Monitoring of a Grid Storage Virtualization Service

Jacques Jorda, Aurelien Ortiz, Abdelaziz M'Zoughi, Salam Traboulsi

Monitoring of a Grid Storage Virtualization Service

Jacques Jorda, Université Paul Sabatier, FRANCE*
Aurélien Ortiz, Université Paul Sabatier, FRANCE
Abdelaziz M'zoughi, Université Paul Sabatier, FRANCE
Salam Traboulsi, Université Paul Sabatier, FRANCE

ABSTRACT

Grid computing is commonly used for large scale application requiring huge computation capabilities. In such distributed architectures, the data storage on the distributed storage resources must be handled by a dedicated storage system to ensure the required quality of service.

In order to simplify the data placement on nodes and to increase the performance of applications, a storage virtualization layer can be used. This layer can be a single parallel filesystem (like GPFS) or a more complex middleware. The latter is preferred as it allows the data placement on the nodes to be tuned to increase both the reliability and the performance of data access. Thus, in such a middleware, a dedicated monitoring system must be used to ensure optimal performance. In this paper, we briefly introduce the Visage middleware – a middleware for storage virtualization. We present the most broadly used grid monitoring systems, and we explain why they are not adequate for virtualized storage monitoring. We then present the architecture of our monitoring system dedicated to storage virtualization. We introduce the workload prediction model used to define the best node for data placement, and show on a simple experiment its accuracy.

Keywords: grid computing, storage virtualization, monitoring systems

INTRODUCTION

The grid concept defines the aggregation of heterogeneous computing and storage nodes across administrative domains to scale the performance of individual computer nodes (Foster, 1999). There are two paradigms in grid computing: large-scale grids, involving at least thousands

of nodes from standard computers, and meta-cluster type grids involving a maximum of hundreds of nodes from high performance computers – typically servers or supercomputers. We will focus on the latter case, mostly used for high performance computing. In such systems, a wide range of distributed physical resources can be used to data processing activity, especially for scientific applications, such as biology research and simulation for weather forecasts. These scientific applications require huge computations in order to achieve their goals, generating access to large amounts of data handled by the deployed grid storage resources.

Before launching such application, data are transferred to the dedicated nodes using tools like GridFTP; the application then runs on these nodes, and the results are merged to a user repository. However, in domains like weather forecasting, the granularity may be refined to produce more accurate results. Thus, these large scale applications involve an increasing number of nodes handling larger and larger amount of data. Consequently, a new problem arises, which is more about data storage than processing unit allocation. Indeed, whatever scientific computation is considered, more accurate results involve more data to store and handle. Then, manual strategies using tools like GridFTP are more difficult to implement, and the performance gain is not linear with the number of nodes: the data distribution on the nodes for HPC programs is complex, and impacts directly the sustained performance. Consequently, the need is to improve data storage on grids to overcome the current limitations.

The most advanced techniques in improving data storage on distributed systems deal with virtualization. Virtualization entails grouping physical resources in virtual ones, in order to mask the hardware complexity. Therefore, like grid middleware, storage virtualization systems handle the heterogeneity and dynamicity of nodes and network workload, while providing a transparent and uniform data access interface to the users, making data access as easy on the distributed environment as on a single machine. The work presented in this paper is part of the ViSaGe project. ViSaGe is a middleware designed to provide the set of functionalities needed for storage virtualization: transparent reliable remote access to data and distributed data management. This storage virtualization is of prime interest when working on grids. In fact, when working either with a meta cluster of a few supercomputers or a larger grid of a few hundreds of servers, the transparent access of data is needed to easily distribute jobs among compute resources. Moreover, efficient mechanism must be employed to guarantee a high level of performance.

Using ViSaGe middleware, data access is easier, and complex data management like data reorganization, data redundancy, or replication can be used to improve performance. However, such data management techniques require an efficient mechanism to guarantee the required quality of service and to ensure maximal performance. This task is even more complex because the characteristics of physical storage resources evolve over time. Therefore the various resources need to be monitored in order to react to any significant changes of observed criteria, which require an adjustment to storage parameters.

A dedicated monitoring system must then be used to trace physical resources usage and to feed back the administration tool to update storage parameters adequately. These parameters are mainly the placement of redundant / spitted data on nodes to improve both data reliability and access performance. In order to give ViSaGe an appropriate Administration and Monitoring component, we first studied the major existing grid monitoring tools to assess their capabilities to be used in the context of storage virtualization. This study will be summarized in section 2, and we will explain why these tools cannot be used in ViSaGe. In the section 3, we will introduce ViSaGe components, and present the architecture of Admon (the administration and monitoring component). We will then present in section 4 the prediction model used, and show on a basic experiment that it accurately predicts the I/O workload variation. Finally, we will conclude the paper and present some future work.

BACKGROUND

Monitoring large scale systems, like grids, implies facing a number of challenges. First, the monitoring system must be scalable. In fact, current grids include hundreds, even thousands of nodes. Monitoring such infrastructures is imperative, considering that the number of nodes will only increase in the future. Secondly, the monitoring of a large scale system needs to address the problems of individual failure. Every single node may be, temporarily or permanently, unavailable, and the monitoring system must continue its job regardless of that. Thirdly, the overhead involved in monitoring must be as low as possible. Indeed, when the number of nodes increases, the global resources consumed by the monitoring tasks should not compromise the primary functionality of the grid. Lastly, the administration overhead should not increase with the number of nodes. Otherwise, managing a grid with several hundreds or thousands of nodes would become impossible.

