirements for allowing a wide variety of higher-level programming abstractions and models more than enforcing a particular approach. So, for example, a grid application can use a group of cache services as a data disseminator while other can use the same infrastructure to implement a particular form of replication. The extensibility principle seeks to expand the interoperation possibilities using the grid environment capabilities and opportunities.

The extensibility circumscribes monitoring and configuration operations. In this way, interaction processes can establish the state of the available resources to implement composed operations: For example, the interaction between several caches requires first to establish the state of available resources on remote cache services. Thus, before invoking a put operation of the remote service, the monitoring operations of remote caches can be invoked to know the resource capacity and availability.

5.3 Cache Model and GCS

In order to accomplish the goal of providing dynamically temporary storage, we propose to administrate the storage resources as cache mechanisms. As administrator of local data and storage resource the CGS is designed following the reference cache model described in Section 4.2. In this section we present how the GCS manages the temporary data based on the reference model.

5.3.1 Storage layer

The GCS is deployed for the management of the content of a local storage resource. The storage resource denotes component(s) like disk file systems or archival storage systems that store the data entities.

The I/O facilities of the underlying storage resources are used by the GCS to support the access operations. The GCS uses the interfaces of the underlying system and exposes them as access operations defined in Section 4.4.2. The implementation of the GCS translates the SetData() and GetData() operations to access API of the particular storage resource.

Our GCS prototype uses the Unix file system partition as a storage resource, and translates the access operations requests to local I/O system calls of the operating system. The GCS maintains information about configuration and characteristics of specific resources: It exposes the resource description using the cache entities information (see Appendix 4.6.1).

5.3.2 Control layer

The key function of the GCS is to register the access actions realised with data entities held in the underlying storage resources. The main functions realized by the control layer are:

- execute the cache replacement policies applied to data entities held in the storage resources
- register information about data entity actions
- check the time to live established for data entities
- register data access requests and data access transfers using the defined cache information

- manage and expose the cache information

The GCS registers individual cache and data activity using data actions elements (see Appendix C.1.2). This information is managed with a database system using MySQL DBMS. A mapping of the XMLSchema of the data information model (see Appendix 4.6.1) is, in fact, stored in the DBMS.

Monitoring operations are internally translated to queries to the database that registers data activity information of the GCS. This makes it possible to establish during cache service working the data and cache activity information in an on-line way.

5.3.3 Collaboration layer

To support the collaboration capabilities in a system composed by a group of caches, the GCS is implemented as a grid service that provides cache capabilities to a wide range of clients including mainly other GCSs. These capabilities are supported by cache operations defined in Section 4.3.

WSDL [32] is used to describe and expose the cache operations in the grid. The cache service is built and deployed using software components and predefined services provided for the Globus Toolkit middleware version 4.1 [54]: it uses Delegation [63] for authorisation and GridFTP [4] to transfer data between cache locations.

The operations are implemented following a protocol based on exchange of XML request-response messages. Each operation is invoked using a request element that contains the parameters of the invocation and a response is given using a response element as defined in Appendix D.

The GCS implementation translates the access operation requests to invocations to the underlying resources. Each grid operation is regulated and registered by the implementation of the control layer. Similarly, registered informations are exposed through cache operations to support monitoring processes of the service operation.

For example, the operation SetData() processes request messages to store a data entity in the GCS; the request contains a description of the data entity defined by the cache information schema described in Appendix 4.6. The operation provides a response message approving or disapproving the request. If the request is approved the data transfer is started using GridFTP.

5.3.4 Coordination layer

The GCS prototype does not implement the coordination layer but it provides the essential information and capabilities to its implementation. The coordination layer can be implemented by an external and specialised module or service, e.g. Chapter 6 describes a system that organizes the collective work of the group of GCSs.

The high level coordination layer is implemented thanks to cache operations supported by each GCS. Organisation and coordination consists in the composition of collective capabilities arranged from basic operations provided by each GCS. Monitoring operations permit to observe and evaluate the individual operation of each cache and configuration operations allow changing the operational parameters.

5.4 GCS Prototype Implementation

In this section we describe some details of the GCS prototype implementation. It is used as a demonstration or "proof of concept" of the approach developed in Chapter 4. The main function of the GCS prototype is to process cache operations in the storage resource location. For example, the GCS prototype mainly supports process cache information about the data and cache instance activity.

This thesis aims to specify, design and implement a software component, the GCS. GCS is the basic collaborative brick that one can use to build temporary data management systems. The goal of prototype is not actually to implement an effective operational GCS system but to check that it is functional, that operations are functional.

Similarly, in this prototype we do not pretend to implement an optimal caching system. Therefore, as proof of concept, no extensive study of the GCS configuration and operational parameters is done and are very basic. These interesting issues are pointed as future work.

5.4.1 Grid Platform

We developed the prototype for the Globus Toolkit 4 [63] platform. Globus Toolkit is a suite of tools to develop and deploy grid systems and applications.

The Globus Toolkit supports the standard Open Grid Services Architecture (OGSA) [55] to build grid systems. OGSA defines a standard open architecture for grid-based applications. OGSA supports service oriented distributed computing with web services. The Web Services Resource Framework (WSRF) [5] is a specification

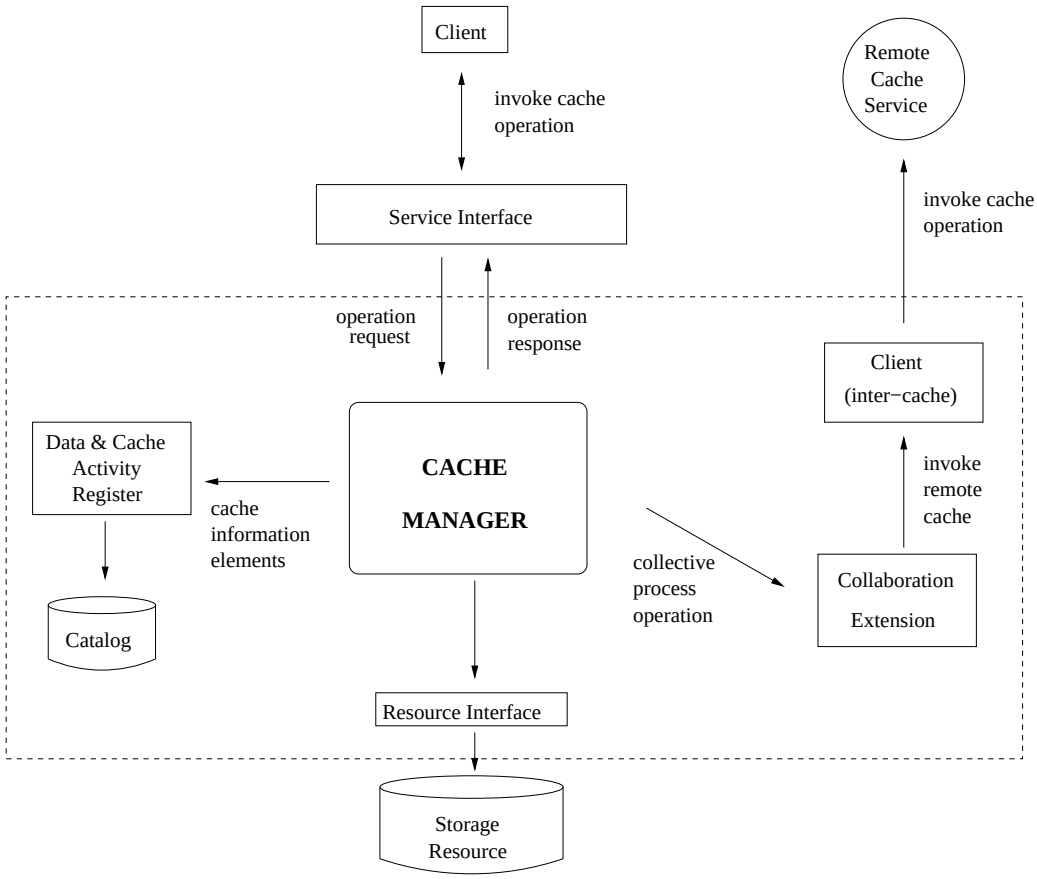

Figure 5.3: GCS Prototype Architecture

to develop service-oriented applications with Web Services. OGSA/WSRF have been standardized by the Open Grid Forum OGF [62].

The use of grid services requires Web Service Description Language (WSDL) [32] which is an XML based language used to describe the interfaces of Web services in a standardised way, and a protocol to exchange Web service requests and responses. The most frequently used protocol for Web service communication is SOAP [132], which is a protocol that enables to exchange XML has been encoded messages using the HTTP communication protocol. The prototype was developed in Java programming language to be executed as Globus WS java container.

5.4.2 Prototype Architecture

The GCS prototype implementation is composed by five main modules. Figure 5.3 shows the general architecture of the prototype.

The *Client* invokes the cache operation of the Service Interface sending the XML *OperationRequest* element as parameter, an example using the SetDataRequest ele-

ments is presented later in Section 5.5.1. The request is enclosed in a SOAP message.

The *Service Interface* receives the cache operations invocations. This module is implemented as a grid service using the tools and libraries provided by Globus Toolkit. A file WSDL describes as *portTypes* the cache operations introduced in Section 4.3, using as parameters the operation definitions of Appendix D.

The Service Interface is deployed into the Globus WS Java Core container. The container takes responsibility for many of the underlying logistic issues related to communication, messaging, logging, and security [63].

The *Cache Manager* module processes the cache operations using cache information elements as data structures. This module is mainly supported by the class called CacheImpl. This class implements the cache replacement mechanisms and a method for each cache operation. An example of the execution of SetData operation is presented later in Section 5.5.1.

The Cache Manager implements three basic replacement methods *LRU*, *LFU* and *SIZE*. It uses activity information registered in activity information elements (see Appendix C.1) and entity information elements (see Appendix 4.6.1) to support the replacement method functions. Section 5.4.4 describes the replacement method implementation. The Cache Manager invokes the Java I/O system as resource interface to handle files in the local filesystem. The Cache Manager invokes the data and cache activity register module to query and update cache information about the executed actions.

The *Data and Cache Activity Register* module manages the data and cache activity catalogue; it uses activity information elements (see Section C.1) to administes activity information. These elements are mapped to tables handled by the database system.

The *Collaboration Extension* represents extension modules that execute collaboration procedures (implemented by operations and specific to collaboration requirements). Currently, the prototype implements simple extensions for the GetData and SetData operations that transfer the original request operation to remote caches.

These extensions select the remote GCS in a *round robin* way; at is, a remote GCS are selected in the order that found in the element "members" of the Cache Group entity information element defined in Appendix 4.6.1. GCS holds an instance of this element for this purpose which is configurable with the SetCacheGroup() operation (see Appendix D.3.3). Later, in Section 6.2, an application scenario is described where the GCS uses a specific extension module: It invokes a special service for coordinating of collective operations between caches.

The Collaboration Extension modules uses an instance of the GCS client implemented with GCS API (described below) to invoke cache operations from other

GCSs.

Instance Configuration

To permit a flexible deployment in the globus container, the GCS is configured through the *SetCache()* configuration operation (see Appendix D.3.1) using a *Cache* entity information element as defined in Appendix 4.6.1. This operation sets the key operational parameters like the replacement method, the default time-to-live and complementary information like the owner organisation and geographical location.

5.4.3 GCS API

A Java API is provided to use the GCS prototype implementation. The API is composed of two groups or packages of classes:

Cache Information package Called *package gridcaching.generated.ci*, this package is composed of classes for Cache Information Element definitions (Section 4.5 and Appendix 4.6) and Cache Operations element definitions (Section 4.3 and Appendix D).

This package consists of the classes that handle the element definitions as Java objects. A Java-to-XML binding is used to convert from Java objects to XML documents; it enables the programmer to deal with the data defined in the cache XML elements through an object model which represents that data. The XML Schema that defines the operations and information cache is used to generate automatically the equivalent Java classes; the Castor library is used for this purpose [50].

Cache Implementation package Called *gridcaching.services.lcs.impl*, this packageis composed of classes that implement the cache "business logic" mainly handling by the information represented in classes of the Cache Information package. The Cache implementation package is composed of the classes that implement the operational modules of the prototype architecture described above (see Section 5.4.2).

Section 5.5.1 presents an example of utilisation of the GCS API for an application client. Figure 5.4 shows the UML class diagram of the GCS prototype generated from the source code.

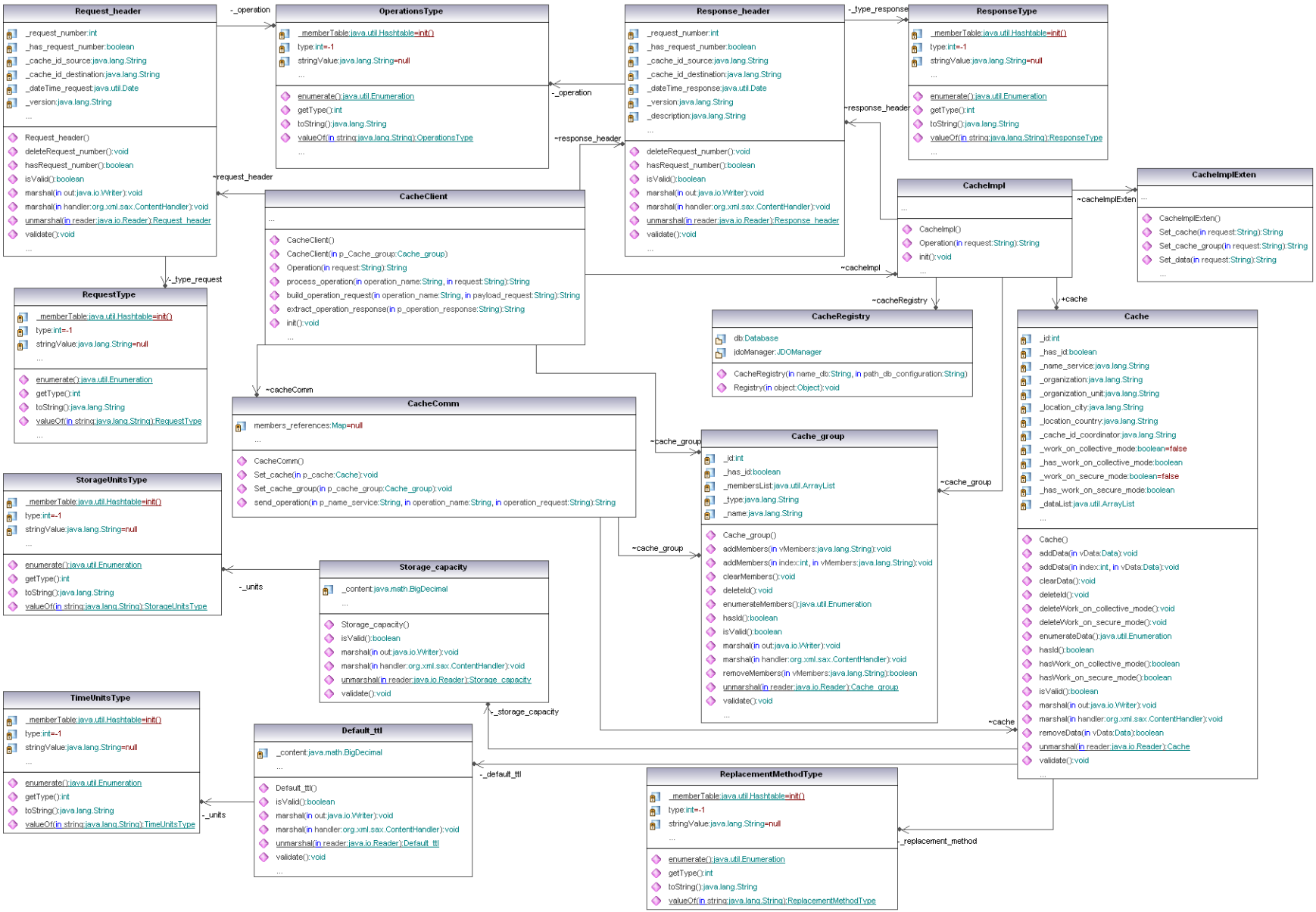

Figure 5.4: UML diagram of GCS prototype classes

5.4.4 Replacement Method Implementation

In this section we present the implementation of the replacement methods available in the GCS. The prototype uses the data activity information registered by the GCS. The GCS prototype implements three basic replacement methods: LRU, LFU and Size. The replacement methods (see Appendix A.1) are algorithms that select the data to be removed from the cache; the replacement method usually requires particular input information, which is related with the activity of the data. This information is fundamental for methods and policies based on recency and frequency strategies.

Cache operation demands, for example, the request operation for adding a new data entity into cache, activate the execution of the replacement methods. The GCS implementation queries the internal database that registers the *DataAction* elements with individual information about the access events related with the data entities stored in the cache.

To apply the LRU replacement method, the GCS submits a query that selects the data entity with the DataAction entry with the eldest *DateTime* field. For the LFU method implementation, the GCS submits a query that selects the data entity with the least number of DataActions registered. The Size method is supported by a query to data entities entries that selects the data entity with the largest size value.

5.4.5 Cache Activity Registry

The GCS continuously registers the events related with access to cache content. These events can be originated by requested operations or internal cache management. This information constitutes the data and cache activity and is handled with a database management system. GCS implementation uses the *DataActions*, *Request* and *Transfer Data* elements, defined in (section C.1), to register data and cache activity information. The API supports the data binding between Java objects, XML documents and relational tables for cache information persistence; the Castor library is used for this purpose. The cache elements are mapped to a mySQL database instance deployed in the same GCS installation.

5.5 Using GCS Operations

In this section we describe the utilisation of a representative cache operation. We present this operation as a typical example; the utilisation of other operations implemented by the GCS is similar. Therefore, our example shows the utilisation of the *SetData()* access operation that clients use to publish or put data into the GCS.

The invocation of this operation causes the execution of the replacement method and the registering of cache information which is described here.

5.5.1 Publishing Data with GCS

GCS clients create an explicit request to store data entities. This request is processed by the GCS as a petition for obtaining available temporary storage space. We call this capability publishing as it permits to expose and share data for a limited period of time between clients that use the GCS. This capability is mainly supported by the *SetData()* (see Section D.1.1) access operation.

To use the publishing capacity the clients create a *SetDataRequest* element that contains all the parameters of the SetData operation to be invoked. The requested element can be created by the client directly in XML or using the API provided to manipulate it as a Java object.

Figure 5.5 shows an example of the request header of the SetData operation; the fields contain information to be used by the client API to invoke the GCS operation, the request number permits to identify each individual operation request. The cache destination contains the reference of the Grid Cache Service instance. The version field is used to distinguish between future versions of the service. The request element also contains the data entity element to be published and an optional metadata element discussed below.

```
<SET_DATA_REQUEST>
  <REQUEST_HEADER>
    <OPERATION> set_data </OPERATION>
    <TYPE_REQUEST> client </TYPE_REQUEST>
    <REQUEST_NUMBER> 454799 </REQUEST_NUMBER>
    <CACHE_ID_SOURCE></CACHE_ID_SOURCE>
    <CACHE_ID_DESTINATION>
       http://liris-7080.insa-lyon.fr:8080/wsrf/services/gridcaching/LCS
    </CACHE_ID_DESTINATION>
    <DATETIME_REQUEST>
       2007-07-26T16:00:00
    </DATETIME_REQUEST>
    <VERSION>0.1</VERSION>
  </REQUEST_HEADER>
</SET_DATA_REQUEST>
```

Figure 5.5: An example of the request header of SetData operation

Figure 5.6 shows an example of a data entity to register with the SetData operation in which the fields contain information that describe the data entity: the *id* permits to identify individual data; the name field contains a Logical File Name (LFN) used

as unique identifier of the data entity, the LFN is defined by the virtual community that deploys the GCS ; and the *parent id* permits to register it as derived from other and the group id permits to register it as part of the group. These elements are used to register hierarchical and collection relations. The share type element indicates if the data entity can be accessed by other GCSs.

```
<DATA ID="78784">
  <NAME>
    research_experiment_2359674_file
  </NAME>
  <TYPE> FILE </TYPE>
  <SIZE UNITS="MEGABYTES"> 256 </SIZE>
  <INTERNAL_PATH_LOCATION>
    /disk7/experiencies/data
  </INTERNAL_PATH_LOCATION>
  <DATA_ID_PARENT> 1258 </DATA_ID_PARENT>
  <DATA_ID_GROUP> 789 </DATA_ID_GROUP>
  <OWNER_USER>
    Peter Morgan
  </OWNER_USER>
  <OWNER_ORGANIZATION> Liris </OWNER_ORGANIZATION>
  <SHARE_TYPE> permited </SHARE_TYPE>
  <CREATION_DATE>
    2007-02-26T16:00:00
  </CREATION_DATE>
</DATA>
```

Figure 5.6: An example of a data entity element for SetData operation

An example of the optional metadata is shown in Figure 5.7 in which the fields contain descriptive information about the data entity: the *id* permits to identify each individual metadata element; the key word list contains items that describe the data entity content; the software provenance contains the software product that produces the data entity; the target software field contains the application that is expected to process or use the data entity; the field external metadata service contains the reference to service that manages metadata; and the field identifier for metadata external permits register an external identifier to this metadata. The description field used is textual and complementary information about other aspects of the data content. The annotation fields permit to add descriptions related with the utilisation of the data entity.

Figure 5.8 shows a simple example of the invocation of the SetData operation using the provided API. Each entity and activity information elements (see Appendix 4.6) is represented by a class that permits manipulating the element attributes. In the example, the SetDataRequest class is mapped into an XML element (marshalling) before operation invocation.

XML is used extensively with Web services as a standard, flexible, and extensible

```
<DATA_METADATA ID="79896">
  <KEY_WORD_LIST>
    astrophysics star2569 optical gama phenomenon
  </KEY_WORD_LIST>
  <SOFTWARE_PROVENANCE>
    optical telescope version 4.2.5
  </SOFTWARE_PROVENANCE>
  <SOFTWARE_TARGET>
    optical analyser 2.6
  </SOFTWARE_TARGET>
  <EXTERNAL_METADATA_ID> 785269 </EXTERNAL_METADATA_ID>
  <EXTERNAL_METADATA_SERVICE>
    metacatalog service
  </EXTERNAL_METADATA_SERVICE>
  <DESCRIPTION>
    raw data captured by  telescope 125 on 2007-02-26t15:00:00
  </DESCRIPTION>
  <ANNOTATION>
    complet phenomenon exposition
  </ANNOTATION>
</DATA_METADATA>
```

Figure 5.7: An example of a metadata element included in SetData operation

data format.

Service Provider Side

The GCS receives the request as a XML element. The GCS maps the XML element into an instance of SetDataRequest class (unmarshalling) for manipulating the operation parameters. This includes the recursive mapping of the Data and Metadata elements into the respective objects. The GCS then accomplishes its functions manipulating objects which represent the different cache elements defined.

