DYNAMIC PLACEMENT OF CONTENT REPLICAS IN DISTRIBUTED MULTIMEDIA SYSTEMS

Julien Gossa, Lionel Brunie, Jean-Marc Pierson
LIRIS, INSA de LYON
Firstname.Name@liris.cnrs.fr

ABSTRACT

We present in this article a multimedia content replica management system, able to satisfy users' constraints and preferences, while limiting the load of the servers and the proxy-caches. Our goal is to optimize the placement of a limited number of replicas via mechanisms of duplication and migration. Our system allows owners to keep control over their multimedia contents even with large scale replica dissemination. We developed an advanced model for integrating multiple metrics on the execution context to better suit the dynamicity inherent to distributed applications. Experiments on simulated networks of proxy-caches validate the approach.

1. INTRODUCTION

An efficient way to improve the performances of distributed multimedia systems has long been to use proxies in order to cache the content retrieved from the sources. In this approach, data are handled on specialized nodes in the network, where they are kept for a certain amount of time. Different mechanisms exist to decide which content is to be kept and for how long (data replacement policies range from the usage of the data, their size, their difficulty to be obtained, the interest of users in similar content and so on), where to keep it (close to the server, close to the client, somewhere in the middle). Some works have investigated the benefit of a collaborative decision, but few have been interested by the dynamic placement of replicas in the network of proxy-caches.

The current trend in distributed multimedia systems is to tackle mobile environment constraints : contents must be delivered any time, anywhere, and to anyone. This imposes to be able to store contents as close as possible to the users. Future distributed multimedia systems need new paradigms to be developed: the dynamicity and the unpredictability of either the users or the infrastructure must find an efficient solution.

Our goal is to ensure within a light infrastructure, the best quality of service for the client, while still maintaining a control on the dynamic number of replicas. Given the fact that multiple replicas of data may exist on the network, questions like the following arise : which one to select to answer a query ? when should we create a new replica ? where shall we create it ? where shall we move it with respect to the users' usage ? how to monitor this usage and control the replica use ?

The first problem to answer a query is the selection of one replica, which can be decomposed in two parts : which is the best metric for comparing two replicas, and how shall we compute it ? While the latter seems easier to tackle and more like a technical problem, the choice of the metric is not that simple. It must take into account the content of the data itself and its usage : Indeed, when the latency is the first parameter to look after for a small file, the bandwidth will be one of the key criteria for a larger file. Many works limit their view on some uniform measure and do not handle multiple criteria for their choice.

The second problem is the placement, the monitoring and the control of the replica. A solution consists in copying all or a subset of data at many places in the network (it is the policy implemented by Akamaï [1] for instance). The number of replicas is controlled by heavy mechanisms and the pertinence of the number of deployed replicas is not addressed : the system may copy in excess a piece of data, whereas the cost to obtain it is less than the cost to maintain a replica on one site. Other solutions lead to distribute the content to well-known clients, limiting *de facto* extensibility.

In mobile environments, none of these solutions adapts well : their responsiveness to the ever changing and moving population of potential clients (geographic mobility but also interest mobility) is limited, as well is their awareness of the network dynamicity (prediction tools to predict the state of the network are rarely used). Moreover, the sensitive contents are often not kept in caches, since a piece of data is sensitive whenever its owner wants it to be. A mechanism to allow at owner's choice for caching such content (thus controlling the use of the replica) must be proposed.

We present in this article a replica management architecture, able to satisfy the client constraints and wills, limiting the workloads on the servers and the proxy caches, and that can adapt to the ever evolving environment. We propose a distributed algorithm able to dynamically keep the replicas close to their usage, based on parameters for measuring and aggregating multiple metrics and to manage the life cycle of the replicas.

The rest of this article is organized as follows: First, we briefly present the Small world theory and discuss the state of the art. Then we study our proposal (for the selection and the placement of replicas) before presenting experiments. We discuss pros and cons of the proposed algorithms. We finally conclude and open perspectives for this work.

2. STATE OF THE ART

2.1 The Small Worlds theory

Many studies [9][10] shows that the Internet is made up of "small worlds", i.e. the very different users can be gathered in small communities sharing common interest. Furthermore these communities often correspond to physical clusters having good internal communications and few connections towards other clusters.

This can be related to local interest and fashion phenomena. Indeed, users from a scientific campus will not be interested by the same documents as those from literature one. Another example is the shifted release of a U.S. movie in France: the French will be interested in its trailer then the Americans have already forgotten it. Moreover an album from a small local group will get a stronger interest within its area of origin than elsewhere.

According to [9], the small world phenomenon opens great algorithmic perspectives in networking and distributed systems. We decide to support it in the following approach with the observation that contents are not as much useful anywhere and anytime, so we could optimize their placement.

2.2 Managements of replicas

Positioning content replicas on network is a key factor in data management. A good strategy allows a good customer QoS, while limiting the resource load. *A contrario*, an ineffective management can be catastrophic when the volume of contents or users scales up.

[2][3] describe replica positioning methods in pervasive environment to optimize inner mechanisms, like adaptation path, but do not consider users needs.

[5][7] deal with the optimization of replica selection to answer a user request, but do not integrate replica placement.

Works [6][8] study the server replicas placement, but not data placement over the servers.

We did not find any work handling the optimization of data placement for customers' requests in pervasive environment, probably because this problem is quite new, emerging with mobile

data processing. And because systematic management was sufficient for traditional cabled network of computers with great communication and storing capacities.
The existing strategies for collaborative systems either leave the replication free for non sensitive contents, or prohibit it to keep a total control over the sensitive ones.
We are convinced that the key factors in mobile computing are flexibility and dynamicity. Those are covered and synthesized in a work in progress [4].
While a majority of works are based either on more or less elaborated but static criteria: kilometric distances, quantity of hops, bandwidth… or on dynamic ones like network state… They do not consider replica nature and use context, what we call flexibility. Authors start from the observation that it is impossible to select an *a priori* sure uniform criterion for effective replicas management.
The concept of contract makes it possible to define the relevant criteria for a certain replica management and to negotiate its replication on proxy-caches.

