End-To-End Distance Computation In Grid Environment by NDS, the Network Distance Service

Julien Gossa, Jean-Marc Pierson
LIRIS - INSA Lyon
20 Av. Albert Einstein 69621 Villeurbanne, France.
{forename.surname}@liris.cnrs.fr

Abstract

This article presents a novel method for computing distances between hosts in a computational Grid. Our method allows to represent the cost to achieve any operation involving some services in a Grid environment. We demonstrate why the current monitoring tools are not sufficient in grid environments that lack high level tools to combine several sources of monitoring while taking into account particular constraints for some given applications. We show how this method has been embedded in a Grid service, namely the Network Distance Service (NDS). We illustrate the benefit of our approach in terms of accuracy, using an algorithm from graph theory on a grid planning scenario. We explain some implementation issues and discuss how the proposed NDS service can be useful to any grid service who needs some accurate evaluation for a decision making process involving such distances. [1]

1 Introduction

Several Distance Vector protocols have been implemented for routing of packet-switched networks in computer communications, as in, for example, the Routing Information Protocol for Internet traffic. Here, the concept of *distance* is actually the number of hops from an end point of the network to another. Since then, this concept has often be reused, not only for routing (RIV v2, IGRP, EIGRP, OSPF,...) but also for very different purposes such as data management, network topology discovering, resource brokering, nodes clustering, etc. We do think that the popularity of this concept comes from its similarity with our real world. So it constitutes a precious help in the understanding of the network and thus in the elaboration of decision making process.

[1]This work is supported by the french ministry of research with the ACI "masse de donnes"

There are many ways to represent an end-to-end distance thanks to raw network metrics measurement. The number of network hops is still used, but it appears to be quite irrelevant for monitoring the performances of high-level applications. The most used metric is the latency or ICMP Round Trip Time (RTT) which advantages are its stability and the easiness and inexpensiveness of its measurement. Some works are based upon bandwidth or throughput which is on one hand more expensive to measure but on the other hand more relevant regarding to data transfer concerns. Few other works are using less common raw network metrics such as loss-rate or TCP initial connection time. But to the best of our knowledge, only one of them is using all these metrics together (see section 2).

Finally, some works are assuming that the distance they use have some properties inherited from Euclidean Distance such as symmetry, reflexivity and triangular inequality. We note that these properties are not trivially satisfied due to the potential asymmetry of routes. Moreover, according to [17], these characteristics are related to mathematical *metrics*, whereas the distance between two points is defined by "the length of the path connecting them" and is free from any particular property.

From our point of view, raw network metrics are well relevant to low-level tasks like packet routing which have simple goals well identified. In grid environment, the tasks are in fact *grid services invokations*. But these *grid services* have much more complex and diverse goals and behaviors. So should be the distances they used. Our purpose is to define a method to design made-to-measure distances for any given *transactions* in grid environment. We call *transaction* an interaction between hosts of a network. Generally, it corresponds to the invokation of a web service. But it may be more basic tasks such as data retrieval or storage. Such distances are meant to be more relevant than raw metrics and thus constitute a considerable help in the improvement of the decision making process for the service using them.

The rest of this article is structured as follows. In section 2 we present different use of distances in networking. In section 3 we present our solution. An example of application in real environment is presented in section 4. Finally, we discuss our proposal in section 5 and we conclude in section 6.

2 State of the Art

In this section, first we study some works using end-to-end distances to show their importance and how they are provided by Distance Map Services. Second we present some grid monitoring tools we use to support our system.

2.1 End-to-End Distances

A abundance of works are using *end-to-end distance* to address a very large panel of problems.

For instance, Karlsson and al. present in [11] an evaluation framework for replica placement algorithms and a rather complete survey on this topic with several algorithms comparison. They define the distance as "a metric such as network latency, number of network hops, or total link *cost*". Most of the listed algorithms use a notion of distance. But this notion is always restricted to a representation by a single raw network metric.

An use case of distance in grid environment is presented in [14]. The authors present a method for automatic nodes clustering. Distances are represented by the minimum RTT and are used to map the hosts to a geometric space. An interesting aspect is that some of the decisions are based on the Euclidean distance. This shows that the notion of distance is much more complex than a simple *score*. Whereas scores are often used only for comparing purposes and so are quite simple, distances have often to satisfy more elaborated properties to be exploitable. We will see further that this approach should be discussed.

Topology discovering is a very active research area. More particularly the Distance Map Services aim to provide a map of the physical network architecture. These maps are graphs where the vertices are the hosts of the network and the weights of the edges are computed thanks to a distance function. The main goal of these works are to define scalable software architectures meant to provide an estimation or a prediction of the distance between all the hosts of a network while minimizing the number and cost of measurements (which can be very expensive). Most of them base their distance computation on network latency which is the easiest and one of the most inexpensive to measure. For instance: Francis and al. with IDMaps in [9], Eugene and al. with the Global Network Positioning (GPN) in [13] and Coates and al. in [6]. Few works are able to integrate other network metrics. For instance [9] can integrate the bandwidth "when possible", but do not give any further details on how this should be done.