The GCS checks the size of the data entity that the client requests to store in the cache. It establishes if there is sufficient space in the resource storage to place the new data, if not it executes the replacement mechanism to obtain available space. In this form it definitively establishes if it can store the data request by the client. It registers the operation request, the replacements and data actions that were done using the respective cache information elements defined for this purpose (section 4.6). Finally, the GCS builds a response to the client using the *SetDataResponse* element that is returned to client.

Figure 5.9 presents an example of the response element returned by the invocation of the SetData operation of the GCS. In the Java code example (Figure 5.8) the

```
CacheClient cacheClient = new CacheClient();
Request_header  request_header = new Request_header();

String lcs="http://liris-7080.insa-lyon.fr:8080/wsrf/services/gridcaching/LCS";
request_header.setCache_id_destination(lcs);
//...

Data data = new Data();
data.setName("research_experiment_2359674_file");
// ...

// Create the operation request
Set_data_request set_data_request = new Set_data_request();
set_data_request.setRequest_header(request_header);
set_data_request.setData(data);

// Marshall request operation to XML element
set_data_request.marshal(stringRequest);

// Invoke Grid Cache Service operation
stringResponse = cacheClient.Set_data(stringRequest);

// Unmarshal operation reponse from received XML element
Set_data_response set_data_response = new Set_data_response();
set_data_response = Set_data_response.unmarshal(new StringReader(stringResponse));

// Check the response
if(set_data_response.getResponse_header().getType_response()==ResponseType.SUCCESSFUL)
System.out.println("Data entity registered in Grid Cache Service");
```

Figure 5.8: An example of the Java client side code to invoke a GCS operation

SetDataResponse class is used by the client to get a response from the GCS. The field type response contains the answer of the GCS to the requested operation: it examines the field type response to determine if the operation was processed with success. In some situations, the complementary actions related with operation processing are done, in this case the transfer using GridFTP. If the GCS can not process the requested operation, the type response field contains the value *failed* and additional textual information can be included in the field description.

5.6 Grid Cache Service Performance

We present some wide area performance measurements for our Grid Cache Service prototype (GCS). This experiment does not pretend to evaluate an optimal caching system; it just presents a wide area experiment that seeks to prove that the GCS is functional and that seeks to establish the system management overhead. Nothing more. Therefore, as proof of concept, no extensive study of the configuration and operational parameters (e.g. number of sites, bandwidth, etc.) is done. These

```
<SET_DATA_RESPONSE>
  <RESPONSE_HEADER>
  <OPERATION> set_data </OPERATION>
  <TYPE_RESPONSE> successful </TYPE_RESPONSE>
  <REQUEST_NUMBER> 454799 </REQUEST_NUMBER>
  <CACHE_ID_SOURCE>
    http://liris-7080.insa-lyon.fr:8080/wsrf/services/gridcaching/LCS
  </CACHE_ID_SOURCE>
  <CACHE_ID_DESTINATION> </CACHE_ID_DESTINATION>
  <DATETIME_RESPONSE>
    2007-07-27T16:08:58.541+02:00
  </DATETIME_RESPONSE>
  <VERSION > 0.1 </VERSION>
  <DESCRIPTION></DESCRIPTION>
  </RESPONSE_HEADER>
</SET_DATA_RESPONSE>
```

Figure 5.9: An example of a SetDataResponse of SetData operation

interesting issues are pointed as future work.

The tests include three the sites that compose the GGM project [94] [86]. The local site, which invokes cache operations, is located in Lyon. Remote sites run a GT4 container and deploy the Grid Cache Service as well as a GridFTP server; they are located at the Toulouse and Lille French cities. We performed 1000 operations initiated from the local site. For each operation, a simple GCS client is created which invokes the cache operation request and gets the response from the remote cache service. The GCS processes individually each requested operation. For each access operation (publish or retrieve data entity) a GridFTP connection is created, then the data transfer is initiated between the sites.

For all operations, we report the average time taken for each cache operation which is composed of request and response phases. This is called the *Request response* operation component. The *Request response* is the time elapsed between sending an operation request and receiving the response from the GCS. For access operations, we report two additional measurements separately. The *Create GridFTP* time is the time required to create the GridFTP connection associated with the operation, and the *Transfer* time is the actual time taken by GridFTP to transfer the data entity required by the operation.

We ran these access operations 1000 times using files of 10 Megabytes and obtained the average time performance shown in Table 5.1. One can see from this table that the internal time spent by the cache procedure is very small, and corresponds roughly to 1% of the total elapsed time. We exhibit also the time to inititate the GridFTP transfer, which counts for 4% of the time. Altogether, the corresponding data rate during the transfer portion of the request is approximately 4.72 Megabits/sec.

Publish data operation		
Component Operation	Time(ms)	Standard Deviation
Request response	171.9	178.6
Create GridFTP	740.0	317.3
Transfer	16927.1	583.1
Retrieve data operation		
Component Operation	Time(ms)	Standard Deviation
Request response	203.2	285.8
Create GridFTP	714.2	268.0
Transfer	16952.1	549.7

Table 5.1: Performance for GCS access operations
Grid cache service performance for access operations of 1000 files of size 10 MB

There is a difference between the average times of publishing and retrieval operations. As expected, the time to request the data is higher than the time to publish the data: Indeed the GCS prototype makes a sequential search of data entries on the cache internal catalogue as part of *retrieve data operation*. The table also shows the standard deviation for each measured value. The variances in operations are due to variations in the wide area traffic between the three cities.

Next, we performed some representative monitoring and management operations: the operations for getting cache content, used storage resource, and replacement cache method. Additionally, we also checked the operations for setting the replacement cache method and for a complete configuration of the cache.

The operation *requests cache content* gets the entire cache content description i.e. the data entity information defined by our XML Schema for each data in cache. In this case it provides a list of data entities pre-stablished to 100 entries with a size of 53015 bytes. The operation *get storage resource* calculates the total storage usage by each time checking the size of data entities registered in grid cache service. The operation *get cache replacement* operation retrieves this cache information element from the current cache service configuration.

The *set replacement* operation changes only one cache configuration element. The *set cache configuration* operation changes several elements together such as default time to live, the replacement method, the group subscription, the description of the organisation that deploys the service, etc. We ran these monitoring and management operations 1000 times and obtained the average times shown in Table 5.2.

We first note that the *request cache content* operation (that retrieves 100 data entity items) is relatively efficient in comparison with request-response operation of the *retrieve data* operation, showed in Table 5.1. The reason is that the cache service prototype keeps the catalogue image in memory for improving cache processing.

Operation	Time(ms)	Standard Deviation
Request cache content	355.5	142.9
Get storage resource used	127.7	110.7
Get replacement method	84.7	9.0
Set replacement method	112.2	55.1
Set cache configuration	223.0	397.9

Table 5.2: Performance for GCS monitor and management operations
GCS performance for some monitor and management operations

The results also show that the time required for cache management is marginal in comparison to data transfers. These tests provide insights into the performance of typical and individual GCS operations. They are the fundamental support for the temporary data in operation grids with a group of GCS working together.

5.7 Discussion

Since our main goal is the operation of temporary data in grid, our solution is based on the utilisation of a group the caches which collectively provide the temporary storage capacities required by the grid environments. However, this functionally is built from individual capacities supported by each cache that composes the system.

The design of the GCS responds to characteristics of the grid environments where the resources are shared and distributed. Our solution, thus, proposes to gather temporary storage from capabilities within a group of GCSs that administrate individual storage resources. This approach requires considerable effort to operate and organise the GCS working together.

The design principles adapt the different requirements and constraints related with the operation of the temporary data in grids. The most important implication is related with the notion of the cache as a distinct and independent component in a distributed system.

The GCS proposes the notion of the cache as a shared and accessible component. This changes the traditional perception of the cache as a hidden mechanism frequently integrated into application implementation or as an internal component of specific solutions (cache in database systems for example). The clients in different levels must be aware of the cache function and must frequently request to interact with it in an explicit way.

The decision to use WSRF specification and Globus middleware was done to obtain interoperability with a wide range of grid services and applications. However, we

do not use the WSRF Resource Properties to expose the cache information. Since the WSRF Resource Properties permit to clients to get multiple resource properties, query resource properties, and subscribe to resource properties.

However, the large amount of cache information is constantly produced. It mainly corresponds to data activity information that includes detailed actions registry of each data entity in the GCS. This characteristic is not suitable to use WSRF Resource Properties to expose the cache information.

The GCS makes a high level abstraction of the access operations. The GCS registers individual actions to the data entity (file) as a whole, however, the cache access operations do not correspond to typical *read* and *write* OS filesystem operations. The granularity of these access actions depend on the client implementation which can retrieve a data entity (file) and make many *read* and *write* actions to the same data entity. The GCS abstraction registers the "macro" operation similar to the *put* and *get* operations to the complete data entity. In this sense, the GCS requires an explicit requirement to put or get a data entity to or from the data storage resource administered by the GCS.

The placement of GCS on grid

The GCS design principles admit a flexible placement of the service in the grid. In general terms, it is possible to deploy a GCS with each storage resource installation to be shared in the grid. However, the proximity to data consumer is frequently the most important criteria used. The proximity to data consumer permits to save network resources and transmission time. The temporary storage resources in the specific location can be administrated by the GCS instance.

A common choice is to use the GCS as a proxy cache for an organisational site, its function is similar to early Internet gateways, where clients inside of the organisation site invoke the local instance of the GCS, and the GCS invokes the services of remote GCS deployed in the virtual organisation by grid partners.

The integration of heterogeneous storage resources

The virtualization of the cache mechanism permits the GCS to incorporate heterogeneous storage resources. This implies that the implementation must invoke the proprietary interfaces of the distinct storage solutions. The GCS prototype implementation wraps underlying filesystem I/O functions but other implementations are possible. In this sense, different systems can be operated by several GCS instances that hide proprietary technologies.

Cache content replacement

The GCS prototype supports three basic replacement methods. The data activity information available in each GCS instance provides information to support other

methods or alternative policies based on data and cache activity. The design the GCS allows the composition of these capabilities in external modules or classes to be loaded by the GCS instance, this is possible because all cache information is exposed in a standard way and the cache operations allows to manipulate the internal data entities.

GCS locally executes different tasks related with the management of the local resources. These tasks constitute essential functions that support the operation of the storage resources for temporary data in grids.

Monitoring

The fundamental feature of the GCS is the capacity of monitoring its operation in realtime. Since this feature is related with the capacity of supervising and observing the detailed data and cache activity, monitoring makes possible measurements to evaluate the operation the system. Monitoring is based on the similar conception for gathering information of each individual instances to build a global view the system.

Configuration

The GCS supports the configuration of its essential parameters to make possible the modification of the operation. The standard configuration support allows to establish common parameters for a minimal of operational coherency between different GCS instances working together. The configuration support is orientated to deal with particular or specific requirements based on relationships among GCS and the status of its instances themselves during operation.

Chapter 6

Temporal Storage Space (TSS)

6.1 Overview

We present the general design of the Temporal Storage Space (TSS) [23], a data management system that operates on temporary data in automatic form. TSSs design seeks to provide insight into the mechanisms of the coordination layer introduced in section 4.2.4. Similarly, it aspires to give a notion of the collaborative cache capabilities to be exploited with GCS.

We first present an overview of the TSS components. Section 6.2 presents examples of composition capabilities for disseminating, collecting, copy proliferation and control of temporary data using GCSs in grids. Similarly, the conception of prospective capabilities for monitoring and configuring the system are explored. Section 6.3 describes an experiment which simulates the TSS capability to provide storage space on-demand. TSS tries to give clients the perception that they have an unlimited capacity for storing temporary data. The clients delegate the temporary data distribution and operation to the TSS. The experiment is motivated by the *Data Dissemination* use case introduced in Section 2.1.1.

The goal of this chapter is not actually to implement an effective operational TSS system but to check that the GCS service is functional, that all operations (API) are functional and that coordination procedures can be implemented that offer interesting opportunities, in other words, this chapter proposes only a functional validation of the service and a "proof of concept" of the coordination layer. That is the reason why many parameters are fixed and some choices are not discussed.

This thesis aims to specify, design and implement a software component, the GCS. GCS is the basic collaborative brick that can be used to build temporary data management systems. There is no optimal management system. For instance, depending on the grid infrastructure, depending on the target applications, it can be beneficial

to collaborate in a peer to peer way or using a centralized coordinator mechanism or using multiple coordinator mechanisms. The appendix B present a description of caching archirectures. However it is not the topic of the thesis to study in depth these issues. As it is stated in the Chapter 4, this thesis does not address the cooperative layer.

The function of the TSS is to store a specified set of temporary data (files) at distributed locations where GCSs are deployed. It finds enough temporary storage in each location to hold large data-sets and then each GCS automatically frees unused storage following the cache techniques.

The functions of the TSS include: space discovery, identifying where there is available storage to place the specified data on the grid; transfer data, move data between remote storage resource efficiently; register and update content description information; and exchange information about the data activity state so that other sites may discover available data content and get information about the access activity associated with this data. Throughout cache operations, the service exposes state information about each data entity (file), including which access actions on the file have been realised.

Aspects that guided the fundamental TSS design decisions included:

- Design of a composable system based on Grid Cache Services (GCS);
- Operation based on reusable lower-level grid services;
- Use of cache operations, data and cache activity information to operate the system.
- Use of a flexible coordinator module to organise specific interaction between caches.
- Utilisation of the OGSA architecture and the [53] Web Services Resource Framework WSRF [55] for interoperability.
- Use of the Globus Toolkit 4.1 as middleware platform.

Following the OGSA/WSRF specifications, the general structure of the TSS consists of a group of GCS instances each one exposed as a grid service deployed in a Globus container. The container takes responsibility for many of the underlying logistic issues related to communication, messaging, logging and security. The cache services capabilities are exposed with the service interface for grid users and applications [55].

In the TSS, a coordinator module organises the interactions between GCSs to process the requests addressed to a set or subset of caches. It monitors the state and behaviour of the cache group. In the context of the TSS we call the individual cache

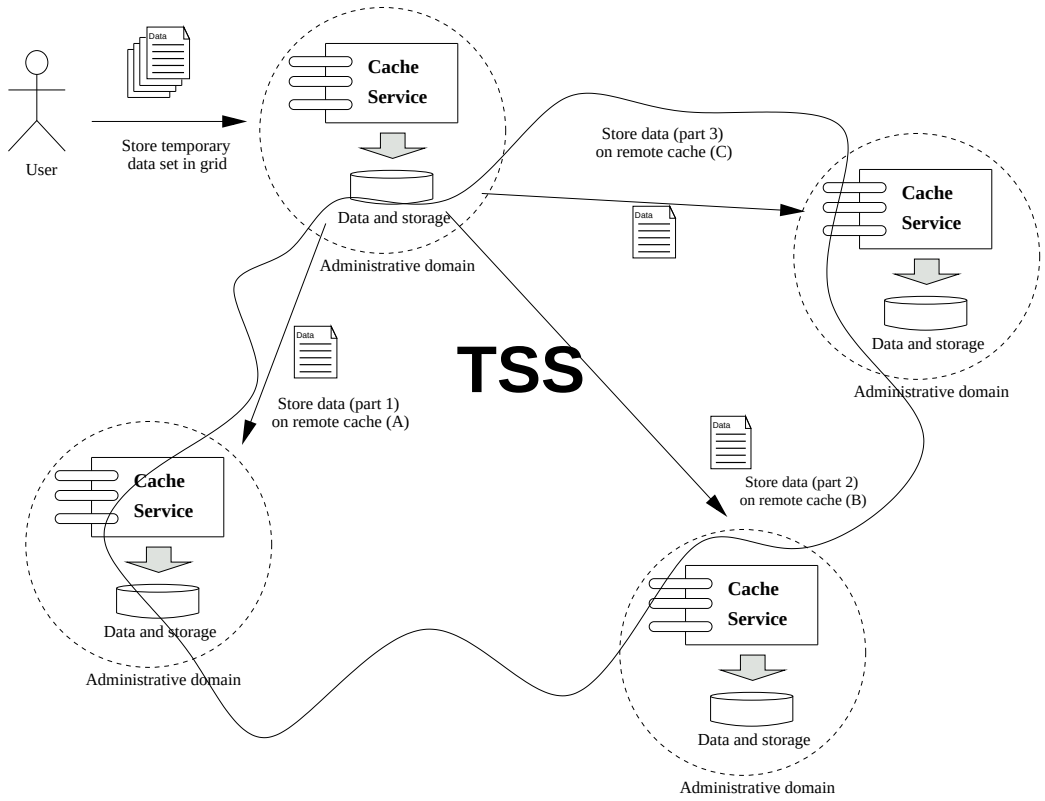

Figure 6.1: Temporal Storage Space (TSS)

service instance a Local Cache Service (LCS). Each LCS exposes cache information defined in Section 4.5. This information is accessible through standard grid interfaces to operations defined in Section 4.3.

To use TSS capabilities, clients invoke cache operations to access cache data content, get monitoring information about data and cache activity, and set values and parameters that configure the basic function of the cache installation. To perform access cache operations, the TSS depends on GridFTP (a non-web service) and two other WSRF services: RFT and Delegation [63].

The TSS is composed of a group of Local Cache Services (LCS) (described in Section 6.1.1) which are deployed at multiple organisation domains, and one Collective Cache Service (CCS) (described in Section 6.1.2). The LCSs work together to provide the base support of the TSS. Figure 6.1 illustrates the TSS components deployed in a grid.

6.1.1 Local Cache Service (LCS)

A Local Cache Service (LCS) is a Grid Cache Service instance deployed inside each location of the virtual organisation. A LCS provides cache functions to internal ap-

plication and user clients of the organisation. Working in collective mode it extends the cache capabilities with the support of the other grid cache services that compose the TSS.

Internally the LCS, like a gateway, serves operations submitted by organisational clients and externally serves operations required by other LCSs. LCSs translate internal data access operations into intercache operations invoking similar operations from the LCS deployed on other organisations of the virtual community.

Clients use these capabilities to share and reuse the data and storage resources inside the organisation. Simultaneously, the LCS offers cache functions to the LCSs deployed in organisations of the same virtual community.

In the collective mode, an LCS interacts with the CCS to process TSS operations in coordination with other LCS. These interactions support the TSS capabilities described later in Section 6.2.

6.1.2 Collective Cache Service (CCS)

The Collective Cache Service (CCS) is a service that allows LCSs to coordinate their interactions. The CCS accepts the same cache operations interfaces defined in Section 4.3 but the treatment is different. The invocations CCS interfaces are translated into one-to-one interactions including two or more LCS. A CCS is a special service for a group of GCS. It accepts access operations from LCS and monitoring and configuration operations from users or grid resources administrators.

The CCS executes a specialised module called the coordinator which implements the specific procedures related with collective decisions; the coordinator invokes classes implemented by users that are specific to application requirements. The coordinator makes its work invoking cache operations from multiple LCSs to arrange interactions. The coordinator implements the logic that embodies rules to achieve the collective cache actions based on the cache operations utilisation.

Typically, a coordinator invokes the cache operations of the LCSs to evaluate their state and arranges intercache data access operations. The coordinator also can implement the procedure for establishing the available storage resource in the system. It invokes the monitoring operations of the LCSs to gather information about the current state and availability of the caches and resources.

The coordinator uses specific algorithms to evaluate the cache information gathered from the TSS. For example, a global data placement algorithm may apply multiple criteria or metrics or combining of them. The input values of these algorithms are obtained with monitoring operations retrieving cache information from LCSs.

6.2 TSS capabilities

This section describes how the LCS and CCS support the main capabilities supported by the Temporal Storage Space (TSS). These capabilities are:

- Data Dissemination (Section 6.2.1)
- Data Collector (Section 6.2.2)
- Copy Proliferation Control (Section 6.2.3)
- Monitoring (Section 6.2.4)
- Configuration (Section 6.2.5).

The capabilities described in this section make reference to the grid implementation using GT4 middleware [63].

TSS does not pretend to implement an optimal caching system; it just presents a example that proves that the GCS is functional, nothing more. Therefore, as proof of concept, no extensive study of the configuration and operational parameters such as number of caches, coordinators, bandwidth, etc. is done and collaboration schemes are very basic. These very interesting issues are pointed as future work in the Chapter 7.

In the particular scenario proposed by TSS, the coordinator is typically a high level cache instance in a collaborative hierarchical organisation (see Section B.1.1). The role of this coordinator is the responsability of the designer of the coordination layer to specify it. This chapter details examples of coordinator for each TSS capability proposed; however, coordinators can be used for many other tasks like as access patterns analysis, data replication, data migration, data indexing, etc.

6.2.1 TSS Data Dissemination

TSS data dissemination capability permits clients to store data in several grid locations. It places the data entities in automatic and dynamic form. It provides the notion of a virtual storage space available for the grid virtual community. TSS processes clients requests to store data in remote grid locations.

Figure 6.2 illustrates a TSS deployment with LCS. A local site disseminates data to several remote sites. The GT4 Delegation [63] service is used by TSS with the services or libraries that need the delegated credentials, for example GridFTP [4].

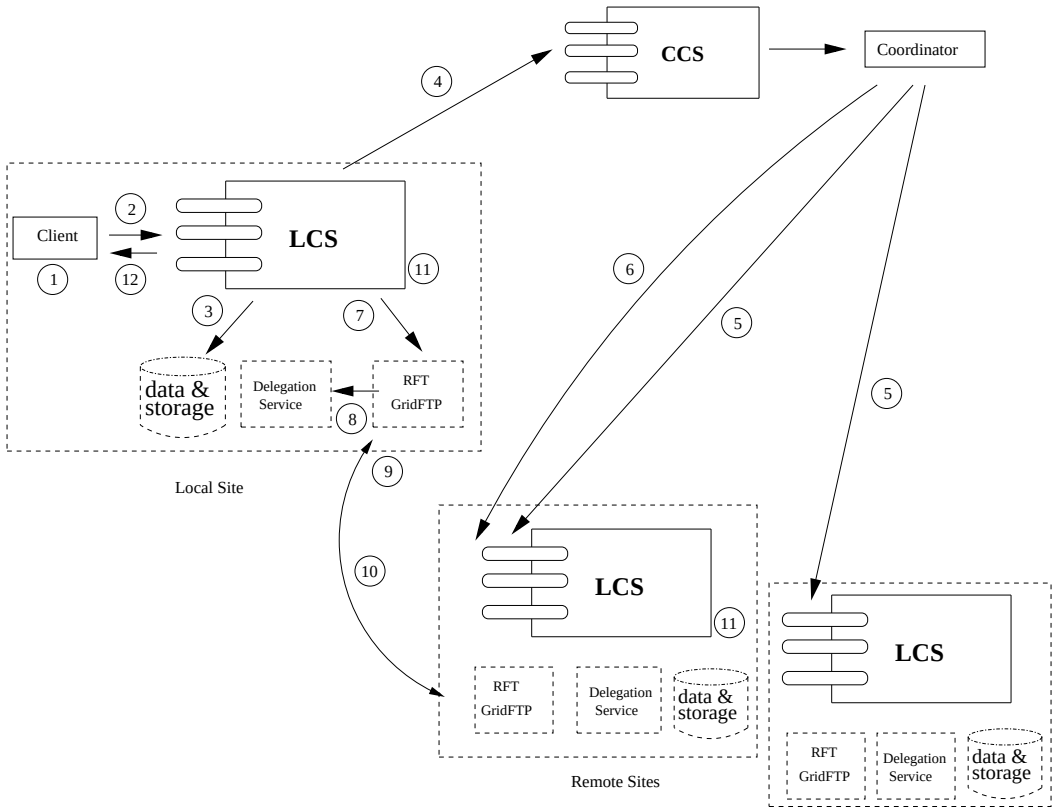

Figure 6.2: TSS Data Dissemination

The design tries to provide a flexible infrastructure to deploy different coordinator modules.

