An Autonomic Framework for Enhancing the Quality of Data Grid Services

Alberto Sánchez[*,a], Jesús Montes[b], María S. Pérez[c], Toni Cortes[d]

[a] *E.T.S. de Ingeniería Informática, Universidad Rey Juan Carlos, Campus de Móstoles, Móstoles, Madrid, Spain*
[b] *CeSViMa, Universidad Politécnica de Madrid, Parque Tecnológico UPM, Pozuelo de Alarcón, Madrid, Spain*
[c] *Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo, Boadilla del Monte, Madrid, Spain*
[d] *Barcelona Supercomputing Center, Universidad Politécnica de Catalunya, Barcelona, Spain*

Abstract

Data grid access and storage services have been used to deal with the increasing needs of applications in terms of data volume and throughput. The large scale, heterogeneity and dynamism of grid environments often make management and tuning of these data services very complex. Furthermore, current high-performance I/O approaches are characterized by their high complexity and specific features that usually require specialized administrator skills. Autonomic computing can help manage this complexity. The present paper describes an autonomic subsystem intended to provide self-management features aimed at efficiently reducing the I/O problem in a grid environment, thereby enhancing the quality of service (QoS) of data access and storage services in the grid.

Our proposal takes into account that data produced in an I/O system is not usually immediately required. Therefore, performance improvements are related not only to current but also to any future I/O access, as the actual data access usually occurs later on. Nevertheless, the exact time of the next I/O operations is unknown. Thus, our approach proposes a long-term prediction designed to forecast the future workload of grid components. This enables the autonomic subsystem to determine the optimal data placement to improve both current and future I/O operations.

Key words: Autonomic storage, self-management, data-intensive applications, data grids, quality of service (QoS), long-term prediction, data management.

[*] Corresponding author
Email addresses: alberto.sanchez@urjc.es (Alberto Sánchez), jmontes@cesvima.upm.es (Jesús Montes), mperez@fi.upm.es (María S. Pérez), toni.cortes@bsc.es (Toni Cortes)

1. Introduction

Grid-based data infrastructures have been used in the last decades to address the increasing needs of applications in terms of data volume and throughput. Data grids have been defined as a set of storage resources and data retrieval elements, which allow applications to access data through specific software mechanisms [6]. The usual manner of providing this functionality is through the creation of data access and storage services. Nevertheless, the use of a grid infrastructure poses an increasing challenge to the efficient management of these data services. This increased complexity may have an impact on the QoS, mainly because the performance of storage services is linked to a very large set of changing characteristics related to the underlying infrastructure.

To date, there is no simple and efficient way of accessing data resources whilst safeguarding a committed QoS level. Ross et al. [34] identified several challenges for I/O systems emphasizing the need to increase the manageability of I/O systems due to their high complexity. Autonomic computing [22] can offer self-management features to I/O systems, which constitutes an important step forward in the I/O field. This approach is commonly known as *autonomic storage*.

Autonomic storage can improve I/O performance and QoS, especially when accessing large volumes of data such as that stored in data grids. Our proposal, called Grid-based Autonomic Storage (GAS), combines concepts from autonomic computing and I/O for the creation of an autonomic subsystem intended to provide self-management features to a data grid. This autonomic subsystem can predict the future performance of the grid components which can vary due to dynamic changes in the grid infrastructure. Based on the prediction, the target resources for data placement can be selected. This contributes to improving the QoS of data access and storage services in the grid.

The rest of this document is organized as follows: Section 2 presents the overview of GAS. Section 3 shows the internal structure of GAS. Section 4 shows our monitoring subsystem aimed at obtaining data that will be used in the following stages. Section 5 deals with all the aspects related to data analysis and scheduling. Section 6 describes a sample scenario where GAS is applied and shows the evaluation of our proposal. Section 7 shows several studies related to our proposal. Finally, Section 8 analyzes the conclusions and describes any open issues.

2. Proposal: Providing Grid Autonomic Storage

Although grid computing enables creating infrastructures to address grand challenge problems, its own characteristics imply a highly complex distributed system. In many cases, the large amount of data required by applications is one of the main causes of this increased complexity. In these scenarios, the I/O access stage limits the overall performance and therefore the QoS. Furthermore, the optimum configuration of these systems is often cumbersome. Dealing with this complexity is a top priority in term of efficiently managing the system.

Obtaining proper information about the system is the first step for its management. Since grid systems are focused on services, monitoring mechanisms should provide

relevant information, i.e. information relating to the overall service rather than the internal composition of each specific element. Thus, there is a case for efficiently adjusting distributed monitoring techniques to highly complex infrastructures. The solution presented in this paper is data aggregation, i.e. monitoring each storage resource as a single entity even if it is composed of several nodes.

Selecting the appropriate data resources to provide a high QoS is another challenging area of grid storage management. The QoS for data accesses is normally more related to the read operation behavior, since data is usually read more times than it is written [34]. As data is not usually required immediately after having been generated, enhancing the overall QoS requires improving the later read accesses instead of current write accesses. Thus, data placement should be chosen depending not only on current behavior of the storage elements but also considering predictions about their future operation. Predictions must be made in the long term as the exact time of future I/O operations is unknown.

This idea, based on monitoring and decision-making, is closely related to the way most autonomic computing applications work. Autonomic computing solutions are based on monitoring managed resources and analyzing retrieved data. After this analysis, they plan and execute the best actions according to the data analyzed and the policies defined. The whole process follows a MAPE (Monitor-Analyze-Plan-Execute) processing loop [20]. GAS is defined in the same way, following the principles of autonomic computing. GAS is also composed of four phases, whose main responsibility is providing autonomic self-management features. These four phases of GAS, following the MAPE loop, are *i*) **System Monitoring** (Monitor), responsible for collecting monitoring data *ii*) **System Analysis and Prediction** (Analyze), responsible for analyzing the monitoring data and constructing a prediction model for the storage elements, *iii*) **Decision Making** (Plan), responsible for deciding the data placement and actions to be carried out and *iv*) **Request Execution** (Execute), responsible for interpreting the decisions made and accessing suitable elements.

Our proposal presents a formal study of all the tasks required to provide autonomic features, reduce management complexity and improve data access QoS: self-managing capabilities, long-term prediction and decision-making. As a result, the proposed GAS system incorporates self-management and self-optimizing capabilities. On the one hand, GAS allows the system to adapt its behavior in the face of environment changes, making appropriate decisions according to the expected behavior. These decisions are intended to improve both current and later I/O operations. On the other hand, GAS adjusts its own internal parameters to improve its operation and future decision making.

3. GAS architecture

In a generic grid scenario, brokering is a useful technique to determine the extent to which the user requirements are met and how efficiently the underlying resources are being used. The broker not only performs the decision-making but is also essential for managing these systems. In this sense, GAS has been conceived as a brokering-based solution and can therefore be used with minor changes in conjunction with any data access service, such as Storage Resource Manager (SRM) [30], Grid Datafarm

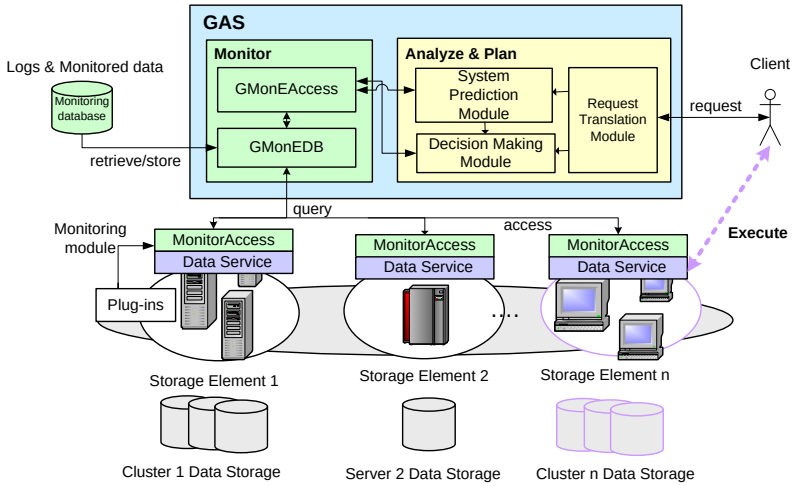


Figure 1: GAS architecture

(Gfarm) [38], MAPFS-Grid [35], etc. designed to provide self-management features to the system.

Furthermore, the broker can act act as a provider of autonomic features. To create an autonomic broker, the GAS architecture covers the following functionalities:

1. Monitoring grid resources.
2. Predicting the future state of the storage elements, according to the data monitored.
3. Decision-making about suitable data placement to ensure a high QoS data access.
4. Decision-making about the adjustment of GAS internal parameters.

Figure 1 shows the GAS architecture. The storage elements can be clusters of workstations, standalone servers with attached disks or storage-based systems, e.g. a Network Attached Storage (NAS). The architecture is divided into two main components: the first is related to the monitoring phase and the second concerned with the Analyze and Plan phases. The following sections describe both components in detail.