Currently many systems are available for monitoring grid resources (computing resources, storage resources, networks links), and grid applications. But the monitoring of a storage virtualization service involves some specificity. In fact, as large amounts of data are stored, constantly accessed, and directly involved in task execution time, the overall performance of running applications must be analyzed to update storage settings as accurately as possible, and to improve fault tolerance. In such distributed storage systems, policies are enforced with storage management protocols, such as data replication. These protocols are complex because of the dynamicity and heterogeneity of grid environments. For example, data replication can be used to optimize the network bandwidth (Frain, 2006), and to increase data access reliability (Shah, 2008). However, nodes being characterized by varying amounts of capacity and availability, the placement and selection of replicated data should consider the workload variations of several components in the end-to-end path linking source and destination (Vazhkudai, 2002), including the I/O workload: data replication implying more disk device read/write operations, creating / using a replica has a direct impact on application execution time.

This implies, for the monitoring system to be used in ViSaGe, to correlate specific counter (like disks IOs) with the traces of applications to update adequately the virtualized storage parameters. Thus, such a monitoring system must:

- collect the physical resources usage accurately, and correlate these results with the traces of running applications;
- keep the resources overhead due to monitoring as low as possible;
- be scalable and resistant to single point of failures / single point of contention;

We are going to present the major monitoring systems dedicated to grid systems, emphasizing the drawbacks of each in our needs for a storage virtualization monitoring system.

NWS: Network Weather Service

NWS (Wolski, 1998) is a project headed by Pr. Wolski from the University of California Santa Barbara (UCSB). It brings together information on the current state of network and computing resources of a platform. It provides a measure of the availability of the system and predicts changes in the short term of measured metrics through statistical treatments. NWS is composed of four components:

- Sensors measure availability of the platform from experiments
- Memory Servers (M) store the computed measurements on disks
- The Name Server (NS) is a system directory including the list of sensors and memory servers in order to locate them (as done by ldap databases)
- Predictors (P) use statistical methods to infer the future series of measurements.

NWS performs measurements on a permanent basis, even when no client requests are submitted, and stores the computed results in memory servers for later use. It measures performance in terms of bandwidth, latency and duration by establishing a socket connection to any TCP / IP link between two of its sensors. The latency is estimated for a small package round

trip: the source computer measures the time required for the recipient machine to return a four-byte packet. Similarly, the source computer measures the time needed to open a new socket and transmit a 4KB message. This measured time is then used by NWS to deduce the latency.

Regarding the bandwidth, the measurement is made by measuring the time needed to send a package of a given size (the default size is 250 KB). It should be noted that NWS sends data in increments of 32KB on a socket configured to have a 32 KB buffer at the local operating system level. Thus, the size of data sent (250 KB) involves some limitations: this value of 250 KB is limited for high performance communications (higher than 1 GB) and too heavy for classic network use (in the order of MB). Added to that, given the heterogeneity of network parameters (such as hardware connections, network cards, etc.), it follows that the results may be biased. However, for NWS, it is not an error, but a functional choice, since the calculated value represents the actual performance of an application using one socket.

NWS is one of the most popular monitoring systems. It is easy to use and performs its actions without privileges. However, the lack of accuracy in the computed measurements presents a very important problem to consider. On the other hand, although NWS collects, in addition to the state of network links, information such as CPU load, the latter has limited usefulness because of the complexity in supporting the scalability of the centralization of these data.

Autopilot

Autopilot (Ribler, 1998) is a monitoring system dedicated to the performance of distributed applications. It is composed of sensors and actuators. The sensors collect data on applications and actuators allow, from a set of decision rules, to configure the behavior of monitored applications. Sensors and actuators are placed in the source code of applications. The study of the behavior of applications is based on hidden Markov models and neural networks.

Two major criticisms can be made about Autopilot. The first criticism is that it is dedicated to monitoring applications, and does not take into account the state of various grid entities. The second criticism concerns the manual annotation of the applications' source code. This action represents a major, tedious, and often fatal constraint. Even if Autopilot relieves the manual annotation phase, a user intervention is still necessary. This intervention is used to describe the distribution of applications, to determine points of interaction consistent with the code within the physical underlying model. These two reasons explain the low use of Autopilot for applications in grid environments.

Hawkeye

Hawkeye is a tool developed by the Condor group. It is fundamentally based on Condor (Litzkow, 1988) and ClassAdd (Raman, 1998) technologies. It is a monitoring system that allows the provision of information pertaining to grid resources (storage resources, computing resources and network links), and also allows predefined tasks to be performed depending on specific conditions. The architecture of Hawkeye is based on four main components:

- Condor pool: set of resources hosting Hawkeye
- Manager: pool master
- Monitoring agents: agents collecting information
- Modules: sensors that probe the requested information

These components are organized in a hierarchical structure of four levels. At the lowest level, the modules collect the requested information using the ClassAd language, and send them to the monitoring agents. Periodically (or on demand), these agents send the state of resources to their manager. Triggers can be created to start a specific job if some event occurs. These triggers are handled by managers.

Note that using the ClassAd language, Hawkeye is highly configurable regarding the resources monitored and the information collected, and is also fully automatic. However, the volume of data collected can increase significantly, and may quickly become critical. Moreover, Hawkeye does not support application traces, which prevent any performance prediction.

Ganglia

Ganglia was developed at Berkeley University (Massie, 2004). This tool allows the monitoring of the grid resources workload, as well as application-specific metrics. It is based on a hierarchical design targeted to meta-clusters, and relies on listen/announce protocols on some specific multicast addresses. Thus, all the nodes in a given cluster have a global view of the state of all cluster nodes. It allows the user, via a Web page, to view the status of nodes over a given period.