A coordinator invokes LCS cache operations to evaluate their state and arranges intercache data access operations. The TSS deploys a coordinator for data dissemination between LCSs. The coordinator disseminator implements the procedure for establishing the available storage resources in the system [1]. It invokes the monitoring operations of the LCSs to gather information about the current state and availability of the caches and resources. Then, it executes an algorithm that distributes the data (files) between LCS. Finally, it invokes the access operations of selected LCSs to execute the data transfers between locations.

The coordinator disseminator uses algorithm not discussed in this thesis, to evaluate which location will be selected to place a specific data entity; it can, for example, select locations according to available space, requests processed (workload), done replacements, etc. The placement algorithms can apply multiple criteria or metrics or combine them. Some of the input values of these algorithms are obtained with monitoring operations by retrieving cache information from LCS.

Placement algorithms permit to adjust the performance of the TSS following, for

[1]We assume that LCS regularly executes the replacement method mechanism to free unused storage space

example, the efficient use of resources or workload. A discussion about sophisticated placement algorithms is beyond the scope of this thesis. We use a simple mechanism for placement determination which selects the location with more space available. Cache storage capacity refers to the level of availability of the individual LCS storage resources: it is obtained invoking the GetStorageCapacity() operation (see Appendix D.2.9) from each LCS.

Data Dissemination Execution

Preliminarily the client prepares the *SetDataRequest* (see Appendix D.1.1) (1), containing an explicit description of the requested operation in terms of the data entities (files) to store, identified by their logical file names, the optional metadata to register in the LCS: it uses entity information elements (see Appendix 4.6) to describe the data entity and optional metadata.

The client invokes the SetData() operation interface of the LCS (2) and passes the SetDataRequest to place the specified data entities into collaborative cache storage space managed by the TSS.

The LCS executes its placement policy to establish if it can store data locally or if it must use any external storage (3). If the data must be stored externally, the LCS sends a intercache SetDataRequest to CCS asking for collective placement operation (4). The CCS gathers cache information about the current storage capacity from the group of LCSs. Then, it invokes the GetStorageCapacity() operation from each LCS in the TSS (5). CCS establishes the current storage capacity of each one LCS in the TSS. It executes the selector class (implemented by user and specific to applications requirements) to select the LCS where to put the data. The selector class also assigns subsets of data entities between the selected LCS. The list of selected LCS is returned to original LCS inside of the SetDataResponse element of the SetData() operation.

For this purpose, the CCS creates a *SetDataRequest* for each LCS where subsets of data entities will be placed. Then CCS invokes the SetData() operation interface of each LCS selected and passes the *SetDataRequest* asking to store a subset of data entities (6) [2].

The transfer process is started between LCS locations by using the RFT service based on the third-party GridFTP control (see Section 3.3.1). An RFT resource is created (7) to operate the transfer: the RFT resource retrieves the credential for the cache service (8). The RFT starts the transfer, which invokes GridFTP servers to perform the data transfer at a low level (9). The LCS source checks periodically the status of each data transfer request to ensure that each file was transferred

[2]We assume that GCS implementation supports third-party caches operations e.i. the CCS could initiate operations between two caches

successfully (10).

Once the transfer process is finished, each LCS updates its internal catalogue with the cache information related with the operation to make the information visible throughout the Grid (11). Finally, the LCS delivers *SetDataResponse* to client (12).

6.2.2 TSS Data Collector

TSS data collector capability permits clients to seek and acquire data dispersed in the grid. TSS provides the notion of an uniform virtual storage space constituted with distributed LCSs.

The TSS processes clients requests to collect data from multiple sources. This capability gathers data content stored in distributed resources that are managed locally by LCSs; they provide detailed information about data description and data activity. This information is analysed to select the desired data sources.

The CCS uses a collective catalogue with information about the content of the LCS group; this catalogue is used to resolve collection data. The coordinator executes queries to the catalogue to find the LCS that contains the requested data. Catalogue entries correspond to a data entity identifier and where the LCS is held. The coordinator maintains a weak consistency catalogue with cache information reports from LCSs.

Figure 6.3 illustrates a scheme of deployment of the Data Collector capacity with LCSs and CCS. The LCS at a local site collects data from one or more remote sites. Similarly to the disseminator function, the TSS uses the GT4 Delegation service with the services that need the delegated credentials and RFT is used for data transfers.

This capability requires that CCS holds a collective catalogue with description of data content of LCSs in the system. The collective catalogue is maintained using regular LCS content reports. LCS must periodically send to CCS information about its cache data content.

Data Collector Execution

We describe the case where the data requested by a client is not stored in the LCS that receives the request. A collaborative resolution process, therefore, is executed with the CCS support.

Preliminarily, the client prepares the *GetDataRequest* (see Appendix 4.4.2) (1) containing a description of the operation request in terms of the data entities (files) to

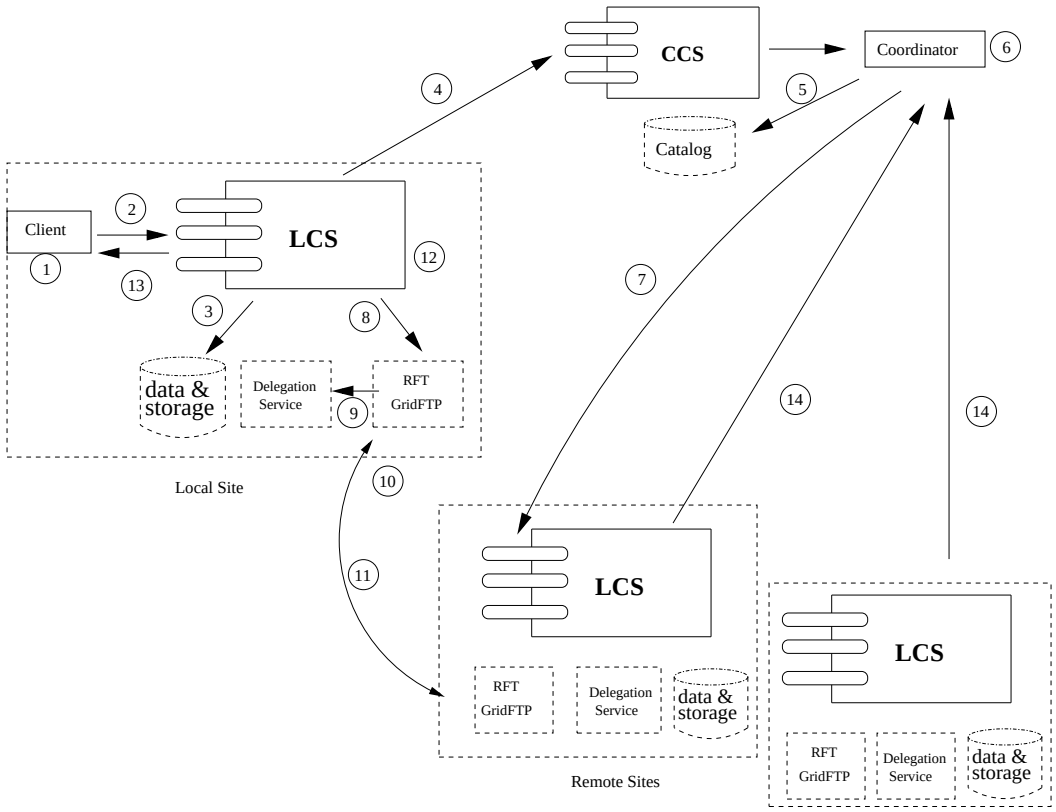

Figure 6.3: TSS Data Collector

retrieve, each identified by their logical file names that corresponds to an specific data entity (see Appendix 4.6.1).

The client invokes the GetData() operation interface of the LCS (2) and passes the GetDataRequest to retrieve the specified data entities from the virtual storage space managed by the TSS.

The LCS queries its local internal catalogue to establish if the data entity request is held locally (3) [3] and, if the data is not local, the LCS sends an intercache GetDataRequest to CCS asking for collective resolution operation (4).

The CCS executes the coordinator that queries its collective catalogue to establish the data location (5) [4]. If the data is held, multiple remote LCSs execute the selector class (implemented by the user and specific to application requirements) to select where the LCS gets the data (6).

The CCS creates a *GetDataRequest* for each LCS where data entities will be retrieved. The CCS then invokes the GetData() operation interface of each LCS

[3]if the data is located in LCS the operation is finished and the LCS delivers data to the client (14).

[4]Before the execution of this capability we assume that the CCS previously has built the collective catalogue with LCS content reports.

selected and passes the *GetDataRequest* asking to retrieve specific data entities (7).

The transfer process is started between LCS locations by using the RFT service: A RFT resource is created (8) to operate the transfer. The RFT resource retrieves the credential for the cache service (9). The RFT starts the transfer which invokes GridFTP servers perform the data transfer in low level (10). The LCS source checks periodically the status of each data transfer request to ensure that each file was transferred successfully (11).

Once the transfer process is finished, each LCS updates its internal catalogue with the cache information related with the operation to make the information accessible to the grid (12). Finally, the LCS delivers data to the client (13).

LCS content updates The collective catalogue is maintained using regular LCS data contents reports. Each LCS periodically builds a data content report then it sends to CCS (14). Information in CCS catalogue must be periodically refreshed by subsequent reports. This catalogue also is used to support Copy Profileration Control capability in next section.

6.2.3 TSS Copy Proliferation Control

TSS data proliferation control capability enables the number of data copies that exist in the system to be established. The capacity exploits the data and cache activity information exposed by each LCS. The goal is to gather to be know information that permits the state and activity of all the different locations and for each one the state of the individual data entities managed by the cache system.

This operational information supports strategies and mechanisms for controlling storage resource utilisation. In this case the reduction of redundant copies based on the individual utilisation analysis. This procedure tries to make effective use of the available storage resources.

TSS implements a special coordinator for copy proliferation. This coordinator is deployed by grid resource administrator for simple and automatic control of copies in the system. The coordinator is regularly executed for this task.

The CCS maintains a collective catalogue with data content description of a subset of the LCSs like was described in Section 6.2.2. The coordinator queries the collective catalogue [5] to discover that data entities have several copies. It builds a list of the data entities duplicated in the LCS where they are stored. It then collects data activity information (see Section 4.5) for each copy detected. CCS applies an algorithm (implemented by the user) that classifies the copies in relationship with

[5]previously built and updated with LCS content reports as described for the Data Collector capability in section 6.2.2.

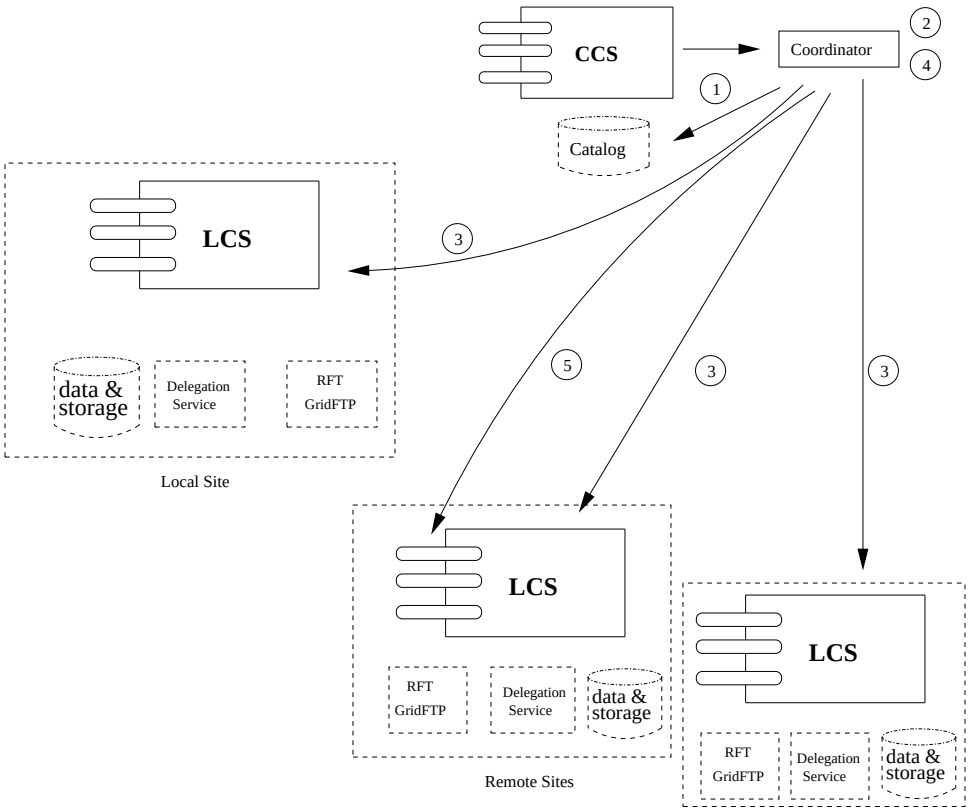

Figure 6.4: TSS Copy Proliferation Control

the individual activity of the data entity and the activity of the LCS. Finally, it selects the copies with smaller activity in the LCS to be eliminated from the system.

Figure 6.4 illustrates the implementation of the copy proliferation control capability using the globus environment. The CCS gathers data activity information from a subset of LCSs to analyse redundant copies. The CCS uses monitoring and configuration operations to support this function.

Copy Proliferation Execution

The CCS queries its collective catalogue to establish the data entities with multiple copies (1). The CCS builds a list of the data entities duplicated and where they are stored in the LCS and prepares a *GetDataActionsRequest* (see Appendix D.2.5) for each copy selected (2). Then, the CCS invokes the GetDataActions() monitoring operation of each LCS selected and passes the *GetDataActionsRequest* asking for actions realised by the the cache service and filtering the specified data entity (3). Next, the CCS executes the classification class (implemented for this function) to classify the copies in relationship with the individual activity of the data entity and the activity of the LCS(4). The CCS creates a *RemoveDataRequest* for each LCS where data entities will be removed. Then CCS invokes the RemoveData() operation

interface of each selected LCS and passes the *RemoveDataRequest* asking to remove the specified data entities (5).

6.2.4 TSS Monitoring

TSS monitoring capability permits the operational state of the group of caches of the TSS system to be established. This capacity exploits the cache activity information exposed by each LCS. The goal is to gather information that permits to know the state and behaviour of the different LCS instances deployed in the TSS system to be known.

This capability is used for basic performance and accounting monitoring of the TSS system. It provides information about the cache access operations supervised. This includes information related with the performance measurements of the system. One of the difficulties in this aspect is in the selection and use the appropriate indicators to measure the operation of the system. In this section, we use the defined monitoring operations (see Section 4.3.2) to get some indicators that are useful for observation and analysis of the TSS operation.

TSS monitoring capacity makes possible the periodic or continuous measurement of different data and cache actions. This provides an ongoing verification of the progress toward the achievement of the TSS system goals; it permits the degree to which the system responds to the requested access operations to be established and the levels of resources to make these operations. TSS supplies some indicators for establishing the effectiveness and efficiency of the access operations and aspects such as cache hit rate (processed requests) and current storage capacity.

The clients of the TSS monitoring capacity are typically grid resources administrators or users with special data management requirements. In contrast with other TSS capabilities, monitoring clients directly invoke the CCS; ihis implies that CCS supports operations addressed to an ensemble of LCS in the TSS.

The TSS implements a special coordinator module for monitoring; this coordinator is deployed by the grid resource administrator. The coordinator can be regularly executed to fulfill this task. The CCS supports the same monitoring operations described in Section 4.3.2 and the coordinator monitor implements the collective execution interacting with the LCS group. Thus the coordinator implementation translates one monitor operation received by CCS in multiple one-to-one operations with each LCS concerned.

Performance monitoring encompasses two main procedures: gathering information; which is the collection of statistics about cache system operation; and information analysis, which consists of processes for reducing and presenting the data acquired. In this work we focus on the gathering information procedure that provides basic

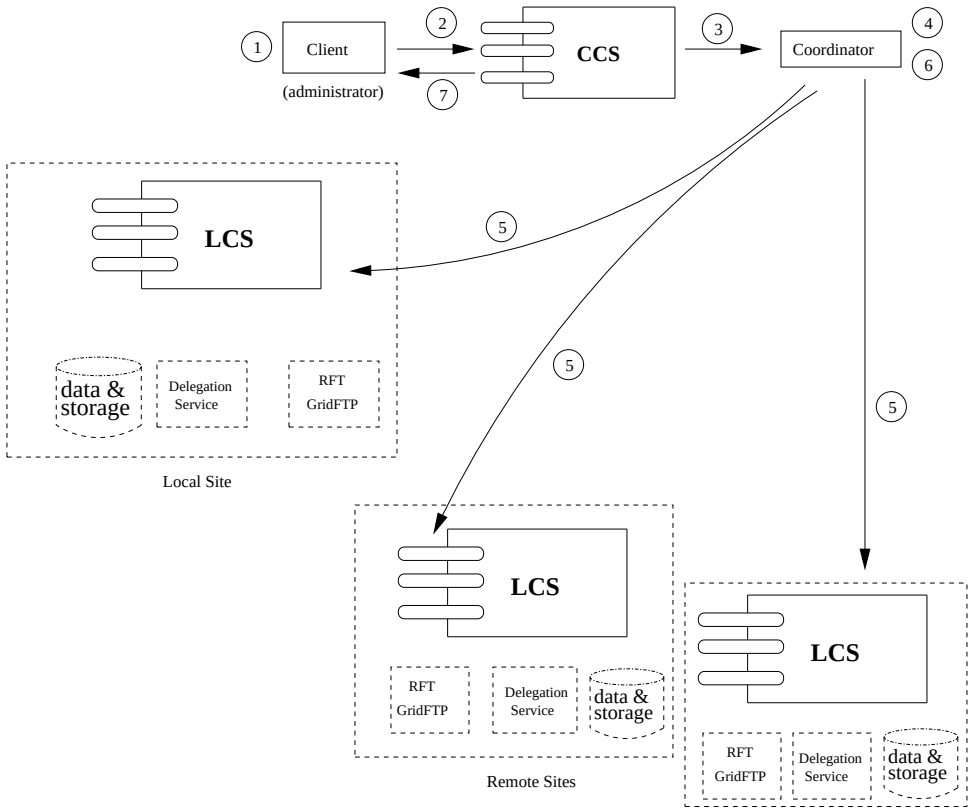

Figure 6.5: An example of gathering information for requests processed in TSS

elements to establish a cache performance measurement. The information analysis procedure is implemented by the user and is specific to the virtual community requirements.

Figure 6.5 illustrates the implementation of the gathering of monitoring information using the globus environment.

Monitoring Capability Execution

Preliminarily, the client (administrator) prepares the *getRequestProcessedRequest* (see Appendix D.2.5) (1), specifying the time period (start and finish datatime). The client (administrator) invokes the *GetRequestProcessed()* operation interface of the CCS (2) and passes the getRequestProcessedRequest to ask for a list of *Request* elements that describes the details of individual requests processed by each LCS in the TSS.

The CCS executes the coordinator that implements the collective monitoring operations (3). The coordinator builds a getRequestProcessedRequest for each LCS in the TSS (4). Next, the coordinator invokes the GetRequestProcessed() operation interface of each LCS in the TSS (5). The CCS builds a list with the set of *getRe-*

questProcessedReponse received from the LCSs (6). Finally, it returns the list as response to the client (7).

The similar procedure is executed for other monitor operations:

- GetTransfers()
- GetDataActions()
- GetDataReplacements()
- GetStorageCapacity()

6.2.5 TSS Configuration

TSS configuration capability permits modification of the parameters that cause actions to be taken by the LCS in the TSS that make up the system to be managed. This capacity exploits the configuration features supported by each LCS. The goal is to modify the basic operational parameters of the set or a subset of LCS in the TSS.

This capability is used for basic configuration control of the TSS system. It changes the functional parameters that alter the system behaviour. This includes parameters such as: replacement method, default time to live, resource storage, cache group, and cache coordinator. It uses the defined configuration operations (see Section 4.3.3) for changing values of the operational parameters.

The TSS configuration capacity makes it possible to adjust the dynamic operation of the TSS. The configuration capability permits changes in response to users or administrator requirements. The configuration capability allows to adjust of the operational behaviour established with the monitoring capability to be adjusted. This provides essential support for the achievement of TSS system goals.

The clients of the TSS configuration capacity are typically grid resource administrators or users with special data management requirements. In contrast with other TSS capabilities, the clients directly invoke the CCS; this implies that CCS supports operations addressed to a ensemble of LCS in the TSS.

TSS implements a special coordination module for configuration; this coordinator module is deployed by the grid resource administrator for configuration control of the system. The CCS supports the configuration operations described in Section 4.3.3 and the coordinator implements its collective execution interacting with the LCS group. The coordinator implementation thus translates one configuration operation received by CCS into multiple one-to-one operations for each LCS concerned.

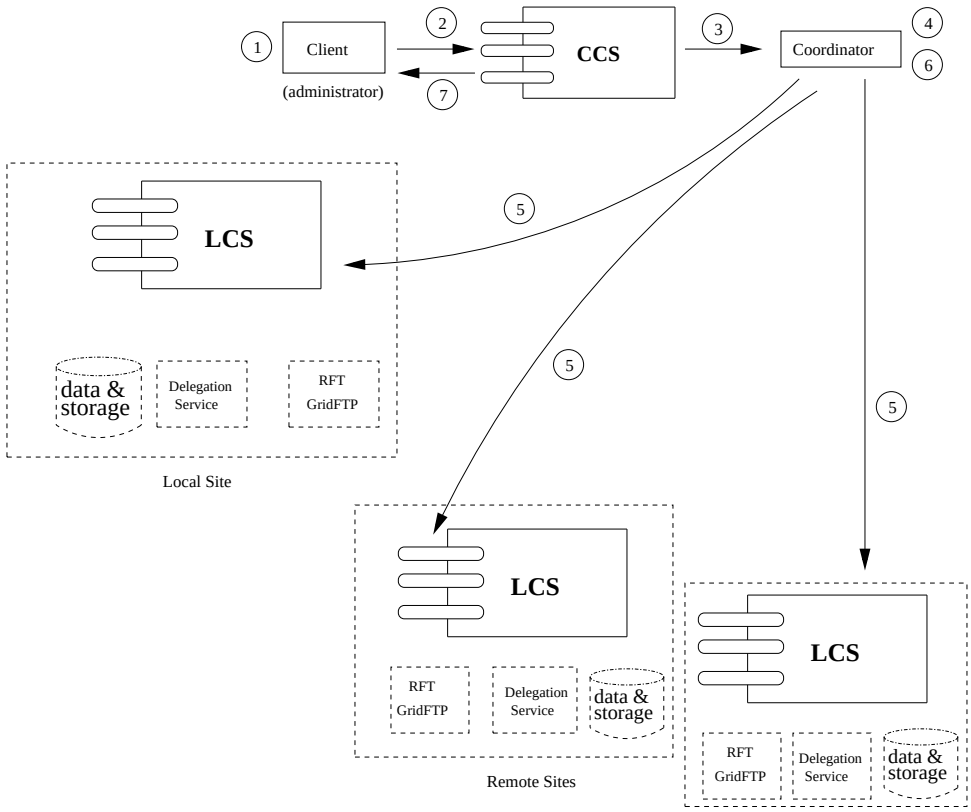

Figure 6.6: An example of modifying the replacement method in TSS

Figure 6.6 illustrates the gathering monitoring information capability implementation using the globus environment.