4. Monitor

In order to manage a large-scale system, GAS needs in-depth knowledge of the system, its state and variability. Thus, GAS needs a powerful monitoring tool which provides the required state information, without affecting the QoS of the system. To provide the required information, we have developed GMonE (Grid Monitoring Environment). Although we have implemented GMonE as a part of GAS, it is a standalone

tool that can run and work independently on any computing environment. GMonE is a set of MDS-compliant services (Monitoring and Discovery Service) [37] designed to work together. GMonE provides efficient access to information, in terms of both response times and data size. It coordinates monitored information from the whole grid, providing a unified interface for information management and queries. It also performs statistical aggregation and pre-processing of the monitored data.

GMonE components are spread throughout the entire infrastructure, monitoring each resource. These components have been designed to cooperate in order to obtain and manage the monitored data, performing four basic tasks (see Figure 1).

First, the raw monitored data is obtained using a service called MonitorAccess. The aim of this service is to provide an efficient way to access monitored information from each grid resource, performing statistical pre-processing of the data. MonitorAccess provides an abstraction layer to access all resource-monitored information from the same interface.

Second, the monitored data is aggregated. Grid systems are composed of different and heterogeneous resources. Clusters stand out among them because of their good relation in terms of power vs. cost. Since most grid resources are clusters, abstracting each cluster by aggregating data as a single entity helps understand the system more easily. In spite of its complexity, it is possible to consider a cluster as a virtual element defined on the basis of monitored data from every node. Using MonitorAccess it is possible to monitor different P_{ij} parameters of each node j of cluster i. Each parameter is grouped to obtain a representative value of the whole cluster, using different methods depending on each parameter. Examples of these methods are arithmetic functions of all the node values (e.g. the average value), pessimistic or optimistic case (e.g. taking the worst or best measurement of all nodes), individual evaluation function of a single parameter, evaluation function including all the parameters (P_{ij}, Q_{ij}, ...), etc. By means of this aggregation, all grid elements can be compared regardless of whether they are clusters or not. The benefits of data aggregation are mainly twofold: *i)* simplified understanding of the system behavior and *ii)* reduced volume of transferred information.

Third, the monitored data is gathered and managed. The GMonEDB service is in charge of gathering and managing the monitored information, once this has been obtained. This means collecting the monitored information from each element and storing it in its own database.

Fourth, the post-processed monitored data is made available to the autonomic system. The GMonEAccess service provides a common interface to query monitored data and manage the GMonE system.

5. Analyze & Plan

After the monitoring stage, GAS *i*) **analyzes** the obtained data, *ii*) creates a **prediction** model and *iii*) **makes decisions** about the target storage elements, in order to enhance the QoS of current and later data accesses.

5.1. Data analysis

Due to the high number of different resources present in a grid, the volume of monitoring information generated can be enormous. Data mining techniques can be used to identify behavior patterns. Since the number of data items is high, it is advisable to group or cluster them in different categories or states, defined by certain parameters[1].

For this task the k-means algorithm [25] was selected, due to its simplicity and fast execution[2]. K-means provides a way of grouping the monitored data. However, it is important to determine the number of clusters. This is not known a priori and, in fact, there might not be a unique answer as to what value it could take. Each method to determine the number of clusters has its own limitations, i.e. there is no definitive method for determining the number of clusters [28]. Although there are approaches [12, 23] that advocate the use of multiple techniques and their mutual comparison, this cannot be done automatically since an expert is required to select the appropriate value observing the results of the different methods. Since the number of states must be automatically determined according to monitored values, Hartigan's rule [17] is used, due to its simplicity and good results.

5.2. Prediction

Different alternatives have been analyzed to solve the problem of predicting the system behavior. *A priori* and *on-line* models provide different approaches to address the prediction task. *A priori* models perform an in-depth analysis of the past system behavior in order to forecast its future operation. However, this is based on the assumption that the system will not change its usual behavior. Long-term predictions can be made based on this assumption. Unlike *a priori* models, *on-line* models are capable of detecting the system changes in the short term. That is, they can adapt themselves to drastic changes in the system behavior due to the fact that they give more importance to current data.

Since GAS provides an autonomic system that can adapt itself to system changes, it should tackle the problem following an *on-line* model approach. However, the specific moment when a grid user makes a data request is unknown. Therefore, GAS must improve future I/O requests as a whole instead of concentrating on a given moment. Thus, in order to improve not just current operations but also future ones, long-term prediction is useful. This kind of prediction can be made by means of *a priori* models. Among the *a priori* models analyzed , Markov chains [27] can be used to predict how every parameter of each grid element usually behaves in the future. Nevertheless, the usual Markov chain approach does not meet the needs of dealing with short-term changes. In order to provide a trade-off solution, we have built an *enhanced* Markov chain approach (see Section 5.2.2), which includes *on-line* model functionalities.

[1] The analysis and data grouping developed in this work are performed for each parameter. This implies that the problem could be simplified to sampling instead of cluster analysis. Nevertheless, clustering methods are used in order to generalize the results.

[2] The system is intended to provide a fast response. Thus, the whole approach is based on methods with low computational cost.

5.2.1. Prediction calculation

The size, complexity, heterogeneity, and constant variability of the system require the creation and solution of several Markov chains, one for each storage element. Predicting a single grid resource performance requires comprehensive knowledge of its past behavior. GMonE is used to obtain all the monitored data required. GAS then obtains basic knowledge about the monitored data by means of the data analysis phase shown above to discover relations between data. This stage defines a set of states describing the expected grid element behavior. These states are dynamically defined, making it possible to self-adapt to the problem. Although it might be easier to use pre-defined and static states – since they can be defined before obtaining the data – , this would restrict the knowledge about the evolution of the grid elements, therefore reducing the system's self-management capabilities. Through this data analysis phase it is possible to predict the reachable states and the probability of reaching them. These states can be used to define a typical Markov chain. Modeling the system in this way enables making long term predictions due to the stationary probability properties. These stationary probabilities can be used for the decision-making phase.

Markov chains represent a system whose state changes over time and which can be represented as a transition matrix. The matrix obtained from a regular chain does not facilitate solving the system. However, it can be turned into a system of equations, which can be easily solved. Among the solving methods available, direct methods have been ruled out due to response-time restrictions. In this case GAS uses approximative methods. The selected algorithm is the Successive OverRelaxation (SOR) [14] which offers performance improvements over the traditional Gauss-Seidel algorithm [21]. The matrix and a vector of independent components are introduced in the SOR algorithm to obtain the probability of each state in the long term. As an initial hypothesis, a vector is defined based on historical data – which is calculated at the time of defining the states – to represent the probability of a certain state. This approach offers some advantages over the usual equiprobability vector, namely that it takes into account the significant influence of past behavior on future operation. All states have their own relevance in the system. If static states were used, the ranges of the states could be too wide for the dataset, leading to a loss of precision and raising the possibility of a transitionless state producing a non-regular Markov chain.

As the system is regulated by external factors, transition probabilities are extracted by studying the system behavior using the GMonE tool. To do this, GAS requests from GMonE a monitored parameter value for each time window. These requests generate a time series. Defining the limit values of each state, the time series is scanned counting transitions from each state to all others. This allows the transition matrix to be directly obtained from the data returned by GMonE. When the transition-counting phase is completed, it is necessary to normalize the resulting matrix. As each matrix element represents a probability, its value must be in the rank [0,1] and the sum of the row values must be equal to one. Normalization can be performed using Laplace's correction formula to obtain a confidence probability value:

$$p_i = \frac{Favorable\ cases + \lambda_i}{Total\ cases + \sum_{i=1}^{n} \lambda_i} \tag{1}$$

where n is the number of classes and λ_i is the priority for class i. In this case, it is set to 1. Equation (1) is applied to each matrix element where the number of *total cases* is the sum of all values of the row and *favorable cases* is the element value. Thus it is possible to obtain the transition matrix for each parameter and grid element. The computational overhead of this matrix calculation is very small, as will be shown in Section 6.2. However, if it were only calculated once, there would be a single prediction about the further behavior of each grid element. Therefore, as new monitored data can vary the expected future state, the predictions need to be reconsidered when new data is acquired. The next section explains our approach in solving this problem.

5.2.2. Adaptation to new data (historical Markov chain approach)

Regarding *on-line* models, it is important to take into account that new data can vary the expected future system behavior. Thus, it is necessary to analyze the past system operation every time new data is acquired. There are several ways to incorporate new behavior-related data in the prediction phase. GAS could perform a new prediction for each client write request. This would create several Markov chains over time, one for each request. Nevertheless, this alternative raises some problems. First, it could cause an overload of the autonomic system and therefore a decrease in the perceived QoS from the client point of view. Second, the monitored data time parameter which needs to be taken into account in the prediction is highly significant, since it represents the total time of the past behavior being analyzed. This means the rest of the time is not considered in the study. Thus, other alternatives have been analyzed where data is gathered since the beginning of the system execution and the monitored data time parameter can be neglected.