3. FLEXIBLE AND DYNAMIC PLACEMENT OF REPLICAS

In this paper, we study the problem of the replicas placement over a network of proxy-caches. Our goals are to optimize the network use, to allow keeping a good control on the replicas and to ensure a good user Quality of Service. Our priority is to deal with a great dynamicity of users, network and contents. The work presented here follows a position paper [12].

3.1 Assumptions

We consider that a logical network of proxy-caches exists (for instance [15]). This represents the middleware allowing nodes to know their direct neighbours, share messages or files and store replicas. We will largely support above this middleware, assuming that it is rather open to allow our work to run.
We mask end-users of data behind their proper mandatory proxy, which forward their requests.
We call replica a copy of a content and request an end-user request for these contents.

We consider moreover than this network is working according to the small worlds theory.

3.2 Approach

The proxy-caches have only local partial information on the topology and the network state. We deal with a very strong dynamicity and a constant need for readjustment within a framework of unpredictability use. Moreover it is impossible to consider one uniform criterion.

We believe that few well managed replicas are more effective than a lot not easily controllable. By obtaining competitive results in term of user QoS with a limited quantity of replicas, we will be able to allow an inexpensive advanced management: to know where all the replicas are, which their uses are, to update them, to easily maintain their consistency and safety, while allowing a real load balancing.

Our approach is based on replicas which are able to know where to position, when to reproduce or commit suicide and to adapt to new needs thanks to simple interactions with their direct environment.

3.3 The replica (re)placement problem and the K-server online algorithm

3.3.1 Original centralized algorithm – the DC-Tree (DoubleCoverture-Tree)

Our replica placement and selection strategy is derived from the on-line approximation algorithm described in [16]. It presents a solution for the k-servers problem, which is formalized as follow:

Let k be the number of mobile servers over a N nodes tree. Requests appear successively anywhere on this tree. A request is served when one of the servers had move up to its position. The total cost of the solution is the sum of the distances covered by the individual servers.

The DC-Tree algorithm controls the k servers positions in order to answer in a competitive time without knowing the request sequence in advance. This algorithm is depicted as follows: "At each time, all the servers neighbouring the request are moving in a constant speed toward the request" [11]

The main idea is that requests influence the positions of all its surrounding servers, and not only the one selected to answer.

Request neighbourhood is defined as the first met servers while following the tree starting from the request.

When answering a request, while the replicas are moving, a replica may overtake another one. This latter is then excluded from the request neighbourhood and must stop moving. Thus, all replicas will not cover the same distance.

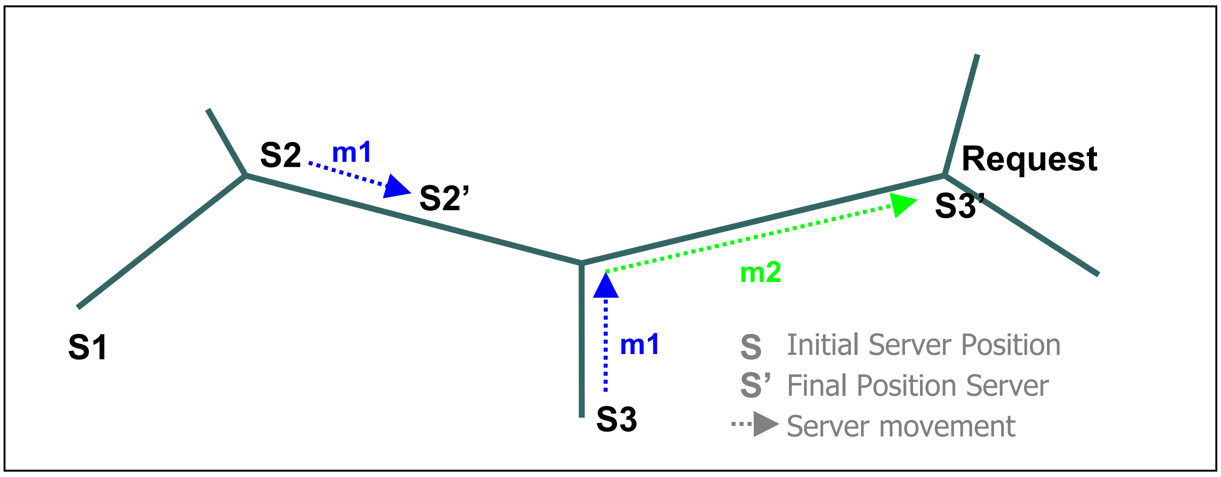


Figure 1: Response to a Request

On the figure 1, to serve the request, each server in the request neighbourhood will move at constant and equal speed towards the request.
Initial step: S1 is located behind S2 on the way towards the request, it is thus out of neighbourhood and will not move. S2 and S3 move towards the request (m1).
End of move 1: S3 passes in front of S2 because of its movement. S2 is then excluded from the neighbourhood and must be stopped at S2'.
S3 will be able to continue (m2) towards the request position.

This method has been proved (N-1)k-competitive. This means that its solution in the worst case is (N-1)k heavier than the optimal one (the optimal could be found with advance knowledge of the request sequence). Fortunately, it works much better in the general case.
Actually, the Small Worlds theory ensures very competitive results in real situation, since the request sequence is not random, but rather presents focuses in different spots.

3.3.2 Adapted reformulation for replica placement.

We can formalize our problem as follows:
Let consider k replicas of the same content on a proxy-caches network. We want to optimize dynamically their position so as to minimize the total distance between these replicas and the whole requests.

The DC-Tree is well adapted to our problem thanks to its efficiency when confronted to dynamicity and unpredictability. Thus we reformulate the original algorithm as: "At each time, all the *replicas* neighbouring the request are moving in a constant speed toward the request".

In the original algorithm, the servers move to the request to serve it. In our adaptation, a replica do not move effectively to the proxy-cache requesting it. Selected replica to serve a request is simply the nearest.

3.3.3 Adaptation to distributed environment: Distribution and Centralization

The original algorithm works in an entirely centralized environment. Complete topology (distances between node, positions of replicas and requests…) is known and used for each decision. Moreover this topology is static.

In network context, one node knows only its local inter-connexion graph and must take decision independently. Dynamicity and flexibility due to instability of network state and usage disparity must also be addressed correctly.