Preliminarily, the client (administrator) prepares the *setReplacementMethodRequest* (section D.3.2) (1) specifying the replacement method to be applied by the LCS in the system. The client (administrator) invokes the *SetReplacementMethod()* operation interface of the CCS (2) and passes the setReplacementMethodRequest to execute the specified replacement method for each LCS in the TSS.

Configuration Execution

The CCS executes the coordinator that implements the collective configuration operations (3). The coordinator builds a setReplacementMethodRequest for each LCS in the TSS (4). Next the coordinator invokes the SetReplacementMethod() operation interface of the each LCS in the TSS (5). The CCS builds a list with the *setReplacementMethodResponse* elements received from LCSs (6). Finally, it returns the list of confirmations to the client (7).

A similar procedure is executed for other configuration operations:

- SetDefaultTimeToLive()
- SetStorage()
- SetCacheGroup()
- SetCacheCoordinator()

6.3 Experiments

In this chapter we illustrate the collective management of a group of GCS providing temporal storage space (TSS). The objective is to extend the caching capabilities individually offered by the caches by implementing collective operations, especially the possibility for a cache to request another cache to store data (*on demand remote storage*).

This experiment illustrates the capacity of extensibility (see section 5.2.5) offered by the GCS.

This experiment has *not* been designed to evaluate cache replacement strategies or data redistribution heuristics. This type of issues has been intensively studied by the research community (e.g. cf. David Coquil's thesis in our team [35]) and is out of the scope of this thesis. Our objective with this experiment is to use the TSS system as proof of concept of the feasibility and effectiveness of the integration of generic GCS instances into a collaborative network of caches. That is, this experiment has been designed to show, on a basic example, that GCSs can be organised into a collaborative TSS system, that provided monitoring tools can be used to analyse the behaviour of this system (individual behaviour of a specific cache instance and global behaviour of the system as a whole) and that re-configuration decisions can be triggered at run-time to optimise the performance of the system.

6.3.1 Grid Simulation

We simulate a grid environment constituted by a group of ten GCSs. These GCSs work at the same level i.e., there is no hierarchy or category between them.

Parameters related with the cache size (space capacity), the data size and the number of files are assumed equal whatever the cache instance. This hypothesis is necessary to eliminate the influence of these parameters on the performance of each individual cache instance. Every GCS manages 100 gigabytes of storage; files have a fixed size of one gigabytes. The total data generated (published) represent 3 terabytes.

Request patterns for the data entities (files) are assumed to verify three locality properties (see section 6.3.3 below):

- Temporal locality: recently accessed files are likely to be accessed again.
- Geographical locality (client locality): data entities recently accessed by a client in a specific site are likely to be accessed by other clients of the same site.
- Spatial locality (data entity locality): data entities related to a recently accessed file are likely to be accessed [35].

Such properties are commonbly used in cache experiments. Indeed they are verified by many distributed applications. These properties are at the basis of all cache systems (caching randomly accessed data shows no effective benefit). In our context, they are even more true since data are put into the TSS on demand by the user himself.

6.3.2 Methodology of the Experiment

Simulator

The simulator consists of three modules. The cache manager module, the client module, and the coordinator module.

The cache manager is an internal module of the GCS prototype described in section 5.4.2. The data and cache activity information used by the manager are registered in memory using an arraylist object. Several instances of the cache manager are concurrently executed as threads in a single machine. The threaded version of the cache manager supports all the specified cache operations (see section 4.3).

The client builds and sends GetDataResquest messages (see appendix D.1.2) and receives and processes GetDataResponse messages from the cache manager. Similarly, the client builds and sends SetDataRequest messages (see appendix D.1.1) and invokes the SetData() method of the cache manager module. The proportion of SetData operations generated is 1/3 of the GetData operations. This ratio is based on the hypothesis that data stored in the cache system are likely to be accessed in the future[6].

[6]In real conditions, SetData and GetData operations are performed on request from the user (human user or program). The benefit of the TSS mainly depends on the access patterns. Storing into the cache a piece of data that will never be accessed in the future is a loss of resource. So, one can reasonably expect that users will store into the cache data that have a reasonable probability of future access.

Parameter	Initial value	After scaling
Number of files	3,000,000	3,000
Number of requests	9,000,000	9,000
Maximum number of files by cache	100,000	100

Table 6.1: Simulation scaling
Experiment parameters before and after scaling

Finally, the coordinator monitors and coordinates the execution of the simulation. It configures and initialises the group of GCSs using the SetCache configuration operation (see appendix D.3.1) e.g., it sets the storage capacity and the replacement method for each GCS thread instance. The coordinator also initialises the client and generates the data request distribution used by the client. Finally, the coordinator analyses the monitoring data to manage the global cache system and optimise its performance (see section 6.3.5 below).

6.3.3 Simulation execution

The simulated grid is composed of a coordinator and ten GCSs instances (cache managers) each logically connected to a client. All modules are executed as Java threads on a single machine.

The coordinator generates the access patterns based on events that follow uniform and Poisson random distributions, each event representing one data access operations (SetData() or GetData()) to be processed by one GCS instance (see section *Access Patterns* below).

When a GetData() access operation is invoked, the concerned GCS checks if the requested piece of data is stored in its cache and returns a response to the client. A SetData() invocation triggers an update of the GCS internal catalogue information. The GCS keeps a record of *all* the requests it receives (using *Request* header elements (see section 4.4.1)); it uses DataActions (see appendix C.1.2) to register successful hits of data (i.e. GetData requests that concern data stored in its cache) and data replacements. These monitoring data have a very high value as they are used by the coordinator to make its management and reconfiguration decisions.

The total amount of simulated data represent 3 terabytes. To enable the simulation, a scale of 1:1000 was used. Table 6.1 illustrates this scaling.

The reason we use this scaling factor is to make the simulation of such a system feasible on a single machine. The cache information elements for a million files would be very memory intensive. Since we need to scale the number of files, the

Parameter	Variability	Value
Total data space	invariable	3 terabytes
Cache size	invariable	100 gigabytes
Data size	invariable	1 gigabyte
Number of caches	invariable	10
Number of files	invariable	3000
Number of requests	invariable	9000
Replacement method	invariable	LFU
Topology	invariable	distributed
GetData operations	variable	generated with Poisson dist.
SetData operations	variable	generated with Poisson dist.
Cache selection	variable	generated with Poisson dist.

Table 6.2: Simulation parameters
Summary of simulation experiment parameters

GCS storage capacity must be scaled accordingly. Indeed the performance of a cache is directly dependent on the percentage of files that it can store. Scaling both the number of files and the cache storage capacity allows keeping a constant ratio between these two parameters[7].

Access Patterns

We use Poisson random distributions to generate data access requests with temporal, spatial and geographical preferences (section 6.3.1). These distributions are used in two simulation phases: to generate the data requests (temporal and spatial locality); to select the cache instance that will process each request (geographical locality).

Replacement method All GCSs in the experiment implement a LFU replacement method.

Parameters setup Table 6.2 summarises the different parameters used in the simulation.

6.3.4 Performance Evaluation

The coordinator thread gathers monitoring information about the state of every GCS instance using the monitoring API provided by the GCSs and described in section 4.3.2:

[7]For this reason, we do not modify the file size since this would have a direct effect on the number of files that can be stored in a cache instance.

- total number of processed requests (get and retrieve)
- number of replacements that have been done
- number of get requests that have been successfully processed, i.e. that concerned pieces of data stored in the cache of the GCS (*successful hits*)

The two first data allow estimating the workload of every cache instance; the latter is a measurement of the performance of the GCS.

These monitoring data are used by the coordinator to analyse the activity of the system, to compare the workloads of the various GCS instances and to take reconfiguration decisions.

6.3.5 Grid Simulation Scenarios

As stated above, this series of simulations aims to show how one can implement a collaborative management of a group of GCSs instances forming a TSS system. More precisely they aim to illustrate how one can monitor cache instances activity, implement reconfiguration decisions and change dynamically cache operational parameters.

We implemented and evaluated three basic scenarios related to the GCS capacity to store remote data[8] (also called *shared storage space*).

Scenario 1: Caches do not store remote data (no shared storage)

The base case against which we compare the other scenarios is when caches do not cooperate, i.e do not use their storage capacity to store remote data. The entire storage capacity is then used for "local" cache operations.

Scenario 2 : Caches reserve a uniform storage space for remote data storage

Each cache reserves 10% of its storage capacity for remote data. As the system is composed of ten GCSs, the total capacity of the shared space is therefore equivalent to the capacity of one cache.

[8]Remote data storage = storage by a GCS instance of a piece of data transmitted by another GCS instance

Scenario 3 : Caches provide shared storage according to their activity

Each cache shares a percentage of its storage capacity according to its activity within the group. In the following experiments, we implemented a strategy such that only the five caches with the lowest activity participate to the "shared space" : the less active GCS reserves 30% of its capacity for remote data, the second less active GCS, 25%; the third, 20%; the fourth 15%; the fifth, 10%. This basic heuristics considers that very active caches (i.e. caches that receive many requests and that make many replacements) should be preserved from additional load ; oppositely, inactive caches can provide a part of their capacity to store data that cannot be stored in overloaded caches.

6.3.6 Experimental results

For each of the three scenarios, simulations were run 10 times. Each simulation was run for 9000 requests of 3000 possible data entities. All data displayed below are average data on 10 simulations.

Table 6.3 shows the results of the execution of scenario 1 (no shared storage). Data are provided for each cache instance.

Rows in table are:

Replacements : percentage of data evicted by the replacement method

Requests : ratio (percentage) between the number of requests (get + retrieve) received by the cache and the total number of requests processed by the system

Individual HR : percentage of the requests that produced a cache hit (individual hit rate)

Participation : participation of the cache to the total hit rate, i.e. ratio between the number of local hits and the total number of hits (percentage)

Several points should be noticed about these results :

- the activity of the GCS instances is very variable. This is an expected consequence of the Poisson distributions that are used (locality patterns)
- some GCS instances (e.g. 1, 2) show a very low activity both in terms of number of processed requests and nb of replacements
- 4 GCSs (4, 5, 6, 7) show a high replacement rate, i.e. they spend much time for managing their storage space

Cache	1	2	3	4	5	6	7	8	9	10
Replacements(%)	0.1	3.7	10.1	14.7	17.6	16.6	15.1	11.6	7.0	3.2
Requests(%)	0.7	3.4	8.8	14.4	18.1	18.2	15.0	10.6	6.8	3.6
Individual HR (%)	12.3	34.7	58.2	67.9	70.3	70.6	68.5	62.7	52.5	37.1
Participation(%)[1]	0.1	1.8	8.0	15.3	20.0	20.2	16.2	10.4	5.5	2.1

Table 6.3: Experimental results by cache in base scenario
Average data for 9000 requests executed 10 times

A clear consequence of these remarks is that some load balancing procedure should be implemented in order to optimize the global performance of the system. A system in which 40% of the participating entities assume 70% of the work is definitely not well balanced.

This argues in favour of the implementation of a global coordination of the cache instances. This coordination suppose that :

- the system is able to monitor all the participating cache instances: cf. table 6.3
- the system is able to change the behaviour of the participating cache instances
- the system implements some load balancing and performance improvement heuristics.

This third issue is definitely out of the scope of this thesis. It has been addressed in our team from two different points of view :

- from a semantic and collaborative point of view (see David Coquil's PhD [35]): data usually show semantic correlations that replacement heuristics and cache collaboration protocols can use to optimize the hit rate;
- from the grid infrastructure point of view (see Julien Gossa's PhD [66]): GCS instances should be placed and cached data should be placed/duplicated/migrated according to the actual operational conditions (e.g., network bandwidth and latency, CPU charge...).

We refer interested readers to these theses and to the extensive bibliography they propose. In our simulations, as explained before, we implemented two basic heuristics, the so-called scenarios 2 and 3.

Implementing cooperation and optimization heuristics supposes, as noted above, that the system is able to monitor and modify the behaviour of the participating

cache instances. For instance, scenarios 2 and 3 require that the coordinator can demand that GCSs reserve storage space for remote data.

We therefore run two additional simulations to illustrate these two scenarios. In these simulations, the coordinator invokes the SetStorage() configuration operation (see appendix D.3.4) to change the storage capacity according to scenario 2 (reduction of the capacity of all the instances by 10%) and scenario 3 (reduction of the capacity of the five less active instances). As noted before, both scenarios reduce the global storage capacity by 10%. This 10% reserved space is "frozen" and not used for storing data. We then measured the cost in terms of hit rate and number of replacements of this storage preemption.

Table 6.4 shows the performance results (average data + standard deviation) measured by these simulations.

Several points can be noticed :

- as expected, reducing the storage capacity of some cache instances reduce the global hit rate and increases the number of replacements
- this overcost is much higher in scenario 2 than in scenario 3 : this is also an expected result : adapting the reduction wrt the load of the GCS is more effective than implementing a uniform reduction
- in scenario 3, the overcost is entirely undertaken by the five less active caches
- in terms of hit rate, this overcost is 3,9% for scenario 2 and 0,4% for scenario 3

In other words, these results show that one can preempt 10% for remote data storage for a cost of 0,4%. For a grid site and GCS instance administrator, such a cost is definitely affordable.

These preempted 10% of the total storage space are available for the system coordinator to optimise the performances of the system.

As noted above, defining optimization heuristics is not the subject of this thesis. We have mentioned research works (within or out of our team) that specifically address this issue. From these works, some basic recommendations can be made :

- optimization strategies should be based on the monitoring and identification of access patterns : when a data is requested from very distant site, a duplication can be very efficient [67]; when data are correlated, prefetching techniques can highly improve the hit rate[35]

Scenario	No shared storage		Uniform storage preemption		Adapted storage preemption	
Value	average	std dev	average	std dev	average	std dev
Replacements	612 .8	29.6	771.5	33.45	682	23.05
Successful hits	5746.9	48.13	5523.3	92.84	5725	89.79

Table 6.4: Preemption of storage space
Performance for 9000 requests executed 10 times

- the focus must be put on overloaded cache instances ; the preempted storage space must be used to reduce the load of the most active caches by redistributing some cached data (using data remote storage, data migration or data duplication)[9]
- grids are very dynamic and heterogeneous platforms : data redistributions must be done and GCS reconfigurations must be decided with respect to the actual operational conditions (network bandwidth and latency, CPU charge...) [67][10]

6.3.7 Summary

In summary, these simulations show that :

- a set of GCS instances can be organised to build an operational TSS system
- the TSS coordinator has the ability to constantly monitor GCS instances using provided monitoring methods
- the coordinator has the ability to modify the operational parameters of the GCS instances
- specifically, the coordinator can preempt storage space on the less active GCSs for remote storage
- under classical data access patterns, the cost for the GCSs concerned by such a preemption is, on average, less than 1%.

[9]Let us consider the following example : assume that 80% of the requests are issued from 4 sites and concern 120 Gbytes of data; assume that the GCS attached to these sites can store only 100 GB of data. Then duplicating the most requested data and remotely storing the 20 GB of data that cannot be stored in these 4 caches can drastically improve the performance of the system.

[10]If the inactive caches used for redistributing data are connected by very bad network links, then the benefit of the redistribution can be very low... or even negative !

6.3.8 Conclusion

The series of simulations presented in this section illustrates how a collaborative system composed of a set of GCSs can provide temporal on-demand data storage. These simulations are motivated by the *Data Dissemination* case introduced in section 2.1.1 and the *TSS Data Dissemination* capability presented early in this chapter (see section 6.2.1).

This series of simulations allows validating the operationality and effectiveness of the monitoring APIs provided by GCS. It also exhibits how collaborative operations can be implemented by invoking reconfiguration facilities.

In particular, these simulations show how one can preempt storage space on the less active GCSs. This "preempted storage space" can be used to optimize and balance the global behaviour of the system by remotely storing data, duplicating the most requested data, migrating data to less active cache instances.

This optimization should be based on the monitoring of data access patterns and grid operational conditions. Experiments are under way to evaluate the benefit, in terms of hit rate, of different optimization heuristics under various conditions of access patterns, network traffic, CPU usage.

6.4 Discussion

In this chapter we presented the design of the Temporal Storage Space (TSS) which is a data management system that operates temporary data in automatic form. We also showed an experience which simulates the TSS capability to provide storage space on-demand.

Not Centralised Coordination

The conception of the TSS is based on the coordination of operations supported by the Local Cache Services (LCS). An external service, the Collective Cache Service (CCS), implements the coordination process between cache services. The decision of executing the coordination in external service permits separate of the implementation of the different layers into specialised components. The LCS components provide the base operational support and the CCS guides the intercache operations between LCSs.

The CCS is proposed as a centralised service, this has as a main advantage the easy implementation in grids that requires a reduced number of LCSs. It does not, however, appropriate to scenarios that require the deployment of a significant number of caches. In this context, other implementation alternatives to be explored for

schemes based on distributed coordination like to peer to peer systems, representative not centralised collaborative cache systems are described in Appendix B.1. The design of not centralised coordinated systems can take advantage of the basic access, monitor and configuration capabilities provided by Grid Cache Services proposed in this thesis.

The flexibility to implement different coordination systems to collaborative caches in grids permits to respond to diverse grid data access patterns and computing models.

Cache as temporary storage provider

The TSS data dissemination capability suggests the idea to use a collaborative cache system as an on-demand storage space provider. This implies a different utilisation of the cache mechanism. Since the cache content conventionally is determined by the data consumer demand, this controversial aspect is expressed often as main difference with replication (see Section 3.6). For this reason the cache metric performance like hit rate is limited to adequate measurement of cache performance used as storage space provider.

The simulation described in Section 6.3 attempts to provide insights on these aspects; it studies a scenario where the collaborative cache execute particular functions. We have calculated the cost, in terms of the sum of the hit rates of the local caches, produced by the implementation of a logical reassignment policy of storage space; we have obtained a cost that we consider very low; the assumption about the invariability of key parameters such as data and cache size, and access patterns distributions limits the representability of the experience.

Cache Resolution

The TSS data collector capability uses the cache resolution mechanism based on cache communication protocols: the coordinator maintains a global catalogue, or directory, to map a data entity identifier associated with whose LCS hold the data. This approach suggests an easy implementation of data resolution mechanisms based on a summary or digest of the cache contents. The *GetDataContent* is a generic access operation (described in Appendix D.1.4) that provides this information. LCS inter-exchange messages indicate their content and keep directories to facilitate finding documents in other caches (see Appendix B.2.2).

This centralised mechanism, however, is not appropriate to support the scalability and deployment concerning a large number of LCSs scattered over the grid. Other mechanisms based on resolution protocols or hashing-based method must be explored to adapt to specific application requirements.

Catalogue consistency

The CCS maintains a weak consistency catalogue with cache information reports

from LCSs. We follow the same assumption used by RLS (see Section 3.6.1); the service needs not to provide a completely consistent view of all available data copies if the query returns to the client a subset of all extant copies, or references that include "false positives" the client may execute less efficiently but it will not execute incorrectly.

TTS capabilities implementation

The algorithm for selection of copies can include a wide range of parameters to take the decision, this includes information about the individual data entity and the individual cache installation. Complex relations can be established in the local and collective context.

The coordinator can build an additional scheme to support global and scalable systems that include a significant number of LCSs and data information. A variety of schemes can be built with different performance features; this includes the organisation of the GCSs in distributed or hierarchical form and type of data redundancy between them.

The monitoring capability discussed in this chapter assumes the worse case situation which detailed information is available. This generates a significative quantity of verbose information that can have a high cost; the implementation of monitoring capabilities must propose alternatives to provide this information, for example, computing values locally from registered activity information.

We now discuss some aspects related with cooperative caching introduced in Section 2.2.2

Interaction

The TSS design is an example of a collaborative cache system composed by LCS interactions. The cache interactions determine the capabilities supported by the collaborative cache system. In the TSS system discussed in this chapter, particular capabilities are implemented to operate and control temporary data between grid locations. The set of caches operations defined in Section 4.3 and Appendix D are used to support the TSS data access, monitor, and configuration capabilities. These operations can be used to support a flexible composition of specific capabilities in grid caching systems.

Organisation

TSS presents the arrangements of LCS elements to operate and control temporary data in a common way. The organisation is strongly related with the required capabilities, functions and performance to be supported by the system. TSS is designed for scenarios concerning a small number of LCSs deployed over the grid. The support of scenarios with a large number of storage locations over the grid does not require

centralised coordination components to adequate scalability and performance. There are diverse alternatives to organise the distributed system components. For example, using hierarchical or distributed schemes depend on requirements and application objectives. GCS can be used as base components to support diverse required capabilities.

Distributed data placement

The TSS design and simulation experience suggests simple strategies to put temporary data in different grid locations based on the particular capabilities for sharing available storage resources between GCS. It is necessary, however, to implement decision processes that support specific and particular data placement requirements. The GCSs provide detailed and accurate information required to support coordination mechanisms.

Load balancing

The experience described in this chapter uses data and cache activity information provided by GCSs to dynamically establish the workload on distributed caches. This information is used to support a strategy to distribute data between a group of GCSs. The strategy deals to minimise the collective performance degradation of the group of caches that compose the system. However, it is indispensable to start research works to obtain accurate knowledge about the utilisation of collaborative caches in order to provide storage capacity on demand

Chapter 7

Conclusions and Future Work

In this thesis we have proposed a specification and implementation of collaborative cache services for grid environments. This work is focused on grid applications manipulating large volumes of data.

The processing of large amounts of data implies data movements that require consumption of network bandwidth and storage space; a significant quantity of the data is typcally used for a limited period of time. Without global coordination, resources requested to manipulate and exchange these temporary data are used in an inefficient and non cost-effective way. This thesis proposes and analyses an approach to managing temporary data in grids.

Caching is recognised as one of the most effective techniques for managing temporary data. Collaborative cache systems, like web caching, have been proposed to scale cache capabilities for distributed systems. In this sense, collaborative cache systems appear as a natural solution for managing temporary data in a grid. However, existing solutions are not directly applicable, due to the specifics of grid environments.