We find out that very strong assumptions are often related to the notion of distances. For instance, some of these systems (like [13]) assume that the generated topology is an Euclidean space. This assumption is valid at small scale with the Euclidean distance

$$dist_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

. But at world scale, the valid geometric space is the spherical space to fit the shape of the earth. Thus the valid distance is the orthodromic distance based on latitude (La) and longitude (Lo):

$$\begin{aligned} dist_{i,j} = 6366 \times \arccos[\sin(La_i) \times \sin(La_j) \\ + \cos(La_i) \times \cos(La_j) \times \cos(Lo_j - Lo_i)] \end{aligned}$$

which is more complex but also more accurate and relevant to earth scale network topologies. The choice between these two distances must be taken according to the application and the topology of the network. The point is that the notion of distance should be refined to fit each peculiar case. Unfortunately, no work deals with the impacts of such assumptions or presents a method helping in the decision between all the solutions with regards to the application and the network topology.

The work of Yan Chen and his colleagues is very interesting to illustrate our approach. They are working on overlay network distance monitoring for large scale network. Their goal is to design tomography-based measurement mechanisms integrated in a scalable architecture. In [5] they consider RTT and TCP initial connection time. In the next article [3], they add the loss-rate. Then in [4] they use the latency, the loss-rate and a measurement method that reflects CPU loads and both uplink and downlink bandwidths. First we note the increasing number of handled raw network metrics. This confirms our intuition that we can not base our distance upon one single independent metric, but we need to combine several ones instead. Second, as one single measurement reflects several metrics, this method is restrictive for reflecting the behavior of any given transaction. Finally, these works are using Euclidean space properties too.

The only work dealing with combination of several network metrics is done by Ferrari and al. in [15]. They present a service called the *Network-based Optimization Service*. It shows how network metrics can be combined to form complex compound metrics. According to the authors, this service can be used in a variety of different use cases involving decision-making functions. It allows Resource Brokers or Data Management to make use of network status to improve their decisions accuracy. The main complex compound metrics they use is called *Closeness* and is defined as follow: Given two reference nodes N_i and N_j, let

- $pl_{i,j}$ be the packet loss probability from N_i to N_j

- $r_{i,j}$ be the maximum average round trip time between N_i and N_j
- $th_{i,j}$ be the average throughput of a TCP stream from N_i to N_j
- T a user defined parameter

the Closeness $C_{i,j}$ is defined by:

$$\begin{cases} C_{i,j} = & 0 & if\ pl_{i,j} > T \\ C_{i,j} = & 1 & if\ pl_{i,j} = 0\ or \\ & & N_i, N_j \in\ LAN\ or\ N_i = N_j \\ C_{i,j} = & \frac{th_{i,j}}{th_{max}} \alpha^{\frac{r_{i,j}}{r_{max}}} & otherwise \end{cases}$$

First, we note that using different cases might be problematic. For instance, this function is not continuous, which might cause many issues depending on the use. Moreover, this distance can not have different values inside one given LAN, which might be restrictive in large scale LANs, like clusters, hosting several instance of a same service. Second, we note that this function does not take into account the nature of the transaction, like the size of the transferred data or the needed computing time. This might lead to accuracy loss for the decision process using it. Finally, the authors aim at defining the architecture of their solution, but give no extended explanation about the method they used to design this function. Their main future work was to extend the number and types of functions available in their service rather than to define a method for this purpose. Unfortunately, the authors do not seem to have work on this topic further.

Another observation is related to the use of the metrics that are not related to the network itself but rather to the characteristics of the hosts. As a matter of fact, metrics such as CPU or disk or memory are never considered in any of these works although they are important in many cases. Obviously, the CPU capacity and load have a major impact on computing tasks performances. But also some other tasks, such as database query optimisation, need advanced monitoring information such as disk throughput or memory capacity and load. Thus, advanced Distance Map Service should be able to take these host metrics into account in the distances they provide.

Finally, these distances are mainly useful in optimization tasks. Two simple instances are: (1) the placement of services according to network topology and clients location; (2) the selection of the best service instance to respond to a given client request. As a matter of fact, computing end-to-end distances allows representing networks by relevant graphs. Such graphs allow appling the graph algorithms to solve the problems of the network itself. But to be relevant, these graphs should not reflect a particular aspect of the network only, but rather any aspect with regards to the tasks. Moreover, this notion of *optimization* may have different goals. For instance, the optimization of customer QoS will not lead to the same decisions as the optimization of global network load. Our method is meant to be adaptable to any task and any objective. Moreover, our method can be applied as well to services as to simple data or objects or physical resources.

2.2 Network monitoring tools

The identification of relevant metrics for grid environment and measurement methods has been made by the Network Measurements Working Group of the Global Grid Forum in [12]. Recent developments in grid infrastructure have lead to effective tools providing the indexing of all the hosts, data and services available in the network, such as the Monitoring and Discovery Service of Globus [2] which is based on the GMA.

