An Adaptive and Historical Approach to Optimize Data Access in Grid Computing Environments

RENATO PORFIRIO ISHII[1]
RODRIGO FERNANDES DE MELLO[2]

[1]UFMS – Federal University of Mato Grosso do Sul
FACOM – Faculty of Computing
P.O. Box 549, 79070-900 Campo Grande (MS) – Brazil
renato@facom.ufms.br,

[2]USP – University of São Paulo
ICMC – Institute of Mathematics and Computer Sciences
P.O. Box 668, 13560-970 São Carlos (SP) – Brazil
mello@icmc.usp.br

Abstract. The data Grid, a class of Grid Computing, aims at providing services and infrastructure to data-intensive distributed applications which need to access, transfer and modify large data storages. A common issue on Data Grids is the data access optimization, which has been addressed through different approaches such as bio-inspired and replication strategies. However, few of those approaches consider application features to optimize data access operations (read-and-write). Those features define the application behavior, which supports the optimization of operations, consequently, improving the overall system performance. Motivated by the need of efficient data access in large scale distributed environments and by the affordable improvements of application characteristics, this paper proposes a new heuristic to optimize data access operations based on historical behavior of applications. Throughout experiments we concluded that applications are better optimized by anticipating different numbers of future events, which vary over the execution. Then, in order to address such issue, we proposed an adaptive sliding window which automatically and dynamically defines how many future operations must be considered to improve the overall application performance. Simulations were conducted using the OptorSim simulator, which is commonly considered in this research field. Our experimental evaluation confirms that the proposed heuristic reduces application execution times up to 50% when compared to other approaches.

Keywords: data access optimization, grid computing, cluster computing, optimization algorithms, resource allocation, modeling and simulation.

(Received February 23rd, 2011 / Accepted May 2nd, 2011)

1 Introduction

The availability of low-cost microprocessors and the computer network evolution have made feasible the development of distributed systems. In such systems, processes communicate one another in order to perform the same computing task. Besides reducing costs, those systems are scalable and more flexible than real parallel machines [7].

Concepts of distributed systems motivated the development of cluster computing, where resources are usually interconnected in a local area network [26]. These cluster environments have encouraged researches

on process scheduling optimization, data prefetching, distributed file systems, fault tolerance and security. As those systems became more available as well as the Internet has allowed the access of long-distance resources, scientists started interconnecting them to solve more complex problems [33]. That approach started the development of grid computing environments, in which resources are usually heterogeneous, geographically distributed and accessible to several users [33]. The intrinsic features of these platforms have required new researches on job scheduling, data access, fault tolerance and security strategies [33]. Besides the evolution of every topic, the Data Access Problem (DAP) is still a major concern when dealing with grids, mainly due to data location and consistency [8]. Other issues that should be considered are the data migration, replication, distribution and also the evaluation of the data access impact on job scheduling approaches [11].

Besides considering the aforementioned topics, related works (Section 2) present three main drawbacks. The first is that most of the works consider exclusively read-only operations [5]. Such situation tends to restrict the execution of real-world applications. They mainly neglect such subject due to the complexity involved in keeping data consistent. The second drawback is that many works consider static data access optimization approaches, i.e. they do not adapt themselves according to dynamic grid computing features, such as users logging into and out, computers connecting and disconnecting. The last drawback is that the overall system performance depends on data access patterns, what varies according to job system calls. In this way, it is very important to understand, estimate and/or predict the job behavior as a way to optimize read-and-write operations.

The three main drawbacks of related works have motivated an initial investigation [19], in which we evaluated a heuristic to optimize job read-and-write operations on distributed environments, based on applications historical behavior. In that paper, we optimized next file access operations by considering a window of future events (every event corresponds to a file access operation). However, in that study, we considered fixed-length windows and confirmed they result in an unsatisfactory response time to grid environments, mainly due to the high heterogeneity of computational resources (CPUs, hard disks and networks) and the variation of reading and writing operations during the process execution. By considering initial experiments presented in that paper, we observed that the interposition of read-and-write events (i.e., events of different types under the same window length) affects the efficiency of the window. Thus, we were motivated to investigate whether a window of future events under the same type of operation could improve data accesses and, therefore, reduce costs.

Based on such hypothesis, this paper proposes an adaptive sliding window to optimize data access by improving replication, migration and consistency decisions. An adaptive window is defined over the historical behavior of applications, and it represents the number of future events that our heuristic analyzes to optimize decisions. The adaptive window constantly adjusts its length according to the dynamic behavior of processes, i.e., the number of similar operations (readings or writings) under execution. As this approach considers historical job behavior, applications need to be executed at least once.

The specific contributions of this paper are: 1) the formalization of the Data Access Problem (DAP); 2) an analytical optimization model to address DAP, aiming at minimizing the overall application execution times; 3) proposal of an adaptive sliding window approach to define how many historical events are considered in the optimization process. 4) simulations to evaluate the efficiency of the adaptive sliding window under a wide range of environments and system configurations (how read-and-write operations are distributed over time).

Besides all the listed contributions, experimental results confirm that this new adaptive heuristic outperforms other commonly considered ones (e.g., LRU, LFU and Economic Model, presented in Section 5) in approximately 50% when dealing with grid environments.

This paper is organized as follows: Section 2 reviews related work; Section 3 models the Data Access Problem (DAP); the proposed adaptive heuristic is presented in Section 4; Section 5 presents simulation results and, finally, concluding remarks.

2 Related Work

Several studies have been conducted to improve data access on grid environments. Such works are mainly focused on data replication, distribution and consistency.

Oliker et al. [27] propose a static data allocation approach and three data-oriented job scheduling algorithms (SI, RI, SYI). The approach attempts to optimize the system overall performance by allocating jobs where data is available. Among the evaluated scheduling algorithms, SI reduces the average execution time by 60% when compared to local approaches, and can execute 40% more jobs.

Rahman et al. [28] present a model which uses a simple data-mining approach (K-Nearest Neighbor,

KNN) to select the best replicas from grid sites. To select the best replica, the authors design an optimization technique that considers the network latency and the disk state. Different file access patterns are investigated and compared to the KNN algorithm. KNN shows a performance improvement for sequential and unitary random file access patterns.