Based on two use-cases, we have presented a list of requirements and constraints that are related to the features of cache systems, the nature of grid environments and the specifics of grid applications.

7.1 Contributions

First Contribution

Our first contribution, the specification of a reference multilayer model for grid caching, defines and organises the main functions needed to operate and control a collaborative cache system. The different layers separate the conceptual functions

of the system.

Four functional layers have been identified and defined: storage, control, collaboration and coordination. The storage layer represents storage resources used to implement the cache system. The control layer regulates and registers the actions in the storage layer. The collaboration layer defines and supports the interactions between caches. The coordination layer supports the organisation and optimisation of the interactions between caches.

The specification defines a set of standard cache operations required to support the model. They are classified into three main types: access, monitoring and configuration. Access operations as used to access data stored in the cache. Monitoring operations are used to exchange information about the cache elements and their activity. Configuration operations are used to dynamically parametrised and manage cache instances. The specification also defines the entity information and activity information elements manipulated by cache instances to represent and exchange information about their activity and about the cached data.

Finally, the specification defines a generic infrastructure for building collaborative cache systems in grid environments. Multiple forms of organisation cache instances accommodate the high variability of grid infrastructures and grid applications.

Second Contribution

Our second contribution; the design and implementation of the Grid Cache Service (GCS) i.e., the base component used to design and build grid caching infrastructures.

In order to satisfy the grid caching requirements, we have elaborated a set of design principles to implement the GCS; these principles establish the main features of the implementation: virtualization, autonomy, accessibility, uniformity, and extensibility. Virtualization provides an abstract view of the behaviour of the data in a cache mechanism. Autonomy defines the cache as a distinct and independent component with specific functions. Accessibility establishes that cache content, resources and management information are accessible and shareable for a wide variety of clients. Uniformity establishes that cache capabilities are provided in a similar way in the whole distributed grid cache system. Extensibility establishes that collaboration between caches is extensible to all possible cache operations including access, monitoring and configuration.

We have implemented a prototype of the GCS which supports and implements all the functions specified by our grid cache model. This prototype has been developed in Java programming language and designed to be executed with the Globus middleware. Performance tests of the GCS prototype show that the time required to execute basic cache monitoring and configuration operations is marginal in comparison to data transfers.

Third Contribution

Our third contribution, the design of the Temporal Storage Space (TSS), is a proof of concept of a data management system that operates temporary data by managing a set of distributed GCSs. The TSS design provides insights to the implementation of the coordination layer defined in our reference grid cache model. Basically the TSS system is composed of a group of GCSs that collaborate under the supervision of a coordinator.

In the TSS system design, the GCS is the essential collaborative software brick used to implement high-level coordination strategies to manage collective cache storage resources in an efficient and cost-effective way.

We have described a series of experiments which simulates the TSS capability to preempt and provide storage space on-demand. These simulations illustrate the way GCS instances can be organised into a collaborative cache system in order to provide temporal on-demand data storage. It shows how it is possible to preempt storage space on the less active GCSs. This preempted storage space can be used to optimise the global behaviour of the TSS system and balance the GCS instances workload.

We present how these contributions satisfy the established initial requirements for grid caching.

Local Operation Requirements

- *Delegation of temporary data operation*

  The specification defines an infrastructure based on specialised cache entities that manage temporary data that will be kept in storage resources: these specialised cache entities are implemented by the GCS. Furthermore, a GCS can request and provide cache operations to other GCSs. The TSS illustrates the collaboration between several GCSs to manage temporary data in a distributed way.

- *Accessibility by different types of clients*

  The specification defines a set of access operations to be supported by a cache instance in a grid. GCS implements grid service interfaces to these accessible operations over a wide range of grid applications, services, and users. TSS is composed by a group of GCSs which provide accessibility in multiple locations.

- *The uniformity of operations and interfaces*

  The specification defines standard access, monitoring and configuration information, plus operations, for grid caching. GCS implements the grid interfaces

to these operations and associated behaviour. In the TSS, these capabilities can be requested, interpreted and processed in the same manner at different grid locations.

- *The capacity to gather resources on demand*

  The proposed model in the specification determines that a grid caching system is built from storage resources: these are handled by the control layer. Similarly, they are shared by a collaboration layer. GCS is implemented as a local administrator of temporary storage and data. TSS is used as proof of concept of a collaborative cache system composed by GCSs to preempt storage space. However we have only demonstrated its feasibility using simulation.

Collective Operation Requirements

- *Enabling the control of storage resource*

  The model specified establishes that each location or organisation allows its partners to use its resources in an automatic way. GCS supports the remote configuration of operational parameters. For instance, in TSS, the coordinator of the system is able to control the data copies proliferation and distribution on behalf of the virtual organization collaboration policy.

- *Coordination for effective operation*

  The coordination layer defined by the grid cache specifications must implement coordination mechanims to support and optimise collective capabilities. GCS supports the remote configuration of its essential operational parameters. TSS experiments illustrate the implementation of basic coordination mechanisms between GCS. However, optimising the resource usage and the system efficiency requires to develop advanced cache interaction coordination heuristics.

- *Accounting and monitoring activity*

  The specification defines that cache components must support monitoring operations to trace their activity. A fundamental feature of the GCS is the capacity of monitoring its operation in real time. This feature is related with the capacity of supervising the detailed processed data and the cache activity. Monitoring makes possible measurements and statistics necessary to evaluate the global state of the system. TSS, for instance, uses these monitoring capabilities to establish the workload of the GCS instances partcipating to the system.

- *Flexibility to choose schemes and strategies*
  Grid caching specifications establish that different collaborative cache architectures must be supported. GCSs are generic software components that provide communication, monitoring, collaboration and configuration facilities. They can be arranged into any organisational form to cope with diverse grid architectures and support diverse application contexts. TSS presents an example of configuration based on a distributed set of GCSs and a centralised coordinator.

7.2 Future Work

Development of coordination heuristics

In this thesis a base infrastructure for grid caching infrastructure has been proposed in which GCSs are the essential bricks used to build high level data management systems. Experiments have shown the feasibility of the proposed approach. However, optimising such systems requires some effective GCS coordination and distributed cache management heuristics. Such heuristics should be adapted to the features of the underlying grid infrastructures and applications.

Scalability and integration tests

To gain additional experience about grid caching capabilities, we plan to run extensive scalability experiments in multiple grid environments. We also plan to integrate our data caching facilities into existing applications in order to evaluate the effectiveness of our system in real operational conditions.

TSS development

The capacity to gather storage on demand for temporary data is a relevant requirement considering the dynamic character of resources in grid. Simulations of the proposed Temporal Storage Space (TSS) system have demonstrated its feasibility. The development of an effective prototype of TSS will require to integrate our system with grid middleware modules, especially the grid information system (metadata catalog), the data replication system, the message passing libraries and the authentication/authorisation system.

Fault Tolerance

Grids are characterised by their high dynamicity and unpredictability. As a consequence, grid infrastructures are subject to strong operational variations (e.g. bandwidth variations) and even faults. Fault tolerance and recovery mechanisms should therefore be studied and integrated within coordination heuristics.

Inter virtual organisation collaboration

The proposed cache infrastructure has been designed to be implemented within a virtual organisation. This limits the capacity of this infrastructure to cope with peaks of data requests. A potential solution is to allows different cache infrastructures attached to different virtual organisations collaborate e.g., exchange or lend storage on demand. This requires to develop negotiation protocols and accounting procedures.

Appendix A

Cache Replacement Methods

A.1 Replacement methods

A key aspect of the effectiveness of cache systems is the data content replacement algorithm which can improve the cache performance. The cache replacement algorithm guides the process will realize when the cache becomes full and old data objects must be removed to make space for new ones. Usually, a cache assigns some kind of value to each data object and removes the least valuable ones. The meaning of valuable varies according to different strategies. Typically, the value of the data object is related to the probability that it will be requested again, thus it attempts maximize the hit ratio. Research communities have proposed numerous replacement algorithms, we present some representative works.

A.1.1 Classical algorithms

There three traditional replacement algorithms:

Least Recently Used (LRU) This strategy removes the least recently referenced data object. It is widely used in different areas such as database buffer management, memory paging and disk buffers.

Least Frequently used (LFU) Removes the least frequently requested object.

SIZE [2] This strategy removes the largest size data object. The LRU strategy is often applied to objects with the same size.

The replacement strategies can be classified in five categories proposed in [95],

- recency-based strategies
- frequency-based strategies
- recency and frequency-based strategies
- function-based strategies
- randomized strategies

We present some representative methods of each category in the following sections:

A.1.2 Recency-Based Strategies

These strategies use recency (time of the last reference to the object) as a principal parameter. Most of them are extensions of the LRU algorithm. LRU is based on the locality of reference established in request flows. Locality of reference characterizes the ability to predict future accesses to data objects from past accesses. Temporal locality refers to repeated accesses to the same data object within small time periods. It implies that recently accessed data objects have a high probability to be accessed again in the future. Another type of locality is spatial locality which refers to access patterns where accesses to some data objects imply accesses to other objects physically near. Recency-based strategies exploit the temporal locality seen in request flows. It implies that references to some objects can predict future references to other objects. The replacement strategies can be classified in five categories proposed

LRU-Threshold [1] . A data object is not cached when its size exceeds a given threshold. Otherwise this strategy works like LRU.

LRU-Min [1]. It is a variant of LRU with predisposition in favor of smaller objects. If there are any objects in the cache which have size being at least S, LRU-MIN evicts the least recently used object from the cache.

History LRU [121]. HLRU proposes a strategy based on history of past references is associated to each cached object. A function hist(x, h) defines the time of the past h-th reference to a specific cached object x. HLRU algorithm will replace the cached objects with the maximum hist value.

Recency-based strategies consider temporal locality as the main factor. This can be favourable for scenarios that present data access with temporal locality like, for example, web requests streams. The utilisation of these methods is rather adaptive according to workload changes. Additionally they are simple to implement and are fast.

A.1.3 Frequency-Based Strategies

These strategies use frequency (number of requests to an object) as a principal factor. Frequency-based strategies are in a certain sense extensions of the LFU algorithm. They are based on the fact that different data objects have different activity access and that this activity result in different frequency request. Frequency-based strategies track this activity values and use them for future decisions.

LFU-Aging [9]. With LFU, objects that were very popular during one time period can remain in the cache even when they are not requested for a long time period. LFU-Aging uses a threshold. If the average value of all frequency counters exceeds this threshold all frequency counters are divided by 2.

Alfa Aging [135]. This is an explicit aging method with a periodic aging function. At a periodic time (e.g., 1 hr) the value of every object is decreased to alfa times its original value. Each hit causes the function to be increased by one.

swLFU (Server-Weighted LFU) [77]. This strategy uses a weighted frequency counter. The weight of an object indicates how much the server of data object appreciates caching of that object. The server can influence caching of the object.

Frequency-based strategies are valuable in static environments where the popularity of objects does not change very much over a specific time period such day or week. In contrast, LFU-based strategies require a more complex cache management. LFU can be implemented, for example with a priority list.

A.1.4 Recency/Frequency-Based Strategies

These strategies use recency and frequency, and often additional, factors to select a data object for replacement:

SLRU (Segmented LRU) [10]. The SLRU techique partitions the cache into two segments: an unprotected segment and a protected segment. The protected segment keeps the popular objects. First time admitted objects in the cache are placed in the unprotected segment, then, if the data object is requested afterwards (cache hit) the object is moved to the protected segment. Both segments are controled with the LRU method, but only data objects from the unprotected segment are removed. Data from the protected segment are transfered back to the unprotected segment as the most recently used object. This strategy requires to establish the parameter that determines what percentage of the cache space is assigned to each segment.

HYPER-G [2]. This strategy combines LRU, LFU, and SIZE. At first the least frequently used object is chosen. If there is more than one object that meets this criterion the cache chooses the least recently used among them. If this still does not give a unique object to replace the largest object is chosen.

A.1.5 Function-Based Strategies

These strategies evaluate a specific function incorporating different factors. They mostly use weighting parameters for different factors. The function calculates the activity or value of a data object. The strategy chooses the data object with the smallest activity or value.

Greedy Dual Size [19]. GD-Size associates a cost with each object and evicts object with the lowest cost/size.

Bolot and Hoschka strategy [15] employs a weighted rational function of the transfer time cost, size, and time of last access.

If designed properly, these strategies can avoid the problems of recency and frequency based strategies but due to the special procedures, most of these strategies introduce additional complexity.

Function-Based Strategies permit a proper choice of weighting parameters, thus is possible to try to optimize the performance metric. They consider a number of factors for handling different workload situations. Choosing appropriate weights is a difficult task. Some proposals assume that the weights are derived from trace studies. This is a simple but error susceptible approach. Server workloads change over time and require some adaptive setting of the parameters.

A.1.6 Randomized Strategies

These strategies use random mechanisms to select a data object for replacement. Randomized strategies constitute a different nondeterministic approach to cache replacement.

RAND . This strategy removes a random object.

HARMONIC [71]. Whereas RAND uses equal probability for each object, it removes from cache one item at random with a probability inversely proportional to its specific cost and size of the data object. The replacement strategies can be classified in five categories proposed

Randomized strategies try to reduce the complexity of the replacement process without sacrificing too much quality . Randomized strategies do not need special data structures for inserting and deleting objects and they are simple to implement. In contrast, they are inconvenient to evaluate: they can provide different results in situations with the same conditions.

Other work explores alternatives for the above categories. Thus, (CODOR) *Coldest Document Replacement* is a semantic approach for cache replacement method proposed in our research team [37] [35]. CODOR is a replacement method based on semantic relevance topics of data content. It gives a numerical value to data objects (documents) that reflects its current interest among a community. This value called 'temperature' is calculated based a previous semantic indexing process of the document and its dynamic access in the cache. A collaborative proxy-cache architecture is proposed based on exchanging information about the temperature variation between caches [36] [35].

Appendix B

Cooperative Web Caching

An important aspect of cooperative caching design is how the caches are organized at different locations of the distributed system. This aspect is essential to establish the possibilities of interactions and the relationships between the group of caches. A review of the literature suggests four main types of schemes or architectures: hierarchical, distributed, organizational based on multicast and peer to peer approach. A concise overview is presented in this section.

B.1 Cooperative Cache Architectures

B.1.1 Hierarchical Cache

The hierarchical cache establishes several levels of relationship between caches. This organization creates a tree-like structure to allow these caches to work between levels and support cooperative interactions. The harvest project [17] [16] introduced the notion of hierarchical cache in web. Initially, the system was implemented caching data objects in HTTP, FTP, WAIS, Gopher and DNS maps. But later the main interest was centered in HTTP traffic. The Harvest cache hierarchy was designed to resolve data queries between several caches. In this context, a cache can resolve data misses through other caches. The cooperative cache resolution mechanism distinguishes between parent and sibling caches.

The parent cache is upper in the hierarchy, a sibling is one in the same level in the hierarchy. When a cache does not have a data object it invokes a remote procedure to all siblings and parents. The request is resolved through the first cache that return the response. The resolution algorithm select through the original server or cache that provides the data most efficiently [28]. Usually, in hierarchical designs, child caches can query parent caches and children can query each other but parents never

query children. When a request is not satisfied by the specific cache the request is redirected recursively to the parent cache. Finally, when the document is found, either at a cache or at the original server, it travels down the hierarchy leaving a copy at each of the intermediate caches along its path. Further requests for the same document travel up the caching hierarchy until the document is hit at some cache level. This creates a cache organi zation where information gradually filters down to the leaves of the hierarchy.

In the [17] authors indicate that measurement of this approach ran approximately twice as fast as the CERN cache (that is not hierarchical), and the distribution of data object retrieval times has a much shorter tail than the cache deployed in CERN. However they do not provide detailed descriptions of the realized experiences.

The hierarchical approach was well accepted and implemented. Squid [111] is an open distribution of a proxy cache with the continuation of the Harvest goals. Squid implementation incorporates several improvements such as heuristic search of data objects among cache neighbors based on weighs. Each cache gives a weight to other caches which permits to select the cache to send queries.

A problem of the hierarchical cache is the placement of data copies through groups of caches. In the initial approach a copy of the data object is stored in all intermediate caches of each level of the path toward the initial request. This characteristic is not always optimal: in [79] an algorithm is proposed to makes copies on some caches in the same level. The proposed algorithm selects the caches that keep the copies, however, it requires detailed information about realtime operational conditions of sling caches and in practice that information is not frequently available. Other work [81] proposes meta algorithms for hierarchical caches: these meta algorithms are charged for selecting a global placement of data objects in the group of caches in the hierarchy.

There are several problems associated with a hierarchy topology [69] [71]: the caches must be placed at strategic points in the network; this often requires significant coordination between caches. Parent or root caches may become bottlenecks and create long queuing delays. Different levels of the hierarchy may add delays. Storing multiple copies of the same data objects create data content proliferation and often unappropriate use of storage resources.

B.1.2 Multicast Approach

Adaptive Web caching considers the problem of global data dissemination [136] [89]. It deals in particular with the problem created by the "hot spot" phenomenon where specific content can, suddenly, become massively popular and high in demand. Adaptive web caching proposes an approach based on multicast to resolve the increasing scale of data dissemination. In that context, when multiple users are

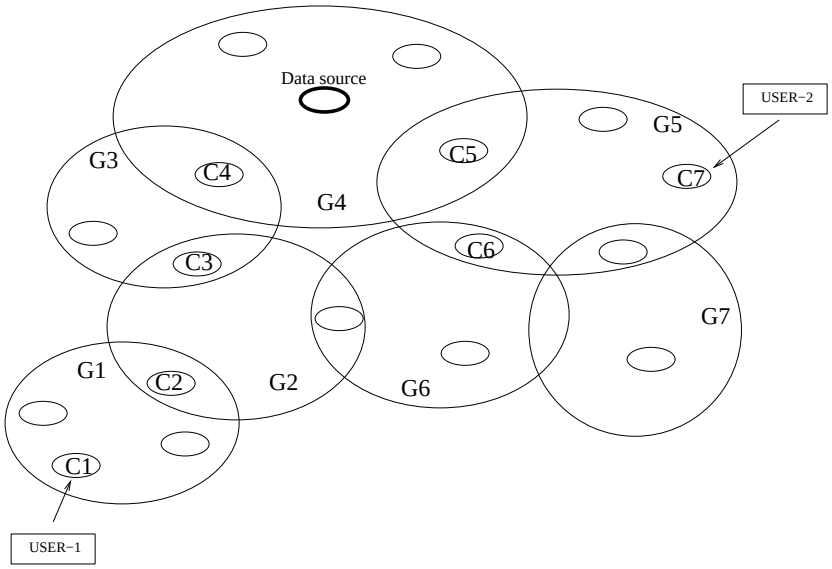

Figure B.1: An example of multicast caching

interested in the same data, a copy of data can be obtained from the original server and then forwarded using muticast to all interested users.

Data content requests on the Internet, however, is asynchronous because different users request web content at different times. Therefore, its proposed to use caches to multicast data dissemination. Adaptive caching consists of multiple distributed cache groups called meshes. These meshes are organized based on content, demand, and also include web servers. In this architecture web servers and caches are organized into multiple, overlapping multicast groups, like shown in the figure B.1.

When a user requests a data object it sends the request to a nearby cache. If it does not find the requested data object in its local cache it multicasts the request to a nearby local group of which it is a member; if some cache in rhe group has the requested data it multicasts the requested data object and the initial cache will forward a copy back to the user. However, in case of a cache miss within the local group, the cache joins more than one multicast group, so that all the cache groups heavily overlap each other. When there is a cache miss in one group each cache of the current group checks to see if its other group lies in the direction towards the originating server of the requested Web document. When a cache finds itself in the right position to forward the request it also informs the current group when doing so.

In case the second cache group has a miss again the request will be forwarded further following the same rules. Proceeding in this fashion the request either reaches a cache group with the data object, or otherwise is forwarded through a chain of overlaping cache groups between the client and the originating server until it reaches the group that includes the original server of the requested data object. Once the request reaches a group in which one or more servers have the requested data, the node

holding the page multicasts the response to the group.

In order for this caching infrastructure to be scalable the organization of Web caches into overlapping groups must be self-configuring. Self-organizing algorithms and protocols are proposed that allow cache groups to dynamically adjust themselves according to changing conditions in network topology, traffic load, and user demands. Thus, the adaptive caching uses the Cache Group Management Protocol (CGMP) and the Content Routing Protocol (CRP). CGMP specifies how meshes are formed and how individual caches join and leave those meshes. In general, caches are organized into overlapping multicast groups which use voting and feedback techniques to estimate the usefulness of admitting or excluding members from that group. The ongoing negotiation of mesh formation and membership results in a virtual topology. CRP is used to locate cached content from within the existing meshes. CRP takes advantage of the overlapping nature of the meshes as a means of propagating object queries betw een groups as well as propagating popular objects throughout the mesh.

An important assumption of the adaptive caching approach is that the deployment of cache clusters across administrative domains is not an issue. If the topologies are more flexible the administrative cache policies must be relaxed so that groups form naturally in different locations of the network.

LSAM

The LSAM Large Scale Active Middleware [119] is an architecture that uses a self-organizing multicast push based on interest groups. The LSAM proxy is deployed near both clients and servers. Near the client the proxy acts as an intelligent cache, allowing multicast channels to preload it with relevant pages. Near the server the proxy acts as an intelligent pump, managing multicast groups and detecting page affinities to multicast related information to a set of interested client caches.

The LSAM uses multicast for automated push of popular web pages. LSAM proxies are deployed as a server pump and a distributed filter hierarchy. These components automatically track the popularity of web page groups, and also automatically manage server push. In this architecture web pages are organized in *affinity groups* in relationship with their popularity. Individual requests trigger multicast responses when these pages are members of active affinity groups. A request is checked at intermediate proxies and forwarded to the server. The response is multicast to the filters in the group by the pump and unicast from the final proxy back to the originating client. Subsequent requests are handled locally from the filters near the clients.

IMPPS

The IMPPS Intelligent Multicast Push and Proxy System [83] is a system for interactive multicast cache running both at the end-user location and at the base station. IMPPS uses reliable multicast instead of TCP/IP to interactively request and reply web based content. Furthermore, IMPPS can be used to keep the web caches up to date by pushing fresh and popular web contents. This architecture is proposed in the context of LMDS (Local Multipoint Distribution Service) [72], a broadband wireless access technology that provide two-way transmission of data and multimedia.

Requests from the clients are received by the IMPPS proxy-cache through unicast IP. If the data is not available the request is forwarded via reliable multicast to the authenticated remote IMPPS proxy-caches. The remote IMPPS proxy/cache checks whether the requested web object can be served from the remote IMPPS proxy-cache or whether it has to be fetched from the Internet. In case of a local hit, the remote IMPPS proxy-cache sends the web data object back via multicast. If the web data object is not cached, it is requested from a nearby proxy via multicast. IMPPS proxies/caches communicate with each other using the MCP (Multicast Cache Protocol) that is provided by a specific transport multicast protocol that replaces the IP multicast. The IMPPS proposes the utilisation of push tecniques to disseminate fresh web content via multicast into the web-caches of the end-users.