Our proposal is based on the idea of gathering data until a drastic change occurs in the system. Instead of performing a prediction for each client request, the predictions are made at certain times and different Markov chains are required:

- A Markov chain Pr_i^T that represents only a time interval $[(i-1)T, iT]$. Each period T a new Markov chain is calculated with all monitoring values corresponding to this period. The predictions performed correspond to the time interval when they have been calculated.
- A historical Markov chain Pr_i^H. This historical chain aggregates all the historical data from the beginning of the execution up until the time iT.

The chain Pr_{i-1}^H can be evolved by using a new Markov chain corresponding to the last time interval Pr_i^T. Pr_i^T is the Markov chain in each step of the evolution.

$$Pr_1^H = Pr_1^T$$
$$Pr_2^H = Pr_1^H \cdot Pr_2^T$$
$$\vdots$$
$$Pr_i^H = Pr_{i-1}^H \cdot Pr_i^T$$

At any time in this process, it is possible to evolve the historical chain to obtain the stationary probability for a certain state in the long term, as described in Section 5.2.1.

To do this, the period T must be constant and the states must not change in all Pr_i^T in order to allow the historical Markov chain to evolve properly. Finally, the stationary Markov chain can be calculated from the historical chain P_i^H taking into account that its states are dynamically defined once, when this matrix is created from Pr_1^T. This allows taking into account all the monitored data from the beginning of the system run. This way of evolving the historical chain allows GAS to adapt itself to gradual changes in the system behavior. In this sense, the predictions are slightly modified as new data is collected enabling to gradually change their meaning in several steps. Nevertheless, drastic changes in the system can abruptly modify this prediction. When a drastic change occurs in the system, the values of the monitored data are modified, causing the new acquired data not to fit with the states defined in the historical matrix. This causes *empty* states or states without transitions that indicate that the matrix does not comply with the regularity property. This case makes the calculation of the stationary probabilities difficult (and sometimes impossible).

Determining whether the matrix is regular or not can be reduced to determining whether all the states can be reached from any other state. This problem can be addressed using graph theory determining if the graph created according to the relations between the states is strongly connected by means of Kosaraju's algorithm. Kosaraju's algorithm calculates the strongly connected components of a graph using two Depth-First Searches [8]. If there is only a single strongly connected component, the graph is strongly connected, in which case the matrix represented by this graph is regular. If the matrix is not regular, it means that the new data, collected after defining the states, does not fit with the known states. This is caused by a drastic change in the system. In this case new states have to be dynamically defined according to the last monitored values so as to calculate a new historical chain.

5.2.3. Interpretation of results

Once the historical chain Pr_i^H of each parameter in each storage element has been evolved, the stationary probability of remaining in each of the defined states in the long term is obtained. Since each grid element can have a different representative pattern, different states can be created for each of them. As a result, the meaning of each state is different for each storage resource even though the same parameters are used. Therefore, the states are not comparable, making the decision process difficult. To overcome this problem a single value is obtained for each parameter and grid resource that represents an aggregation of the probabilities obtained in each state for the parameter in question, i.e. a numerical value that represents the average expected value in the long term. The problem is approached as a piecewise defined function that indicates the probability of the possible values of the parameter studied. The *centroid* or *center of gravity* is the most common method to obtain a single value that represents the whole function. The centroid calculates an average of the probability function, concentrating it in a single point. The value of the centroid g is obtained by means of the following formula:

$$g = \frac{\sum_{i=1}^{n} x_i f(x_i)}{\sum_{i=1}^{n} f(x_i)} \tag{2}$$

where x_i are equidistant points which cover the whole range of values where the function is not 0. The separation between these points can be adjusted according to the precision needed in the result. This single numerical value represents the observed parameter in the long term. This result is consistent with the expected probabilities of remaining in each state and it is possible to make decisions by comparing with the values obtained for the same parameter in every storage element.

5.2.4. Influence of the monitoring data period

The monitoring data period D_P^s refers to the time interval used by GAS to request a parameter P from GMonE. This parameter, monitored in a specific resource s, determines the data set used in the prediction phase. If D_P^s is too high, the system changes occurring during this period are not visible resulting in a loss of precision. A reduction of the monitoring data period allows more detailed tracking of the system changes but also requires a higher amount of monitoring data from GMonE, causing a performance decrease when a prediction is performed. Thus, careful adjustment of this parameter allows the system to better adapt itself to environment changes avoiding overheads in the prediction algorithm (i.e. the lowest amount of data should be used). The proposed system adjusts D_P^s autonomically, finding a trade-off between the size of D_P^s and the overhead caused. In order to automatically select a suitable value, it is possible to compare results obtained in the prediction phase for different values of D_P^s.

As the initial aim is the reduction of D_P^s to enhance operation, a prediction is carried out with a smaller value than the current D_P^s (the minimum size of this value is the monitoring period). Its result then is compared with the result previously obtained for the same parameter and grid resource. If the results differ[1], it means that system changes are not properly recognized and therefore it is necessary to reduce the monitoring data period, even though this involves an overhead. The initial monitoring data period used is 5 minutes because it is considered relevant enough, although it is adjusted depending on the data supplied. If both predictions are similar, reducing the number of observations does not obtain higher accuracy. In this case, a new prediction is carried out with a larger D_P^s to know if it is possible to reduce the amount of data without losing accuracy (the maximum size of D_P^s is a fraction of the prediction period T). If results are similar to the current prediction, then both predictions are similar and it is possible to increase the size of the monitoring data period in order to work with fewer observations. The amount by which to reduce or increase the monitoring data period is a fraction of the difference between the maximum and minimum period. Since this proportion is not a multiple of the current monitoring data period, periodic system changes can be discovered.

5.3. Decision-making

Decision-making means selecting the most suitable storage elements to deal with client requests. Whereas future behavior prediction is made at certain intervals, decision-making is carried out when a client requests an I/O operation from the system. Dif-

[1]The degree of similarity indicates the adjustment between D_P^s and the overhead caused. A similarity of 5% is considered sufficient to prevent loss of accuracy

ferent methods have to be applied depending on the kind of I/O operation. Depending on the specific data access service used, such as MAPFS-Grid, SRM, SRB, etc., the degree of development of each of these methods is different. The reason is that each grid I/O system uses a different data access mode. To give a comprehensive overview of the decision-making involved, the most difficult process is shown in terms of the use of data parallelism and the replication of the most used parts of each file instead of whole files. Subsequently, the problem can be adapted more easily to simpler data grid systems.

5.3.1. Decision-making for read requests

This type of requests requires reading a file. File data can be replicated into the system to provide fault tolerance capabilities. The replication involves the broker selecting the replica of each file part that currently offers the highest quality access.

The basic discovery of file data in GAS is carried out by means of MDS. That way, each data access service provides information on how to locate data. Thus, the autonomic GAS system knows which storage elements store each file data replica. Important information is then extracted from the meta-data stored in its corresponding *WS-resource properties*, which allows GAS to analyze the completeness of all the data that makes up the whole file. Finally, GAS checks if all the information is complete, ensuring that there is at least one replica of every portion[1]. If the file is complete, GAS decides which replicas provide the most efficient access to the requested file. In order to make this decision, the transfer time T_r of each replica r is calculated. T_r takes into account both the latency Lat_s^c and the data transfer rate TR_s^c. The former represents the time delay from the instant the transfer starts at the storage element s until the client c receives the first communication. The data transfer rate TR_s^c represents the time needed to send the data size DS_r. DS_r corresponds to the size of the replica and is the same in each data service resource that contains a replica of this information. Furthermore, this information is included in the meta-data since it is useful to know the file size to book space. The data transfer rate TR_s^c is the average number of bits per unit time between the resource and the client. Since this article is concerned with distributed storage, it is worth noting that both the I/O and network bandwidths are limiting factors. Thus, TR_s^c is obtained as the minimum between the internal read bandwidth, IR_s, of the storage element and the network bandwidth E_s^c between the client and the storage element. Both parameters IR_s and E_s^c are obtained by means of GMonE. If GMonE does not provide a value E_s^c (traffic between client c and element s may not yet have been observed by the monitoring system) ∞ is taken as the value, which turns E_s^c into a non-limiting factor. Normally IR_s will be higher than E_s^c. By setting E_s^c to ∞, the system will select those replicas stored in the resources not previously accessed by the client. As this access implies a connection, in later accesses other replicas will be selected, namely those that are stored in resources which the client has never previously connected to. Though this does not improve the data operations in the short term, it does, however, increase the number of connections between the client and new

[1]Depending on the policy used for the parallel distribution the method to ensure the right file reconstruction can be different.

elements thus obtaining more knowledge. This knowledge will enhance the QoS later on. Using the parameters mentioned, Lat_s^c, TR_s^c and DS_r, the usual transfer time $T_r^{s,c}$ of every replica r from the storage element s to the client c can be calculated using the following formula:

$$T_r^{s,c} = Lat_s^c + \left(\frac{DS_r}{TR_s^c}\right) \tag{3}$$

Lat_s^c is provided by GMonE. If there is no value for Lat_s^c, the client has never connected to this element. In this case, 0 is taken as the value, which implies a higher probability of selecting the non-connected resources. This results in increased knowledge about the system, allowing GAS to enhance the QoS.