The use of a multicast protocol and the replication of data about the state of cluster nodes may significantly increase the network load, and may prevent optimal performance on data access through the nodes. Moreover, this may become critical as the number of nodes increases in a cluster.

R-GMA: Relational Grid Monitoring Architecture

R-GMA (Cooke, 2004) provides an implementation of the Grid Monitoring Architecture (GMA) proposed by the Global Grid Forum (GGF) in a relational way. GMA (Tierney, 2002) defines three types of components: Producers, which collect data and make it available, Consumers which receive performance data and handle them, and Directory Services which support information publication and discovery. This structure allows the use of intermediaries to forward, broadcast and/or filter data. Producers are in charge of registering themselves to Directory Services, and sending ad-hoc (query) or periodical (subscription) data to Consumers. The Directory Service maintains information about each Producer and the data it produces. Consumers contact the Directory Service in order to retrieve specific Producers, given the data needed, and then connect to those Producers to query a data sample or to subscribe to periodical data feeding.

R-GMA is an implementation of the GMA architecture previously described, which uses a relational database to maintain the current state of the monitoring system, and a subset of the SQL language to query/update the information system. R-GMA presents a simple, consistent view of performance analysis in distributed systems. Its major drawback is the single directory service, which may be a single point of failure of the whole system, and a bottleneck for Producers/Consumers.

Netlogger

Netlogger (Networked Application Logger) (Tierney, 1998) represents both a methodology for analyzing the performance of applications, and a set of tools for implementing this methodology. The tools provided include some for nodes and network monitoring, a global timestamp delivery service, and an API for tracing applications. Developers can use the API for including timestamps at critical points of their applications, and use the logs provided by Netlogger to analyze the time used for every such critical operation and the state of the corresponding resources at that time.

Netlogger uses a binary format for serializing performance data, which has a minimal impact on the analyzed system. But like Autopilot, it requires manual annotation of the code for performance analysis, which is a fatal constraint. Moreover, it allows only postmortem analysis, and then cannot be used for proactive performance adjustments. Lastly, it is not structured hierarchically which may lead to scalability problems.

Monitoring and Discovery System

The Monitoring and Discovery System (MDS) is a set of tools integrated into Globus Toolkit (current release: GT4) for resource discovery and monitoring (Czajkowski, 2001). It provides two fundamental services: an index service which collects data and allows subsequent queries, and a trigger service which collects data and allows actions to take place when specific conditions are met. It should be noted that MDS is not, by itself, an information provider - it retrieves data collected by an external software component, and aggregates it in its information model. From this point of view, it is just a tool that should be used in conjunction with, for example, Ganglia, in order to achieve scalability of non-grid-dedicated monitoring systems.

This constraint is very important, since it adds a new layer to an existing system. Using such a system would imply adapting an existing monitoring system to some specific monitoring needs, and then interacting with MDS to publish/retrieve the collected information.

What we need in ViSaGe

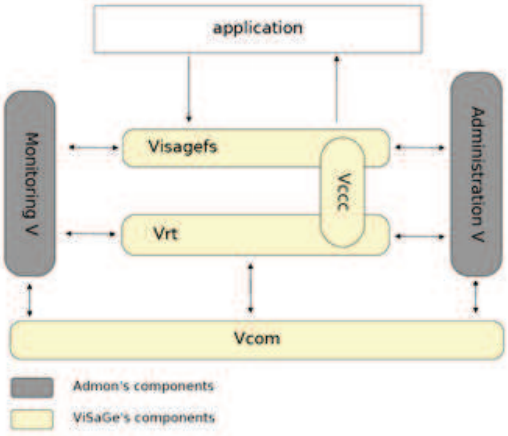
The aforementioned monitoring systems are designed to produce and deliver measurements on grid resources and applications efficiently. Nevertheless, none of these systems possess all of the pertinent characteristics for a virtualization system like ViSaGe. ViSaGe needs a monitoring system providing a full view of nodes' workloads in order to choose nodes efficiently according to its goal (data replication, workload distribution, etc.). Our monitoring system should be at the intersection of these existing tools. It should trace applications, and collect information about grid resources and the network. It should provide an accurate performance prediction time from the information collected on the I/O workload in order to choose the nodes to be used for data replication, as the latter must ensure both data transfer performance and minimal response time. In fact, many solutions are static, attempting to fix a schedule for all data transfers (Shen, 2003). Our purpose is to use monitoring knowledge to predict the time taken to place data during I/O workload variations.

VISAGE ADMINISTRATION AND MONITORING SERVICE

ViSaGe provides a storage virtualization service to the grid community. ViSaGe allows the creation and management of shared virtual storage resources in a transparent way. It joins heterogeneous and geographically dispersed physical storage resources into virtual spaces featuring various qualities of services (QoS). Each virtual space consists of logical volumes. The logical volumes, in turn, consist of data storage units. Finally, a data storage unit represents the physical placement of the data on the grid nodes, with a data placement policy being associated to each of them. This service respects the grid hierarchical architecture, which is composed of three levels: the grid level (which represents the grid gate), the site level (the frontal site) and the node level. The grid consists of several sites connected by network links. The sites themselves provide several computing and/or storage nodes. The grid is represented by a set of services which allow the user to exploit the grid, for example the storage resources virtualization. They are deployed on the grid respecting its hierarchical architecture: the grid level (representing the grid gate), the site level (the frontal site) and the node level.

ViSaGe consists of five main components (see Fig. 1): Vcom, Visagefs, Admon, Vrt and Vccc.