Our architecture is divided in two distinct parts: a completely distributed autonomous one and an optional centralized one.

The distributed part concerns the vital functionalities, therefore the adaptation of the algorithm described above. These functionalities concerns the evaluation of network state and replica placement/selection. Each proxy-cache interacts autonomously with others : Thus it benefits from classical Peer to Peer properties and remains flexible, adaptive, robust and scalable.

The centralized part relates to the monitoring servers. They support two independent functions: first they ensure the control of replicas through duplication/suicide authorisation; second they gather information on replicas life cycle through usage monitoring. The monitoring servers are administrated by each contents owner according to their needs. Actually, these functionalities levels are independently configurable. For example an owner could allow duplication and record each access.

3.4 Realization: implementation of the distributed part

In the original algorithm, servers "move at constant speed" on the network. This concept of "constant speed", understood in the algorithm like a Euclidean speed, is not at all transposable to network context. Moreover, in the original algorithm, replicas have to stop in the middle of connections between two nodes, which once more is unrealizable. From the example of figure 1, difficulties are to detect that S1 is out of neighbourhood and to represent the fact that S2 "stops" in the middle of the connection.

We transpose the concept of "constant speed" towards the concept of "distance"[1], which keeps sense on network. We virtualize movement according to these distances. We will see how we materialize distances in the section 3.5.

We separate replicas movement in two distinct functions:

- Updating virtual positions – attraction vector updating
- Concretization of these positions – attraction vectors maintenance.

3.4.1 Updating Virtual positions – attraction vector updating

Each proxy-cache maintains a virtual position of each of its managed replicas, according to its local knowledge of topology. We call "link" a physical link between two neighbour proxy-caches. We call "attraction" the value of the distance to the virtual position towards a particular link. We call "attraction vector" of a particular replica the vector containing the value of attraction for each link of a proxy-cache, plus the local attraction. This vector represents the virtual position of the replica. Initially it is null, then it evolves in time according to requests positions. An attraction towards a link is increased when the replica is requested on this link. The added value corresponds to the covered distance envisaged in the original algorithm, multiplied by a parametric factor DISTANCE2ATTRACTION.

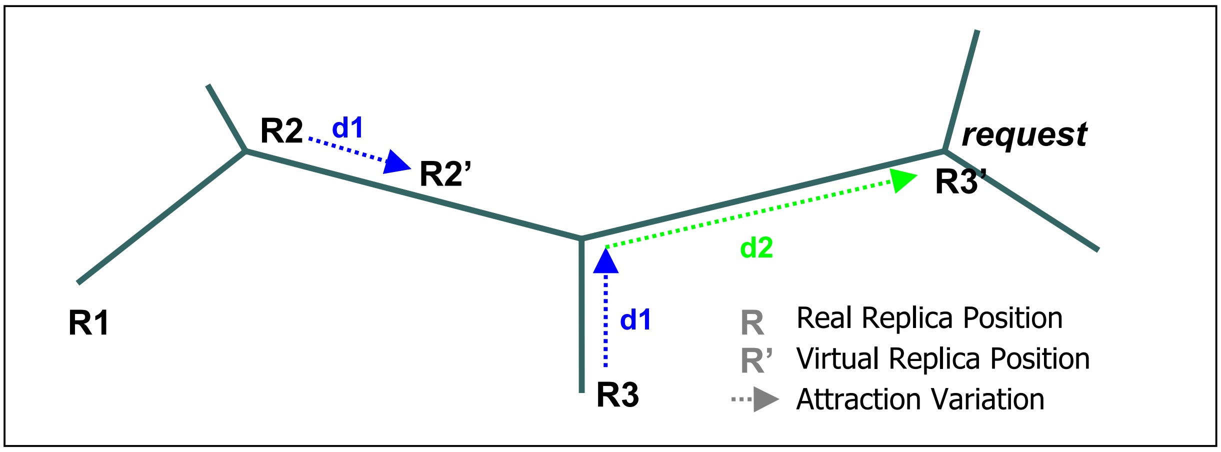


Figure 2 : example of attraction variation due to request trigger

[1] Note here that "distance" might be understood in an intuitive way, like remoteness for instance, and not as a mathematical metric distance (our "distance" lacks properties of metric distance like triangular inequality and symmetry).

The example of figure 2 shows attraction variation and virtual position, according to fig 1 instance.
R1 is out of neighbourhood, so it will not have any added attraction.
R2 having to virtually move distance d1, this one will be added to its attraction towards the corresponding link.
R3 will have an added attraction of the total distance d1+d2.

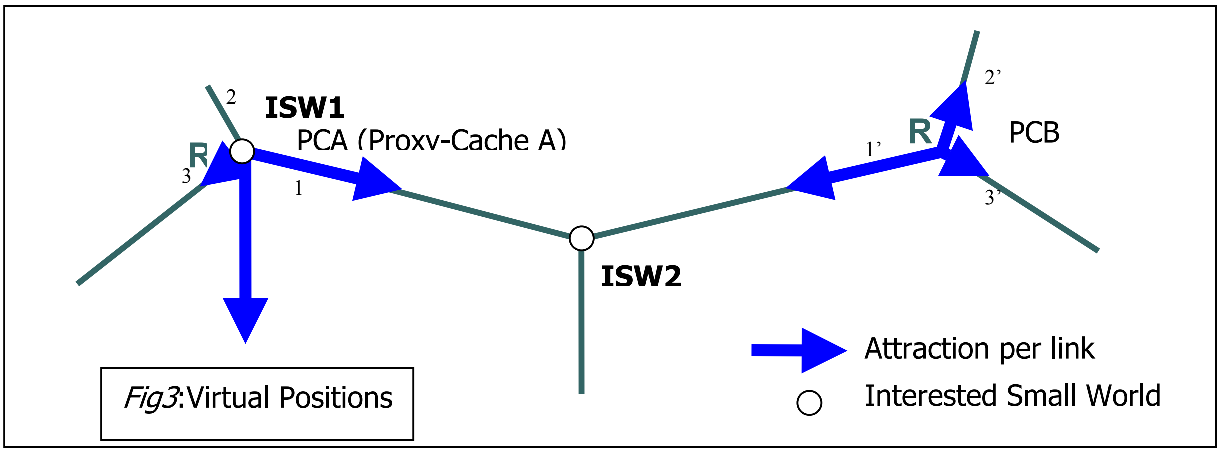


Figure 3 : Example of attraction vectors at a given time with two Interested Small Worlds

We call Interested Small World (ISW), a physical cluster whose actors are currently interested by the replicated content.