Sun & Xu [34] propose two consistency algorithms: Lazy-Copy (LC) and Aggressive-Copy (AC). LC updates replicas only when needed, i.e., when a user requires them. This may reduce the bandwidth consumption, avoid unnecessary transfers when data are modified but not required. AC updates replicas whenever a change occurs in the original file. Therefore, AC fully guarantees the consistency, while LC partially guarantees it. In the comparison between the two algorithms, the Aggressive-Copy reduces the access latency, whilst Lazy-Copy reduces the bandwidth consumption.

Wang et al. [36] consider the parallel access of data replicas. The access time is minimized by overlapping requests, what tends to increase the throughput. The proposed solution (called MSDT) carries on replicating data in idle intervals as a way to improve system performance. That work does not consider data consistency. Results present a speedup factor in the range of $2.72 \sim 3.06$ when comparing MSDT to another technique (called NoObserve) [15].

Oldfield & Kotz [26] propose the Armada framework which launches applications and defines how files are distributed. It also provides access control and data access mechanisms. It builds graph structures to represent the processing and data flow. Experiments compare restructured applications to original ones. Armada improves throughput of wide-area networks in 40% when compared to original applications.

Dang & Li [11] propose a tree-type structure to correlate data in grid regions aiming at reducing file transfers. When a job needs a file, the approach looks for high correlation data before asking for transfers. This reduces network costs and improves the overall application performance.

Elghirani et al. [14] define a data management service to replicate files on sites as well as a Taboo Search approach to schedule jobs aiming at optimizing runtime and system utilization. The Taboo Search attempts to find good data replication solutions, considering job data accesses and processing time. Results present performance improvements from 8% to 35%, depending on the replication and job scheduling approaches.

Kim et al. [22] propose a technique to improve data access, matching nodes to the best remote data sources. The authors consider a trace-based synthetic scenario on PlanetLab to evaluate their heuristic. Results show that the resource selection outperforms conventional techniques such as latency-based or random allocations.

Chervenak et al. [10] propose a framework called Replica Location Service (RLS) which maintains and provides information on physical locations of replicas. RLS is used in a variety of production environments such as the Laser Interferometer Gravitational Wave Observatory (LIGO) [3], Earth System Grid (ESG) [1] and Pegasus [12]. Authors presented a performance study demonstrating that the individual RLS servers have performed well and scale up to millions of entries compared to the native MySQL using ODBC clients.

AL-Mistarihi & Yong [4] propose an approach to address the replica selection problem. The authors consider the Analytical Hierarchy Process (AHP) to solve that problem, and they evaluate this approach in an extension of the OptorSim simulator. AHP was employed to solve this optimization problem using a simplification of multiple objectives into a single one. However, the authors evaluated AHP comparing it only against a random approach. They could and should at least compare the performance of AHP against strategies included in the OptorSim simulator, such as LRU, LFU and the Economic Model.

It is important to observe that all previous presented studies do not consider the dynamic behavior of applications when taking decisions, likewise, they do not take advantage of future read-and-write operations to optimize data accesses.

The dynamic behavior of applications motivated Ishii & Mello [19] to propose a heuristic that adopts a fixed length window of future events which aims at anticipating reading and writing operations. Using future events, the heuristic optimizes decisions on replicating, migrating and keeping consistency. This study confirmed that windows of future events can indeed improve application performance in some scenarios. For example, when considering environments with read-only operations, the heuristic improved as much as 100%, however under a low frequency of writing operations (5% of writing and 95% of reading operations) and low frequency of reading operations (95% of writing and 5% of reading operations), the heuristic could not reduce application execution times.

In this previous study [19], the window length is fixed and defined by the system administrator. Based on that we attempted different window lengths and analyzed experimental results, such additional work motivated us to study whether a window of future events, under the same type of operation, could improve data

accesses and, therefore, reduce application execution times. Thus, in this paper, we propose an adaptive sliding window to optimize data accesses by improving replication, migration and consistency decisions.

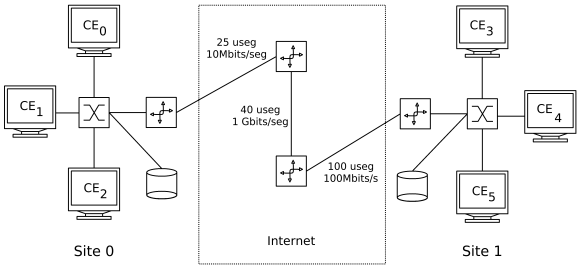

Figure 1: Example of network interconnection

3 The Data Access Problem

This section aims at defining the Data Access Problem (DAP). We start with an empirical case study and then we proceed with the formal definition of DAP.

3.1 An Empirical Case Study

In order to empirically state the DAP, we propose the following hypothetical case study: let two parallel applications be composed of the processes presented in Table 1, where MI represents the million of instructions executed by processes (during their lifecycles); MR and MW are, respectively, the number of KBytes/sec read and write from/into the main memory; HDR and HDW are, respectively, the number of KBytes/sec read and write from/into the hard disk (or secondary memory); NETR and NETS are, respectively, the number of KBytes/sec received and sent through the computer network – in this case, the sender (for NETR) and the receiver (for NETS) processes are presented. For example, Table 1 row 1 shows that process p_0 consumes $1,234$ MI, reads 123 KBytes/sec (and does not write data into memory, i.e. MW$=0$), it still reads 78 KBytes/sec from the hard disk (and it does not write data into the hard disk, i.e. HDW=0), receives 12 KBytes/sec from process p_1 and sends 532 KBytes/sec to process p_1 .