Figure 1: Visage's architecture



- Vcom is a communication layer between ViSaGe components. It provides a secure and adaptable way to communicate without worrying about the details related to the grid infrastructure.
- Visagefs (Thiebolt, 2007) is a grid filesystem. Visagefs allows grid users to access their data in the virtual storage spaces.
- Admon is the administration and monitoring service, used to manage and monitor grid nodes. It also provides a means of communication between grid users and ViSaGe.
- Vrt is the ViSaGe virtualization layer. Vrt carries out the Admon requests to aggregate physical storage resources in virtual spaces. Vrt places these virtual spaces at the disposal of Visagefs.
- Vccc is an extensible coherency and consistency library. It provides to the Vrt and Visagefs a set of coherency and consistency protocols.

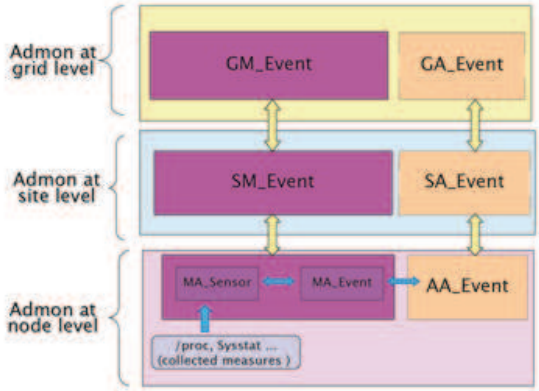
We will focus on Admon, the Administration and Monitoring Service of ViSaGe. We will first present the general architecture of this service and the API to be used to access its functionality. We will then show how the collected data are processed to guarantee the lowest impact on the nodes' workload.

ADMON Architecture

Admon consists of two modules. The administration module manages the use of ViSaGe, and the monitoring module collects pertinent information needed for overall performance management. Admon offers an interface to grid users to manage the virtual storage system lifecycle (e.g. to create or modify virtual spaces, share storage resources, access shared storage resources, etc.). It traces both ViSaGe applications and system resource consumption (CPU, Disks and Networks) in a scalable way. Thus, using its workload prediction model, Admon allows dynamic data placement according to resource usage (Traboulsi, 2008), ensuring best performance on data placement while limiting the monitoring overhead.

A hierarchical multi-tiered architecture was adopted to implement the Admon structure (see Fig. 2).

Figure 2: Admon's Architecture



As previously stated, Admon consists of two modules. Each module consists of three components corresponding to:

- The agents, on the node level, which manage and monitor nodes in order to send reports to the upper level.
- The site tools, on the site level, which collect nodes' reports to manage their site. These tools also communicate with the upper level.
- The grid service and the grid tools, on the grid level. The grid service allows grid users to reach the Admon functionality by means of a grid gate. This grid service respects the WSRF standard.

We are now going to present in more depth the various components of Admon.

ViSaGe Monitoring

The monitoring part of Admon consists of three main components: the monitoring agents, the site monitoring tools and the grid monitoring tools.

- *Monitoring Agent*: a monitoring agent is launched in all single nodes where the ViSaGe processes are running. It is represented by two routines: MA_sensor and MA_event (Fig. 2). MA_sensor collects and computes the percentage of the system resource utilization: CPU, Disk and Networks. This information is stored in a Berkeley database. The monitoring agent gets information from the Sysstat utility. The information collected is used to compute system resource throughput. MA_event manages messages sent to the monitoring agent, in an event driven way. For example, for identifying the state of ViSaGe (up or down) in a node, the monitoring site contacts the node's MA_event. Furthermore, this information is optimized and summarized by MA_event in order to send significant node reports (significant change) to the monitoring tools of its site. The monitoring agent collects information at five second intervals. It summarizes information at one minute intervals in order to detect significant changes. This selected time interval is arbitrary.
- *Site monitoring tools*: a site monitoring tool is represented by SM_event (Fig. 2). SM_event exploits the collected and optimized information of each monitored node. This information is used to observe the lifecycle of the site's nodes. For instance, if a node does not exist anymore because of a ViSaGe failure, the site monitoring tool contacts the administration part to remove this node from the live nodes list. We must mention that the administration part stores the node's virtualization history in the database (ViSaGe mount point, shared storage resources).
- *Grid monitoring tool*: a grid monitoring tool is represented by GM_event (Fig. 2). GM_event collects and communicates pertinent virtualization information, for instance virtual storage resource identification, storage resource information (node name, site name, identification, directory path), etc. The information collected by the grid monitoring tool represents a system history that allows ViSaGe to have sufficient information, for example, when the system is restarted.

ViSaGe Administration

The administration service manages the grid resources using the summarized information. It consists of the grid administration tool, the site administration tools and the administration agents.

- *Grid Administration tool*: a grid administration tool is represented by GA_event (Fig. 2). GA_event manages the requests of the grid user via the Admon grid service.
- *Site Administration tools*: the site administration tool is the Admon's means of communication between node level and grid level. Moreover, the site administration tool knows the available physical storage resource location on the nodes of its site to be shared. It is represented by the SA_event, placed on the frontal of each site.
- *Administration Agent*: an administration agent is represented by the AA_event (Fig. 2), and distributed on the node level. AA_event constitutes a way of communicating between the site administration tool and other ViSaGe services. AA_event facilitates the deployment and configuration of ViSaGe's nodes. When ViSaGe starts on a node, AA_event contacts the administration tool of its site in order to add it to the ViSaGe nodes list.