Attraction Vector PCA	
self	7
Link 1	5
Link 2	0
Link 3	1

For Proxy-Cache A and replica R, one notes:

- a strong attraction towards itself, since it contains ISW1
- a strong attraction towards link 1, leading to ISW2
- a light attraction towards link 3, leading to non represented ISWs
- no attraction towards link 2, leading to no user

Attraction Vector PCB	
self	0
Link 1'	5
Link 2'	2
Link 3'	1

For the Proxy-Cache B and replica R, one notes:

- a strong attraction towards the link 1', leading to ISW2
- a light attraction towards the links 2' and 3' leading to other ISW
- no local attraction

In order to update this attraction vectors, we describe an inter proxy-cache protocol.

Inter proxy-cache NCasting protocol.

This protocol allows maintaining the state of the attraction vectors. We call this mechanism NCasting (for NeighbourCasting).

```
On Reception of NCasting(Replica, ReceivedDistance) by the proxy-cache PCre from PCor

If Replica owned by PCre
    add ReceivedDistance for the link to PCor to Replica attraction vector
Else
    let PCpp be the nearest proxy-cache owning Replica
    send NCasting(Replica, ReceivedDistance) on the driving link to PCpp
    send NCasting(Replica, (PCre-PCpp)Distance) to all neighbours except of PCpp and PCor
```

Figure 5 : pseudo code of NCasting protocol

When a proxy-cache receives an end-user request, it redirects it towards the nearest replica. Then it initialize update of different concerned attraction vectors with a simple flooding of NCasting(Replica, distance to nearest Replica).

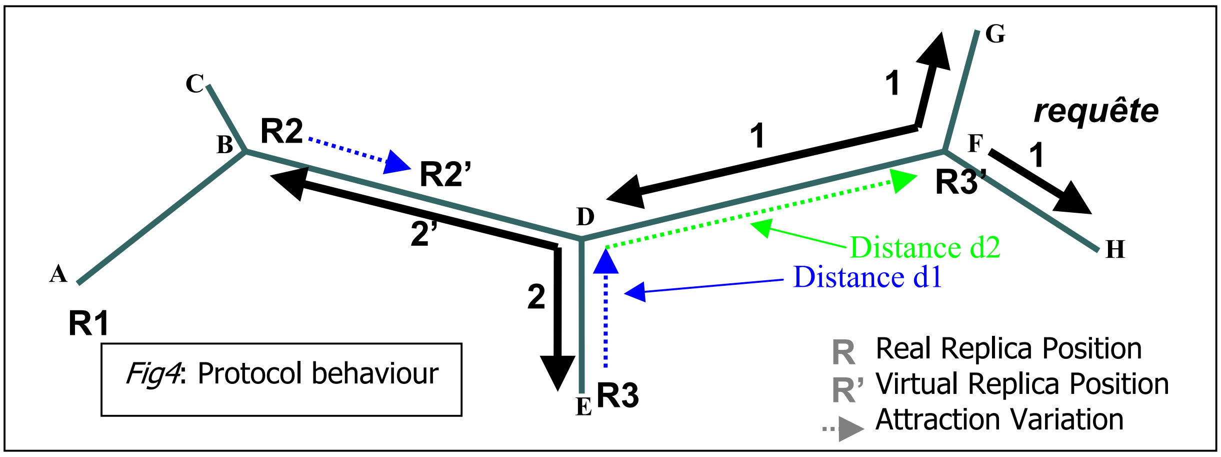


Figure 6 : Example of the behaviour of the protocol

- init: F requires contents R, the distance to the nearest replica of R (R3) is D1+D2.
    - 1: flood NCasting(R, d1+d2) starting from F
- D receives NCasting(R, d1+d2) and doesn't own the replica, so it sends:
    - 2: NCasting(R, d1+d2) towards E, which owns the nearest replica.
    - 2': NCasting(R, d1) towards B, D1 being the distance to the nearest replica
- E receives NCasting(R, d1+d2) and owns R:
    - it increments the attraction vector towards D by d1+d2
- B receives NCasting(R, d1) and own R:

- it adds e1 towards D to its attraction vector

Communication complexity of the protocol in terms of message quantity

Let consider a tree of N nodes.

The worst case occurs when replicas are positioned on each leave of the tree and requests are on any other node. So, the request's neighbourhood is the complete tree. NCasting messages will be sent to all nodes. Which means a complexity of O(N).

Actually the request's neighbourhoods are often less than the complete graph because of the restraint number of replica. Moreover in small world context, as replicas move towards the requests, neighbourhoods are progressively closing, and then message complexity is progressively reducing.

Moreover, if we dispose of a good replica indexing (allowed by a powerful middleware), we can limit the number of messages by contacting interested proxy-cache directly by multicast.

Another optimization is to accumulate a certain number of NCastings for a same replica without triggering the rest of the system, and then send one only NCasting with the sum of accumulated distances.

This protocol can also work on any graph by adding loop control.

3.4.2 Concretizations of virtual positions – attraction vectors maintenance.

We call "concretization" the materialization of replicas virtual positions. It triggers the operations of migration, duplication or suicide.

The concretization is performed during the attraction vectors maintenance. This operation is regularly run on each proxy-cache for each of its stored replica every time step defined by MAINTAINANCE_TIMESTEP.

Migration

For migration, concretization happens when attraction vector exceeds distance to the linked proxy-cache.