Now consider that these processes are allocated on a set C of computers which is described in Table 2, where CE is a computing element; MIPS represents the processing capacity in million of instructions per second; TMR and TMW are, respectively, the read-and-write throughput of the main memory (in KBytes/sec); THDR and THDW are, respectively, the read-and-write throughput of the hard disk (also in KBytes/sec). Let

Table 1: Processes behavior

App 0

	MI	MR	MW	HDR	HDW	NETR	NETS
p_0	1,234	123	0	78	0	$12 - p_1$	$532 - p_1$
p_1	1,537	23	89	0	12	$532 - p_0$	$12 - p_0$

App 1

	MI	MR	MW	HDR	HDW	NETR	NETS
p_2	1,221	823	70	78	543	$10 - p_3$	$321 - p_4$
p_3	1,137	223	179	324	212	$423 - p_4$	$10 - p_2$
p_4	2,237	23	17	12	0	$321 - p_2$	$423 - p_3$

the allocation operator be defined by $\propto$, where an example of allocation, assuming an application composed of 5 processes, is given by $p_0 \propto \texttt{CE}_0$, $p_1 \propto \texttt{CE}_1$, $p_2 \propto \texttt{CE}_2$, $p_3 \propto \texttt{CE}_3$ and $p_4 \propto \texttt{CE}_4$. Computers in C are interconnected according to Figure 1, which also presents an example of network bandwidths and latencies.

Table 2: Grid site characteristics

CE	MIPS	TMR	TMW	THDR	THDW
$\texttt{CE}_0$	1,200	100,000	40,000	32,000	17,000
$\texttt{CE}_1$	2,100	120,000	50,000	42,000	19,000
$\texttt{CE}_2$	1,800	100,000	30,000	22,000	9,000
$\texttt{CE}_3$	1,700	95,000	20,000	25,000	11,000
$\texttt{CE}_4$	2,500	110,000	60,000	62,000	30,000
$\texttt{CE}_5$	2,000	110,000	45,000	40,000	17,000

Also consider that all 5 processes (described in Table 1) of the 2 parallel applications access a set of files $F = \{f_0, f_1, \ldots, f_9\}$. In order to solve the DAP, in an optimal way, we must explore all possible file distributions over the 6 computing elements and evaluate the access cost for every process p_i. In such situation, permutations would be performed to find out all possible solutions. For example, let a solution where the subset of files $\{f_0, \ldots, f_6\}$ is allocated on $\texttt{CE}_0$ and all others, i.e. $\{f_7, f_8, f_9\}$, are placed on $\texttt{CE}_1$. The set of all possible solutions, for any instance of the problem, is obtained by computing n^z, where n is the number of computing elements and z is the number of files. Consequently, in the previously mentioned instance, the solution space is equal to $6^{10} = 60,466,176$.

Besides the presented example, it is necessary to study the problem in real-world conditions. For illustration purposes, assume that the target environment contains more than 256 computers. In such situation, the

expected problem solution space would follow the orders of magnitude defined in Table 3, in which we expect to address thousands of computers storing millions of files.

Table 3: Solution space to distribute files over computing elements

CE's	# files	Solutions: n^z
256	320	$\approx 4 \times 10^{770}$
256	3,200	$\approx 2 \times 10^{7706}$
256	32,768	$\approx 1 \times 10^{78913}$
512	320	$\approx 9 \times 10^{866}$
512	3,200	$\approx 4 \times 10^{8669}$
512	32,768	$\approx 2 \times 10^{88777}$
1,024	320	$\approx 1 \times 10^{963}$
1,024	3,200	$\approx 9 \times 10^{9632}$
1,024	32,768	$\approx 3 \times 10^{98641}$

3.2 Formal Definition

A grid computing environment can be represented as a non-directed graph $G = (S, L)$, where the set of vertexes S represents the grid sites (networks of workstations, parallel machines, clusters, etc.) and the set of edges L is composed of communication links in between grid sites. Also consider a set F containing z files, whose sizes are modeled by function $\theta(.)\colon F \rightarrow \mathbb{Z}^+$. Each grid site $s \in S$ has none, one or more elements with the storage capacity defined by $\Omega\colon S \rightarrow \mathbb{Z}^+$ which is the size in MBytes of the storage element into a grid site s. Function $\alpha_{j,i}\colon F \times S \rightarrow \mathbb{Z}^+$ defines the cost of storing file j on site s_i. In the same way, the function $\delta(.)\colon F \times S \rightarrow \mathbb{Q}^+$ considers the links in L to model the communication cost in between grid sites. Function α is constrained such as $\alpha_{j,i} \leq \Omega_i \ \ \forall j, i$.

Consider the set A which contains k parallel applications executing on a large scale distributed environment. Let the function $\phi(.)\colon A \rightarrow \mathbb{Z}^+$ represent the number of processes of each parallel application. Consider set P which contains all processes of all k parallel applications, i.e. the number of terms in P is equal to $h = \sum_{a \in A} \phi(a)$. All processes are previously allocated on the distributed system according to any scheduling criterion. Every element in P has a set R associated, which contains m read-or-write file requests. A process contains particular features, here defined as behavior, such as processing, memory and input-and-output utilization. Every process, consequently, requires different amounts of resources provided by set S of grid sites as well as the set of communication links L. A process $p \in P$ may access none, one or many files in F.

The cost to transfer file f in between two grid sites s and s' is modeled in Equation 1. It depends on file size $\theta(f)$ and distance $d(s, s')$ in between both sites, which is measured in terms of the network latency and bandwidth. Cost ψ is assumed when a constraint forbids the replication of file f to grid site s'.

$$\delta(s, s') = \begin{cases} \theta(f_j) d(s, s') & \text{where } s \in S,\ j = 1, \ldots, z \\ \psi = \infty & \text{otherwise} \end{cases} \tag{1}$$

A file f must be transferred from the shortest path site $s \in S$ whose cost is minimum and defined by function $d(s, s')$. The access cost for reading a file f, stored in a site s, is, therefore, given by Equation 2. Write operations induct replica updates for every file $f \in F$, which consumes resources as described in Equation 3.

$$r_{\text{cost}}(f) = \delta(s', f) = \arg\min(\theta(f) d(s, s')) \ \forall\ s \in S \tag{2}$$

$$w_{\text{cost}}(f) = \sum_j \delta(s', f_j) \ \forall j \text{ local and remote replicas} \tag{3}$$