The transfer time shown in (3) does not take into account the number of messages sent between the storage element and the client to transfer the size of the replica. Not all the data blocks are sent in the same message, if the transfer is carried out slice by slice. The block size of the slice is a key factor that indicates the number of messages. As in the case of DS_r, the block size BS_r^s – which is is calculated during file creation to optimize I/O access (as explained in Section 5.3.2) – is stored in the meta-data. To calculate the number of sent messages $M_r^{s,c}$, the data size of the replica DS_r and the size of the slice of the same BS_r^s must be considered in the following way:

$$M_r^{s,c} = \lfloor \frac{DS_r}{BS_r^s} \rfloor + 1$$

Finally, the number of messages only has an impact on the I/O operation latency, since it increases the time spent to establish the communication. Therefore, the transfer time is expressed as:

$$T_r^{s,c} = (Lat_s^c \times M_r^{s,c}) + \left(\frac{DS_r}{TR_s^c}\right) \tag{4}$$

Once all the transfer times of each replica have been calculated, GAS chooses the replica with the shortest $T_r^{s,c}$ time to read this part of the file. It then provides the client with a list of the data services used to store the selected replicas including their appropriate block size BS_r^s. This allows the client to find and efficiently access all the parts of the file in a parallel way using BS_r^s as block size.

5.3.2. Decision-making for file creating requests

These requests require creating a new file of a certain size. GAS must not only determine the currently suitable data locations to create and write the data with their indicated size, but it must also estimate the appropriate storage elements in order to perform later I/O operations. This estimate is made by means of the prediction phase shown before. GAS then selects the data placements that improve both current and later operations. In the selected resources, the required size is booked. Thus the autonomic system asks GMonE for the free storage space in each resource and checks if there is enough room for the file.

The decision-making to obtain a suitable trade-off between improving current and later operation is a very complex process. When a huge amount of elements are work-

ing together, each element can have a different representative pattern. Section 5.2.3 explained how the results obtained from the prediction phase can be interpreted and concentrated in a single value. This value represents a parameter observed in a storage element in the long term.

Since several parameters can be taken into account, it is necessary to determine a goodness value X_s for every storage resource s. This value concentrates all the parameters in an aggregated value representing current and later behavior of the storage resource. To model this problem, the goodness is calculated from the different behavior of each storage element. The influence of every parameter on the goodness is weighted according to its importance. Since these weights constitute a key factor, the system itself should automatically decide these values following high-level policies determined by administrators. This can be defined as an optimization problem between all the parameters affecting the system, thereby maximizing the goodness of the storage elements to ultimately simplify decision-making.

Before defining the constraints and the objective function, it is necessary to define the parameters to be considered in the decision-making process. These parameters must be related with the monitoring of the I/O system phase in a data grid. The transfer rate for read TRR_s^c and write TRW_s^c operations and the latency Lat_s^c stand out as parameters defined by both the client c and the storage element s. Parameters that are related with the behavior of each storage element s include the percentage of available storage capacity CP_s, the workload WL_s and the number of simultaneous I/O requests R_s[1]. Furthermore, since the QoS should not only be related to current but also later data accesses, the prediction of parameters must be taken into account in the calculation of the goodness. The parameters that have an influence on later accesses are those related with read accesses and the behavior of the resource, such as $\widehat{TRR_s^c}$,$\widehat{R_s}$, $\widehat{WL_s}$ and $\widehat{CP_s}$[2], and thus they have to be predicted following the method explained in Section 5.2. Additionally, since a write access occurs just after creating the file, the prediction of parameters related with this write, such as TRW_s^c and Lat_s, is not considered. Finally, the importance of every monitoring parameter in the goodness calculation can be established by means of a weight W, which allows GAS to decide which parameters have more influence during decision-making.

$$X_s = W_{TRR} \times \widehat{TRR_s^c} + W_{TRW} \times TRW_s^c + W_{CP} \times \widehat{CP_s} + \\ + W_{WL} \times \widehat{WL_s} + W_R * \widehat{R_s} + W_{Lat} \times Lat_s^c \quad (5)$$

Equation (5) shows the goodness of each grid element. Since each parameter has its own units, their values are standardized before adding. W_P represents the importance weighting assigned to each parameter P. As the aim is to obtain the values of W_P that yield the best decision-making process, the decision will be easier if the goodness of every storage element is the highest. Thus, the objective function should represent the calculation of the weights $W_{P_1}, W_{P_2}, \ldots, W_{P_n}$ that obtain a goodness X_s for each

[1] Note that any other parameters can easily be included in the model.

[2] The symbol $\widehat{P}$ stands for the predicted value of parameter P.

element s as long as there are no other weights $W'_{P_1}, \ldots, W'_{P_n}$ whose goodness assigned to a resource X'_s is higher than the one calculated with the previous weights X_s. However, this problem is NP-complete. To avoid this, the joint system goodness can be optimized instead of the goodness of each element. In this sense the objective function is:

$$\max(X_1 + X_2 + \ldots + X_n) \tag{6}$$

The relations between these weights must be defined through high-level policies. The policies indicate which parameters are more important for the definition of the QoS and they can be expressed by means of constraints on the relation between weights. Since policies are intended to improve the I/O phase so as to enhance data-intensive applications, the following restrictions have been defined based on expert knowledge in the I/O field[1]:

1. The transfer rate of read operations has a higher priority than that of write operations because the aim is to improve the QoS of later read accesses, i.e. $W_{TRW} \leq W_{TRR}$
2. Due to the great volume of data to store, the transfer rate for write operations is more important than the workload in the storage element, i.e. $W_{WL} \leq W_{TRW}$
3. Since the problem is focused on storage in a distributed environment, latency is a key factor. Therefore, its weight has to have more priority than the processing capacity, i.e. $W_{WL} \leq W_{Lat}$
4. It is important not to saturate the storage capacity of the few grid elements which have the best characteristics, since data should be distributed between all the resources. Therefore, the free capacity percentage is more important than the number of expected simultaneous connections and the current transfer rate for write operations, i.e. $W_{TRW} \leq W_{CP}; W_R \leq W_{CP}$

This problem is solved by means of linear programming, namely the Simplex method [10], since all the constraints shown and the objective function are linear. The problem is solvable since there is a basic solution available, i.e. $\forall i \in [1, n], W_{P_i} = 1/n$.

The dynamic calculation of weights allows GAS to asses the goodness of each element in order to automatically make a decision about where to place the data when creating a new file. To take full advantage of the available client network bandwidth BW_c a sufficient number of elements are selected. This involves analyzing the transfer rate TR_s^c that each element provides to the client. This analysis is performed by order of goodness and includes checking that the sum of the transfer rates of the previously selected elements is less than BW_c, to prevent the client being saturated.

[1]These policies are adapted to large amounts of data. For smaller volumes the constraints need to be changed accordingly.

Data distribution and block size. The way to access data services has a clear influence on the QoS. Data is usually accessed by means of slices. Thus, the size of the stripe sent to each storage element affects the QoS obtained. In view of the importance of this parameter, GAS calculates a suitable block size BS.

Chen and Paterson [5] indicate the block size that maximizes the performance in a striped disk array. Disk arrays can be regarded as a previous step to the proposed storage element arrays. Their proposal consists in balancing the benefit and the cost of the operation. The benefit is the transfer time of a single request, and the cost is the time spent until data actually starts being accessed. For disk arrays, the benefit can be expressed as the slice size divided by the transfer rate of the disk, and the cost is its positioning time. In a data grid, where disks are replaced by storage resources distributed over WAN, the benefit and cost are represented by different parameters. These parameters must take into account the operation of the network instead of the disk, since the network establishes the communication between the client and the data. Following the previous idea, the benefit would be the block size divided by the transfer rate of the connection and the cost would be its associated latency. Thus, a suitable block size to optimize the access between a storage element s and a client c can be expressed as:

$$BS_s^c = Z \times Lat_s^c \times TR_s^c \tag{7}$$

where Z is the zero-knowledge coefficient, that is, if no information about the workload is known. Chen and Patterson demonstrate that Z is roughly $\frac{2}{3}$. Since (7) is based on the Chen and Paterson formula developed for homogeneous disk arrays, this slice size can be used when a single storage element is accessed to obtain data. If parallelism between heterogeneous networks and elements is used, it is possible to asynchronously send the suitable block size BS_s^c to each resource. The block size of each element is obtained independently by means of (7). In this sense, faster storage elements receive more blocks according to their transfer rate. This makes reconstructing and recovering the file difficult. In fact, any grid element must inform GAS[1] about the file parts it contains by publishing each file block through MDS using its corresponding *WS-resource*. The size of a file block in each storage element s is equal to the calculated slice BS_s^c. The present proposal provides a seamless approach, since parallelism logic does not require waiting for the lowest element.