B.1.3 Distributed Cache

Caches can be also organized to work in the same level: in this option it is necessary to create mechanisms or protocols to decide which cache will be contacted. In these cases, the protocols usually require information about the content of every other cache. In this context, cache content information is distributed among caches in the system. Sometimes, a hierarchical distribution mechanism is employed. However, the hierarchy is used only to distribute index information, not data content. Some distributed cache protocols are described later in this section.

Distributed caching allows better load balance work between caches and provides more capacities for fault tolerance. Nevertheless, it can create other troubles such as high connection times, more bandwith usage, and administrative conflicts.

CRISP

CRISP (Caching and Replication for Internet Service Performance) is a distributed architecture proposed in [58] [57]. In CRISP caches are structured as a collective of autonomous web proxies sharing their cache directories through a common mapping service that can be queried with at most one message exchange.

The CRISP architecture consists of a set of caching servers (proxies), which directly serve requests from clients, and a mapping service, which is responsible for maintaining the global directory of objects stored in the caching servers. On a local cache miss, each caching server queries the mapping service to determine if the object is resident elsewhere in the collective cache. If the mapping service reports a hit, then the object is fetched directly from the peer caching server. Caching servers keep the global directory current by reporting object fetches and evictions to the mapping service.

Individual servers may be configured to replicate all or part of the global map in order to balance access cost, overhead and hit ratio, depending on the size and geographic dispersion of the collective cache.

In [99] studies the cases in which distant copies of objects in the cache may not be worth fetching and for which the source server may be a better choice. Thus, it proposes that instead of maintaining the global directory with information of cached data objects, each node maintains a directory of objects cached in its vicinity. For each CRISP proxy this directory is different.

Distributed Caching

A distributed caching approach is proposed in [96] and while the hierarchy is still present in this scheme, it exploits the hierarchy for indexing distributed documents. It proposes to avoid the use of caches at the root and upper levels of the hierarchy to resolve requests and store cached documents. Instead, only the leaf caches would be responsible for retrieving and storing the objects, while the upper level caches would be used to maintain information about the contents of these caches. This scheme removes the requirement for upper level nodes to maintain large storage resources.

Figure B.2 illustrates how such approach works for a hierarchy of three levels. In this case, neither nodes A, B or C are caching objects: they are instead used only to propagate cache information. For example; if node F requires an object it queries node B. Here node B does not know where a copy of the object can be found. However, instead of returning a miss immediately to F it propagates the query to A. If Node A is also unable to locate the object returns a miss to B which then propagates this information back to F. This recursive querying approach allows the entire cache contents to be searched very quickly so that node F can correctly determine that there is no cached copy of the requested object and thus retrieve it from its primary site.

When a miss is encountered by a leaf cache and it resolves the request from its primary site, a mechanism is required to indicate to the upper level nodes in the hierarchy that the document has been cached. This procedure is known as an *advertisement*. Node F sends an advertisement message to its parent B. The advertisement

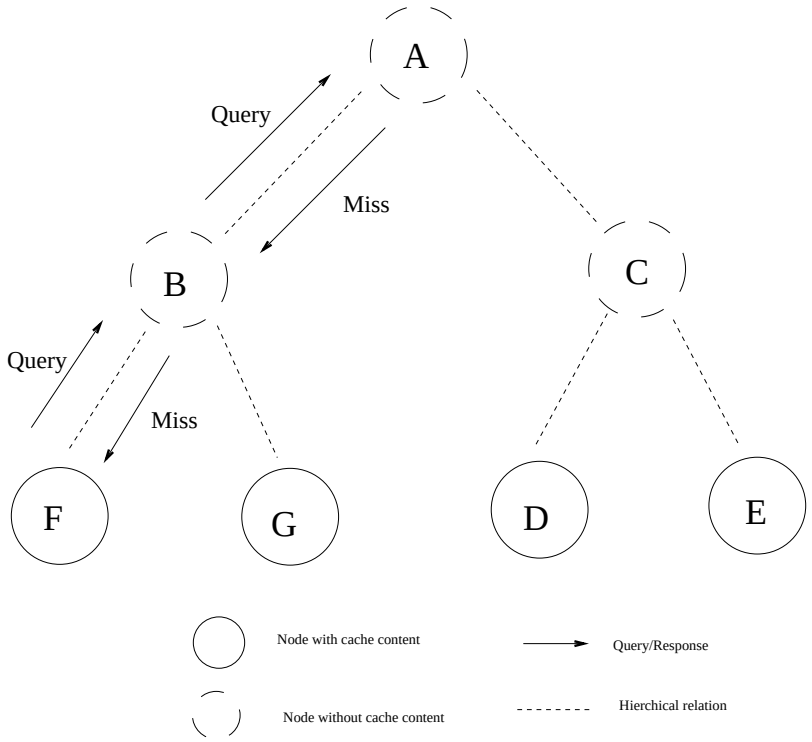

Figure B.2: An example of distributed cache

is then propagated recursively to the root node. If a child of node B requests the object then B will have information about where to find it. If a child of node C requests an object by recursively querying, then the root node A will indicate that a copy may be found at F. Rather than returning the actual object from its cache, the upper level cache returns a reference to the cached object consisting of a Uniform Resource Locator (URL) and information about the size and modification time of the object.

Distributed Hint Hierarchy

This architecture is described in [117]: it separates data paths from metadata paths and maintains a hierarchy of metadata that registers where copies of data are stored. It maintains location hints so that caches can locate nearby copies of data: it uses a direct cache to cache data transfers to send the data directly back.

This strategy uses a hierarchy to distribute hints. It only stores data at the leaves of the hierarchy and it always transfers data directly between caches rather than through the hierarchy. To facilitate widespread sharing of data among large numbers of caches, it compresses the size of hints so that hint caches can track the contents of several caches and it uses the hierarchy to distribute hints efficiently.

The system propagates hint updates through an index hierarchy to reduce the

amount of information sent globally. When a node in the index hierarchy learns about a new copy of data from a child, or the local data cache if it is the leaf, it propagates that information to its parent only if the new copy is the first copy stored in the subtree rooted at the parent. The node determines this by examining its local hint cache; if the parent had already informed the node of a copy in the subtree of the parent, then the node terminates the update propagation. Similarly, when a node learns about a new copy of data from a parent, it propagates that knowledge to its children if none of its children had previously informed it of a copy.

RELAIS

The Relais system [73] [88] allows a group of caches and mirrors to cooperate in order to improve web caching within an organization. Relais guarantees to its clients a monotonic and rapid progress on the retrieval versions of the documents. Relais provides a distributed directory with the contents of all data providers in the organization. The directory is always replicated, ensuring that name resolution is always a local operation.

The Relais architecture comprises four components: intermediate providers, provider agents, location proxies and directory managers.

intermediate providers stores document copies within the organization. Some examples of intermediate providers include web caching proxies, mirrors, archives, file systems, etc.

provider agent allows an intermediate provider to register documents in their posession with the shared distributed directory. An intermediate provider notifies its provider agent when adding or removing a documen and when detecting a newer version of a document.

location proxy enables a user to transparently access the group of intermediate providers that make up the organization-wide distributed cache.

directory managers perform registration and removal of document locations, invalidation of cached copies when their originals are updated and lookup of available locations.

Directory service of Relais permits efficient and transparent location of document copies cached within the organization boundaries. Relais provides a basic structure for efficient sharing of information among well delimited groups of users, possibly distributed at geographically separate sites. An important characteristic is that permits connects different kinds of data sources such as web caching proxies, file systems, and replicated databases.

B.1.4 Peer to Peer Cache

Squirrel

Squirrel is a cooperative cache system based on a peer-to-peer scheme [75]. The key idea in Squirrel is to allow mutual sharing of data objects among client nodes. Thus, web browsers on every node maintain a local cache of the web objects recently accessed by the browser. Squirrel enables these nodes to export their local caches to other nodes in the corporate network, in this way they build a large shared virtual cache. Each node then performs both web browsing and web caching.

Squirrel uses a self-organizing and peer-to-peer routing protocol called Pastry as its object location service, to identify and route to nodes that cache copies of a requested object. Pastry provides the functionality of a scalable distributed hash-table that permits to map a given data object key to a unique active node in the network. The system assigns uniformly a portion of URL space to participating caches using a hash function.

Each client runs an instance of Squirrel software and configures the web browser in that node to use this Squirrel instance as its proxy cache. The browser and Squirrel share a single cache managed by the Squirrel instance. User browsers issue their requests to the Squirrel proxy running on the same node. The proxy checks the local cache, just as the browser would normally have done to exploit the client-side locality. If a copy of the object is not found in this cache, then Squirrel essentially tries to locate a copy in some other cache. It starts by mapping the object URL (the key) to a node in the network and then it invokes the Pastry routing procedure to forward the request to the node with the node indentifier numerically closest to this object indentifier. It designates the recipient as the home node for this object.

This architecture proposes two approaches for request resolution based on the question of whether the particular node actually stores the data object, or whether it only maintains a directory of the information about a small set of nodes that store the object. These nodes, called delegates, are nodes whose browsers have recently requested the object, and are likely to have it in their local caches.

BuddyWeb

BuddyWeb [125] is a peer-to-peer based collaborative web caching system designed for enterprise networks. It combines the power of mobile agents into peer-to-peer systems to perform operations at sites of peers. In BuddyWeb, all the local cache of participating nodes are available to be shared, and the nodes within the enterprise network will be searched first before remote external accesses are invoked. BuddyWeb is supported by local components called BestPeer.

Whenever the web browser submits a URL request or keyword queries, the local proxy will receive and rewrite the query into the input format of the BestPeer. The query will then be passed to BestPeer component and it generates a mobile agent and dispatches it to the network to search for matching documents. Upon receiving a match, BestPeer passes the information about document location, back to the local proxy. In this way, the local proxy will issue HTTP request directly to the peer that has the documents. The peer, upon receiving the HTTP request, will process it by the HTTP daemon, and sends the requested documents to the requester.

The peer network can dynamically reconfigure itself based on the similarity of its interests. This similarity is established analysing the user fetched documents. The routing strategy is also based on the idea of similarity of content of the peers. Query are routed from a peer to its neighbor that has the highest similarity value.

Kache

The Kache [84] is a cooperative caching system implemented over a peer-to-peer system called Kelips. Kelips implements an index over a set of participating caches (nodes) where applications can insert (key,value) pairs into an indexing structure, and can perform lookup operations on the keys, retrieving the associated value. In this peer-to-peer indexing structure the objects can be inserted, retrieved, and deleted from a distributed collection of nodes. The key of a cached web object copy consists of its original URL, and the value would specify the location of the cached copy.

Kelips consists of virtual affinity groups; each node lies in an affinity group determined by using a consistent hashing function to map the identifier of node (IP address and port number).

For query routing, Kelips starts with a single query in the hope that it will locate the desired data resource rapidly. However, if a resource lookup or insert query fails, the querying node retries the query in a more aggressive multi-hop mode. Query retries occur along several axes: first, the querying node can concurrently send the query to multiple contacts; second, contacts could be asked to forward the query within their affinity group; and finally, the querying node could request the query to be executed at another node in its own affinity group (if this is different from the affinity group of the resource).

B.2 Cache Communication Protocols

Typical cache interaction consists in the exchange of information about cache content or direct exchange of data content. Different cooperative architectures need a

intercache communication protocol to support this function. Several protocols have been proposed for different works. An over view is presented in this section.

B.2.1 Internet Cache Protocol (ICP)

The Internet Cache Protocol (ICP) [127] [126] was initially developed by Peter Danzig, et. al. [128] at the University of Southern California as a central component of hierarchical caching architecture in the Harvest research project [16].

ICP defines a lightweight message format used for communicating among web caches. This communication protocol is primarily used in a cache group to locate specific Web objects in neighboring caches. One cache sends an ICP query to its neighbors. The neighbors send back ICP replies indicating a hit or a miss.

When a cache does not hold a requested object, it may ask via a query message ICP_QUERY whether any of its neighbor caches has the object. If any of the neighbors has the requested object it receives a message ICP_HIT, and the cache will request it from them. If none of the neighbors has the object, it receives messages ICP_MISS, and the cache must forward the request either to a parent, or directly to the originating server.

By default a cache sends an ICP_QUERY message to each neighbor cache. Normally, it expects to receive a ICP_HIT reply from one of the neighbors. Because ICP uses UDP as underlying transport, ICP queries and replies may sometimes be dropped by the network. The cache installs a timeout event in case not all of the expected replies arrive.

ICP protocol is supported by several proxy cache implementation like Squid, Microsoft ISA, Cisco Cache Enging and BlueCoat.

The Hypertext Caching Protocol (HTCP) was designed as a successor to the ICP. Its objective is to increase the integration with HTTP. In general the global operation is similar, HTCP fixes some limitation of ICP, thus it permits full request and response headers to be used in the cache management in contrast with simple URL used in ICP. The HTCP requests are designed for enabling HTCP replies to more accurately describe the behavior that would occur as a result of a subsequent HTTP request of the same resource.

B.2.2 Cache Digests

A cache digest [102] proposes a technique which allows caches to efficiently inform each other about their contents without query reply scheme. Instead of transfer

the entire list of data objects from a cache to another, a summary or digest of the contents of each cache is exchanged. These digest are compressed in a special array of bits called *Bloom Filter*.

To add an entry to the bloom filter, a small number of independent hash functions are computed for the key of entry (usually the URL). The hash function values specify which bits of the filter should be turned on. To check whether a specific entry is in the filter, it calculates the same hash function values for its key and examine the corresponding bits. If one or more of the bits is off, then the entry is not in the filter. If all bits are on, there is a probability that the entry is in the filter.

However, Bloom Filters do not perfectly represent all of the items they encode. Occasionally the Bloom Filter will incorrectly report some item is present, when in fact it is not. For web caches, this means to generate a remote cache miss when it was expecting a cache hit.

Cache Digests use a pull technique for disseminating cache digests. The caches exchange digests via HTTP. Pull fits very well with the current distribution and access control schemes for web objects. For digest updates, after a proxy gets a digest from its neighbor, the digest is no longer synchronized with the contents of the cache of the neighbor. In that case is necessary for a proxy to notify its digest users about recent changes in its cache contents.

The Summary Cache [51] is a similar protocol based on exchange of summary or digest of the cache contents generated using a hash function. To reduce the overhead the summaries are stored as a Bloom Filter and it does not have to be updated every time the cache directory is changed; rather, the update can occur upon regular time intervals or when a certain percentage of the cached documents are not reflected in the summary.

B.2.3 CARP

The Cache Array Routing Protocol CARP) describes a distributed caching protocol based on a known membership list of loosely coupled proxies and a hash function for dividing URL space among those proxies [101] [122]. CARP provides a deterministic request resolution path; there is no query messaging between proxy servers.

The basic mechanism of CARP is:

- All proxy servers are registered through an array membership list which is automatically updated through a time-to-live (TTL) countdown function that regularly checks for active proxy servers.
- A hash function is computed for the name of each proxy server.

- A hash value of the requested URL is calculated.
- The hash value of the URL is combined with the hash value for each proxy. Whichever URL plus the proxy hash comes up with the highest value becomes owner of the information cache.

The result is a deterministic location for all cached information, meaning that the downstream proxy server can know exactly where a requested URL is either already stored locally, or will be located after caching. Because the hash functions used to assign great values, the result is a statistically distributed load balanced across the array.

The deterministic request resolution path provided by CARP avoids the need to maintain massive location tables for cached information. The client simply runs the same math function across an object to determine where the object is.

The division of URL space among a group or array of proxy caches permits to eliminate the duplication of contents that otherwise occurs on a group of proxy servers. For example, with the ICP protocol a group of proxy servers can rapidly evolve into essentially duplicate caches of the most frequently requested URLs. The hash-based routing of CARP keeps this from happening, allowing all proxy servers to exist as a single logical cache. The result is a more efficient use of server resources.

B.2.4 Web Cache Coordination Protocol (WCCP)

The Web Cache Communication Protocol (WCCP) [33] [34] was originally developed by Cisco. WCCP enables routers or switches to transparently redirect the HTPP packets to caches rather than to intended host sites. It also balances the traffic load across a group of caches.

WCCP defines a *Service Group* of one or more routers plus one or more web-caches working together in the redirection of the traffic. The purpose of the interaction in WCCP is to establish and maintain the transparent redirection of selected types of traffic flowing through a group of routers. The selected traffic is redirected to a group of caches with the aim of optimizing the resource usage and lowering the response times.

Unlike other cache protocols, WCCP uses the IP address of the web server instead of the URL to calculate the hash function. A redirection hash table is maintained by the router. This table maps the hash index derived from a HTTP packet to be redirected to the IP address of a destination cache. Thus a router checks the destination IP of the HTTP packets passing through it against its set of caches. A primary key is formed from the packet and hashed to yield an index into the redirection hash table.

If the entry contains a cache index then the packet is redirected to that cache. If the entry is unassigned the packet is forwarded normally.

Once the service group has been established, one of the caches is designated to determine load assignments among the caches. The role of this cache is to determine how traffic should be allocated across caches. The assignment information is passed to the entire service group from the designated cache so that the routers of the group can redirect the packets properly and the cache of the group can manage their load better.

Appendix C

Activity Information Elements

C.1 Activity Information Elements

C.1.1 Storage Usage

Storage usage is the information activity about storage capacity used by the storage entity at a specific time. It is used to establish the available storage capacity of a storage resource in a dynamic way. It permits to establish historical utilisation of the storage resource. The figure C.1 shows the XML Schema element definition of the storage usage activity information.

```
<xsd:element name="storage_usage">
  <xsd:complexType>
    <xsd:sequence>
    <xsd:element name="dateTime" type="xsd:dateTime"/>
    <xsd:element name="used_storage">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:integer">
          <xsd:attribute name="units" type="storageUnitsType" use="required"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
    </xsd:sequence>
  <xsd:attribute name="id" type="xsd:integer"/>
  </xsd:complexType>
</xsd:element>
```

Figure C.1: XML Schema element definition of the storage usage activity information

C.1.2 Data Action

Data action is the information that registers the individual activity of each data entity. This information permits tracing in detail the access operations realised on a specific data object. This includes the main record of the data access operations. It also includes the internal operations initiated by the cache system: it comprises the source or entity (local cache instance, remote cache instance, client etc) that causes the action. Making data action information available makes it possible to establish the degree of activity of particular data objects. The figure C.2 shows the XML Schema element definition of the data action activity information.

```
<xsd:element name="data_action">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="operation" type="dataActionsType"/>
      <xsd:element name="source" type="xsd:string" />
      <xsd:element name="dateTime" type="xsd:dateTime" />
      <xsd:element name="data_id" type="xsd:integer" />
      <xsd:element name="data_name" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:integer" />
  </xsd:complexType>
</xsd:element>
```

Figure C.2: XML Schema element definition of data action activity information

C.1.3 Data Transfer

Data Transfer is the activity information that registers the individual data transfers between cache instances. This information establishis in a detailed manner the data transfers realised between cache instances. The analysis of this information permits to know the behaviour of the system in relation to resource usage in the context of a global data movement. Similarly it is the base to establish data movement patterns in the distributed system. Recognising data movement patterns allows the execution of sophisticated data distribution strategies. Transfer data information comprises the source and destination locations (cache instances) and transfer time elapsed. The figure C.3 shows the XML Schema element definition of the data transfer activity.

C.1.4 Request

Request is the activity information element used to register the individual request processed by a cache instance. This information permits to measure the performance

```
<xsd:element name="data_transfer">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="data_id" type="xsd:integer" />
      <xsd:element name="data_name" type="xsd:string" />
      <xsd:element name="cache_source" type="xsd:string" />
      <xsd:element name="cache_destination" type="xsd:string" />
      <xsd:element name="start_date_time" type="xsd:dateTime" />
      <xsd:element name="finish_date_time" type="xsd:dateTime"/>
      <xsd:element name="status" type="statusType" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:integer" />
  </xsd:complexType>
</xsd:element>
```

Figure C.3: XML Schema element definition of data transfer activity information

and effectiveness of the cache instance. Request information include the name the operation demanded, the type of request (client or intercache), the type of response (successful, failed, not supported), and the time elapsed for processing of the request. The figure C.4 shows the XML Schema element definition of the request element. The *status* element indicates the state of requests like submitted, started, finish, cancelled or undefined.

```
<xsd:element name="request">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="operation" type="operationsType"/>
      <xsd:element name="type_request" type="requestType"/>
      <xsd:element name="type_response" type="responseType"/>
      <xsd:element name="start_date_time" type="xsd:dateTime" />
      <xsd:element name="finish_date_time" type="xsd:dateTime"/>
      <xsd:element name="status" type="statusType" minOccurs="0"/>
      <xsd:element name="source" type="xsd:string" />
      <xsd:element name="destination" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:integer" />
  </xsd:complexType>
</xsd:element>
```

Figure C.4: XML Schema element definition of request activity information

Appendix D

Cache Operations Definitions

D.1 Access Operations

The operations discussed in this section need appropriate security priviledges to succeed. The users need to be authorised and authenticated to perform data object access actions: we assume that the users are authorised and authenticated by a service or system defined by the community that deploys the grid. In other work our research team proposes an architecture for the protection of confidential data on Grids [107] [106] [105].

D.1.1 SetData()

SetData() is used to register in the cache service a *data entity* element (section 4.6.1) that corresponds to a new data object placed in the storage resource managed by the cache instance. The *SetDataRequest* needs as parameter a *data entity* element with *name* as a unique identifier (for example Logical File Name). Additional items are optional and merely descriptive information; including *Metadata entity* 4.6.1.

The operation implementation updates values for *size*, *internal path location* and *creation date* using local underlying I/O facilities. For intercache working, the implementation operation starts the data transfer between cache services using the mechanism of data transport (see section 3.3). The cache service creates a unique numerical identification *id* for each data entity registered which is used internally for the cache system.

The operation returns a *SetDataResponse* element with the operation confirmation which can be successful, failed or not supported. The figure D.1 shows the XML Schema element definition of the SetData() operation.

```
<xsd:element name="SetDataRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element ref="data" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="SetDataResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.1: XML Schema element definition of the SetData() operation

D.1.2 GetData()

GetData() is used to retrieve from the cache service a *data entity* element (section 4.6.1) that corresponds to data object storage in the cache instance. The *GetDataRequest* needs as parameter a *data name* supplied as unique identifier (for example Logical File Name) with the *SetData()* operation (see section D.1.1) or the *id* registered by the cache service.

In the working intercache, the cache service starts the data transfer between cache services using a mechanism for data transport (see section 3.3). The cache service can execute a mechanism for the collaborative resolution between several caches, this only will be possible if collaborative mechanism are implemented and configured by the cache group.

The *GetDataResponse* return the *data entity* element if the data object is present in the cache system. On the contrary, the *type response* of the response header is set to failed. The figure D.2 shows the XML Schema element definition of the GetData() operation.

D.1.3 RemoveData()

RemoveData() is used for removing; each specified data entity from the cache services a *data entity* element (section 4.6.1) that corresponds to the data object storage in the cache instance. Similar to *GetDataRequest* the *RemoveDataRequest*, it needs as parameter a *data name* supplied as a unique identifier (for example a Logical File

```
<xsd:element name="GetDataRequest">
  <xsd:complexType>
  <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="data_id" type="xsd:string"/>
      <xsd:element name="data_name" type="xsd:string"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="GetDataResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="data" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.2: XML Schema element definition of the GetData() operation

Name) with the *SetData()* operation (see section D.1.1) or the *id* registered by the cache service.