By unifying Equations 2 and 3, the total cost to access a file f is determined in Equation 4, considering all processes in P. Table 4 describes each model parameter, afterwards the data access problem is formalized.

$$\Lambda(f) = \sum_{p \in P} \sum_j r_{\text{cost}}(f_j) + w_{\text{cost}}(f_j) \tag{4}$$

Consider a set P of processes which were previously scheduled on grid sites in S with storage capacities defined by α and transfer costs by $\delta(s, s')$. Let a set of files F with a given initial file distribution x_{ij} and size $\theta(f)$. Assume a set of quintuples TR which describes read-or-write operations on files in F. The optimization problem, DAP, consists in determining a new file distribution y_{ij} according to the energy function defined in Equation 5, which is constrained according to Equations 6 and 7 and follows the domains: $x_{ij} \in \{0, 1\},\ \ \forall\ i, j$ and $y_{ij} \in \{0, 1\}\ \ \forall\ i, j$.

$$\Gamma(\textbf{DAP}) = \min \sum_i^n \sum_j^z (x_{ij} - y_{ij}) \Lambda(f_j) \tag{5}$$

Table 4: Model parameters

Parameter	Description
G	The graph of environment $G(S, L)$
S	Set of grid sites
s	Grid site which contains computers, workstations, nodes or similar
n	Number of grid sites
L	Set of communication links
l	Data link in between sites, e.g. $\{s, s'\}$
$\alpha(s)$	Cost function to store data on site s
$\delta(s, s')$	Communication capacity in between s and s'
$\theta(f)$	Size of file f
Ω_i	Total storage capacity of site s_i
A	Set of parallel applications
a	An application
k	Number of applications
$\phi(a)$	Function which determines the number of processes of application a
P	Set of processes
h	Total number of processes
R	Set of read-and-write requests
r	Read-or-write request to a file f
m	Total number of requests
F	Set of files
f	A file
z	Total number of files
i	Grid site index in S ($s_i \in S$)
j	File index in F ($f_j \in F$)
x_{ij}	Equals to 1 if f_j is stored on site at index i. 0, otherwise
y_{ij}	Equals to 1 if the new solution allocates f_j on the site at index i. 0, otherwise
TR	Set of quintuples to describe read-and-write operations to files in F
tr	A quintuple in TR
u	Total number of quintuples

$$\sum_{i}^{n} x_i f_j \leq n \tag{6}$$

$$\sum_{j}^{z} y_{ij} = 1, \;\; i = 1, \ldots, n \tag{7}$$

The energy function (Equation 5) attempts to reduce the cost to request files, considering the distance in between grid sites, amount of data and storage capacity. The constraint presented in Equation 6 limits the number of file replicas, which can not surpass n sites, i.e. if $j = n$ all storage elements in the grid have one replica of f each. The constraint given by Equation 7 defines that every replica must be allocated on one site only. This constraint implies that storage elements do not have more than one replica of file f. All computing elements connected to a grid site access the replica on the storage element of that grid site.

In order to prove that the data access problem is NP-complete, we demonstrate that it is contained in the NP set and it is NP-hard. To demonstrate that the DAP is in NP, a reduction is conducted from the allocation of multiple copies of the same file on a distributed environment (here called Multiple File Allocation (MFA)) which is proven to be NP-complete according to [16]. The MFA problem is defined as:

Instance: *Graph $G(V, E)$, for each $v \in V$ a usage $u(v) \in \mathbb{Z}^+$ and a storage cost $s(v) \in \mathbb{Z}^+$, and a positive integer K.*

Question: *Is there a subset $V' \subseteq V$ such that, if for each $v \in V$ we let $d(v)$ denote the number of edges in the shortest path in G from v to a member of V', we have:*

$$\sum_{v \in V'} s(v) + \sum_{v \in V'} d(v) \times u(v) \leq K? \tag{8}$$

Theorem 1. *DAP is NP-complete.*

Garey & Johnson [16] present three different approaches to prove NP-complete problems: restriction, local replacement and component design. Consider a problem $\Pi \in NP$, the proof using the restriction approach consists in showing that Π contains a known NP-complete problem Π' as a special case. The main idea lies in the specification of additional restrictions on instances of Π, so that the resulting problem will be identical to Π'. The problem Π does not need to be exactly the same as Π', but it must preserve yes-and-no correspondence in their outputs. In local replacement, we must identify the components, or building blocks, which integrate the instance of a known NP-complete problem Π'. In order to prove that a problem Π is

NP-complete, we look for similarities of the Π' basic components and relate them to the Π problem. The last type of proof is the component design, which considers the constituents of the target problem instance to design components to be combined and represent the instance of the already known NP-complete problem.

Proof. In this paper, we prove that DAP is NP-hard by using the restriction-proof approach. DAP is contained in NP due to it is possible to build a non-deterministic machine to verify the graph $G(S, L)$ in polynomial time. When building the machine, the following is non-deterministically defined: for each $s \in S$ an allocation x_{ij} in which the file f_j is mapped, and, for each $l \in L$ the cost $\alpha(s, s')$. The verification step checks out, for each $f \in F$, whether the grid site mapping is valid, this is, if $\sum_{j}^{n} x_j \leq n$. Finally, the condition $\sum_{j}^{n} y_{ij} = 1, \quad i = 1, \ldots, n$ is evaluated for the new allocation of files in F.

Consequently, an instance of DAP is translated into a MFA one. We initially map every element of the set V in elements in S. Then, we define the number z of files. Those files are initially allocated on sites in S, in a random way. This file allocation respects the constraint that each grid site $s \in S$ has the maximum storage capacity of $s(v)$. Besides that, $d(v)$ is mapped into $d(s, s')$ what represents the cost to transfer file $f \in F$ in between two grid sites s and s', and the function $u(v)$ is mapped into $\theta(f)$, which models the file size. For every element $e \in E$, there is a correspondent $l \in L$ with network latency and bandwidth associated.