Admon API

Admon is used to observe and achieve activities on the ViSaGe system; to attempt these goals, a grid service was implemented. This grid service, respecting the WSRF standard, allows commands to be used via Globus services. Globus is used only to provide a standardized interface to access the Admon commands. The grid service consists of two parts: server part and client part.

The client part is developed by our industrial partner "CS SI". This part is an implementation in Java and is connected to our partner's grid virtualization services. These services of virtualization were developed to be deployed on a grid using a distributed file system. Their implementation is based on the Globus Toolkit 4.0 (GT4.0), on compatible and safe, open technologies: XML, Java, SOAP, etc.

The server part is related to Admon API. This API supports various administration and monitoring commands. By means of a web page, the grid user uses Admon commands. These commands enable the grid user to manage the virtual storage resources efficiently, and also the constraints used by the Admon administration decision-making process. The Admon grid service was validated on the elis@ grid.

The Admon commands allow the configuration, management and easy usage of the ViSaGe storage virtualization system. The outcome of these commands is an XML file. The administration commands consist of a set of commands allowing ViSaGe management and usage. The administration commands are sent to the ViSaGe virtualization layer (the Vrt component). These commands allow the virtual storage space creation (CreateVs), the logical volume creation (CreateLv), the storage resource sharing (ShareSrs), the logical volume format (FormatLv), the Visagefs mount and the data placement (AddReplica and RemoveReplica). In addition, the monitoring commands are sent to the Admon monitoring agents and tools. These commands allow, for example, information to be obtained about the state of nodes (GetNodeState) or sites (GetSiteState), the list of physical storage resources on a specific node (GetSrListNode) or on a site (GetSrListSite), the available nodes list (GetNodeSiteList), and so on.

Performance of the monitoring system

The performance of a monitoring system requires summarizing the information collected to avoid the storage of huge amounts of data at each level of Admon components. Therefore, the monitoring agent adopts an algorithm for sending efficient information to the site monitoring tool. The percentage of use of system resources (CPU, Disk, and Networks) is strongly related to the node workload. The monitoring agent collects the average percentage of use of system resources and uses Admon monitoring algorithms in order to manage and improve monitoring performance. Therefore, for each monitored measurement we have an old monitored mean and a new monitored mean. Furthermore, we incorporated a method of alerting the upper levels to the fact that either the CPU, Disk or Networks were operating at 50% or more, which would indicate the presence of overloaded nodes. Therefore we used a threshold that may reflect the workload state. This threshold is defined by this equation:

$$\varepsilon = \left| \frac{m_o - m_n}{m_o} \right| \qquad (1)$$

where m_o represent the old monitored mean and m_n the new monitored mean.

After collecting information, the monitoring agent computes the threshold, allowing contact with the upper level in order to update the state, or to contact the administration agent to help manage the storage system. We will give an overview of the ViSaGe monitoring algorithm:

- *Case 1:* if $\varepsilon = 0$, it means that there is a stable state, so we do not need to send the information except if:
    1. Case a - The site monitoring tool contacts the monitoring agent: the site monitoring tool sends a message to test if the node is live, taking the opportunity to update the criteria values or to delete the node from the list.
    2. Case b - The node is in a sensitive (loaded) state: for example if the monitored criteria is the CPU, the monitoring agent sends the information if the percentage of CPU usage is greater than 50%.
- *Case 2:* if $\varepsilon > 1$, it means that we have suddenly reached an important value (peak). So, this value will not be sent the first time. The monitoring agent doesn't know, at this time, if this

value is just an epiphenomenon or a new stable state; it will then wait for the next threshold to make a decision.

- *Case 3:* if $0 < \varepsilon \leq 1$, it means that the system workload is in progress or is trying to stabilize its state. We can take several decisions according to the mean value.

We must mention that the monitoring tool can change the time interval between two consecutive threshold evaluations on a node if the latter sends information constantly - thus avoiding network overload, and excessive resource consumption.

However, monitoring the performance is not limited to monitoring system resource usage, but must include the generation of application traces. The data access trace can be implemented by two methods (Hidrobo, 2002):

- Making a change to the source code of the application, or changing the library used by the program.
- Making a change to the I/O operating system stack to capture events.

Working on the first method is difficult, because the source code of the application is needed, and it is very difficult to know what is happening with physical operations on the higher software layers. Thus, the modification of the source code of the operating system is preferred, which is always dangerous and can permanently affect the stability of the system.

Using timestamps for important functions, such as read/write operations, Admon traces events at two levels: the application level (represented by the filesystem) and the physical level (represented by the virtualizer). Therefore neither the modification of the source code of the applications nor patching the operating system is needed, as the required information can be taken directly at the lowest level, transmitted by the other components of ViSaGe. Moreover, according to its location, these timestamps are collected without significant overhead, guaranteeing the low impact of the monitoring on the system.

In this section, we have shown the architecture of the administration and monitoring component. We have seen the mechanism guaranteeing the low impact of the monitoring on the nodes' workload. Using these collected measurements, Admon must now predict the workload variation of the nodes to feedback the administration components. In the next section, we are going to present the Admon prediction model and show its accuracy.

ADMON'S WORKLOAD PREDICTION MODEL

In ViSaGe, Admon, connected to an extendable coherency and consistency library "Vccc", may choose to use any consistency management protocol, such as the replica protocols (Frain, 2006), or more generally data distribution among nodes, according to the various nodes' workload. Whatever the mechanism retained for improving performance or reliability, taking into consideration the distance variation between various nodes (Vazhkudai, 2002) is not sufficient because data need to be write on disk. The disk write performance being related to the I/O workload, the latter is of prime importance for choosing the best node for data placement.