5.3.3. Decision-making for write requests to an existing file

This kind of requests requires writing a file that is in the file system. Since file data can be replicated into the system, a write involves selecting the replica of each file part that enhances the QoS of not only current but also later data accesses. However, since replication and file creation aim to optimize later access, decision-making for write operations is only concerned with current access. Therefore, the decision can be made as for read operations.

[1]GAS checks if the whole file is in the system obtaining the total number of parts in which the file was distributed and checking if all the parts are stored by any elements.

To select the most efficient replicas, the storage location of the old data must provide sufficient capacity for the new file size. This is verified during the whole process of selecting a replica of each file part. If there is not enough space available to store the file in the data distribution, decisions need to be made about other suitable locations for the new file size in the way explained in Section 5.3.2. The file must then be reconstructed following the selected new distribution. This task can be performed using selective operations. These operations read data slices of the most effective replicas that belong to the first topology and subsequently write them into the new distribution of storage elements. [31] shows the file reconstruction stage by using selective operations in cluster environments. In a grid these operations are applied analogously, as this alternative is more efficient than *brute force*. Even so, data redistribution is an expensive task, which affects system operation. The system must provide service while executing this task. The original file must not be eliminated until the whole operation is completed in order to maintain the service during the redistribution. Therefore, the system has two copies or views of such files, which are managed by GAS. During the redistribution task, a file map is created, in such a way that if another client accesses slices that have been written by the redistributing process, they access the new view. Otherwise, they access the old view.

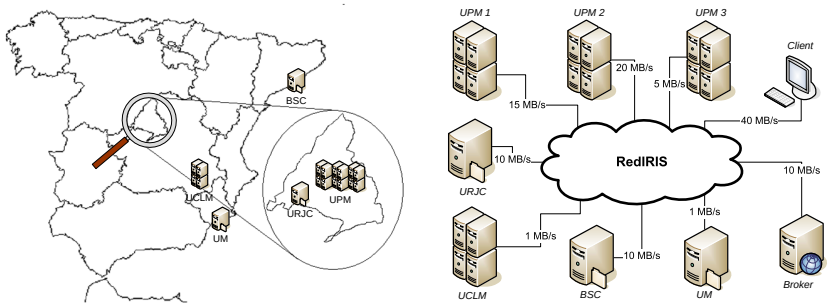
If there is enough space, GAS decides which replicas provide the most efficient write access. The transfer time is the key factor in selecting the best replicas (see (4)). Nevertheless, the data transfer rate TR_s^c for sending from the client c to each storage element s is different, since parameters affecting write operations must be considered. In this case, the data transfer rate is obtained as the minimum between the internal I/O bandwidth of writing IW_s in each site and the network bandwidth E_s^c between the client and the storage resource.

When a write is performed on the replica that provides the most efficient write access, the non-selected replicas must be marked as obsolete because their data is not updated. The consistency problems derived from this data update can be solved by using a Consistency Service [11] that propagates the update to the other sites.

6. Evaluation

In order to evaluate GAS, some non-idle storage elements distributed on the Internet have been used to build a simple but complete grid testbed. The storage elements are distributed over Spain and run the GridFTP version of the MAPFS-Grid service [35] for parallel file storage, called MAPFS-DSI. Both the storage elements and the client, are connected via the Spanish scientific wide area network, called RedIRIS. The resources are heterogeneous and belong to different sites with different administrators and security policies:

- Universidad Politécnica de Madrid: its resources are three clusters, UPM 1, UPM 2 and UPM 3, located in Madrid, Spain. UPM 1 has 8 Intel Xeon 2.40GHz nodes with 1 GB of RAM memory, their hard disks providing approx. 30 MB/s each. UPM 2 has 8 Intel Xeon 3.0GHz nodes with 2 GB of RAM memory, their hard disks providing approx. 50 MB/s each. UPM 3 has 2 Intel Pentium IV 3.20GHz



(a) Geographical location (b) Network topology

Figure 2: Grid testbed

nodes with 512 MB of RAM memory, their hard disks providing approx. 30 MB/s each.

- Universidad Rey Juan Carlos: the URJC node is an Intel Pentium IV 2.80GHz node with 1 GB of RAM memory located in Mostoles, Spain.
- Barcelona Supercomputing Centers: the BSC resource is an Intel Pentium IV 3.0GHz node with 1 GB of RAM memory located in Barcelona, Spain.
- Universidad de Castilla La Mancha: the UCLM cluster is composed of 8 Intel Pentium IV 3.0GHz nodes with 2 GB of RAM memory located in Albacete, Spain.
- Universidad de Murcia: the UM node is an Intel Pentium III 1.0GHz node with 384 MB of RAM memory located in Murcia, Spain.

The GAS broker is running on an Intel Xeon 3.00GHz node, with 2 GB of RAM memory. It is located at the computer science faculty of the Universidad Politecnica de Madrid. Finally, the client is running on an Intel Core 2 2.13GHz node with 1 GB of RAM memory and network bandwidth limited to 40MB/s. It is located at the Operating Systems Group's laboratory of the Universidad Politecnica de Madrid. Figure 2 shows the grid testbed with the topology of the network that interconnects the different grid elements. The network bandwidths shown refer to the maximum values of their network interfaces, their actual value being lower because of third-party network traffic and the internal RedIRIS topology. Furthermore, in order to understand the results it is important to emphasize that the whole set of grid elements used in the testbed are shared by other local users.

6.1. Evaluation of GMonE

GMonE is designed to provide information about the whole grid by means of an information movement similar to a client-server architecture. Clients are represented by monitored storage elements and the server is the broker where monitored data is finally stored. This client-server design implies studying three clearly differentiated

items of the system, i.e. client, environment and server, used to evaluate GMonE: *i*) data monitoring in storage elements (clients), *ii*) communication between storage elements and broker (environment), and *iii*) data recovery in the broker (server).

6.1.1. Data monitoring

An important aspect of a monitoring system is the time needed to request a value of each monitored parameter. This time refers to the response time of the MonitorAccess service, which is running in every storage element.

The monitoring time of each parameter depends on several factors, most of them related to the way parameters are monitored. The average monitoring time of the 9 required parameters is 842.2 ms. For instance, if GMonE monitors data every 5 minutes the overhead or workload of the monitoring system in the monitored system is 0.28 %.

Furthermore, monitored data by GMonE is more useful than data provided by other systems, since data is aggregated considering a compound storage element as a single machine regardless of its composition. Parameter aggregation by means of the defined functions not only makes the system easy to understand but also reduces the quantity of information that is transferred through the network, taking less than 1 ms on average, if 1000 values are aggregated.

6.1.2. Communication

It is important to analyze the network overhead produced by both monitoring services and the communication method used in a geographically scattered environment. The communication method used in GMonE for data transfer mechanisms is the standard grid monitoring information service MDS. The capabilities of MDS in terms of adaptability and scalability to heterogeneous and geographically distributed environments over WAN networks have been described in [9]. Some tests on performance, throughput and query response times shown for MDS in [36] confirm the capabilities mentioned and the low network overhead generated by the communication method. Therefore, GMonE inherits these scalability and low traffic network features.

6.1.3. Data recovery

Monitoring systems aim at providing monitored data to be processed by high-level applications. Therefore, the access time to monitored data is a key factor of a monitoring system.

GMonEAccess is the part of the proposed system that is in charge of accessing the monitoring database and providing data. Different volumes of monitored data have been requested from GMonEAccess in order to evaluate access time. Figure 3 shows the average access time regarding different volumes of required information indicated by the request period in hours (the volume of information increases as the request period is increased). The database file size is around 300 MB.

Results show very low times when accessing the database, whatever the volume of required data. The use of a proper database index significantly improved access speed because of efficient ordering of access to records by means of the date of the monitored data. Furthermore, the access time increase is not linear regarding the increase of the

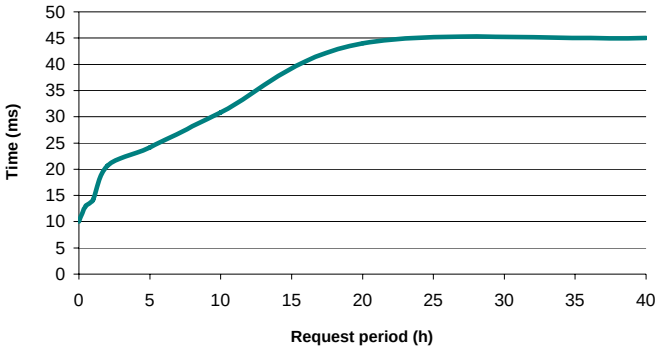


Figure 3: Access time to monitored data

request period or data volume. As a result, the use of GMonE is suitable for requesting high volumes of monitored information. Moreover, the information provided is aggregated, reducing the later processing of high-level applications.