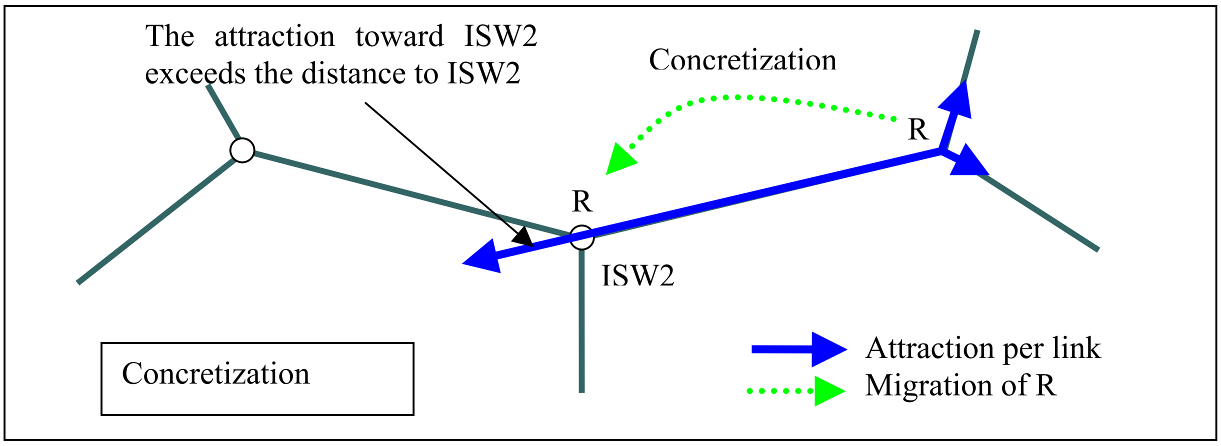


Figure 7 : Example of Concretization situation

On figure 7, one can see that the attraction of R towards ISW2 exceeds the distance to ISW2, so the concretization will migrate R to ISW2.

Duplication

The original algorithm drives a fixed number of servers.
With flexibility and dynamicity needs, we have wanted the number of replicas can evolve automatically, while remaining under supervisor control.

Our architecture places the duplications/suicide authorization service on the monitoring servers. Those are centralized and administered by each replicas owner. These last select the monitoring level for its own replicas trough a metadata DUPLICATION_AUTHORISATION which stipulates explicitly if duplication is authorized/prohibited/controlled. In case of controlled duplication, the monitoring server knows the exact quantity of replicas, but the duplication is impossible in case of breakdown.

Actually, the attraction vector represents the directional popularity of replica. Thus many significant values mean that the replica is useful in the many corresponding directions.

- If (DUPLICATION_AUTHORISATION=authorized) OR
  (DUPLICATION_AUTHORISATION=controlled AND authorization is delivered)
  - The replica is duplicated on the neighbour proxy-caches indicated by the attraction.
- If (DUPLICATION_AUTHORISATION=prohibited) OR
  (DUPLICATION_AUTHORISATION=controlled AND no authorization is delivered)

- The replica cannot be copied and tries to migrate (which don't change the number of replicas) to find a better position in function of the attraction.

Suppression

We deal with episodically isolated request out of interested small world, which could be considered as noise. Moreover we must slowly forget past demands in order to consider new ones. So, we need to reduce the attraction values with time.

During the maintenance operation, we decrement the attraction vector values according to a parameter ATTRACTION_DEGRADATION.

Replicas must be deleted when its attraction vector remains null for a parametric period TIMETOSUICIDE, which means it keeps useless for this period.

Maintenance mechanism

This algorithm is regularly run by the proxy-caches for each of its replicas. It decides duplications/migrations toward attractive link and suicide of useless replicas.

```
For a given replica R, let:
  V its attraction vector;
  L the list of links which attraction exceeds the distance to linked proxy-cache,
    sorted by descending order of attraction;
  NDA the amount of authorized duplications, case of DUPLICATION_AUTHORISATION
    if authorized:∞; if prohibited:0; if controlled: asked to the monitoring server | 0 if unreachable
  TTS time before suicide

For i = 0 to min(NDA-1, size(L))
    If L[i]!=self
        Duplicate R towards L[i]
        Update the attraction vector to 0 for L[i]
If L[NDA ]!=self
    Migrate R towards L[NDA]

Update V: V=V-ATTRACTION_DEGRADATION
If V==null
    TTS=TTS-MAINTAINANCE_TIMESTEP
Else reset TTS : TTS=TIMETOSUICIDE
If TTS==0
    Delete R
```

Figure 8 : pseudo code of maintenance algorithm

3.5 Metrics management

As seen before, our algorithms rely on distance computation on network. Actually, attraction has the same nature as distances between two nodes.

Unfortunately, distances computation over the network is not trivial and the context constraints imply that we cannot choose one uniform metric, such as the latency or the kilometric distance.

So, we propose to use an advanced metric model in order to ensure a flexible and dynamic distance computation.

3.5.1 Metric Modelling

A metric, *M*, can be either dynamic, when associated to the network state, or static when related to resources capacities. Dynamic metrics need to be evaluated in real time, whereas static ones are declared.

The value of the individual computation of a metric M_i is noted $Measure_i$.

The relevance of a metric is strongly related to the replica characteristics and its use. This relevance is modelled by a coefficient α, between 0 and 1, defined for each metric M_i and each class of replica *R*. We will see example of coefficients in section 3.5.5.

3.5.2 Temporal aggregation

We consider small variability of network, like ephemera breakdown, as noise. Moreover certain metrics are expensive to measure, so we want to avoid remaking all measurements for each response.

So, we aggregate the measurements according to time. This aggregation is done either by sampling over a relative period (X last minutes), or by sampling over static absolute periods (hour/day/…), which can prove relatively heavy, but rather close to the customers QoS constraints.

3.5.3 Node to node route distances computation.

This computation is made by a simple weighted average of normalized measurements of all metrics.

In order to incorporate metrics of different nature (for example a latency in second and a distance in kilometre), normalization functions are used. They are declared together with associated metrics. These normalization functions stand from metric measurement space towards R [0..1].

$$Measure_{route}^{replica} = Average\left(\alpha_i^{replica} \times fn_i\left(Measure_i\right)\right)$$, where

$Measure_{route}^{replica}$	is the distance of *route* for *replica*, between 0 and 1
$Measure_i$	is the temporal aggregation of metric i on *route*
fn_i	is the normalization function for metric i
$\alpha_i^{replica}$	is the weight of metric i according to the class of *replica*.