The Grid Monitoring Architecture (GMA) responds to the main constraint of grid context which is users and resources dynamicity. The GMA-compliant grid resource monitors differ by the kind of entities they manage. MDS [2] and R-GMA [7] allow to monitor physical resources like CPU power and storage when other monitors like SCALEA-G [16] allows managing software resources too. They differs also by some implementation considerations. First, the stored information at the top level of the architecture: In MDS, only resource availability is mentioned. Users interested by an available ressource must then request more information from the local levels of monitoring. The second difference is about the storage format and query language. In R-GMA, storage is implemented like a virtual relational database queried with an SQL-Like language, when in SCALEA-G[16], information is stored in an XML format and queried using Xquery language. MDS stores ressource information in an LDAP-like directory.

Nevertheless, we decide to mainly use the Network Weather Service [18] because it seems to be the most complete monitoring tools. It is able to capture the condition of both network and hosts. It can provide the raw measurements of the classical metrics as well as forecasts based on aggregations of the set of raw measurements. Though NWS is enough for the application presented in this article, our service is designed to be able to easily integrate any information from any monitoring tools.

3 NDS: The Network Distance Service

Our proposal is a method to compute *end-to-end distances* made-to-measure for any given *transaction* in grid environment. Typically, a transaction is a grid or web service invokation. But it can be more basic tasks such as data placement or database request submission. A *transaction* involves at least one *customer* and one *provider*.

3.1 Model and notations

As the term *node* is ambiguous, from now on we will denote by *hosts* the real computers and by *vertices* their representation in our graph.

We define a network as a graph $G = (\mathcal{V}, \mathcal{E})$ where

- $\mathcal{V}$ is a finite nonempty set of vertices representing the hosts of the computer network.
- $\mathcal{E}$ is a finite set of edges. An edge is an ordered pair of vertices in $\mathcal{V}$. Each edge can be labelled with a weight corresponding to the value of the distance from the source to the destination of the edge.

The main problem we address is handling the large diversity of grid services: The nature of the considered *transaction* strongly influences the relevance of the different involved parameters. For instance, the metrics about CPU are very relevant for job submission, but not for data retrieval. It also influences the choice of the measurement method. For instance, the current observations are relevant for an immediate action, but the forecasts are preferable for long-term purposes. Moreover, the metrics relevance may vary according to some inner characteristics of the transaction. For instance, the latency is a relevant factor when dealing with small size data, but the bandwidth is preferable for large size data.

According to the GGF NM-WG in [12], a metric is a quantity related to the performance and reliability of the Internet. More precisely, a metric is a primary characteristic of the Internet, or of the traffic on it. A metric is the characteristic itself, not an observation of that characteristic. A measurement is an observation of a metric. Measurements may be either raw or derived. Raw measurements are something that can be measured directly, such as measuring latency using ping. Derived measurements are measured indirectly, and might be an aggregation or estimation based on a set of low-level measurements. For convenience purposes, we consider that a *metric* is a primary characteristic measured with a particular method. For instance, the last observation of the latency is one metric and the mean of all the available observations of latency over the past day is another metric.

We call either *observation* or *measure* the actual value related to one particular metric. We note M the set of metrics handled by the NDS server.

We define four classes of metrics:

- $M_{\mathcal{V}}$ is the set of metrics which domain is $\mathcal{V}$. These are network metrics such as latency or bandwidth.
- $M_{\mathcal{E}}$ is the set of metrics which domain is $\mathcal{E}$. These are host metrics such as CPU or memory or disk capacity.

$$M = M_{\mathcal{V}} \cup M_{\mathcal{E}} \text{ and } M_{\mathcal{V}} \cap M_{\mathcal{E}} = \emptyset$$

- $M^{\mathbb{R}}$ is the set of metrics which observations range in $\mathbb{R}$
- $M^{\mathbb{S}}$ is the set of metrics which observations range in $\mathbb{S}$

Where $\mathbb{S}$ is the set of the strings on the ASCII alphabet and

$$M = M^{\mathbb{R}} \cup M^{\mathbb{S}} \text{ and } M^{\mathbb{R}} \cap M^{\mathbb{S}} = \emptyset$$

Please note that $M^{\mathbb{S}}$ must be define to handle qualitative metrics such as the operating system or the machine architecture of the host. But those metrics are useful only in very peculiar cases. Thus, we will not use them anymore in this article.

The main useful metrics are:

- the bandwidth: $BW \in M^{\mathbb{R}}_{\mathcal{E}}$ in Megabits/second
- the latency: $L \in M^{\mathbb{R}}_{\mathcal{E}}$ in Milliseconds
- the CPU capacity: $CPUc \in M^{\mathbb{R}}_{\mathcal{V}}$ in MegaHertz
- the CPU availability: $CPUa \in M^{\mathbb{R}}_{\mathcal{V}}$ in percents
- the free memory space: $RAM \in M^{\mathbb{R}}_{\mathcal{V}}$ in Megabytes

These observations can be represented by matrices called BW, L, $CPUc$, $CPUa$ and RAM. we note $m_{i,j}$ the measurement of the metric m from the host i to the host j. A sample of these matrices is given in the table 1.