In the MFA problem, Equation 8 formalizes the objective function which aims at minimizing costs related to the allocation of multiple copies of the same file. This equation is bounded by K. Similarly, in the DAP problem, the objective function is defined by Equation 5 which also attempts to minimize access costs.

Finally, consider an instance for the DAP where: given a request $r \in R$ which is launched by a process $p \in P$ when reading or writing on a file $f \in F$. For any instance of DAP where $|S| \geq 2$ and $|L| \geq 2$, the problem presents exponential characteristics and it is considered NP-complete.

□

4 Data Access Approach

After stating the DAP, we may address it by using exact or approximation approaches. Exact approaches guarantee optimal solutions for the problem. However, they may be very time consuming, depending on the problem and the instance. For small instances, they might offer acceptable run-time, but, for large instances, they are prohibitive [35]. This fact motivated the development of approximation strategies considering heuristics and metaheuristics. A heuristic is an algorithm that, based on the problem knowledge or experience, leads to appropriate solutions, but there is no guarantee to obtain optimal solutions [16].

Heuristics are commonly considered due to their trade off in between the solution quality and the time complexity. Metaheuristic is a type of heuristic to solve a class of problems and not only a specific one. Examples of metaheuristics are: Genetic Algorithms (GA) [18], Ant Colony Optimization (ACO) [13] and Simulated Annealing (SA) [23].

Genetic algorithms have been used as search and optimize techniques in several domains [30]. They are based on the natural selection theory, which guarantees the survival of the most adequate individuals, i.e. the ones that represent good solutions. This approach does not ensure optimal solutions for all problem instances, but provides appropriate solutions for a reasonable number of NP-Complete problems [30].

ACO-based algorithms support the search for paths in between a given source and a destination. This bio-inspired approach is based on the stigmergy strategy and the pheromone concentration. Stigmergy is a communication mechanism used by ants to coordinate global functions. As ants randomly walk looking for food, they lay down pheromone (chemical component released by ants) on trails. When another ant is looking for the food path, it has a certain probability to follow the previous crossed path (according to the pheromone concentration). This approach iteratively reinforces good paths, supporting the search for shortest-path solutions on graphs.

Simulated Annealing aims at finding a global minimum for a given energy function [23]. This nomenclature comes from an analogy to the metallurgic process of annealing, which consists of the controlled heating and cooling of materials as a way of finding stable energy states. Those states, for instance, help metallurgic processes to reduce physical defects on different materials. This technique introduces the system temperature concept, which defines the annealing scheduling (heating and cooling operations). SA supports the search of global minimal of energy functions.

After stating the DAP, we may address it by using metaheuristic, which is a type of heuristic to solve a class of problems and not only a specific one. Examples of metaheuristics are: Genetic Algorithms (GA)

[18], Ant Colony Optimization (ACO) [13] and Simulated Annealing (SA) [23]. However, one issue prevents the adoption of such techniques, which is the dynamic behavior of data accesses. Those metaheuristics would model the DAP by evaluating the current accesses as well as future ones. However, their objective functions would compose such information in the current moment, in a way that they tend to optimize the average behavior. Therefore, such approaches may privilege remote accesses while they could improve the system overall performance by replicating data and supporting local operations. Better solutions could be obtained by assessing behavior changes and using them to anticipate data operations. In order to develop such method, we must identify the dynamics involved in the process behavior.

Given the disadvantages presented by the metaheuristics, we propose a novel heuristic to approach the data access optimization using historical application behavior and the adaptive sliding window length. As the heuristic considers the historical process information, we can anticipate process read-and-write operations and, therefore, replicate data locally before needed, what tends to reduce access costs. When data is locally available, read-and-write operations execute faster, what avoids access delays. The anticipation of process events is used to take decisions on data replication, migration and consistency.

The proposed approach follows the workflow defined in Figure 2, which is composed of the modules: 1) application knowledge acquisition; 2) adaptive sliding window length; and 3) the Heuristic. The following sections describe the features of each module, including their integration.

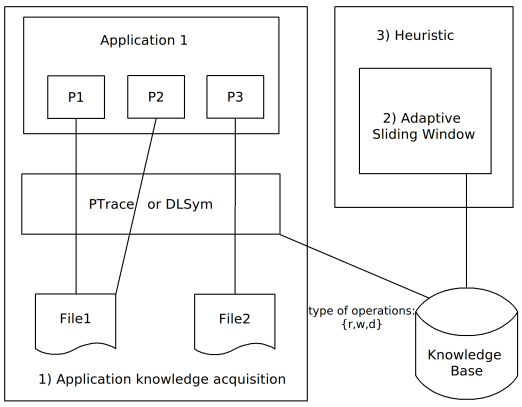

Figure 2: The proposed approach

4.1 Application Knowledge Acquisition

This paper considers applications behavior to guide a novel heuristic to improve decisions on replication, migration and consistency of files. The behavior is composed of read-and-write operations issued by application processes.

There are two approaches to extract process behavior: monitoring and event interception. Monitoring periodically requests information of a given system. Examples of monitoring tools are SMART (Self-Monitoring Analysis and Reporting Technology) [31], Linux Vmstat [37] and Tcpdump [20]. System monitoring is usually based on counters. Counters are variables which account the system utilization within specific time intervals. For example, consider SMART. Let it request the number of hard disk reads and obtain 0 at a given instant. After 10 seconds, it asks for the information again and the system returns 5, what means that five read operations were executed in between the first and the second monitoring points. The monitoring approach grabs the amount of data and what kind of event has happened in a certain time interval, but it does not inform when exactly an event happened and its detailed information. The second approach intercepts process events and calls a procedure to inform about them (this approach obtains the exact event moment as well as its detailed information). However, depending on the event complexity (such as process system calls) and cost, the interception may become prohibitive. Examples of the intercepting tools are the Unix DLSym [21] and Ptrace [32].