We can evaluate and compare network distances in a flexible way whatever the number of aggregated metrics is.
The beauty of this approach is that final customers will consider overloaded proxy-caches farthest than underused ones. As the customers are redirected to the "nearest replica", we ensure final customers QoS while allowing real load balancing, because the users will more requisition underused proxy-caches than overloaded ones.

3.5.4 Metrics Declaration

We want to remain as open as possible. So, we allow content providers to integrate new metrics. A metric is described by four characteristics:

- Its name
- Its individual measurement tool (like "ping")
- Its temporal aggregation function
- Its normalization function

This description largely depends on the underlying middleware. We will not describe it more in detail. Nevertheless, we initially identify the metrics which appear *a priori* adequate.

On a route	
Dynamic	Static
latency (RTT)	
available band-width	ideal band-width (theoretical)
On a resource (Proxy-Cache)	
Dynamic	Static
system load (in term of CPU)	system capacity
storage load	storage capacity.
host availability time ratio	

Figure 10 : example of adequate metrics

3.5.5 Choice of weighting

The choice of metrics weights according to replica nature occupies a significant place in the effectiveness of our system. First we declare basic weightings related to replica characteristics. For example, a small file will strongly weight latency, system load and availability ratio in order to privilege reactive hosts. *A contrario*, a heavy video will weight available band-width, load and capacity storage, in order to avoid overloads. These two different files categories correspond to two different replicas classes describing these needs. Thereafter, each content owner will refine theses basic weightings according to its specific needs.

3.5.6 Replica Classes

We set up a hierarchical system of replica classes.
A replica class maps replica, according to its characteristics called intrinsic metadata, with a set of parameters useful to its management, called exploitation metadata.
These classes are derivable to allow refining the choices by describing more and more precisely the weights to be used. To find the membership class of a replica, we simply have to descend this hierarchy.

Intrinsic Metadata	Description
Name	The name of the cached file
Type	Its type (extension)
Size	Its size in byte
Date	Its date of last update
Owner	Its initial owner
Exploitation Metadata	**Description**
$\alpha_0 \ldots \alpha_1$	set of weight
Temporal aggregation	absolute | relative
Monitoring server IP	IP of the server controlling the replica use
Duplication authorization	False | True | Controlled
Times before suppression	Time without attraction variation before suppression

Figure 11 : Description of replicas metadata

The creation of a new class is announced by flooding all the proxy-caches in order to keep consistency. This operation should be only specific and carried out with the objective of managing a great amount of replicas. It should not thus involve network overload.

These classes are defined in the portable understandable XML format.

The Figure 12 shows an example of intrinsic metadata for a given replica identified by *re17845*.

```
< intrinsic_metadata replica_id = "re17845" >
        < name > alizee  – ta meilleure amie.mpg </name >
        < type > mpg </ type >
        < size > 17845746 </size >
        < date > 2004/03/21 17:32:47 </date >
        < owner > universal company </owner >
        < monitoring_server_ip > monitoring.universal.com </monitoring_server_ip >
</intrinsic_metadata >
```

Figure 12 : Example of intrinsic metadata to contents

```
< replica_class id="rcid147825" parent_id="recl45621" >
        < intrinsic_metadata_condition >
                < name > * </name >
                < type > mpg | avi </ type >
                < size > * </size >
                < date >
                        < min > 2000/01/01 00:00:00 </min >
                </date >
                < owner > universal company </owner >
        </intrinsic_metadata_condition >
        < exploitation_metadata >
                < weighting >
                        < latency > 1 </latency >
                        < available_bandwith > 8 </available_bandwith >
                        < ideal_bandwith > 7 </ideal_bandwith >
                        < system_load > 3 </system_load >
                        < system_capacity > 2 </system_capacity >
                        < disk_load > 5 </disk_load >
                        < disk_capacity > 4 </disk_capacity >
                        < host_availability > 9 </host_availability >
                </weighting >
                < temporal_aggregation> relative </temporal_aggregation >
                < duplication_authorization > controlled </duplication_authorization >
                < suicide_delay > 1 day </suicide_delay >
                < exploitation_vocation > QoS Customer </exploitation_vocation >
        </exploitation_metadata >
</replica_class>
```

Figure 13 : Example of a replica class

The figure 13 gives an example of the definition of a replica class (this class - *rcid147825*- is a sub-class of *recl45621*, not described here).

One can note on the examples of figure 2 and 3, that replica *re17845* may belong to the class *recl45621* (provided that the hierarchy leads to this class), since the replica intrinsic metadata are included in the intrinsic metadata conditions of the class. Thus, we can obtain the exploitation metadata of a replica from its intrinsic metadata.

4. VALIDATION AND EXPERIMENTATION

We have implemented a simulator of NCasting and concretization mechanisms.
This simulator generates a random interconnection tree.
The nodes represent the proxy-caches. The simulator emulates hot spots and fashion phenomena : End-user's requests occur following a Gaussian distribution rate.
The edges represent the network links and are weighted by random numbers, between 0 and 1, representing the distances.

We first illustrate the behaviour of our algorithms on a simple example, with only 4 proxy-caches and 100 time-steps. The interconnection graph and the distances between proxy-caches (PC*) are given on figure 14.

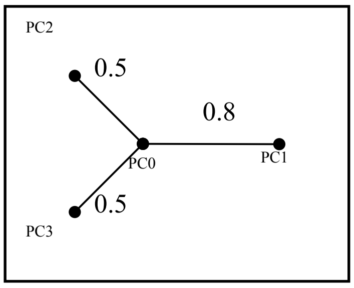


Figure 14 : Interconnection graph

The figure 15 indicates the temporal distribution of requests per proxy-cache: Initially, a peak of requests occurs on PC0; then a weaker peak appears on PC1; finally PC2 and PC3 receive approximately the same number of requests; PC3 lasts a bit longer.