One can note that monitoring services can not provide observations of network metrics from one host to itself. These peculiar values must be configured by the user for each particular metric. Here, we decide to have

$$\forall i,\ BW_{i,i} = \infty \text{ and } L_{i,i} = 0$$

because we want to consider that the cost of local data transfer is null (in term of network usage) with the formulae we define in section 3.2.3.

3.2 NDS Step by Step

In this section, we explain the three steps of our method which is meant to lead to distances made-to-measure for any given grid transaction.

3.2.1 Step 1: Identification of Sources and Destinations

A vertex of our graph can be either source or destination or both of one edge. The sources represent the customers of the transaction and the destinations represent the providers. One host can be both source and destination when, for instance, one want to estimate whether it is better to submit a task to a more powerful remote host or to locally run this task and save the data transfer time.

We note these sets:

- $\mathcal{S}$ the set of sources (customers). $\mathcal{S} \subset \mathcal{V}$.
- $\mathcal{D}$ the set of destinations (providers). $\mathcal{D} \subset \mathcal{V}$.

Although the identification of sources and destinations is not a very difficult task, it must be done carefully. As a matter of fact, for a service placement instance, a bad identification of the customers might lead to false results. Moreover, in general cases, only a little part of the network might be represented. An example of involvement of the whole network is a global reorganization, which is a possible but rather rare application.

3.2.2 Step 2: Identification of Transaction Properties

We call *Transaction Property* one parameter involved in the computation of the distances which is not pure network metric. It is just one named value without any particular constraint: $< name, value >$.

For a particular kind of transaction, we gather all its properties in one *Transaction Properties Set* noted TPS.

$$TPS = \{< name, value >, ..., < name, value >\}$$

For instance, the TPS for a job submission should contain the problem data size, the result data size and the number of CPU cycles needed:

$$TPS = \{< prob, 10^5 >, < res, 4 >, < cyc, 10^{10} >\}$$

3.2.3 Step 3: Identification of the Distance Function

The Distance Function is the real-valued function that gives the final value of the distance between two nodes. We note it:

$$df : \mathcal{S} \times \mathcal{D} \rightarrow \mathbb{R}$$

It is a nonlinear combination of observations of metrics in M and values of properties in TPS. This function is the core of the distance computation. It must be relevant to the aspects the user wants to assess. Generally, its result must tend to zero when the reflected performances tend to the perfection, as distances generally represent *costs* and algorithms generally tend to minimize it.

A grid transaction, like a service invokation, is composed of three steps:

1. the customer sends a request to the provider with its problem data.
2. the provider computes the response.
3. the provider returns the response data to the customer.

This is well adapted to a large diversity of computation tasks, such as multimedia contents adaptation or data mining or database request. According to the task, one step can be ignored in the distance computation. For instance a simple database "select" query does not involve important request data and thus the step 1 might be irrelevant. On the other hand for a basic data transfer, the step 2 and 3 should be ignored.

The steps 1 and 3 correspond to data transfer across the network while the step 2 corresponds to a computation by a host (or a cluster hidden behind this host). One can note that if the transaction is very simple, for instance a time service, none of these three steps are important. As a matter of fact in that case, our method is practically useless.

Now we will see how we can model simple compound metrics representing the cost of a data transfer and the cost of a computation.

First, we have to build a compound metric called DTC, for *Data Transfer Cost*, involved in the steps 1 and 3. It must reflect the time needed to transfer a piece of data from one host to another according to its size x and the network capacity and condition. This time can vary according to many parameters, such as the protocol used, the MTU, the client and server configuration... We will define a basic metric here only based on the main parameters and on a raw TCP/IP data transfer. According to [8], "the Raw Bandwidth model using NWS forecasts can be used effectively to rank alternative candidate schedules". Then, we decide to base our function on it:

$$\begin{aligned} &\forall (i,j) \in \mathcal{S} \times \mathcal{D}, \\ &\quad DTC_{i,j}(x) = \frac{x}{BW_{i,j}} \end{aligned}$$

But this model does not consider the influence of the size of the data to place. Actually, we observe that if we have to place small size data, like a counter for instance, the key factor must be the latency. On the other hand, if we have to place large size data, the key factor is the bandwidth. Then, the size of the considered data does not influence only the distance itself, but also the relevance of the different metrics. The integration of the latency in our distance function is done according to the TCP/IP protocol: Opening and closing the connection needs 3 round trips (respectively $SYN \rightarrow SYN/ACK \rightarrow ACK$ and $FIN \rightarrow FIN/ACK \rightarrow FIN$). Then we obtain:

$$\begin{aligned} &\forall (i,j) \in \mathcal{S} \times \mathcal{D}, \\ &\quad DTC_{i,j}(x) = \frac{x}{BW_{i,j}} \\ &\qquad + 3 \times (L_{i,j} + L_{j,i}) \end{aligned}$$