Although access to the GMonE database is efficient, it should be remembered that an increase of the database size can have an impact on its performance. This growth can be estimated taking into account the size of each database record (S_R), the number of parameters analyzed (N_P), the number of elements monitored (N_S) and the monitoring period (D_G). The growth trend (G) of the GMonE database in bytes per day (1440 min) can be expressed in the following way:

$$G = S_R \times N_P \times N_S \times \frac{1440}{D_G} \tag{8}$$

The average size of GMonE database records is 30 bytes ($S_R \cong 30$ b). For these experiments 9 parameters were monitored ($N_P = 9$) with a 5 minute time period ($D_G = 5$ min). As a result, approximately 76 KB per day and storage element were stored in the GMonE database. Even if the GMonE database is properly indexed and accessed, if N_S is high enough the access time could increase significantly. In this case, it would be advisable to either increase D_G or to use a hierarchy of independent GMonEDB services, each one with its own database, monitoring a subset of elements and sharing the information.

6.2. Evaluation of the prediction model

This section analyzes the time required to make predictions, the weight of selected algorithms in the workload and the evaluation of the predictions.

First, several predictions have been made by GAS to evaluate the complete prediction time. Every period T, a new prediction is made for each storage element and parameter required in the decision-making. Since the system is designed to work in

the long term and the aim is not to overload the system, 1 day is considered an acceptable minimum value fo T. The average prediction time for each grid element and parameter is then analyzed. On average, it takes approximately 20 ms to predict every parameter of each resource. As four parameters are predicted per resource, this means less than 80 ms per storage element. This low time illustrates the high capacity offered by the proposed system because a typical machine can predict the future state of 12.5 resources per second, i.e. 45198 per hour. Furthermore, the possibility of increasing the period T shows the scalability of the system.

It is important to note that most of the prediction time is spent on gathering data, especially accessing database. Around three quarters of the prediction time (73.68%) is consumed by the database accesses to gather the information required for making predictions. This indicates the importance of having fast database access. On the other hand, only around a quarter (26.32%) is consumed in processing data, defining the states and making the prediction. Moreover, the prediction model designed in this work takes less than 2 ms for each parameter and storage element.

On the other hand, with the aim of understanding how well the predicted model works, the prediction of the expected distribution should be checked against real observations. Goodness of fit and more specifically the chi-square (χ^2) test can be used to check this. The χ^2 statistic calculates a discrepancy measure between observed values Obs and the values Exp of the expected distribution. In this test, both states i and expected frequencies Exp_i of every parameter p of each storage element s are calculated by means of the data analysis and prediction phases shown previously. The predictions were made based on 10 days of historical data. These predictions are compared with the data set Obs_i obtained for each parameter and grid element during the following 20 days (measured every 4 hours) classifying each value according to the states previously calculated. Once the χ^2 statistic is calculated, it can be compared to the χ^2 distribution to determine its goodness of fit and therefore its confidence level. The confidence level indicates the probability of the result being correct, i.e. if the prediction has been correct. In the tests, 57.14% of the predictions made achieve a statistically meaningful confidence level (higher than 90%). An analysis of the predictions that reach a statistically meaningful confidence level allows further conclusions. The parameter for internal I/O bandwidth of reading (Ir) showed the highest probability of a statistically meaningful confidence level, reaching 71.48%. Regarding the confidence level of the predictions of the different storage elements, no statistically meaningful differences were found.

6.3. Evaluating decision-making

In evaluating decision-making attention should be focused not only on the time consumed in read/write operations but also the time dedicated to file create operations. These operations consume much more processing time because they have to calculate the most efficient data placements to enhance the QoS of later data accesses.

The decision-making performance of read/write operations depends not only on the number of parts into which the file was divided when it was created but also the number of replicas of each part. For each replica, the broker has to obtain the corresponding current data by means of GMonE to make the decision. The average time to request this data is 30 ms per replica. Nevertheless, the processing time to select the appropriate

replicas only represents a very low percentage (13.4%) of this time because of the high cost of database accesses.

The study of file create operations requires a deeper analysis due to its higher complexity. For this purpose, the weights of the different stages are examined step by step: *i)* analysis of the active grid resources to obtain their predictions and the historical information of each of the different parameters analyzed, *ii)* obtaining the objective function and sorting the storage resources from higher to lower expected QoS of current and later data accesses according to the predictions and the high-level policies defined, and *iii)* selection of the set of storage elements required to store the file, taking advantage of most of the client network bandwidth and calculating a suitable block size to access each resource. The processing time (200 ms on average) is mostly spent on the first step (44.5%) because this phase requests a higher number of data from the monitoring system. Most of this time (72.25%) is consumed in accessing the database whereas only a minor percentage (27.75%) is used for processing data. The next high workload stage (37.8%) is the resource selection and block size calculation step, since it has to make the decision. Finally, the objective function calculation is the step that requires the least workload (17.7%) ensuring the selection of suitable algorithms to obtain the goodness of each resource.

6.4. Autonomic capabilities

GAS aims at making system management easy by incorporating autonomic capabilities. On the one hand, the self-management capability is intrinsic to the GAS operation. Section 5.3.2 analyzes in depth how GAS configures the importance (W_P) of every parameter P in order to make decisions following high-level policies. On the other hand, self-optimizing aims at enhancing the QoS of not only data accesses but also the GAS itself. The autonomic configuration of the internal parameter data period (D_P^s) allows the system to adapt better to changes in the environment. Figure 4 shows the average time taken by the system to make predictions according to different values of D_P^s. If GAS detects a stable behavior for a specific storage element s and parameter P, it increases D_P^s achieving faster access to the database and avoiding overheads in the prediction algorithm. If the behavior is too variable, GAS reduces D_P^s to obtain more information about the resource and the parameter thus adapting itself better to changes. In this case, prediction results can change around 16% moving closer to actual resource behavior as D_P^s is decreased.

6.5. GAS performance and QoS

To measure the GAS influence on the system performance and the quality of the overall data services provided, a MAPFS-DSI client application has been used. This application performs read and write operations on the selected resources in a parallel way. For comparison purposes, the client and GAS have been configured in four different working modes during these experiments:

- Random mode: the client randomly selects how many and what storage resources are used to perform parallel read and write operations. It uses a 2 MB block size, since results obtained in previous analyses show that this size is suitable in all tests.

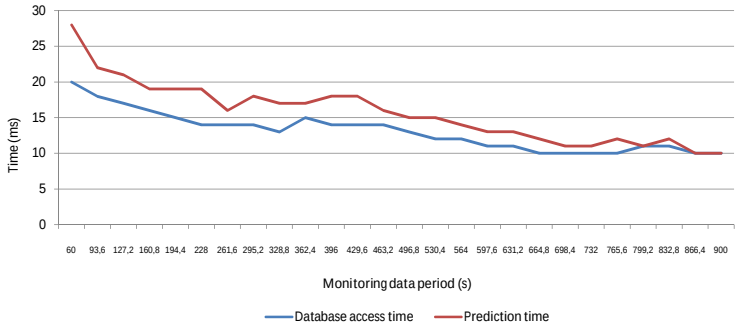


Figure 4: Average time to make predictions regarding the monitoring data period D_p^s. *GAS automatically adapts this value in accordance with the behavior of the parameter P observed in the storage element s*

- Best resource mode: the broker selects the best resource for read and write operations based on static characteristics of grid elements instead of dynamic workload. In these experiments *UPM 2* was selected as the best resource. The appropriate block size for effectively accessing the selected resource is calculated by the broker.
- Decision mode: the broker selects data placements according to the decision making phase explained in this work. However, the prediction phase described in this paper is not performed, i.e. the broker only uses the current state of each resource. As in the case of best resource mode, the client obtains the appropriate block sizes to access every resource.
- Prediction & decision mode: the GAS broker is in charge of calculating the predictions and the decision-making. The client obtains the suitable block sizes from the broker.

The last two working modes (*decision*, and *prediction & decision*) contain the main elements of the GAS proposal. They are the center of this experimental evaluation and represent the main contribution of this paper. The first two working modes (*random* and *best resource*) represent typical distributed storage configurations and are included in the study for comparison purposes. Several experiments have been performed in order to analyze the QoS of I/O operations according to the client access mode and the file sizes. Due to capacity restrictions, these tests consisted in creating and writing 10 MB, 100 MB and 1GB files for the purpose of subsequently performing consecutive read operations from the same files every six hours. For read operations, average and standard deviation bandwidth values were selected as representative descriptors of each operation. Also, brokering times were measured in order to study the overhead of the decision and prediction phases.

Regarding brokering overhead, the experiments performed show that decision and prediction times are not dependent on the specific characteristics of each I/O opera-

Figure 5: Average Read Bandwidth

tion. This means that the brokering overhead is basically constant for all operations, regardless of the amount of data that is being accessed or any other parameter.

The *random* and *best resource* modes do not have any brokering overhead, since the resource selection is performed by the client in the first case (simply by selecting random resources) and is static and performed before the experiments in the latter case. The other two working modes (*decision* and *prediction and decision*) present a very similar constant overhead of around 0.86 seconds. The *prediction and decision* mode presents the higher overhead of the two (0.89 seconds), which is not surprising given that it presents the most complex brokering process. However, the small difference between this and the *decision* mode is explained by the fact that most of the prediction process (Markov chain calculations, etc.) is performed asynchronously, and therefore adds no direct overhead to the I/O operations time. Therefore, the extra time spent due to the prediction process is insignificant.