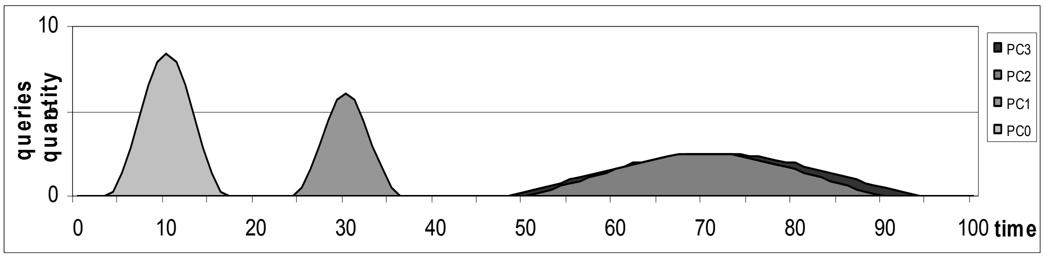
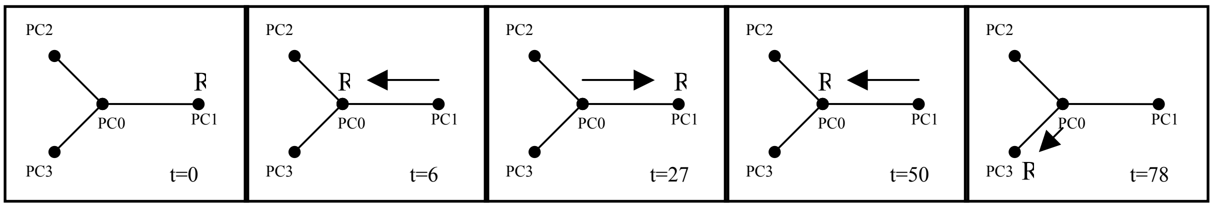


Figure 15 : Query distribution over time

Given this distribution, with one replica R (initially on PC1) and duplication prohibited, the behaviour of the system is:



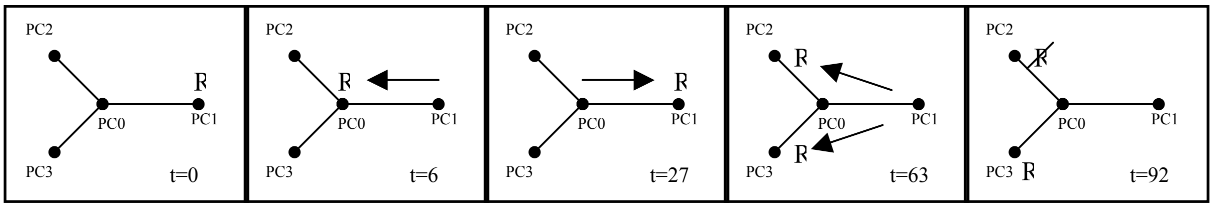
t=6: migration towards PC0, single site of requests at this time

t=27: migration towards PC1, single site of requests at this time

t=50: not being able to duplicate itself, R is attracted both by the simultaneous requests from PC2 and PC3; R moves to a balanced position in PC0

t=78: balance is broken since the number of requests on PC2 is decreasing. R thus moves to PC3 which remains an active applicant.

With the same request distribution, but with authorized duplication, the behaviour becomes :



Differences arise when two sites start requesting the replica R.

t=63: R is duplicated on the two sites of requests PC2 and PC3

t=92: PC2 stops emitting requests : its replica is deleted.

The previous example shows that the algorithm is working as expected.

Then we were interested in the evaluation of the average distance between the set of requests and the nearest replicas, as a function of the authorized replica number. This average distance indicates the customers Quality of Service.

We configured the different parameters to obtain :

- a very reactive behaviour, i.e. with a high DISTANCE2ATTRACTION value (so each request is important);
- and a low memory capacity, i.e. with a high ATTRACTION_DEGRADATION value (so requests are soon forgotten).

We simulated 100, 300 and 1000 nodes, with 1 to infinite authorized duplications. Each condition was tested on 100 different generated graphs.

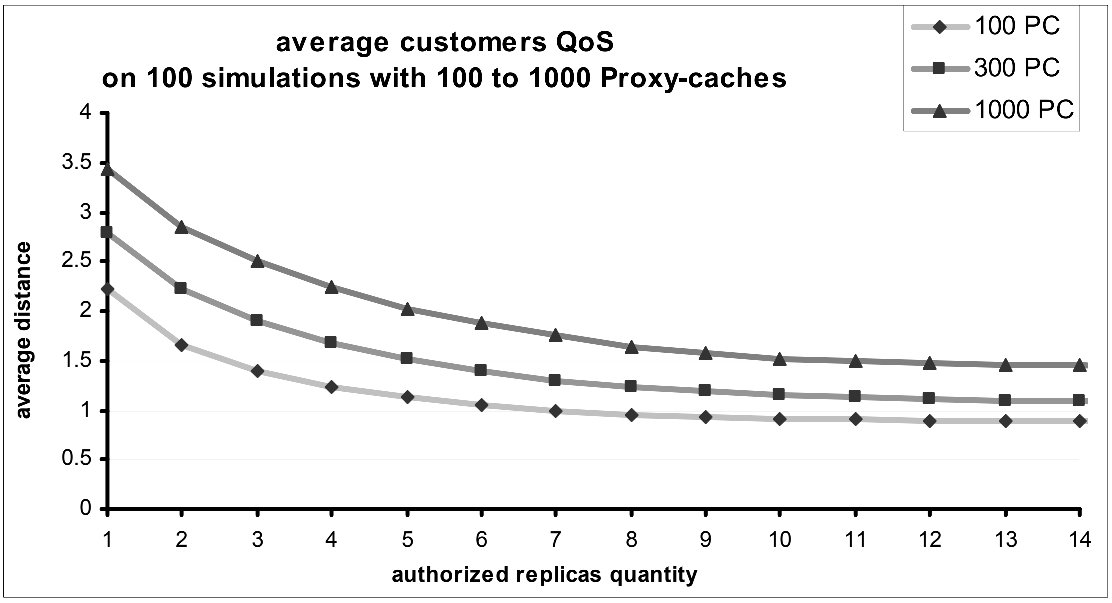


Figure 16 : average customers QoS in function of the authorized replicas quantity.