DLSym allows the interception of dynamic procedure calls (from dynamic libraries). This is, when the program calls a function, instead of having the code internally, it loads a shared library and runs the procedure. This approach avoids procedure rewriting and also allows the interception of calls. Consequently, any procedure in a dynamic shared library can be intercepted, what helps to build monitoring tools. On the other hand, Ptrace transparently intercepts process signals and system calls. It is usually employed to build diagnosis and debug tools. Ptrace does not use dynamic libraries and can intercept any Unix application.

Researchers must evaluate the options in between monitoring and intercepting and choose the best approach to address the system under study. Some systems can only be monitored, while others, only intercepted. Given the physical characteristics, hardware are usually monitored, while software can be monitored or intercepted. In this paper, the interception approach was chosen due to it allows the continuous extraction of application behavior over time. Furthermore, the monitor-

ing approach may hide behavior in between sampling intervals.

After extracting the application behavior, we transform the sequence of events in series of numerical values which represent time instants. Every event is described by the quintuple $\mathrm{tr} = \{\mathtt{pid}, \mathtt{inode}, \mathtt{amt}, \mathtt{time}, \mathtt{op}\}$ where `pid` is the identifier of the process that performs the operation, `inode` is the file identifier, `amt` is the amount of data read or written, `time` is the time interval in between operations and `op` is the operation type (whether reading or writing). A series for u sampling intervals is, therefore, defined as $\mathrm{TR} = \{\mathrm{tr}_0, \mathrm{tr}_1, \ldots, \mathrm{tr}_{u-1}\}$ which is used to analyze relations among events. Those events are included in a trace file for future usage. Table 5 presents an example of a trace file.

Table 5: Example of trace file

index	pid	inode	amt	time	op
1	p_{41}	f_0	313	24	**r**
2	p_{89}	f_3	94	171	**w**
3	p_{10}	f_9	92	80	**w**
4	p_{32}	f_0	826	132	**w**
5	p_{69}	–	–	76	**d**
6	p_1	f_5	292	70	**w**

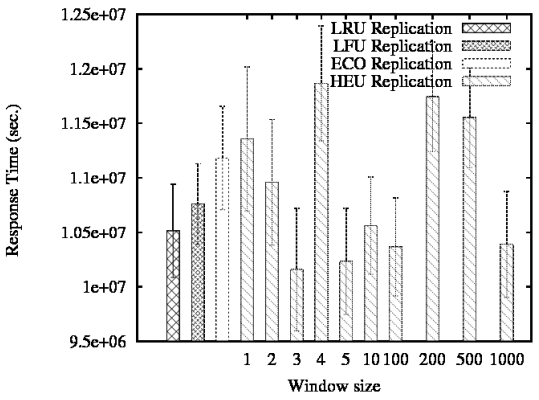

Figure 3: The impacts of sliding window variation

4.2 Adaptive Sliding Window Length

The sliding window length defines the number of future events that the heuristic analyzes to optimize decisions, which impact on the system performance. An example of the impact the sliding window has on applications execution is shown in Figure 3, that considers a sliding window approach with fixed length. Four techniques were evaluated: LRU, LFU, Economic model (see Section 5), and the heuristic with fixed window length (label HEU in Figure 3) [19].

We observe that, for the sliding window length 4, the application execution time is higher. We observed this fact occurs due to the heterogeneity and the interposition of writing and reading operations. This also happens for sliding windows 200 and 500, where there is also high heterogeneity in terms of operations. We experimentally confirmed that the more homogeneous is the execution order of reading and writing operations (i.e. similar operations are arranged consecutively), the longer can be the prediction horizon, i.e. more future events can be considered and, therefore, we can set larger window lengths.

This fact motivated this work that proposes of an adaptive sliding window approach which considers the type and number of operations. Thus, we proposed parameter β to adapt the window length according to consecutive homogeneous operations (i.e. reading or writing). Equation 9 defines the Adaptive Sliding Window (ASW), where W_{t+1} is the next window length, W_t is the current window length, Op is the number of homogeneous operations and β is the factor which determines modifications in the window length.

$$W_{t+1} = W_t \times (1 - \beta) + \mathrm{Op}^{2 \times \frac{\mathrm{Op}}{W_t}} \times \beta \qquad (9)$$

For example, given $W_1 = 10$, $\mathrm{Op} = 4$ and $\beta = 0.10$, we would obtain a next window length equals to $W_2 = 9$. Note that the window adapts according to the number of operations under the same type, see Figure 4. On the other hand, when Op is equal to W_t (see W_4 in Figure 4) the next window length increases considerably, $W_5 = 11$. Parameter β is experimentally defined (as presented in Section 5.4).

Figure 4 presents a sample trace file with process behavior information (in this example we focused only in reading (r) and writing (w) operations) obtained through the interception approach (Section 4.1). The first part of the Figure 4 shows the initial window length ($W_1 = 10$) and all consecutive operations. The second part shows how to compute of the next sliding window length based on Equation 9. Finally, the last part shows the subsequent windows $W_2 = 9, W_3 = 8, W_4 = 7$ and $W_5 = 11$, which clearly demonstrates the adaptation of window length according to the behavior of reading and writing operations.

Initializing the Adaptive Sliding Window

$$\overbrace{rrrrwwrwwr}^{W_1=10}rwrrwwrwwwwwwwrrwrrrrrrr$$

Calculating window length

$$\underbrace{rrrr}_{\text{Op=4}}wwrwwr \rightarrow W_2 = 10 * (1-\beta) + 4^{2*\frac{4}{10}} * \beta = 9$$

$$\underbrace{r}_{\text{Op=1}}wrrwwrww \rightarrow W_3 = 9 * (1-\beta) + 1^{2*\frac{1}{9}} * \beta = 8$$

$$\underbrace{wwwww}_{\text{Op=5}}rrw \rightarrow W_4 = 8 * (1-\beta) + 5^{2*\frac{5}{8}} * \beta = 7$$

$$\underbrace{rrrrrrr}_{\text{Op=7}} \rightarrow W_5 = 7 * (1-\beta) + 7^{2*\frac{7}{7}} * \beta = 11$$

Window length after executing the ASW approach

$$\overbrace{rwrrwwrww}^{W_2=9}\overbrace{wwwwwrrw}^{W_3=8}\overbrace{rrrrrrr}^{W_4=7}\overbrace{rwrrrwwrrwr}^{W_5=11}$$

Figure 4: Example of the Adaptive Sliding Window with $\beta = 0.10$.