The Prediction Model

Ensuring the performance of data access in a grid environment had led to the problem of determining which node can be chosen to be accessed most efficiently in order to place redundant data. Because of various characteristics and I/O workload variations of the disk device, and various nodes' architecture, selecting the best node requires prediction of the time needed to place data.

Considering the effect of the I/O workload variation on the application throughput, Admon adjust the storage parameters. Before starting with our model, we should determine the metrics to be used. According to (Gossa, 2007) and (Lowekamp, 2004), the metric is a primary characteristic of the system, to be measured by a suitable method. In our case, we attempt to

calculate the distance between nodes in order to replicate data. The data replication implementation requires two tasks:

1. Transfer of data across networks,
2. Placing of data to destination disk.

So the Time $T(S)$ will include:

- The time $t_n(S)$ needed for data transfer across networks,
- The time $t_p(S)$ needed for data placement on disk (S being the data size).

We deduce that $T(S)$ can be defined by:

$$T(S) = t_n(S) + t_p(S) \quad (2)$$

In the above equation, the components have the same order of magnitude (seconds). The assumption of accuracy will be validated further. $t_n(S)$ consists of:

- the time needed to establish a communication, denoted here by the latency L,
- the time needed to send data to the remote disk, $t_s(S)$.

Thus, $t_n(S)$ is as follows:

$$t_n(S) = L + t_s(S) \quad (3)$$

The time needed to place data is defined by this equation:

$$t_p(S) = \frac{S}{B} \quad (4)$$

where B represents the disk bandwidth. The main task is to evaluate $t_p(S)$ as a function of the data size S during I/O workload variations. B is computed using traces of time used to place data on disks, and will be used to predict the I/O workload variation.

To highlight the impact of disk workload variation on I/O operations execution time, we have developed a module that generates requests to the disks in order to load them, thus increasing disk bandwidth workload percentage. This module progressively increases the number of queries issued to the disk and therefore increases the utilization of bandwidth, which is in fact reducing the available bandwidth. The module consists of two components:

- A client representing a virtual user attempting to read or write an S bytes data (here, $S = 100MB$ and the client will try to write the data). This client measures the write time of the operation, defining $t_p(S)$. The time measurements collected to place data at each level will be used to study the workload variations effect on the data access time.
- A server, increasing the I/O workload percentage, thus minimizing the available disk bandwidth, searching for stability at specific percentage levels. When a stable level is reached, the server proceeds with the next level (upper workload percentage), increasing or decreasing the number of requests sent to the disk device. It should be noted that the methodology adopted in this module was used in (Harbaoui, 2008).

Figure 3: I/O time vs Workload variation

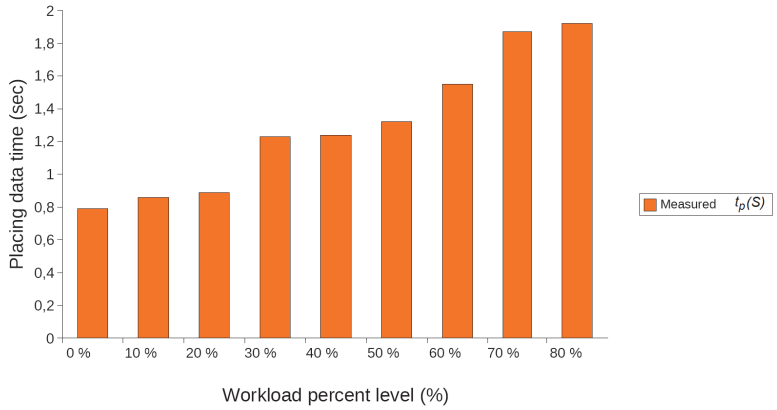


Figure 3 shows the measured $t_p(S)$ at the different percentage levels. This result proves the effect of workload variation on the time needed to place data, which was not taken into account in previous work on I/O prediction models (Shen, 2003). However, in the grid context where the distance between sites is static, the aforementioned problem degrades data transfer performance.

To predict the time $t_p(S)$, we must find a method to compute the B. B is a component of equation 4. B is calculated from the trace of several collected time measurements needed for data placement on the disk, representing the time used by the physical operations.

$$B = \frac{S}{t_p(S)} \qquad (5)$$

This value of B will be used to predict the behavior of the I/O workload, and (in future work) the data placement time on the disk devices.

We will summarize the Admon role in ViSaGe and we will validate our approach in the following section.

Experiments and validation

We have conducted a simple experiment showing how the I/O variation modifies the time needed to place data. We used three nodes of the ViSaGe grid. These nodes are node1, node2, node3. Each node is based on AMD Opteron Processors (2P), and an SCSI disk drive. Each node can reach a maximum writing bandwidth of 129 MB/s.

We launched ViSaGe on the three nodes. We installed the Admon grid tools and site tools on node1. We assumed node1 to be a source node, and node2 and node3 to be the destinations. We prepared the experiment using the following commands:

1. Create virtual space (CreateVs), create logical volume (CreateLv) on node1;
2. Share storage resources in the same logical volume (ShareSrs) on node2 and node3;
3. Format the Logical volume (FormatLv);
4. Mount the ViSaGe filesystem (MountVisagefsNode) on the chosen mount point, be it /mnt/visagefs on node2 and node3.

When ViSaGe was ready, Admon had to choose between node2 or node3, in order to add a replica of the node1 data, according to the time $T(S)$ of equation 2. “Add replica” is an administration command, calling a related function of the virtualizer API. In order to compute $T(S)$, the required information was gathered by the monitoring agents of Admon, which were started on every node where ViSaGe ran. This preliminary step was essential, as it allowed virtualization events related to I/O physical operations (the vrt_sr write data function of the virtualizer) to be collected.