The figure 16 displays the results we obtained. The Y-axis represents the average distance between a user and the serving replica. A distance of 1 means that the replica is on average 2 hops from the user in the interconnection tree.
The average diameter of the interconnection trees: 11 for 100 nodes; 13 for 300 nodes; 17 for 1000.
The graph shows that with an average distance converging to [1..1.5], the system ensures a good response time. This QoS is obtained with a limited number of replicas: With 100 nodes, optimal results are obtained with only 9 replicas; 11 replicas are needed for 1000 node tree. Moreover, the small difference between results with 100 and 1000 nodes indicates that our solution is scalable in terms of number of replicas. This allows implementing resource consuming mechanisms like real time monitoring or replica indexation.

The experiment described in this section show the efficiency of our solution in the context of the assumptions described in section 3.1. We are implementing series of experiments using traces coming from real proxy-caches and network monitoring tools.

5. DISCUSSION

Distributed multimedia systems take advantages of replica management systems where the type of data is used to better fit both the needs of users and the cost of management. Our

proposal allows to manage replicas differently with respect to their class. It is obvious that this information improves the global response time of the system, since the management becomes adapted or personalized for each replica.
The presented methodology is particularly well adapted to Small Worlds where requests on data are expressed by communities of interest, geographically close (in network terms). Indeed, replicas of data appropriately located serve a set of users. A completely random requests pattern without replica duplication could lead to a non effective management of replicas : attractions to opposite directions may stick replicas on their creation sites, and the cost of their management would overcome the benefit of an optimal placement of the replicas.
As seen in section 3, some parameters impact the behaviour of the algorithm : Finding optimal ones for a given context (network topology, number of users, properties of data to handle –size, privacy, time to live...-, ...) may be a difficult and long process. This makes the full deployment of the infrastructure limited in the general case, even if a basic configuration may help : The task to configure optimal parameters should be limited to well known situations when specific needs have to be taken into account (either bandwidth, latency or disk usage limitation, privacy, ...).
Management of sensitive data in a community, when users want to precisely control their dissemination, obtains a great benefit using the framework. For medical data, typically, MRI brain images acquired in an hospital interest mainly some practicians in this hospital, as well as colleagues or researchers at specialized centres (cancerologists, neurologists, psychologists, …), but have no intensive use in a maternity or a stomatology centre. These image replicas must be controlled and monitored carefully. The solution presented in this article is not self sufficient for this purpose and must be used together with strong access control to the actual content of data : This aspect is also studied in the research team [13].

6. CONCLUSION AND FUTURE WORKS

We have presented in this article a dynamic replica management and selection mechanism : Selection of the best replica to answer a user request is done using multiple criteria and is personalized for each class of replicas (even for each replica if needed). The management of a replica among the set of proxy-caches takes into account the usage of this replica by the users, as well as the wills of the owner of the original data : Replicas can be duplicated, can be deleted, can migrate from one proxy-cache to another. Algorithms have been proposed and have demonstrated their efficiency in our experiments on simulated networks of proxy caches.

Future work includes "real world" experiments, based on traces coming from distributed proxy-caches, given that we find a way to tackle all running conditions at the sites (see section 4). Another and complementary extension is to apply the infrastructure in a real framework : We will integrate this work in the context of dynamic management of medical and genetic data in the framework of a distributed warehouse and data-mining project [13].

REFERENCES

[1] Akamai, www.akamai.com.

[2] S. Buchholz, A. Schill. *Adaptation-Aware Web Caching: Caching in the Future Pervasive Web*. In 13th GI/ITG Conference Kommunikation in verteilten Systemen (KiVS), Leipzig, Germany, 2003.

[3] S. Buchholz, T. Buchholz. *Replica Placement in Adaptive Content Distribution Network*. Proc. of the ACM Symposium on Applied Computing (SAC'04), Nicosia, Cyprus, Mar 14-17, 2004.

[4] C. Ferdean and M. Makpangou. *Flexible Contract Substrate for an Adaptable Replication System*. Concurrent Information Processing and Computing Advanced Research Workshop, July 5-10, 2003

[5] R. Vingralek. *Web++: A System For Fast and Reliable Web Service*. Proceedings of the USENIX Annual Technical Conference, Monterey, California, USA, June 6-11, 1999 ; p171-184.

[6] M. Rabinovich. *Issues in Web content replication*, Data Engineering Bulletin, Invited paper , Vol. 21 No. 4. December 1998.

[7] M. Rabinovich, A. Aggarwal. *Radar: A scalable architecture for a global web hosting service*. In The 8th Int. World Wide Web Conf, May 1999.

[8] L. Qiu, V. N. Padmanabhan, G. M. Voelker. *On the Placement of Web Server Replicas*. Proceedings of IEEE INFOCOM 2001, Anchorage, AK, April 2001.

[9] J. Kleinberg. *The Small-World Phenomenon: An Algorithmic Perspective*. Proceedings of the 32nd ACM Symposium on Theory of Computing, 2000.

[10] K. Briggs, S. Tay. *Efficient Network Topologies*. 2000.

[11] A. Borodin, R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[12] J. Gossa, JM. Pierson, L. Brunie. *Replica placement and selection in pervasive computing.* UbiMob'04, Sophia-Antipolis, France. June 1-3, 2004 ; p195-198

[13] L. Seitz, JM Pierson and L. Brunie. Semantic Access Control for Medical Applications in Grid Environments. ACM Europar'03, Klagenfurt, Austria, august 2003, LNCS Springer Verlag, LNCS2790, pp374-383.

[14] L. Brunie, M. Miquel, JM Pierson, A. Tchounikine, C. Dhaenens, N. Melab, EG Talbi, A. Hameurlain, F. Morvan: Information grids: managing and mining semantic data in a grid infrastructure; open issues and application to geno-medical data. Workshop PADD (Parallel and Distributed Database) in 14th International Workshop on Database and Expert Systems Applications (DEXA'03), september 2003, Prague, Czech Republic, pp 509-514.

[15] Kacimi M., Chbeir R., Yetongnon K., "DM^{2}P", Information Resources Management Association conference, IRMA'04, May 23-26, New Orleans Marriot Hotel, 2004, To appear.