By using this mechanism, we compute the number of similar operations and define the length of the next window of future events. Thus, it limits how many future events will be analyzed. Then, we consider the window to anticipate data transfers, bringing information locally and, therefore, reducing future access costs.

Without the adaptive mechanism, we should set a fixed length for the sliding window, which is only possible experimentally (requiring the evaluation of a range of values for the length). Besides that, a fixed length might be good for part of the application execution, while it would reduce the performance in other situations (time instants). Therefore, the length needs to be dynamically adapted according to the instantaneous behavior of the process (we consider one specific adaptive window per process). Moreover, using the adaptive mechanism, the window fits the process behavior over time, making it more flexible and efficient.

4.3 The Proposed Heuristic

Let $P = \{p_0, p_1, \ldots, p_{h-1}\}$ be the set of all processes previously scheduled on grid sites, $F = \{f_0, f_1, \ldots, f_{z-1}\}$ be the set of files to be accessed, and $O = \{r, w, d\}$ (where r is read, w is write and d represents idle moments) represent the set of types of operations. Let TR model the series of process behavior, where each element tr $\in$ TR describes an operation $\mathtt{op}$ over a file f, executed by process p.

Thus, we anticipate data access operations over TR by using an adaptive sliding window ASW $\rightarrow [t_y, t_x]$ (considering $x > y$). This sliding window models the number of future events (process behavior) considered when optimizing process data-access operations.

The mechanism of anticipation aims at transferring files before they are requested by jobs in execution. This mechanism reduces the access cost due to files are requested beforehand and they are locally replicated. Consequently, all this scenario reduces the total execution time of applications. In Figure 5, a file will be requested at t_1, and it was totally transferred in advance. On the other hand, in Figure 6, the file is requested in t_1, however the transfer was not completed and, in this case, the heuristic still needs to transfer the spare data. In this last scenario, the total cost is $t_2 - t_1$.

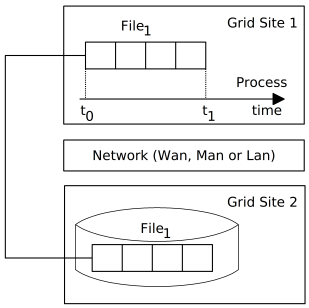

Figure 5: The file is completely transferred

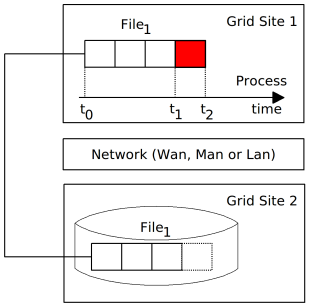

Figure 6: The file is partially transferred

As previously presented in Table 5, every trace line (event) corresponds to a quintuple tr $\in$ TR. Thus, the proposed heuristic (Algorithm 5) considers the historical process behavior, represented by TR, and a given sliding window ASW of future operations to optimize data accesses on distributed environments. The anticipation of future operations supports the minimization of communication/synchronization costs related to file transfers and consistency.

In this way, consider a process p which executes operations over a file f. Now, let p be at the current execution or time instant t_c and there is a window of events up to the time instant t_f, where $t_f > t_c$. The proposed heuristic evaluates if file f can be retrieved (i.e. to find the latest version of a replica f') to the local site where it is needed. If so, transfer costs and access delays are reduced and the process performance is improved. In this circumstance, there are two ways of retrieving f, both imply cost Γ (Equation 5):

1. if $t_c + \Gamma \leq t_f$, then we can completely retrieve file f, which reduces the process execution time, in Γ units, to access it;

2. if $t_c + \Gamma > t_f$, then we can reduce $t_f - t_c$ time units in overall process execution.

The proposed heuristic is presented in Algorithm 5 and in other auxiliary procedures as follows (procedures listed in Algorithms 1, 2, 3 and 4).

The method `Retrieve`(f) (Algorithm 1) establishes a new local copy (replication) of the remote file f and assesses the copy removal option (for data migration purposes). The method `Read`(tr) (Algorithm 2) receives a quintuple tr and, if file f is *invalid* then it applies the method `Retrieve`(f) and finally reads it. The method `Write`(tr) works similarly (Algorithm 3). The method `Invalidate`(f) (Algorithm 4) receives a file and updates all replicas as *invalid*.

Algorithm 1 `Retrieve`(f)

Require:
The f file.
Ensure:
f' replica file retrieved from nearest site.

1: $S \leftarrow$ sites subset where there are f copies
2: **for** `each` $s \in S$ **do**
3: $\Lambda(f) = \sum_{p \in P} \sum_j r_{\text{cost}}(f_j) + w_{\text{cost}}(f_j)$
4: **end for**
5: return f'

Every grid site $s \in S$ (where the environment is represented by the graph $G(S, L)$) has an associated set of processes which were previously scheduled according to any policy. The method `getNextProcess`(s) (Algorithm 5, line 3) returns a process p scheduled on site s. Every grid site s contains the historical traces of processes (TR, line 4). The pseudocode in between lines 5 and 13 performs operations tr $\in$ TR, considering the possible types: read, write or idle. From line 14 to 44, the heuristic assesses the feasibility of replication,

Algorithm 2 `Read`(tr)

Require:
Quintuple {`pid`, `inode`, `amt`, `time`, `op`}.
Ensure:
Reading `amt` from file referenced by `inode`.

1: **if** f is "*invalid*" **then**
2: `Retrieve`(f)
3: **end if**
4: reads

Algorithm 3 `Write` (tr)

Require:
Quintuple {`pid`, `inode`, `amt`, `time`, `op`}.
Ensure:
Writing `amt` to the file referenced by `inode`.

1: **if** f is "*invalid*" **then**
2: `Retrieve`(f)
3: **end if**
4: writes

Algorithm 4 `Invalidate`(f)

Require:
The f file.
Ensure:
The subset S of outdated f replicas.