Finally, the harmonization of the units of the different metrics must be done in, for instance, bytes and seconds:

$$\begin{aligned} &\forall (i,j) \in \mathcal{S} \times \mathcal{D}, \\ &\quad DTC_{i,j}(x) = \frac{x}{BW_{i,j} \times 10^6 \times 8^{-1}} \\ &\qquad + 3 \times (L_{i,j} + L_{j,i}) \times 10^{-3} \end{aligned}$$

Second, we have to build a compound metric representing the *Computation Task Cost* of the step 2, noted CTC. It must take into account the complexity of the computation according to the request data size and the provider capacity and load. Basically, we can represent this cost by:

$$\forall i \in \mathcal{V}, \quad CTC_i(x) = \frac{x}{CPUc_i \times CPUa_i}$$

where x is the number of computation cycle needed by the task. One more time, this representation is very basic and must be refined to fit more accuracy needs as it depends on number of parameters such as the host architecture, its OS, buffers size, scheduler configuration, etc.

Moreover, this cost model implies that the behavior of the task is deterministic and predictable. As a matter of fact, a lot of algorithms, and more particularly approximation ones, are not so predictable. Nevertheless, our method allows deciding the best behavior to model. As soon as one have minimal information on the complexity of the algorithm ran by the provider, one can decide to assess a mean cost and to use it. On the other hand, one can have complete information about the algorithm behavior. Although these ones are complex and involve several parameters, it is absolutely possible to include all of them in a user-made compound metric.

For instance, if we want to take into account the amount of memory space available in our cost:

$$\begin{aligned} &\forall i \in \mathcal{V}, \\ &CTC_i(x, y) \quad = \quad \frac{x}{CPUc_i \times CPUa_i} \\ &\qquad\qquad\quad \times \quad (y \leq RAM_i) : 1?2 \end{aligned}$$

where y is the size of the memory space needed and $(y \leq RAM_i) : 1?2$ represents a $2\times$-malus if not enough memory space is available.

Finally, we have to harmonize the units in cycles and bytes and seconds:

$$\begin{aligned} &\forall i \in \mathcal{V}, \\ &CTC_i(x, y) \quad = \quad \frac{x}{CPUc_i \times CPUa_i \times 10^6} \\ &\qquad\qquad\quad + \quad (y \leq (RAM_i \times 10^6)) : 1?2 \end{aligned}$$

Whatever, this model is enough for the task we show in the next section. It is important to note that our purpose is to show the feasability of such model and its usefulness more than proposing ready-to-be-used cost functions. For instance, databases have well identified and particular cost functions that can be easily integrated in our service.

4 Implementation and Experimentation

In this section, we succinctly present the implementation of the presented method and an example of it application in a real grid environment.

4.1 Implementation

NDS is a web service developed with JAVA and The Globus Toolkit 4. Its main function is to provide the value of distance between the hosts in the given *source* and *destination sets* and according to the given *Transaction Properties Set* and *Distance Function*. The evaluation of the *df* is made by JEP [1] and thus the programming effort to express the *df* by the programmer using NDS is at the same time easy and rather complete: JEP supports all the classical functions, the characters strings and the Boolean tests.

We decide to access the monitoring tools by command line execution. Thus, NDS is compatible with any monitoring tool that is accessible from its execution host. Moreover its configuration is designed to be easily extended with new metrics. NDS can provide the list of the metrics it handles for information purpose and to help the design of an appropriate *df* for one given purpose. Moreover while NDS have a library of predefined compound metrics, it is designed to allow its user to easily declare new ones, either in the library or directly in the *df* he uses.

4.2 Example of application

We have evaluated our proposition on a representative simple planning scenario of a storage service. We show how our service can be used to take optimal planning and operative decisions. The four classical problems we solve are:

1. What is the optimal number of instances of the service?
2. What is their optimal placement?
3. What is the instance to optimally store a given data?
4. What is the instance to optimally serve a given request?

Our goal here is to optimize the cost of the invokations by the customers of the service. We assume that each instance stores the same set of data and that the size of these data are 1 (or 10^0), 10^3 and 10^7, and that there is an equal number of data for each size.

Our experimentations are made on a test grid deployed over three remote sites (which are cities from France): Lyon, Lille and Toulouse. This grid is composed of 4 hosts called `Toulouse`, `Lille`, `Lyon1`, `Lyon2`.

Now we will see how we can apply our method.