Figure 5 shows the different average read bandwidths of each client access mode and file size. Regarding small operations (10 MB), the *random* mode obtains better results since the file size is very small. In *best resource*, *decision* and *prediction & decision* modes, the client requests the data location information from the broker whereas in the random mode the client itself has the required location information. The effective read bandwidth shown is calculated using the request time and the file size transfer. The request time is the sum of the time of the connection to the broker and the decision-making to select the appropriate replicas. The request time affects the 2 broker-based modes and therefore has an impact on their performance. The high influence of the request time due to the low file size means that the best option to read low-size files is to use the random mode.

The 100 MB read operation performance with a *random* mode client has a high average read bandwidth in the transfer time compared to the 10 MB case. Whereas I/O operations with a random mode benefit from the parallel access if several resources are randomly selected, only one resource is used in *best resource* mode, limiting its

performance. Regarding the two proposed decision-based modes, which represent the fundamental contribution of this paper, the average read bandwidths of the *decision* and the *prediction & decision* modes are very close because there are several suitable combinations for this file size in which the bandwidths are not saturated during a long time.

The 1 GB read operation bandwidth in each mode is noticeably different. A subset of the grid testbed resources has considerably higher bandwidth making their intervention decisive. The *random* mode client is clearly penalized because it is equally likely to select either the best or the worst resources. On the other hand, both proposed decision-based mode approaches guarantee the use of optimal resources because decisions are focused on obtaining high-performance. Thus, although the *random* mode client does not contact the broker, eliminating the extra overhead, it takes around 38% more time than *decision* mode client and around 46% time more than *prediction & decision* mode client. In the case of 1 GB read operations, the difference observed between the two decision-based modes is due to the prediction enabling more constant data access in the long term. The *decision* mode client is penalized by the grid changes because write operations do not take into account the behavior prediction. The *prediction & decision* mode client achieves an improvement of around 10% with regard to the *decision* mode client. On the other hand, as in 10 MB and 100 MB file size accesses, the *best resource* mode obtains the worst performance since clients only access a single grid resource. The average improvement of using the parallelism provided by the *decision* mode client vs. the non-parallel access of the *best resource* mode when performing 1 GB read operations is around 143% whereas the use of the *prediction & decision* mode client involves an enhancement of around 175%. From a general perspective, both decision-based modes (the fundamental part of this contribution) obtain significantly better results.

When the file size is increased, performance improves because of the high cost of the initial connection and the request time for broker-based modes. The performance is significantly improved by using the decision-based modes. The *decision & prediction* mode obtains the highest bandwidths for both read and write operations, with their transfer rates exceeding 25 MB/s and 20 MB/s, respectively. These are higher than the theoretical network transfer rate of every storage element of the grid testbed. Despite obtaining slightly worse results than the *decision & prediction* mode, the *decision* mode still presents an important improvement over the other two mechanisms, proving that both key elements of the GAS framework (decision and prediction) are necessary to obtain the highest performance.

These results show how the proposed techniques implemented in GAS can help to improve data access performance. However, to obtain quality of service a certain bandwidth stability in all operations and the ability to adapt to system changes have to be guaranteed. Therefore, not only the average bandwidth has to observed, but also the degree of dispersion of the performance observed in each experiment (basically how close each individual operation is to the average value). To provide a descriptor of this dispersion, Figure 6 shows the different read bandwidth standard deviations for each client access mode and file size.

In the case of small operations (10 MB), the dispersion is negligible, with all deviation values less than 0.1 regardless of the access mode. In this scenario the small

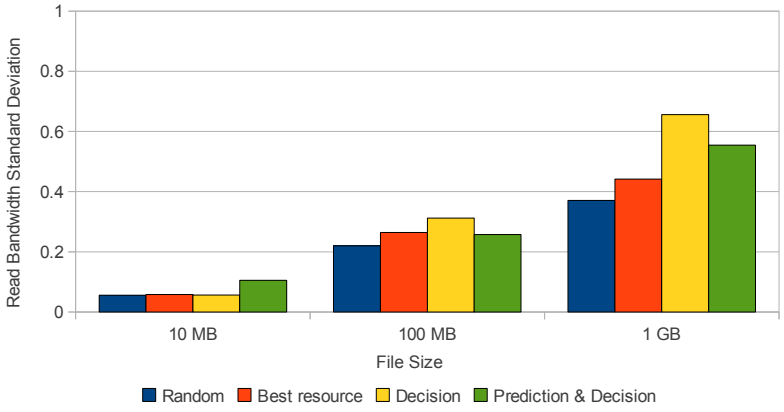


Figure 6: Average Read Bandwidth Standard Deviation

file size and access time make most read operations almost identical, regardless of the resource selected.

In the 100 MB read access scenario higher deviations have been observed, showing that in large I/O operations resource selection is a key factor. However, the dispersion observed is still very low, between 0.2 and 0.3. An important element that can be seen in this scenario is that the *random* mode presents a lower deviation than both decision-based modes. If we consider again the data from Figure 5, showing that the *random* mode performance is clearly lower than the other two mentioned, the conclusion is that this mode always produces low performance results. The *best resource* mode brings up the rear, with even lower performance and a deviation similar to the decision-based modes.

Finally, the 1 GB scenario shows that the significant performance improvement obtained by the decision-based modes is concomitant with an increase in deviation. This is due to the fact that the grid constantly changes, and the best possible performance that can be obtained varies with time. The decision-based modes try to effectively obtain the best performance at each moment, reflecting the variability of the grid infrastructure in their results. However, the deviation values observed are still very low (less than 0.7), guaranteeing a very stable service with substantially better performance than in the *random* and *best resource* modes.

From a general perspective, the increase in file size has an impact on service stability, since resource selection is the key element of operation QoS. The *random* and *best resource* modes generally provide poor results, with low dispersion but also low performance. These modes fail to take advantage of the grid potential and present sub-optimal solutions. Our decision-based proposals (both *decision* and *decision & prediction* modes) show slightly higher dispersion due to the fact that they adapt to the grid variability and try to achieve optimal performance in each situation. The slightly higher deviation observed is still very low, guaranteeing a higher performance and stable operation, and therefore increasing QoS.

7. Related Work

The main difference between the work presented in this paper and the previous work done on data grids is the way write operations are performed. In most systems, files are written locally and once the write is completed the files are either sent to the server that will finally store them or replicated to the right places (according to the replication policies) [2, 3, 24, 41]. Our proposal is based on the idea of providing efficient access to large volumes of data, such as those stored in data grids. Therefore, it is necessary to enhance the quality of data services, writing data in those storage elements that will offer high quality access when the file is to be read.

Although, as far as the authors are aware, no previous work has followed the proposed approach, many projects have tried to model grid resources to be able to make decisions on how and where to replicate data. The first approach, which has been widely used in grid systems, is based on measured performance values. In these systems, the load is measured once (peak value), periodically, or every time an event occurs. This observed performance serves as the basis for making decisions [4, 13, 24, 33, 41]. This approach is based on the assumption that the future will be like the measured state, i.e. that the system is somehow static. A second approach, not widely used in data grids, consists in modeling using analytical models [1, 18, 29, 32, 40]. Another family of models is based on correlation between events. The idea is to be able to predict that some event will occur by the correlation to previous events [7] or workload characteristics [42]. As an evolution of these models, there have also been proposals going beyond building the model, by checking the accuracy of the model and deciding to remodel whenever needed [39]. The proposed models are not adequate for our objective because they cannot predict the behavior at a given time when this time is not known *a priori*, making it impossible to dynamically adapt to changes in the system behavior.

Finally, the main cluster file systems that have autonomic capabilities [15, 16, 19, 26, 43] need to be mentioned. Although these file-systems have different target environments, they are the precursors of autonomic storage for grid systems.

8. Conclusions and Open Issues

Grids, and more specifically data grids, are highly complex systems. The autonomic framework GAS has been designed and implemented to manage this complexity, enhancing the quality of data storage services. Firstly, GAS allows the system to adapt its behavior facing environment changes. Additionally, it adjusts internal parameters to improve the QoS of the overall system and the decision-making processes. This paper describes in depth all the required phases to provide autonomic features: monitor, analyze and plan. The analyze & plan phases are based on data analysis, Markov chain-based prediction and decision-making, whereas the monitoring phase is provided by GMonE. GMonE is a proposed set of services for GAS designed to work together and intended to provide a complete monitoring infrastructure for the grid.

As can be seen in Section 6, the use of GAS provides substantial benefits both in the areas of data access performance and, most importantly, quality of service. These benefits have been observed in real experimental scenarios, further validating the scientific contribution of this autonomic framework. Additionally, the *decision* and *pre-*

diction techniques here presented are based on an *enhanced Markov chain model*, an extension of traditional Markov chain models that addresses the special needs of grid resource behavior modeling.