The *RemoveDataResponse* returns the *data entity* element if the data object is present in the cache system. In the contrary case, the *type response* of the response header is set in failed. The figure D.3 shows the XML Schema element definition of the RemoveData() operation.

D.1.4 GetDataContent()

GetDataContent() returns the list of all *data entity* elements (section 4.6.1) that are stored in the cache instance. The *GetDataContentRequest* does not take parameters in addition to *requestHeader*.

This operation can be used to exchange content indexes between cache instances. Other processes can be implemented based on this information, for example, control copy proliferation among the cache group.

The *GetDataContentResponse* returns a list of *data entity* elements. On the contrary case, the *type response* of response header is set to failed. The figure D.4 shows the XML Schema element definition of the GetDataContent() operation.

```
<xsd:element name="RemoveDataRequest">
  <xsd:complexType>
  <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="data_id" type="xsd:string"/>
      <xsd:element name="data_name" type="xsd:string"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="RemoveDataResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="data" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.3: XML Schema element definition of the RemoveData() operation

```
<xsd:element name="GetDataContentRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="GetDataContentResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="data" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.4: XML Schema element definition of the GetDataContent() operation

D.1.5 SetMetatada()

SetMetatada() updates the *metadata* entity information (section 4.6.1) that corresponds to *data entity* (section 4.6.1) managed by the cache system. The *SetMetatadaRequest* element takes as parameters the *data name* or *data id* of the data entity and the *data metadata* element with information to register.

This operation permits to update the description information about data content managed by the cache. This is basic functionality that does not substitute a metadata management mechanism. Metadata in the cache system can be used to add a complementary description of data in cache. The operation implementation can be supported by a metadata catalog based on a database management system.

The operation returns a *SetMetatadaResponse* element with operation confirmation. If the data entity does not exist in the cache system, the *type response* in the responseHeader is set to failed. The figure D.5 shows the XML Schema element definition of the SetMetatada() operation.

```
<xsd:element name="SetMetatadaRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="data_name" type="xsd:string"/>
      <xsd:element name="data_id" type="xsd:integer"/>
      <xsd:element ref="data_metadata"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="SetMetatadaResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.5: XML Schema element definition of the SetMetatada() operation

D.1.6 GetMetatada()

GetMetatada() retrieves the *metadata* entity information (section 4.6.1) that corresponds to a *data entity* (section 4.6.1) in the cache system. The *getMetatadaRequest* element takes as parameters the *data name* or *data id* of the data entity.

This operation is used to get a basic and complementary information about the data content in the cache system. The operation implementation can be supported by a metadata catalogue based on a database management system.

The operation returns a *getMetatadaResponse* element with the *metadata* entity requested. In a negative case, if the data entity does not exist in the cache system, the *type response* in the response header is set to failed. The figure D.6 shows the XML Schema element definition of the GetMetatada() operation.

```
<xsd:element name="getMetatadaRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="data_name" type="xsd:string"/>
      <xsd:element name="data_id" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getMetatadaResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="data_metadata"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.6: XML Schema element definition of the GetMetatada() operation

D.2 Monitor Operations

D.2.1 GetCache()

GetCache() retrieves the *cache* entity information (section 4.6.1) that corresponds to a cache service instance. The *getCacheRequest* element does not take as parameters in addition to *requestHeader*.

This operation permits to know the current description and configuration of a cache service instance. Gathering cache entity information from a group of caches permits to establish the global state of the collective system.

The operation returns a *getCacheResponse* element with the *cache* entity information requested. If the cache entity does not exist in the cache instance, the *type response* in the response header is set to failed. The figure D.7 shows the XML Schema element definition of the GetCache() operation.

D.2.2 GetReplacementMethod()

GetReplacementMethod() retrieves the name of *replacement method* applied in a cache instance. It is registered in the *replacement method* element of *cache* entity information (section 4.6.1). The *getReplacementMethodRequest* element does no take

```
<xsd:element name="getCacheRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getCacheResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="cache" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.7: XML Schema element definition of GetCache() operation

parameters in addition to *requestHeader*.

This operation permits to know dynamically the *replacement method* applied by a cache service.

The operation returns a *getReplacementMethodResponse* element with the name of the *replacement method* in execution. The figure D.8 shows the XML Schema element definition of the GetReplacementMethod() operation.

```
<xsd:element name="getReplacementMethodRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getReplacementMethodResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element name="replacement_method" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.8: XML Schema element definition of the GetReplacementMethod() operation

D.2.3 GetStorage()

GetStorage() retrieves the *storage* entity information (section 4.6.1) that corresponds to a storage resource registered in a cache service instance. The *getStorageRequest* element takes as parameter the storage identification of the *storage* element.

This operation permits to establish the configuration of a storage resource in a particular cache service instance. It is used to analyse and evaluate the features of the available storage resources.

The operation returns a *getStorageResponse* element with the *storage* entity information required. If the cache entity has not registered the storage resource, the *type response* in the responseHeader is set to failed . The figure D.9 shows the XML Schema element definition of the GetStorage() operation.

```
<xsd:element name="getStorageRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="storage_id"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getStorageResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="storage"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.9: XML Schema element definition of the GetStorage() operation

D.2.4 GetCacheGroup()

GetCacheGroup() retrieves the *cache group* entity information (section 4.6.1) registered in a cache instance. The *getCacheGroupRequest* element takes as parameters the *cache group identifier* or the *cache group name* of the *cache group*.

This operation permits to monitor the configuration information of a cache service. This information is essential to support the implemented intercache collaboration mechanisms between groups of caches.

The operation returns a *getCacheGroupResponse* element with the *cache group* entity required. If the cache entity does not have registered a group, the *type response* in the response header is set in failed. The figure D.10 shows the XML Schema element definition of the GetCacheGroup() operation.

A cache service can participate to several groups. Each cache instance keeps one or several Cache Group information entities (see section 4.6.1) that record the members of different groups.

```
<xsd:element name="getCacheGroupRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="cache_group_id" type="xsd:integer"/>
      <xsd:element name="cache_group_name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getCacheGroupResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="cache_group" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.10: XML Schema element definition of GetCacheGroup() operation

D.2.5 GetRequestProcessed()

GetRequestProcessed() returns a subset of the total *request* elements (section C.1.4) registered by the cache instance. The *getRequestProcessedRequest* takes as parameters the initial and final date of the time period required.

This operation permits to obtain the detailed information about the request processed by the cache service at specific time period. An extension of the operation can include other parameters such as the *type request*, the *type response*, the *status*, the *source* or the*destination*. The operation implementation can be supported by queries to a database management system that registers the requests processed by the system.

The operation returns a *getRequestProcessedResponse* element with a list of the *request* elements. The figure D.11 shows the XML Schema element definition of the GetRequestProcessed() operation.

```
<xsd:element name="getRequestProcessedRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="start_date_time" type="xsd:dateTime"/>
      <xsd:element name="finish_date_time" type="xsd:dateTime"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getRequestProcessedResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="request"  maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.11: XML Schema element definition of the GetRequestProcessed() operation

D.2.6 GetTransfers()

GetTransfers() returns a subset of the total *transfer data* elements (section C.1.3) registered by the cache instance. The *getTransfersRequest* takes as parameters the initial and final date of the time period required.

This operation permits to obtain the detailed information about the data transfers realised by the cache service at specific time period. An extension of the operation can include other parameters in addition to time period such as the *data entity*, the *source*, the *destination* or the *status*. The operation returns a *getTransfersResponse* element with a list of the *transfer data* elements required. The figure D.12 shows the XML Schema element definition of the GetTransfers() operation.

D.2.7 GetDataActions()

GetDataActions() returns a subset of the total *data action* elements (section C.1.2) registered by the cache instance. The *GetDataActionsRequest* takes as parameters the initial and final date of the time period required.

This operation permits to obtain the detailed information about the data actions realised by the cache service at specific time period. An extension of the operation can include other parameters in addition to time period such as the *operation*, the

```
<xsd:element name="getTransfersRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="start_date_time" type="xsd:dateTime"/>
      <xsd:element name="finish_date_time" type="xsd:dateTime"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getTransfersResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="transfer_data"  maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.12: XML Schema element definition of GetTransfers() operation

source or the *data entity*. The combination of these parameters permit the extraction statistical information derived from *data action* activity information registered. For example, it is possible to establish the number of access to a specific *data entity* at a particular time period. The operation implementation can be supported by queries to a database management system that registers the data actions realised by the system.

The operation returns a *GetDataActionsResponse* element with a list of *data action* elements. The figure D.13 shows the XML Schema element definition of the GetDataActions() operation.

D.2.8 GetDataReplacements()

GetDataReplacements() returns the number of data replacements realised by the cache service instance. This value is obtained by calculation on a subset of the total *data action* elements (section C.1.2) registered by the cache instance. The *GetDataReplacementsRequest* takes as parameters the initial and final date of the time period required.

This operation obtains the number of data replacements realised by the cache service at specific time period. It is a result derived from information registered with *data action* elements. This measure permits to monitor an operational parameter in relationship with the cache service load. The operation implementation can be supported by queries to a database management system that registers the data

```
<xsd:element name="GetDataActionsRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="start_date_time" type="xsd:dateTime"/>
      <xsd:element name="finish_date_time" type="xsd:dateTime"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="GetDataActionsResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element ref="data_action" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.13: XML Schema element definition of the GetDataActions() operation

actions realised by the system.

The operation returns a *GetDataReplacementsResponse* with the number of data replacements. The figure D.14 shows the XML Schema definition element of the GetDataActions() operation.

```
<xsd:element name="GetDataReplacementsRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="start_date_time" type="xsd:dateTime"/>
      <xsd:element name="finish_date_time" type="xsd:dateTime"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="GetDataReplacementsResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element name="number_replacements" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.14: XML Schema element definition of the GetDataActions() operation

D.2.9 GetStorageCapacity()

GetStorageCapacity() returns the current quantity of storage available as managed by the cache instance. The *getStorageCapacityRequest* does not takes parameters.

This operation obtains the current storage capacity available in a cache service. This value permits to gather information from several caches to establish dynamically their collective capacity. The operation implementation can be obtained from *data* elements 4.6.1 registered or from functions supported for the storage management subsystem in the local installation.

The operation returns a *getStorageCapacityResponse* with the capacity value. The figure D.15 shows the XML Schema definition element of the GetStorageCapacity() operation.

```
<xsd:element name="getStorageCapacityRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getStorageCapacityResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
      <xsd:element name="capacity" type="xsd:integer"/>
      <xsd:element name="capacity_unit" type="storageUnitsType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.15: XML Schema element definition of the GetStorageCapacity() operation

D.3 Configuration Operations

Configuration operations are invoked by grid resource administrators that require to configure collaborative cache systems.

D.3.1 SetCache()

SetCache() updates the *cache* entity information (section 4.6.1) that corresponds to a particular cache service instance. The *setCacheRequest* element takes as parameters the *cache* element with information that describes the cache service instance to register.

This operation updates the configuration information of a cache service. It is used to start the service or updates one or more elements of a cache entity information. It updates several values with only one operation. The operation implementation must initiate the actions related with the configuration modifications.

The operation returns a *setCacheResponse* element with operation confirmation. If the cache entity was not updated in the cache system, the *type response* in the response header is set to failed. The figure D.16 shows the XML Schema definition element of the SetCache() operation.

```
<xsd:element name="setCacheRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element ref="cache"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="setCacheResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.16: XML Schema element definition of the SetCache() operation

D.3.2 SetReplacementMethod()

SetReplacementMethod() actives the *replacement method* in the cache instance. It also updates the value in the *replacement method* element of *cache* entity information (section 4.6.1). The *setReplacementMethodRequest* element takes as parameters the name of the new *replacement method*.

This operation configures dynamically the *replacement method* applied by a cache service. It is used to modify the operation and possibly the performance of the cache

service. In the operation implementation different replacement methods can be supported. The specified method is applied immediately the operation is processed. Cache instance implements the necessary procedures to update the parameters required by each configuration.

The operation returns a *setReplacementMethodResponse* element with *response header*. Thus the *type response* element can have successful values if the method was changed, or not supported if the cache mechanism does not implement the method required. The figure D.17 shows the XML Schema definition element of the SetReplacementMethod() operation.

```
<xsd:element name="setReplacementMethodRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="replacement_method" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="setReplacementMethodResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.17: XML Schema element definition of the SetReplacementMethod() operation

D.3.3 SetCacheGroup()

SetCacheGroup() updates the *cache group* entity information (section 4.6.1) that describes a cache group which the cache service is member. The *setCacheGroupRequest* element takes as parameters the *cache group* element with the list of members of the group.

This operation permits updates the group configuration information of a cache service. It is used to inform each member the list the members belong to a group. It also supports the mechanisms that manage the addition and retire of members. This cache group information is essential to support the intercache collaboration mechanisms.

The operation returns a *setCacheGroupResponse* element. The figure D.18 shows the XML Schema definition element of the SetCacheGroup() operation.

```
<xsd:element name="setCacheGroupRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element ref="cache_group"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="setCacheGroupResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.18: XML Schema element definition of the SetCacheGroup() operation

D.3.4 SetStorage()

SetStorage() updates the *storage* entity information (section 4.6.1) that corresponds to a storage resource controlled by a cache service instance. The *setStorageRequest* element takes as parameter the *storage* element with information that describes the storage resource to register.

This operation permits to assign the configuration information of a storage resource to be managed by a cache service instance. It is used to update storage resources configuration in a cache service. The operation implementation interacts with configured resource using the I/O facilities in the local installation.

The operation returns a *setStorageResponse* element. The figure D.19 shows the XML Schema definition element of the SetStorage() operation.

D.3.5 SetDefaultTimeToLive()

SetDefaultTimeToLive() sets the minimal time that a data object must be kept in cache before it can be evicted. It updates the *default ttl* element of *cache* entity information (section 4.6.1) of the cache service instance in execution. The *setDefaultTimeToLive* element takes as a parameter the *time* to be applied.

This operation permits to configure dynamically the default time to live applied by a cache service to data entities. It is used to regulate cache replacement rate. The operation implementation can support this option with different replacement

```
<xsd:element name="setStorageRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element ref="storage"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="setStorageResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.19: XML Schema element definition of the SetStorage() operation

methods. The specified time value is applied immediately the operation is processed.

The operation returns a *setDefaultTimeToLiveResponse* element. The *type response* element can have the values: successful if the value was applied, or not supported if the cache mechanism does not implement this option. The figure D.20 shows the XML Schema element definition of the SetDefaultTimeToLive() operation.

```
<xsd:element name="setDefaultTimeToLiveRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="time" type="timeUnitsType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="setDefaultTimeToLiveResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.20: XML Schema element definition of the SetDefaultTimeToLive() time to live operation

D.3.6 SetCacheCoordinator()

SetCacheCoordinator() sets the cache instance coordinator by default of a cache service. The coordinator is typically a high level cache instance in a collaborative hierarchical organisation (section B.1.1). The *setCacheCoordinatorRequest* element takes as parameter the *cache identification* that identify the coordinator cache.

This operation permits to assign dynamically a cache instance as coordinator or high level cache in a collaborative hierarchical organisation. It is used to modify the relationship between a group of caches. The operation implementation depends of the collaborative cache mechanisms supported by the cache services involved.

The operation returns a *setCacheCoordinatorResponse* element. The *type response* element can have the values: successful, if the value was applied, or not supported if the cache mechanism does not implement this option. The figure D.21 shows the XML Schema element definition of the SetCacheCoordinator() operation.

```
<xsd:element name="setCacheCoordinatorRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="cache_id" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="setCacheCoordinatorResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure D.21: XML Schema element definition of SetCacheCoordinator() operation

D.3.7 SetCacheCollectiveWork()

SetCacheCollectiveWork() actives or inactives the cache operation in collective mode. With the collective mode active the cache service works invoking capabilities from other cache services. With the collective mode inactivate the cache service does not invoke external capabilities. The *setCacheCollectiveWorkRequest* element takes as parameter a boolean value that actives or inactives the option.

This operation turns off the capacity of the cache instance for request operations

from other caches. It is used to stop or start the collective operation of an individual cache instance.

The operation returns a *setCacheCollectiveWorkResponse* element. The *type response* element can have the values: successful if the new mode was applied, or not supported if the cache mechanism does not implement this option. The figure D.22 shows the XML Schema definition element of the SetCacheCollectiveWork() operation.

```
<xsd:element name="setCacheCollectiveWorkRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="requestHeader"/>
      <xsd:element name="name="work_on_collective_mode" type="xsd:boolean" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="setCacheCollectiveWorkRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="responseHeader"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Figure D.22: XML Schema element definition of the SetCacheCollectiveWork () operation

Bibliography

[1] ABRAMS M., STANDRIDGE C., ABDULLA G., et al . *Caching Proxies: Limitations and Potentials*. Technical report [**on-line**]. Virginia Polytechnic Institute & State University, Blacksburg, VA, USA, 1995. Available from: `http://ei.cs.vt.edu/succeed/www4/www4.html` (webpage visited on 29.03.2007).

[2] ABRAMS M., STANDRIDGE C., ABDULLA G., et al . *Removal Policies in Network Caches for World-Wide Web Documents*. **In :** SIGCOMM, 1996, pp 293–305.

[3] AHMED M., ZAHEER R., QADIR M. *Intelligent cache management for data grid*. **In :** ACSW Frontiers '05: Proceedings of the 2005 Australasian workshop on Grid computing and e-research, 2005, Darlinghurst, Australia, Australia. Australian Computer Society, Inc., pp 5–12.

[4] ALLCOCK W. *GridFTP: Protocol Extensions to FTP for the Grid*. Technical report [**on-line**]. Global Grid Forum, 2003. Available from: `http://www.ogf.org/documents/GFD.20.pdf` (webpage visited on 16.04.2007).

[5] ALLIANCE GLOBUS, IBM . *The WS-Resource Framework (WSRF)*. Technical report [**on-line**]. Globus Alliance, 2004. Available from: `http://www.globus.org/wsrf` (webpage visited on 11.08.2007).

[6] ALTINEL M., BORNHÖVD C., KRISHNAMURTHY S., et al . *Cache Tables: Paving the Way for an Adaptive Database Cache*. **In :** VLDB, 2003, pp 718–729.

[7] ANTONIOLETTI M., ATKINSON M., PATON N., et al . *The design and implementation of Grid database services in OGSA-DAI*. Concurrency - Practice and Experience, 2005, vol. 17, n°2-4, pp. 357-376.

[8] ANTONIOLETTI M., COLLINS B., PATON N., et al . *Web Services Data Access and Integration*. Technical report [**on-line**]. Global Grid Forum, 2006. Available from: `http://www.ogf.org/documents/GFD.76.pdf` (webpage visited on 16.04.2007).

[9] ARLITT M., CHERKASOVA L., DILLEY J., et al . *Evaluating Content Management Techniques for Web Proxy Caches*. SIGMETRICS Perform. Eval. Rev., 2000, vol. 27, n°4, pp. 3–11.

[10] ARLITT M., FRIEDRICH R, JIN T. *Performance evaluation of Web proxy cache replacement policies.* Perform. Eval., 2000, vol. 39, n°1-4, pp. 149-164.

[11] BAKKER A., AMADE E., BALLINTIJN G., et al . **In :** USENIX Annual Technical Conference, FREENIX Track, 2000, pp 141–152.

[12] BARU C., MOORE R., RAJASEKAR A., et al . *The SDSC Storage Resource Broker.* **In :** Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research, November 30 - December 3, 1998, Toronto, Ontario, Canada. 1998, IBM, pp 5.

[13] BIOMEDICAL INFORMATICS RESEARCH NETWORK . *BIRN* [**on-line**]. Available from: `http://www.nbirn.net` (webpage visited on 16.04.2007).

[14] BLAZE M., ALONSO R. *Dynamic Hierarchical Caching for Large-Scale Distributed File Systems.* **In :** ICDCS, 1992, pp 521–528.

[15] BOLOT J-C., HOSCHKA P. *Performance Engineering of the World Wide Web: Application to Dimensioning and Cache Design.* Computer Networks, 1996, vol. 28, n°7-11, pp. 1397-1405.

[16] BOWMAN C., DANZIG P., HARDY D., et al . *The Harvest Information Discovery and Access System.* Computer Networks and ISDN Systems, 1995, vol. 28, n°1–2, pp. 119–125 (or 119–126).

[17] BOWMAN C., DANZIG P., MANBER U., et al . *Scalable Internet Resource Discovery: Research Problems and Approaches.* Commun. ACM, 1994, vol. 37, n°8, pp. 98–ff.

[18] BRAY T., PAOLI J., SPERBERG-MCQUEEN C. *Extensible Markup Language (XML).* World Wide Web Journal, 1997, vol. 2, n°4, pp. 27-66.

[19] CAO P., IRANI S. *Cost-Aware WWW Proxy Caching Algorithms.* **In :** USENIX Symposium on Internet Technologies and Systems, 1997.

[20] CARDENAS Y., PIERSON J-M, BRUNIE L. *Service de Cache pour les Grilles de Calcul.* **In :** F. CLOPPET , J. PETITN. VINCENT Extraction des connaissances: état et perspectives, RNTI E-5 ISBN: 978-2-85428-707-3, chapter 6. Editions Cépaduès, 2005.

[21] CARDENAS Y., PIERSON J-M., BRUNIE L. *Uniform Distributed Cache Service for Grid Computing.* **In :** DEXA Workshops. 2005, IEEE Computer Society, pp 351–355.

[22] CARDENAS Y., PIERSON J-M., BRUNIE L. *Management of Cooperative Cache in Grids.* **In :** 2nd International Workshop on Data Management in Grids DMG VLDB 2006 32nd International Conference on Very Large Data Bases, VLDB Data Management in Grids Workshop 2006, sep 2006, pp 38–49.

[23] CARDENAS Y., PIERSON J-M., BRUNIE L. *Temporal Storage Space for Grids*. **In :** Second International Conference on High Performance Computing and Communications (HPCC 2006), Lecture Notes in Computer Science. sep 2006, Springer, pp 803–812.

[24] CARDENAS Y., PIERSON J-M., BRUNIE L. *Management of a Cooperative Cache in Grids with Grid Cache Services*. Concurrency and Computation:Practice and Experience, nov 2007, vol. 19, n°16, pp. 2141–2155.