4.2.1 Identification of sources and destinations

As all the host must be served by our service, the set of sources is:

$$\mathcal{S} = \{\texttt{Toulouse}, \texttt{Lille}, \texttt{Lyon1}, \texttt{Lyon2}\}$$

Moreover, we consider that the service can be supported by `Toulouse`, `Lille`, `Lyon1` only. Thus the set of destinations is:

$$\mathcal{D} = \{\texttt{Toulouse}, \texttt{Lille}, \texttt{Lyon1}\}$$

4.2.2 Identification of Transaction Properties

The main property of the transactions our storage service will treat is the size of the stored data. We consider that we have three types of data. Their respective sizes in Bytes are 1 (or 10^0), 10^3 and 10^7. Thus, the TPS we use are:

- $TPS = \{< \mathit{data\_size}, 10^0 >\}$,
- $TPS = \{< \mathit{data\_size}, 10^3 >\}$,
- $TPS = \{< \mathit{data\_size}, 10^7 >\}$.

4.2.3 Identification of the Distance Function

As we want to optimize the time needed to retrieve data by customers, the cost to be optimized here is related to transfer only. Thus the distance we use is:

$$\mathit{df}(i,j) = DTC_{i,j}(\mathit{data\_size})$$

4.2.4 Exploitation of the NDS results and validation

The real measurements of our test grid used in the distance computation are given in the table 1. For instance, the first line corresponds to the measurements from `Toulouse` to `Toulouse`, `Lille`, `Lyon1` and `Lyon2` respectively. The results of the distance computation by NDS are given in the table 2. df_0, df_3 and df_7 correspond to the three different TPS. The three columns correspond to the three sources: `Toulouse`, `Lille`, `Lyon1` while the lines correspond to the destinations.

Now we will see how this distance computation allows to use a graph algorithm. NDS implements a trivial algorithm to solve the k-centers problem. This problem can be defined as:

`Given a set` S `of points in a metric space` M `endowed with a metric distance function` D`, and given a desired number` k `of resulting clusters, partition` S `into non-overlapping clusters` $C_1, \ldots, C_k$ `and determine their ''centers''`

$$\mu = \{\mu_1, \ldots, \mu_k\} \subset M$$

`so that` $max_j max_{x \in C_j} D(x, \mu_j)$ `(i.e. the radius of the widest cluster) is minimized.`

In our scenario, S is actually $\mathcal{V}$, D is df, k is the number of instances of the service we want to place, while $\mu_1, \ldots, \mu_k$ are their optimal location. We show in [10] how properties of the distance produced by NDS can be validated to ensure that we have a "metric distance".

The "score" and rank of the possible solutions of this problem is given in the table 3.

The score is computed by:

$$\sum_{i \in \mathcal{V}} min_{j=1}^{k}\, \mathit{df}(i, \mu_j)$$

.

One can make some observations based on this table:

- The performances should be improved with more instances of the service, unsurprisingly.
- If we want to limit to one single instance, `Lyon1` seems to be the best location for the service.
- The scores with $k = 2$ is approximatively the half of the scores with $k = 1$. But the score with $k = 3$ corresponds to a real improvement.
- The rankings for sizes 10^0 and 10^3 are the same, whereas there are differences with 10^7.

  This is explained by the MTU which is approximatively 10^3 bytes over the Internet, which means that sending 1 byte or 10^3 bytes have the same cost since they use one single IP packet only.
- If we consider all the sizes together, only 10^7 have a real impact.

  This is normal because the cost of the transfer of very small data is negligible face to the cost of a large data transfer.

With regards to these results, one can decide to execute an instance of the service on `Toulouse`, `Lille` and `Lyon1` since it is obviously the best solution to ensure good performances for the customers.

The same method can be used to decide which the best instance to store a given data. In order to check if the location pointed out by NDS is truly the best, we have timed the invokation of a real storage service. The data used have size from 1 to 10^{10} bytes, but we only present representative results with the 3 sizes used before. We have timed 20 transfers at 1 hour interval for each case to obtain representative means. The results of these experiments are given in the table 4.

Table 1. Matrices of the measurements of our scenario metrics

$$BW = \begin{bmatrix} \infty & 1.56 & 2.48 & 2.17 \\ 5.37 & \infty & 3.17 & 3.14 \\ 3.36 & 3.27 & \infty & 87.68 \\ 3.44 & 3.24 & 87.19 & \infty \end{bmatrix} \quad L = \begin{bmatrix} 0 & 16.5 & 10.0 & 9.5 \\ 16.5 & 0 & 15.6 & 15.2 \\ 9.8 & 15.7 & 0 & 15.2 \\ 10.0 & 15.7 & 0.6 & 0 \end{bmatrix}$$

Table 2. Distances computed by NDS

$$df_0 = \begin{bmatrix} 0.0000 & 0.0331 & 0.0200 \\ 0.0330 & 0.0000 & 0.0313 \\ 0.0195 & 0.0315 & 0.0000 \\ 0.0200 & 0.0315 & 0.0012 \end{bmatrix} \quad df_3 = \begin{bmatrix} 0.0000 & 0.0382 & 0.0232 \\ 0.0345 & 0.0000 & 0.0338 \\ 0.0219 & 0.0339 & 0.0000 \\ 0.0223 & 0.0339 & 0.0013 \end{bmatrix} \quad df_7 = \begin{bmatrix} 0.0000 & 51.348 & 32.343 \\ 14.936 & 0.0000 & 25.300 \\ 23.815 & 24.496 & 0.0000 \\ 23.283 & 24.723 & 0.9190 \end{bmatrix}$$