There are, however, pending tasks to be considered for possible future work. GAS is a broker-based solution, which means that every client query is first submitted to the broker, thus making it a central element (creating a *single point of failure*). To address this problem, GAS could evolve into a distributed architecture using a distributed database, thus allowing more efficiency, availability, scalability and fault tolerance. Furthermore, although the *prediction* phase is relatively short, scalability is a crucial feature since there could be thousands of storage elements in a grid. Thus, an alternative could be to move the prediction capabilities to the server-side and analyzing the possible benefits and drawbacks.

Acknowledgments

This work is partially supported by the Madrid Regional Authority (Comunidad de Madrid) and the Universidad Rey Juan Carlos under the URJC-CM-2010-CET-5185 contract and the Marie Curie Initial Training Network (MCITN) "SCALing by means of Ubiquitous Storage (SCALUS)". We thank all the research centers participating in the grid testbed required for a complete evaluation of this work.

References

[1] Anderson, E., Jul. 2001. Simple table-based modeling of storage devices. Tech. Rep. HPL-SSP2001-4, HP.

[2] Andronikou, V., Mamouras, K., Tserpes, K., Kyriazis, D., Varvarigou, T., 2011. Dynamic qos-aware data replication in grid environments based on data "importance". Future Generation Computer Systems In Press, Corrected Proof, –.

[3] Baru, C. K., Moore, R. W., Rajasekar, A., Wan, M., 1998. The SDSC storage resource broker. In: MacKay, S. A., Johnson, J. H. (Eds.), CASCON. IBM, p. 5.

[4] Cameron, D. G., Casey, J., Guy, L., Kunszt, P. Z., Lemaitre, S., McCance, G., Stockinger, H., Stockinger, K., Andronico, G., Bell, W., Ben-Akiva, I., Bosio, D., Chytracek, R., Domenici, A., Donno, F., Hoschek, W., Laure, E., Lucio, L., Millar, A. P., Salconi, L., Segal, B., Silander, M., 2004. Replica management in the european datagrid project. J. Grid Comput. 2 (4), 341–351.

[5] Chen, P. M., Patterson, D. A., 1990. Maximizing performance in a striped disk array. In: Proceedings of the 17th annual international symposium on Computer Architecture (ISCA '90). ACM Press, New York, NY, USA, pp. 322–331.

[6] Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S., 2000. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Journal Of Network And Computer Applications 23 (3), 187–200.

[7] Cohen, I., Goldszmidt, M., Kelly, T., Symons, J., Chase, J. S., 2004. Correlating instrumentation data to system states: a building block for automated diagnosis and control. In: OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation. USENIX Association, p. 16.

[8] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (Eds.), 2001. Introduction to Algorithms, second edition Edition. MIT Press, McGraw-Hill, Ch. 22.3: Depth-first search, pp. 540–549.

[9] Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C., 2001. Grid information services for distributed resource sharing. In: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC '01). IEEE Computer Society, Washington, DC, USA, pp. 181–194.

[10] Dantzig, G. B., 1963. Linear Programming and Extensions. Princeton University Press.

[11] Domenici, A., Donno, F., Pucciani, G., Stockinger, H., Stockinger, K., Nov. 2004. Replica consistency in a Data Grid. Nuclear Instruments and Methods in Physics Research A 534, 24–28.

[12] Everitt, B. S., 1980. Cluster Analysis. Halsted.

[13] Feng, J., Humphrey, M., 2004. Eliminating replica selection - using multiple replicas to accelerate data transfer on grids. In: ICPADS. IEEE Computer Society, pp. 359–366.

[14] Frankel, S. P., 1950. Convergence rates of iterative treatments of partial differential equations. MTAC 4, 65–75.

[15] Ganger, G. R., Strunk, J. D., Klosterman, A. J., Aug. 2003. Self-* storage: brickbased storage with automated administration. Tech. Rep. CMU-CS03 -178, Carnegie Mellon University.

[16] Halem, M., Schauer, R., 2005. A mass storage system administrator autonomic assistant. In: Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC-05). IEEE Computer Society, Los Alamitos, CA, USA, pp. 300–301.

[17] Hartigan, J., 1975. Clustering Algorithms. Wiley.

[18] Hidrobo, F., Cortes, T., 2003. Towards a zero-knowledge model for disk drives. In: Active Middleware Services. IEEE Computer Society, pp. 122–130.

[19] Hidrobo, F., Cortes, T., 2004. Autonomic storage system based on automatic learning. In: Bougé, L., Prasanna, V. K. (Eds.), HiPC. Vol. 3296 of Lecture Notes in Computer Science. Springer, pp. 399–409.

[20] IBM, August 2004. Autonomic Computing Toolkit. Developer's Guide. IBM - International Business Machines Corporation, 2nd Edition.

[21] Kahan, W., 1958. Gauss-seidel methods of solving large systems of linear equations. Ph.D. thesis, University of Toronto, Toronto, Canada.

[22] Kephart, J. O., Chess, D. M., 2003. The vision of autonomic computing. Computer 36 (1), 41–50.

[23] Ketchen, D. J., Shook, C. L., 1996. The application of cluster analysis in strategic management research: An analysis and critique. Strategic Management Journal 17, 441–458.

[24] Kunszt, P. Z., Laure, E., Stockinger, H., Stockinger, K., 2005. File-based replica management. Future Generation Comp. Syst. 21 (1), 115–123.

[25] MacQueen, J. B., 1967. Some methods of classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathemtical Statistics and Probability. pp. 281–297.

[26] Magoutis, K., Sarkar, P., Shah, G., May 2006. OASIS: Self-tuning storage for applications. In: Fourteenth NAASA Goddard, Twenty-third IEEE Conference on Mass Storage Systems and Technologies. College Park, Maryland, USA.

[27] Markov, A. A., 1971. Extension of the limit theorems of probability theory to a sum of variables connected in a chain. reprinted in Appendix B of: R. Howard. Dynamic Probabilistic Systems. John Wiley and Sons volume 1: Markov Chains, 552–577.

[28] Milligan, G. W., Cooper, M. C., 1985. An examination of procedures for determining the number of clusters in a data set. Psychometrika 50, 159–179.

[29] Montes, J., Sánchez, A., Valdés, J. J., Pérez, M. S., Herrero, P., August 2010. Finding order in chaos: a behavior model of the whole grid. Concurr. Comput. : Pract. Exper. 22, 1386–1415.

[30] Perelmutov, T., et al., January 2007. The storage resource manager interface specification, version 2.2. Lawrence Berkeley National Laboratory.

[31] Pérez, M. S., Sánchez, A., Peña, J. M., Robles, V., 2005. A new formalism for dynamic reconfiguration of data servers in a cluster. Journal of Parallel and Distributed Computing 65 (10), 1134–1145.

[32] Qin, X., January 2007. Design and analysis of a load balancing strategy in data grids. Future Gener. Comput. Syst. 23, 132–137.

[33] Ranganathan, K., Foster, I., 2001. Design and evaluation of dynamic replication strategies for a high performance data grid. In: Proc. of the Int. Conf. on Computing in High Energy and Nuclear Physics.

[34] Ross, R., Thakur, R., Choudhary, A., 2005. Achievements and challenges for I/O in computational science. Journal of Physics: Conference Series 16, 501–509.

[35] Sánchez, A., Pérez, M. S., Montes, J., Cortes, T., 2010. A high performance suite of data services for grids. Future Generation Computer Systems 26 (4), 622 – 632.

[36] Schopf, J. M., September 2005. Distributed monitoring and information services for the grid, bath, UK.

[37] Schopf, J. M., D'Arcy, M., Miller, N., Pearlman, L., Foster, I., Kesselman, C., April 2005. Monitoring and discovery in a web services framework: Functionality and performance of the globus toolkit's mds4. Tech. Rep. ANL/MCS-P1248-0405, Argonne National Laboratory.

[38] Tatebe, O., Morita, Y., Matsuoka, S., Soda, N., Sekiguchi, S., 2002. Grid datafarm architecture for petascale data intensive computing. In: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '02). IEEE Computer Society, Washington, DC, USA, p. 102.

[39] Thereska, E., Narayanan, D., Ailamaki, A., Ganger, G. R., 2007. Observer: keeping system models from becoming obsolete. In: HotAC II: Hot Topics in Autonomic Computing. USENIX Association, p. 10.

[40] Vazhkudai, S., Schopf, J. M., 2003. Using regression techniques to predict large data transfers. Int. J. High Perform. Comput. Appl. 17 (3), 249–268.

[41] Vazhkudai, S., Tuecke, S., Foster, I., 2001. Replica selection in the globus data grid. In: Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID'01). IEEE Computer Society, Washington, DC, USA, pp. 106–113.

[42] Wang, M., Au, K., Ailamaki, A., Brockwell, A., Faloutsos, C., Ganger, G. R., 2004. Storage device performance prediction with cart models. mascots 0, 588–595.

[43] Zhang, Z., Lin, S.-D., Lian, Q., Jin, C., 2004. Repstore: A self-managing and self-tuning storage backend with smart bricks. In: ICAC. IEEE Computer Society, pp. 122–129.