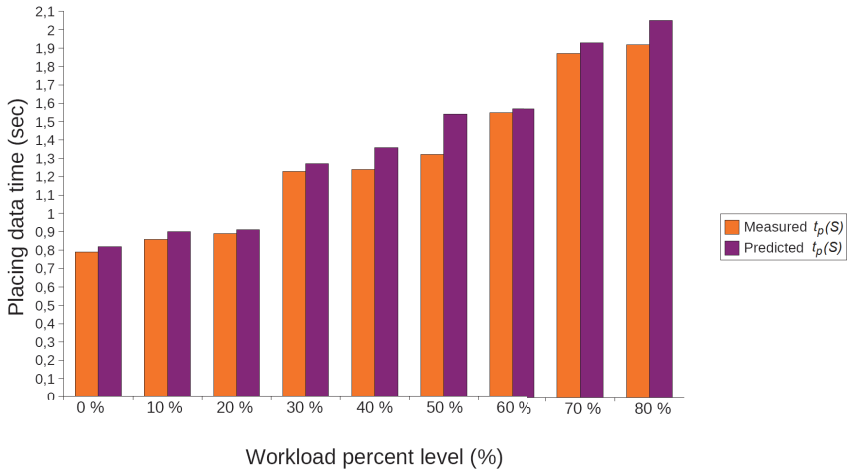
Since the challenge here was to show the I/O workload variations effect and how to establish the prediction model in order to place data, the network was not involved in the experiment. So,

we did not vary the network value, and therefore, the MA_event did not submit the network bandwidth value (required for computing $t_n(S)$, the first component of $T(S)$) to the site level, expected at the initialization phase. On the other hand, to compute the other component of equation 2 - the time to place data - Admon used information included in its traces. The measured times allowed the monitoring agents to compute the mean of the data transfer rate from the virtualizer to the disk device, which represented the available disk bandwidth (B). B was then used to predict the time needed to place data.

In our experiment, node2 was the overloaded node (that is, the node on which we launched the module described in the previous section), and node3 was an idle node. Thus, we showed how Admon proceeded to take efficient decisions.

The module consisted of two parts; the client part was attempting to write in ViSaGe and the server part was the I/O request injector. The server attempted to increase the I/O workload percent in increments of 10 %, and during each percentage increase, the virtual user (client) wrote on the /mnt/visagefs data of size 100 MB. The client which ran on the node source (node1) tried to write on node2 and node3, in order to compare Admon traced data. The server part ran on node2 to increase disk workload. Therefore, the I/O workload percentage was increased gradually, thus increasing the time needed to write on destination node2. Admon monitored the disk workload, and to avoid processes to/on the overloaded node, an alert message was sent to the upper level (node1). Furthermore, at each client writing operation, the monitoring agents of each node measured the time needed to place data in that specific workload condition. From this information, Admon computed the data transfer rate representing the available disk bandwidth. Based on this history, Admon used the computed disk bandwidth mean B in order to predict, in the future, the time required to place data (equation 4). Finally, Admon had information to measure time using its relationship with the virtualizer, and to compute predicted time using the data transfer rate. We must mention that at the first stage and before instigating the increasing disk workload module, the predicted time was computed according to the maximum data transfer rate represented by the maximum disk bandwidth. Figure 4 shows the results of our experiment.

Figure 4: Predicted vs Measured Time for Data Placement on Disk



It represented both the measured and the predicted time to place data on node2. These results show that the proposed Admon method provides very accurate times, making this model a promising method for improving storage system performance.

The prediction accuracy of this experiment was demonstrated by using the percentage error calculation.

$$Error = \left| \frac{v_m - v_p}{v_m} \right| \qquad (6)$$

where v_m represent the measured value and v_p the predicted value. We consistently observed errors of less than 5% (Table 1), which proves the performance prediction accuracy of our methodology.

Table 1: Error vs I/O workload variation

I/O Workload percentage level	Error
0	3.27
10	4.91
20	1.65
30	1.21
40	2.61
50	2.33
60	0.44
70	3.01
80	0.37

Thus, having a workload in progress on node2, and node3 being an idle node, Admon has to make a decision according to the nodes' workload state before executing the administration replica command. Table 2 shows data used by Admon to compute the $T(S)$ time of equation 2.

Table 2: information used by Admon

From node1 to node2		From node1 to node3	
$t_n(S)$	$t_p(S)$	$t_n(S)$	$t_p(S)$
0.43	1.36	0.43	0.79
0.43	1.54	0.43	0.79
0.43	1.57	0.43	0.79
0.43	1.93	0.43	0.79

This table shows the effect of the I/O workload variation on the predicted I/O time, and shows that the results presented in (Shen, 2003) may be inaccurate in case of workload variations. According to the $T(S)$ value, Admon (acting in a proactive way) will dynamically choose node3 to create its replica, and communicating with the virtualizer component, will finally place the data at that chosen node.

Referring to the problem posed at the beginning of this section, it has been shown through experiments that the predicted time computed by Admon is very closed of the measured time. Thus, Admon has led to a relevant prediction model supporting dynamic and accurate node selection for data placement by the virtualization service of ViSaGe.