Table 3. k-centers results

data_size	10^0		10^3		10^7		sum	
μ	score	rank	score	rank	score	rank	score	rank
{1}	0.0725	6	0.0787	6	62.034	6	62.1852	6
{2}	0.0961	7	0.1060	7	100.57	7	100.769	7
{3}	0.0525	5	0.0583	5	58.562	5	59.1453	5
{1,2}	0.0395	4	0.0442	4	47.098	4	47.1817	4
{1,3}	0.0325	3	0.0351	3	15.855	2	15.9226	2
{2,3}	0.0212	2	0.0245	2	33.262	3	33.3077	3
{1,2,3}	0.0012	1	0.0013	1	0.9190	1	0.9215	1

Table 4. Summary of experiment results

	data_size	10^0		10^3		10^7	
data location		score	rank	score	rank	score	rank
`Toulouse`	$\sum_{i=1}^{4} cf(i,1)$	0.0725	2	0.0787	2	62.034	3
	Experimental Mean (s)	0.615	2	0.618	2	2170.6	3
`Lille`	$\sum_{i=1}^{4} cf(i,2)$	0.0946	3	0.1011	3	65.74	2
	Experimental Mean (s)	0.803	3	0.765	3	1178.5	2
`Lyon1`	$\sum_{i=1}^{4} cf(i,3)$	0.0520	1	0.0569	1	49.22	1
	Experimental Mean (s)	0.455	1	0.445	1	879.9	1

One can note that the NDS and experimental ranking are exactly the same. Moreover, NDS recommends different decisions according to the size of the data: small size data are better located on `Toulouse` than on `Lille`, whereas it is quite the contrary for large data. This is due to the better uplink latencies from the `Toulouse` site while `Lille` have a better uplink bandwidths. Please note that this recommendation is confirmed by the real data retrievals. This proves the accuracy of NDS results which actually takes into account the network topology and the task characteristics to propose very adapted decisions.

Finally, NDS can be used to select which is the best location to retrieve a given data with very few efforts from the programmer: The only things that need to be changed are the sets of sources and destinations. For instance, to select the best provider for `Lyon2` which requests a data of 10^3 bytes replicated on the three storage services:

1. Identification of sources and destinations

$$\mathcal{S} = \{\texttt{Lyon2}\}$$

$$\mathcal{D} = \{\texttt{Toulouse}, \texttt{Lille}, \texttt{Lyon1}\}$$

2. Identification of Transaction Properties

$$TPS = \{< data\_size, 10^3 >\}$$

3. Identification of the Distance Function

$$df(i,j) = DTC_{i,j}(data\_size)$$

Then, the minimum distance returned by NDS simply indicates the best provider among the different possibilities.

One can think that this scenario is too restrictive to truly prove NDS accuracy. Please note that our goal is to show that NDS can be used for many purposes with few efforts from its user while the results are computed with regards to a lot of parameters. Moreover, the small size of our test grid have the advantage of producing very readable results, which is not the case with more hosts since the size of the matrices explodes. Finally, as our test grid is very heterogeneous and connected through the Internet, the experiments are more interesting than with a cluster-based architecture with very homogeneous hosts connected by private links.

Another observation might be that the use case is very peculiar due to the set of assumptions (requests pattern equiprobability, data sizes fixed to 3 static values, etc.). Actually, these parameters do not influence the relevance of the produced distance, but rather the feature of the used k-centers algorithm. Whereas taking into account varied request probabilities for each host does not seems to be very difficult, tackling with a wide range of continous data size might be difficult. This implies to be able to compute a primitive of the distance function and is an important part of the future works.

Finally, some results might seem to be quite trivial, for instance the fact that putting an instance of the service on each node is the optimal solution. Actually this aspect can not be well demonstrated on such a small size infrastructure. But the derivation on the distance function on a large infrastructure can show what is the amount of instances which represents real improvements: one of these amounts should correspond to the number of clusters (or LANs). Detecting this number is a quite difficult problem in grid environment which are administered by several independent administrations. Moreover NDS can determine the optimal host inside these clusters at the same time.

5 Discussion

We found out that our distance computation is a more complete approach than any previous work on this topic. The presented method allows the computation of made-to-measure distances for any transaction in grid environment. The embedment of a k-centers algorithm allows solving the most important problems of grid planning and operating. Moreover this work is possible with few efforts from the user and is reusable at every step of the service life cycle.

Nevertheless, our method is also much more complex and so presents some disadvantages.