[25] CARON E., DESPREZ F., NICOD J-M., et al . *A Scalable Approach to Network Enabled Servers*. **In :** Euro-Par '02: Proceedings of the 8th International Euro-Par Conference on Parallel Processing, 2002, London, UK. Springer-Verlag, pp 907–910.

[26] CENTRE EUROPEEN POUR LA RECHERCHE NUCLAIRE . *Cern Advanced Storage Manager (CASTOR)* [**on-line**]. Available from: `http://castor.web.cern.ch/castor` (webpage visited on 16.04.2007).

[27] CENTRE EUROPEEN POUR LA RECHERCHE NUCLAIRE . *LHC Computing Grid Project* [**on-line**]. Available from: `http://lcg.web.cern.ch/LCG` (webpage visited on 16.04.2007).

[28] CHANKHUNTHOD A., DANZIG P., NEERDAELS C., al . *A Hierarchical Internet Object Cache*. **In :** USENIX Annual Technical Conference, 1996, pp 153–164.

[29] CHERVENAK A., DEELMAN E., FOSTER I., et al . *Giggle: a framework for constructing scalable replica location services*. **In :** SC, 2002, pp 1–17.

[30] CHERVENAK A., FOSTER I., KESSELMAN C., et al . *The Data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets*. Journal of network and computer applications, 2000, vol. 23, n°3, pp. 187-200.

[31] CHERVENAK A., SCHULER R., KESSELMAN C., et al . *Wide Area Data Replication for Scientific Collaborations*. **In :** GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, 2005, Washington, DC, USA. IEEE Computer Society, pp 1–8.

[32] CHRISTENSEN E., CURBERA F., MEREDITH G., et al . *Web Services Description Language (WSDL)*. Technical report [**on-line**]. World Wide Web Consortium (W3C), 2001. Available from: `http://www.w3.org/TR/wsdl` (webpage visited on 11.08.2007).

[33] CIESLAK M., FORSTER D. *Web Cache Coordination Protocol V1.0* [**on-line**]. Available from: `http://www.wrec.org/drafts/draft-ietf-wrec-web-pro-00.txt` (webpage visited on 28.03.2007).

[34] CIESLAK M., FORSTER D., TIWAN G., et al . *Web Cache Coordination Protocol V2.0* [**on-line**]. Available from: `http://www.web-cache.com/Writings/Internet-Drafts` (webpage visited on 28.03.2007).

[35] COQUIL D. *Conception et Mise en Oeuvre de Proxies Sémantiques et Coopératifs*. PhD thesis. Villeurbanne, France : INSA de Lyon, March 2006, 208 p.

[36] COQUIL D., BRUNIE L., PIERSON J-M. *Semantic collaborative web caching*. **In :** Ling Tok Wang, Dayal Umeshwar, Bertino Elisa, Ng Wee Keong, Goh Angela 3rd International Conference on Web Information Systems Engineering (WISE 2002). 2002, IEEE Computer Society, pp 30–42.

[37] COQUIL D., BRUNIE L., SIMON S. *Software Architectures for Collaborative Proxies in Wide Area Information Systems*. **In :** Tjoa A. Min, Wagner Roland 12th International Workshop on Database and Expert Systems Applications (DEXA 2001). 2001, IEEE Computer Society, pp 146–150.

[38] DAIS-WG . *The OGSA-DAI Project* [**on-line**]. Available from: `http://www.ogsadai.org.uk` (webpage visited on 16.04.2007).

[39] DATAGRID PROJECT , GLOBUS ALLIANCE . *Replica Location Service (RLS)* [**on-line**]. Available from: `http://www.globus.org/toolkit/data/rls` (webpage visited on 16.04.2007).

[40] DEHNE F., LAWRENCE M. *Cooperative Caching for Grid Based DataWarehouses*. **In :** Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007.

[41] DEL-FABBRO B., NICOD J-M., LAIYMANI D., et al . *DTM: a service for managing data persistency and data replication in network-enabled server environments*. Concurr. Comput. : Pract. Exper., 2007, vol. 19, n°16, pp. 2125–2140.

[42] D'ORAZIO L., JOUANOT F., RONCANCIO C., et al . *Distributed Semantic Caching in Grid Middleware*. **In :** WAGNER R., REVELL N., PERNUL G. DEXA, volume 4653 of *Lecture Notes in Computer Science*. 2007, Springer, pp 162–171.

[43] D'ORAZIO L., VALENTIN O., RONCANCIO C., et al . *Services de cache et intergiciel pour grilles de données*. **In :** LAURENT D. 22èmes Journées Bases de Données Avancées, BDA 2006, Lille, 17-20 octobre 2006, Actes (Informal Proceedings), 2006.

[44] DUBOIS M., SCHEURICH C., BRIGGS F. *Synchronization, Coherence, and Event Ordering in Multiprocessors*. IEEE Computer, 1988, vol. 21, n°2, pp. 9-21.

[45] E.OTOO , OLKEN F., SHOSHANI A. *Disk cache replacement algorithm for storage resource managers in data grids*. **In :** Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, 2002, Los Alamitos, CA, USA. IEEE Computer Society Press, pp 1–15.

[46] ESCHMANN F., KLAUER B., MOORE R., et al . *SDAARC: An Extended Cache-Only Memory Architecture*. IEEE Micro, 2002, vol. 22, n°3, pp. 62-70.

[47] EUROPEAN COMMISSION . *Enabling Grids for E-science project* [**on-line**]. Available from: `http://www.eu-egee.org` (webpage visited on 22.10.2007).

[48] EUROPEAN DATAGRID PROJECT , PARTICLE PHYSICS DATA GRID PROJECT . *Grid Data Mirroring Package (GDMP)* [**on-line**]. Available from: `http://project-gdmp.web.cern.ch/project-gdmp` (webpage visited on 16.04.2007).

[49] EUROPEAN UNION . *The DataGrid Project* [**on-line**]. Available from: `http://eu-datagrid.web.cern.ch` (webpage visited on 22.10.2007).

[50] EXOLAB GROUP . *Castor Project* [**on-line**]. Available from: `http://www.castor.org` (webpage visited on 11.08.2007).

[51] FAN L., CAO P., ALMEIDA J., et al . *Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol*. IEEE/ACM Trans. Netw., 2000, vol. 8, n°3, pp. 281-293.

[52] FERMI NATIONAL ACCELERATOR LABORATORY . *Fermilab Mass Storage System (MSS)* [**on-line**]. Available from: `http://www-isd.fnal.gov/enstore` (webpage visited on 16.04.2007).

[53] FOSTER I. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. **In :** First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 15-18, 2001, Brisbane, Australia. 2001, IEEE Computer Society, pp 6–7.

[54] FOSTER I., KESSELMAN C. *The Globus project: a status report*. Future Generation Computer Systems, 1999, vol. 15, n°5–6, pp. 607–621.

[55] FOSTER I., KESSELMAN C., NICK J., et al . *Grid Services for Distributed System Integration*. IEEE Computer, 2002, vol. 35, n°6, pp. 37-46.

[56] FOSTER I., TUECKE S., UNGER J. *OGSA Data Services*. Technical report [**on-line**]. Global Grid Forum, 2003. Available from: `http://forge.ggf.org/projects/dais-wg` (webpage visited on 16.04.2007).

[57] GADDE S., CHASE J., RABINOVICH M. *A Taste of Crispy Squid*. **In :** Proceedings of the Workshop on Internet Server Performance (WISP'98), 1998.

[58] GADDE S., RABINOVICH M., CHASE J. *Reduce, Reuse, Recycle: An Approach to Building Large Internet Caches*. **In :** HOTOS '97: Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI), 1997, Washington, DC, USA. IEEE Computer Society, pp 93.

[59] GHARACHORLOO K., LENOSKI D., LAUDON J., et al . *Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors.* **In :** 25 Years ISCA: Retrospectives and Reprints, 1998, pp 376–387.

[60] GIBSON G., KLEIMAN S., SHEPLER S., et al . *High performance NFS: facts and fictions.* **In :** SC. 2006, ACM Press, pp 68.

[61] GLOBAL GRID FORUM . *Reliable File Transfer (RFT) Service* [**on-line**]. Available from: `http://www.globus.org/toolkit/docs/4.0/data/rft` (webpage visited on 11.08.2007).

[62] GLOBAL GRID FORUM AND ENTERPRISE GRID ALLIANCE . *Open Grid Forum* [**on-line**]. Available from: `http://www.gridforum.org` (webpage visited on 11.08.2007).

[63] GLOBUS ALLIANCE . *Globus Toolkit* [**on-line**]. Available from: `http://www.globus.org/toolkit` (webpage visited on 11.08.2007).

[64] GOLDSTEIN J., LARSON P. *Optimizing Queries Using Materialized Views: A practical scalable solution.* **In :** SIGMOD Conference, 2001, pp 331–342.

[65] GOODMAN J. *Cache Consistency and Sequential Consistency.* Technical Report 61, IEEE SCI Commitee, 1989.

[66] GOSSA J. *Modélisation et outils génériques pour la résolution des problémes liés a la répartition des ressources sur grilles.* PhD thesis. Villeurbanne, France : INSA de Lyon, December 2007, 218 p.

[67] GOSSA J., PIERSON J-M., BRUNIE L. *Dynamic Placement of Content Replica in Ditributed Multimedia System.* Technical Report RR-Liris-2005-004, LIRIS INSA Lyon, feb 2005.

[68] GRAY J., HELLAND P., O'NEIL P., et al . *The Dangers Of Replication And A Solution.* **In :** SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data, 1996, New York, NY, USA. ACM Press, pp 173–182.

[69] GRIPHYN . *The GriPhyN Project* [**on-line**]. Available from: `http://www.griphyn.org` (webpage visited on 16.04.2007).

[70] HIGH PERFORMANCE STORAGE SYSTEM . *(HPSS)* [**on-line**]. Available from: `http://www.hpss-collaboration.org/hpss` (webpage visited on 16.04.2007).

[71] HOSSEINI-KHAYAT S. *Replacement algorithms for object caching.* **In :** SAC '98: Proceedings of the 1998 ACM symposium on Applied Computing, 1998, New York, NY, USA. ACM Press, pp 90–97.

[72] IEEE WORKING GROUP ON BROADBAND WIRELESS ACCESS STANDARDS . *LMDS Local Multipoint Distribution Service* [**on-line**]. Available from: `http://grouper.ieee.org/groups/802/16` (webpage visited on 22.10.2007).

[73] INRIA . *The Relais project* [**on-line**]. Available from: `http://www-sor.inria.fr/projects/relais` (webpage visited on 26.03.2007).

[74] IVOA . *International Virtual Observatory Alliance* [**on-line**]. Available from: `http://www.ivoa.net` (webpage visited on 16.04.2007).

[75] IYER S., ROWSTRON A., DRUSCHEL P. *Squirrel: A Decentralized Peer-To-Peer Web Cache*. **In :** PODC, 2002, pp 213–222.

[76] JANG H., MIN K., JOU W. SEOK, et al . *A Path Based Internet Cache Design for GRID Application*. **In :** LI M., SUN X., DENG Q., et al GCC (2), volume 3033 of *Lecture Notes in Computer Science*. 2003, Springer, pp 455–458.

[77] KELLY T., JAMIN S., MACKIE-MASON J. *Variable QoS from shared Web caches: User-centered Design And Value-Sensitive Replacement*. **In :** In Proceedings of the MIT Workshop on Internet Service Quality Economics (ISQE 99), Cambridge, MA, 1999 [**on-line**]. Available from: `http://citeseer.ist.psu.edu/article/kelly99variable.html` (webpage visited on 29.03.2007).

[78] KISTLER J. *Disconnected Operation in a Distributed File System*, volume 1002 of *Lecture Notes in Computer Science*. Springer, 1995.

[79] KORUPOLU M., PLAXTON C., RAJARAMAN R. *Placement Algorithms for Hierarchical Cooperative Caching*. J. Algorithms, 2001, vol. 38, nº1, pp. 260–302.

[80] LAMPORT L. *How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs*. IEEE Trans. Computers, 1979, vol. 28, nº9, pp. 690-691.

[81] LAOUTARIS N., SYNTILA S., STAVRAKAKIS I. *Meta Algorithms for Hierarchical Web Caches*. **In :** Proceedings of the 2004 IEEE International Conference on Performance, Computing, and Communications. 2004, IEEE, pp 445 – 452.

[82] LILJA D. *Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons*. ACM Comput. Surv., 1993, vol. 25, nº3, pp. 303-338.

[83] LINDER H., CLAUSEN H., STERING W. *A Multi-Level, Multicast Web Caching System for Interconnected LMDS Networks* [**on-line**]. Available from: `http://citeseer.ist.psu.edu/554248.html` (webpage visited on 24.03.2007).

[84] LINGA P., GUPTA I., BIRMAN K. *Kache: Peer-to-Peer Web Caching Using Kelips* [**on-line**]. Available from: `http://www.cs.cornell.edu/projects/quicksilve` (webpage visited on 26.03.2007).

[85] LIP COMPUTER SCIENCE LABORATORY OF THE ENS LYON . *Diet Project* [**on-line**]. Available from: `http://graal.ens-lyon.fr/diet` (webpage visited on 11.11.2007).

[86] LIRIS , IRIT , LIFL . *GGM Project* [**on-line**]. Available from: `http://liris.cnrs.fr/projects/ggm` (webpage visited on 11.08.2007).

[87] LUO Q., KRISHNAMURTHY S., MOHAN C., et al . *Middle-tier database caching for e-business.* **In :** SIGMOD Conference, 2002, pp 600–611.

[88] MAKPANGOU M., PIERRE G., KHOURY C., et al . *Replicated Directory Service for Weakly Consistent Replicated Caches.* **In :** Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99), may 1999.

[89] MICHEL B., NGUYEN K., ROSENSTEIN A., et al . *Adaptive Web Caching: Towards a New Global Caching Architecture.* Computer Networks, 1998, vol. 30, n°22-23, pp. 2169-2177.

[90] OBJECT MANAGEMENT GROUP . *Common Object Request Broker Architecture (CORBA)* [**on-line**]. Available from: `http://www.omg.org` (webpage visited on 08.03.2007).

[91] OPEN GRID FORUM . *Info Dissemination Working Group (INFOD-WG)* [**on-line**]. Available from: `https://forge.gridforum.org/sf/projects/infod-wg` (webpage visited on 16.04.2007).

[92] PEARLMAN L., C.KESSELMAN , GULLAPALLI S., et al . *Distributed Hybrid Earthquake Engineering Experiments: Experiences with a Ground-Shaking Grid Application.* **In :** HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04), 2004, Washington, DC, USA. IEEE Computer Society, pp 14–23.

[93] PETERSEN K., SPREITZER M., TERRY D., et al . *Flexible Update Propagation For Weakly Consistent Replication.* **In :** SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles, 1997, New York, NY, USA. ACM Press, pp 288–301.

[94] PIERSON J-M, GOSSA J., CARDENAS Y., et al . *GGM Efficient Navigation and Mining in Distributed Geno-Medical Data.* IEEE Transactions on NanoBioscience, may 2007, vol. 6, n°2, pp. 110–116.

[95] PODLIPNIG S., LASZLO B. *A Survey Of Web Cache Replacement Strategies.* ACM Comput. Surv., 2003, vol. 35, n°4, pp. 374–398.

[96] POVEY D., HARRISON J. *A distributed Internet Cache.* **In :** Proceedings of the 20th Australian Computer Science Conference, Sydney, Australia, 1997.

[97] PPDG . *PARTICLE PHYSICS DATA GRID COLLABORATORY PILOT* [**on-line**]. Available from: `http://www.ppdg.net` (webpage visited on 16.04.2007).

[98] PRZYBYLSKI S. *The Performance Impact of Block Sizes and Fetch Strategies*. **In :** ISCA, 1990, pp 160–169.

[99] RABINOVICH M., CHASE J., GADDE S. *Not All Hits Are Created Equal: Cooperative Proxy Caching Over A Wide-Area Network*. Computer Networks and ISDN Systems, 1998, vol. 30, n°22–23, pp. 2253–2259.

[100] RAMACHANDRAN U., SHAH G., KUMAR R., et al . *Scalability Study of the KSR-1*. Parallel Computing, 1996, vol. 22, n°5, pp. 739-759.

[101] ROSS K. *Hash-Routing for Collections of Shared Web Caches*. IEEE Network Magazine, 1997.

[102] ROUSSKOV A., WESSELS D. *Cache digests*. Computer Networks and = ISDN Systems, 1998, vol. 30, n°22–23, pp. 2155–2168.

[103] SAN DIEGO SUPERCOMPUTER CENTER . *Storage Resource Broker (SRB)* [**on-line**]. Available from: `http://www.sdsc.edu/srb` (webpage visited on 16.04.2007).

[104] SATYANARAYANAN M. *Scalable, Secure, and Highly Available Distributed File Access*. IEEE Computer, 1990, vol. 23, n°5, pp. 9-21.

[105] SEITZ L. *Design and Implementation of Secure Mechanisms for Sharing Confidential Data; Application to the Management of Biomedical Data in a Grid Computing Environment*. PhD thesis. Villeurbanne, France : INSA de Lyon, July 2005, 180 p.

[106] SEITZ L., MONTAGNAT J., PIERSON J-M., et al . *Authentication and Authorization Prototype on the grid for Medical Data Managements*. **In :** Press IOS Confrence Healthgrid 2005, Technology and Informatics, apr 2005, pp 222–233.

[107] SEITZ L., PIERSON J-M, BRUNIE L. *Encrypted Storage of Medical Data on a Grid*. Methods of Information in Medicine, jan 2005, vol. 44, n°2, pp. 198–202.

[108] SHOSHANI A., SIM A., GU J. *Storage Resource Managers: Middleware Components for Grid Storage*. **In :** In Nineteenth IEEE Symposium on Mass Storage Systems (MSS'02), 2002, pp 209–223.

[109] SKILLICORN D. *The Case for Datacentric Grids*. **In :** IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium, 2002, Washington, DC, USA. IEEE Computer Society, pp 313.

[110] SMITH A. *Cache Memories*. ACM Computer Survey, 1982, vol. 14, n°3, pp. 473-530.

[111] SQUID PROJECT . *Squid Web Proxy Cache* [**on-line**]. Available from: `http://www.squid-cache.org` (webpage visited on 26.03.2007).

[112] STANFORD LINEAR ACCELERATOR CENTER . *The Babar Detector* [**on-line**]. Available from: `http://www-public.slac.stanford.edu` (webpage visited on 16.04.2007).

[113] STOCKINGER H., SAMAR A., HOLTMAN K., et al . *File and Object Replication in Data Grids*. **In :** 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 2001), San Francisco, CA, USA. 2001, IEEE Computer Society, pp 76–86.

[114] STORAGE RESOURCE MANAGEMENT WORKING GROUP . *Storage Resource Manager* [**on-line**]. Available from: `http://sdm.lbl.gov/srm-wg` (webpage visited on 16.04.2007).

[115] TANENBAUM A. *Structured Computer Organization*, chapter Parallel Computer Architectures. Prentice Hall, Upper Saddle River, 5 edition edition, 2005.

[116] TANENBAUM A., STEEN M. VAN. *Distributed systems principles and paradigms*. Prentice Hall, New Jersey, 2002.

[117] TEWARI R., DAHLIN M., VIN H., et al . *Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet*. Technical Report TR98-0, UTCS, 1998.

[118] TIERNEY B., JOHNSTON W., LEE J., et al . *A data intensive distributed computing architecture for grid applications*. Future Gener. Comput. Syst., 2000, vol. 16, n°5, pp. 473–481.

[119] TOUCH J., HUGHES A. *LSAM Proxy Cache: A Multicast Distributed Virtual Cache*. Computer Networks, 1998, vol. 30, n°22-23, pp. 2245-2252.

[120] UNIVERSITY OXFORD. *eDiaMoND grid computing project* [**on-line**]. Available from: `http://www.ediamond.ox.ac.uk` (webpage visited on 16.04.2007).

[121] VAKALI A. *LRU-based Algorithms for Web Cache Replacement*. **In :** EC-Web, 2000, pp 409–418.

[122] VALLOPPILLIL V., ROSS K. *Cache Array Routing Protocol v1.0* [**on-line**]. Available from: `http://icp.ircache.net/carp.txt` (webpage visited on 28.03.2007).

[123] VENUGOPAL S., BUYYA R., RAMAMOHANARAO K. *A taxonomy of Data Grids for distributed data sharing, management, and processing*. ACM Comput. Surv., 2006, vol. 38, n°1, pp. 3.

[124] WALLACH D., DALLY. W. *A Hierarchical Protocol for Shared memory on a Distributed Memory Machine*. **In :** International Symposium on Shared Memory Multiprocessing, 1991.

[125] WANG X., NG W., OOI B., et al . *BuddyWeb: A P2P-based Collaborative Web Caching System*. **In :** Web Engineering and Peer-to-Peer Computing: Networking 2002 Workshops, may 2002, Pisa, Italy. pp 247–251.

[126] WESSELS D., CLAFFY K. *Application of Internet Cache Protocol (ICP), version 2*, 1997.

[127] WESSELS D., CLAFFY K. *Internet Cache Protocol (ICP), version 2*, 1997.

[128] WESSELS D., CLAFFY K. *ICP and the Squid Web Cache*. IEEE Journal on Selected Areas in Communication, 1998, vol. 16, n°3, pp. 345–357.

[129] WIESMANN M. *Comparison of Database Replication Techniques Based on Total Order Broadcast*. IEEE Transactions on Knowledge and Data Engineering, 2005, vol. 17, n°4, pp. 551–566. Member-Andre Schiper.

[130] WILKES M. *Slave memories and dynamic storage allocation*. IEEE Transactions on Electronic Computers, 1965, vol. EC-14, n°2, pp. 270-271.

[131] WOLSKI R., SPRING N.T., HAYES J. *The network weather service: a distributed resource performance forecasting service for metacomputing*. Future Generation Computer Systems, 1999, vol. 15, n°5–6, pp. 757–768.

[132] WORLD WIDE WEB CONSORTIUM . *Simple Object Access Protocol (SOAP)* [**on-line**]. Available from: `http://www.w3.org/TR/soap` (webpage visited on 11.08.2007).

[133] WORLD WIDE WEB CONSORTIUM . *Web Services Architecture* [**on-line**]. Available from: `http://www.w3.org/TR/ws-arch` (webpage visited on 11.08.2007).

[134] XU Z., SOHONI S., MIN R., et al . *An Analysis of Cache Performance of Multimedia Applications*. IEEE Trans. Computers, 2004, vol. 53, n°1, pp. 20-38.

[135] ZHANG J., IZMAILOV R., REININGER D., et al . *Web Caching Framework: Analytical Models and Beyond*. **In :** WIAPP '99: Proceedings of the 1999 IEEE Workshop on Internet Applications, 1999, Washington, DC, USA. IEEE Computer Society, pp 132.

[136] ZHANG L., FLOYD S., JACOBSON V. *Adaptive Web Caching* [**on-line**]. Available from: `http://citeseer.ist.psu.edu/zhang97adaptive.html` (webpage visited on 22.03.2007).