CONCLUSION

This paper presents the multi-tiered architecture of a grid administration and monitoring service named "Admon". Admon is tightly coupled to a storage virtualization system called ViSaGe, which is a middleware designed to provide the set of functionalities needed for storage virtualization, namely transparent reliable remote access to data and distributed data management. It supports the creation and management of shared virtual storage resources, aggregating distributed physical storage resources. However, ensuring the performance of data access in a grid environment is a major issue, as large amounts of data are stored and constantly accessed,

and directly involved in task execution time. In particular, the placement and selection of replicated data are made especially difficult because of the dynamicity and heterogeneity of grid environments.

In this paper, we displayed "Admon" architecture and functionality. Admon traces both ViSaGe applications and system resource consumption (CPU, Disks, and Networks) in a scalable way. Thanks to a workload prediction model extending previous work (Traboulsi, 2008; Frain, 2006), Admon allows data to be placed dynamically according to resource usage, and especially disk I/O workload, ensuring the best performance on data placement while limiting the monitoring overhead.

We are currently implementing DRMAA (Distributed Resource Management Application API) in Admon in order to fully automatize job scheduling based on Admon predictive models. DRMAA is the specification produced by the Open Grid Forum to standardize job submission and management in distributed environments. Using this specification will allow Admon to interact with many job scheduling systems like Condor or Sun Grid Engine.

REFERENCE

Cooke A., & Gray A. (2004). The relational grid monitoring architecture : Mediating information about the grid. Journal of Grid Computing, 2(4), 323–339.

Czajkowski K., Fitzgerald S., Foster I., Kesselman, C. (2001). Grid information services for distributed resource sharing. 10th IEEE International Symposium on High Performance Distributed Computing (HPDC'01), pp. 181-194.

Foster I., & and Kesselman C., (1999). The grid : Blueprint for a future computing infrastructure (pp. 259–278). Morgan-Kaufmann.

Frain I., M'zoughi A., & Bahsoun J.-P. (2006). How To Achieve High Throughput With Dynamic Tree-Structured Coterie. IEEE International Symposium on Parallel and Distributed Computing (ISPDC), (pp. 82–89).

Gossa J., Pierson J.-M., & Brunie L. (2007). When the Grid becomes pervasive to save the day even Mitch Buchannon needs decision-making support. IEEE International Conference on Pervasive Services 2007, (pp. 1–10).

Harbaoui A., Dillenseger B., & Vincent J.-M. (2008), A. Performance characterization of black boxes with selfcontrolled load injection for simulation-based sizing. CFSE 2008.

Hidrobo F., & Cortes T. (2002). Racing I/O operations in Linux. Retrieved September 31, 2011, from ftp://ftp.ac.upc.es/pub/reports/DAC/2002/UPCDAC- 2002-25.ps.Z.

Litzkow M., Livny M., & Mutka M. (1988). Condor : Hunter of idle workstations. 8th International Conference on Distributed Computing Systems, (pp. 104–111).

Lowekamp B., Tierney B., Cottrell L., Hughes-Jones R., Kielmann T., , & Swany M. (2004). A hierarchy of network performance characteristics for grid applications and services. Global Grid Forum Proposed Recommendation. Retrieved September 31, 2011, from http://www.gridforum.org/documents/GFD.23.pdf.

Massie M., Chun B., & Culler D. (2004). The ganglia distributed monitoring system : Design, implementation, and experience. Parallel Computing (30), ELSEVIER, (pp. 817–840).

Ortiz A., Jorda J., & M'zoughi A. (2006). Toward a New Direction on Data Management in Grids. 15th IEEE International Symposium on High Performance Distributed Computing , (pp. 377–378).

Raman R., Livny M., & Solomon M. (1998). Matchmaking : Distributed resource management for high throughput computing. 7th IEEE International Symposium on High Performance Distributed Computing, (pp. 140-146).

Ribler R., Vetter J., Simitci H., & Reed D. (1998). Autopilot: Adaptive Control of Distributed Applications. 7th IEEE Symposium on High-Performance Distributed Computing, (pp. 172-179).

Shah C., & Shah S. (2008). Minimizing latency and optimizing resources. (COIT'08) National Conference on Challenges and Opportunities in Information Technology.

Shen X., Choudhary A., Matarazzo C., & Sinha P. (2003). A distributed multi-storage resource architecture and I/O performance prediction for scientific computing. Cluster Computing, 6(3), pp. 189–200.

Thiébolt F., Ortiz A., & M'zoughi A. (2007). VisageFS: Dynamic Storage features for wide-area Workflows. 2007 International Conference on Parallel and Distributed Computing Systems (PDCS), (pp. 61–66).

Tierney B., Aydt R., Gunter D., Smith W., Swany M., Taylor V., & Wolsky R. (2002). A Grid Monitoring Architecture. Retrieved September 31, 2011, from http://www-didc.lbl.gov/GGF-PERF/GMA-WG/.

Tierney B., Johnston W., Crowley B., Hoo G., Brooks C., & Gunter D., (1998). The NetLogger Methodology for High Performance Distributed Systems Performance Analysis. HPDC '98 Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing, (pp. 260-267).

Traboulsi S., Ortiz A., Jorda J., & M'zoughi A. (2008). Admon: ViSaGe Administration And Monitoring Service For Storage Virtualization in Grid Environment. IEEE International Conference on Information and Communication Technologies: from Theory to Applications 2008, (pp. 1-6).

Vazhkudai S., & Schopf J. M. (2002). Predicting sporadic grid data transfers. 11th IEEE International Symposium on High Performance Distributed Computing (HPDC '02), (pp. 188-196).

Wolski R. (1998). Dynamically Forecasting Network Performance Using the Network Weather Service. Journal of Cluster Computing, 1(1), pp. 119-132.