First, the identification of the *Transaction Properties Set* and the *Distance Function* may be a long and difficult task, particularly for complex transactions. In case of mistake, the whole distances would be irrelevant and thus the decision taken upon them might be inaccurate. Nevertheless, our method allows to use any information one have on the transaction whatever the granularity is. We have presented basic compound metrics that can be used for a majority of services. But our method allows also using more precise compound metrics in case one needs it.

As a matter of fact, NDS is not designed for the end users of the system, but rather for administrators and developers which needs easy and/or accurate responses to particular well identified problems. In particular, NDS can be used by other grid services. It represents an important way of improvement in their decision making process, especially when these decisions must be made according to the network and hosts conditions. NDS can be used via a web interface by administrator to monitor the current network state and can help them to take decision for infrastructures and applications deployment with more relevance than any previous Distance Map Service.

6 Conclusion and Future works

We have presented a novel method designed to define made-to-measure distances for any given transaction in grid environment. The relevance of the provided distance is obviously enhanced compared to the distance provided by the preceding works, including the characterization of the produced distance to ensure graph algorithm working (as explained in [10]. We have implemented a web service called NDS for Network Distance Service. It is able to compute such distances between given sets of sources and destinations. NDS provides an important help at every steps of Service life-cycle and can optimally solve the most important problems related to grid services planning and operating. We have shown its accuracy on a simple but representative grid planning scenario corresponding to the k-centers problem.

At present time, NDS integrates the Network Weather Service measurements, but it is designed to be able to easily integrate any other monitoring tool. In the first instance, the future works will be to build a library of general *Transaction Properties Sets*, *Compound metrics* and *Distance Functions*. Second, we will study how the produced distances can be derived and integrated. This is an important step in the decision taking process of placement of services since the derivation might help to detect the optimal number of instances and integration allows obtaining results for a wide continous range of TPS values. Moreover, we do think that NDS represents an opening of the computer network world to the classical graph theory world which is very rich in solutions for a wide variety of problem. That is an important application opportunity for very numerous application classes.

References

[1] Jep - java math expression parser. www.singularsys.com/jep/.

[2] G. Alliance. Monitoring and discovery service. http://www.globus.org/mds/.

[3] Y. Chen, D. Bindel, and R. H. Katz. Tomography-based Overlay Network Monitoring. In *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement (IMC'03)*, pages 216 – 231, Miami Beach, Florida, USA, October 2003.

[4] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An algebraic approach to practical and scalable overlay network monitoring. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, pages 55–66, Portland, OR, USA, September 2004.

[5] Y. Chen, R. H. Katz, and C. Overton. Internet iso-bar: A scalable overlay distance monitoring system. In Spring, editor, *Journal of Computer Resource Management, Computer Measurement Group*. Computer Measurement Group, 2002.

[6] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang. Maximum likelihood network topology identification from edge-based unicast measurements. volume 30, 1 of *SIGMETRICS Performance Evaluation Review*, pages 11–20. ACM Press.

[7] A. Cooke, A. Gray, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Cordenonsi, R. Byrom, L. Cornwall, A. Djaoui, L. Field, S. Fisher, S. Hicks, J. Leake, R. Middleton, A. Wilson, X. Zhu, N. Podhorszki, B. Coghlan, S. Kenny, D. O'Callaghan, and J. Ryan. The relational grid monitoring architecture: Mediating information about the grid.

[8] M. Faerman, A. Su, R. Wolski, and F. Berman. Adaptive performance prediction for distributed data-intensive applications. In *Proceedings of the ACM/IEEE SC99 Conference on High Performance Networking and Computing*, Portland, OR, USA, 1999.

[9] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Transactions on Networking*, 9(5):525–540, October 2001.

[10] J. Gossa. Evaluation of network distances properties by nds, the network distance service. In *3th International Workshop on Networks for Grid Applications (GRIDNETS'06)*. IEEE/Create-Net, october 2006.

[11] M. Karlsson and M. Mahalingam. We need replica placement algorithms in content delivery networks? In Boulder, editor, *7th International Web Content Caching and Distribution Workshop (WCW 2002)*, pages 117–128, August 2002.

[12] B. Lowekamp, B. Tierney, L. Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany. A hierarchy of network performance characteristics for grid applications and services. In *Global Grid Forum*, june 2004.

[13] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. volume 1 of *Proceedings IEEE INFOCOM 2002*, pages 170–179. IEEE Computer Society.

[14] J. S. Qiang Xu. Automatic clustering of grid nodes. In IEEE/ACM, editor, *6th IEEE/ACM International Workshop on Grid Computing*, pages 227–233, Seattle, 13-14 november 2005. IEEE/ACM.

[15] F. G. Tiziana Ferrari. Network monitoring for grid performance optimization. In *Computer Communications Journal*, volume 27, pages 1357–1363. LNCC, january 2004.

[16] H.-L. Truong and T. Fahringer. Scalea-g: a unified monitoring and performance analysis system for the grid.

[17] E. W. Weisstein. Mathworld–a wolfram web resource, april 2006.

[18] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.