1: $S \leftarrow$ site subset where there are f copies.
2: **for** `each` $s \in S$ **do**
3: update copy c_s of f as "*invalid*" in site s.
4: **end for**
5: return S.

Algorithm 5 DAP Heuristic

Require:
Set of process $P = \{p_0, p_1, \ldots, p_{h-1}\}$;
Set of files $F = \{f_0, f_1, \ldots, f_{z-1}\}$;
Set of quintuples TR $= \{\text{tr}_0, \text{tr}_1, \ldots, \text{tr}_{u-1}\}$;
initialASW $= 100$; ASW > 1; $\beta = 0.10$.
Ensure:
Set of file replicas F'.

```
 1: for each p ∈ P do
 2:   p ← getNextProcess()
 3:   for each tr ∈ TR do
 4:     if p = getProcess(tr) then
 5:       if getOperation(tr) = "Read" then
 6:         Read(tr)
 7:       else if getOperation(tr) = "Write" then
 8:         Invalidate(f)
 9:         Write(tr)
10:       else
11:         Sleep(tr.time)
12:       end if
13:     else if tr.index ≤ u then
14:       j ← tr.index
15:       ASW ← initialASW
16:       k ← 1; Op ← 0; diff ← true
17:       prevOp ← getOperation(tr)
18:       while k < ASW AND j < |TR| do
19:         if p = getProcess(tr_j) then
20:           if getOperation(tr_j) = "Read" then
21:             if f is invalidate OR f is not local AND
                none bringing the data then
22:               Retrieve(f)
23:             end if
24:           else
25:             if f is invalidate OR f is not local then
26:               Retrieve(f)
27:             end if
28:           end if
29:           k ← k + 1
30:           currentOp ← getOperation(tr)
31:           if prevOp = currentOP AND diff then
32:             Op ← Op + 1
33:             prevOp ← currentOp
34:           else
35:             diff ← false
36:           end if
37:         end if
38:         j ← j + 1
39:       end while
40:       initialASW ← ASW × (1 − β) + Op^(2× Op/ASW) × β
41:       if initialASW < 2 then
42:         initialASW ← 2
43:       end if
44:     else
45:       Sleep(tr.time)
46:     end if
47:   end for
48: end for
```

migration, consistency and retrieval of files, according to the historical events in sliding window. Lines 32 to 34 compute the number of similar operations in the current window.

Line 41 computes the next window length and line 43 is a special case when window length is less than 2. This special case is a restriction which denies the absence of further updates in order to avoid shorter-length windows (which could risk the adaptive approach). Line 46 corresponds to the situation in which site s is not responsible by process p and there is no event in the sliding window, therefore, s remains idle for a period of `time`.

4.4 Complexity evaluation

In order to understand the complexity of the proposed heuristic, we focused on the dominant computations in the Algorithm 1. The outer loop (line 1) is repeated at most P times, therefore, its time complexity is $O(|P|)$. The loop at line 3 is execute, in worst case, |TR| times. Further, for each iteration of the inner loop (line 18), lines from 19 to 38 have the time complexity of $O(|\text{TR}|)$. We concluded that the total time complexity of the heuristic algorithm is $O(P \times \text{TR}^2)$. Thus, our heuristic algorithm is low-order polynomial and can be run quickly for various grid environments with several sites.

5 Experiments

Grid computing researches are based on two types of experiments: real and simulations [11]. Experiments in real environments are usually more reliable and confirm proposed approaches in practice. However, applications may run for long periods and, furthermore, they need to be correctly and functionally implemented. Most likely failures occur during those executions, which influence the results accuracy. Other workloads, imposed on the environment, may also interfere in experiments. Finally, it is very difficult to have fully dedicated environments to avoid those interferences, mainly when considering the inherently large scale of grid computing scenarios.

Simulations approximate real experiments by modeling (using equations) those environments and their iterations (computers, processes, operations, etc.). The advantages of simulations over real experiments are: no need of building real systems; no limits of experimental scenarios; full control of the environment and reproducibility of experiments.

Due to the advantages of simulation, this paper adopts the OptorSim simulator, which is part of the Eu-

ropean DataGrid EDG Project [5]. This simulator was originally developed to model the dynamic effects of data replication, and it is used to model the LCG (Large Hadron Collide Computing Grid), also considered by other works [22, 4, 10].

OptorSim models grid environments by using sites interconnected through communication links (Figure 7). Every grid site contains at least one computing element (CE, or computer), and one storage element, or both. Every grid site executes a local replica optimizer (RO) to take replication decisions and there is a single global resource broker (RB) to decide on job scheduling. OptorSim models jobs to access a set of files, which may be replicated on grid sites, according to the RO. A replica catalog (RC) maps logical names to physical files and a replica manager (RM) handles replications and also registers them in the catalog. Figure 7 presents all components of the OptorSim simulator.

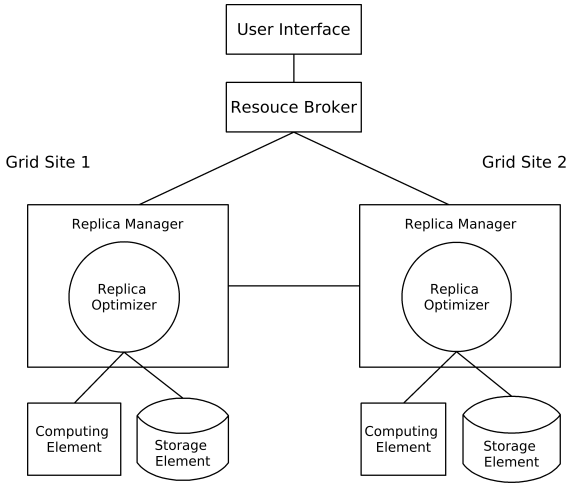

Figure 7: Simulated Data Grid architecture

This simulator allows users to specify the grid topology by providing parameters for CE's and communication links. In this work, we consider a topology generated by BRITE [25], a complex-network topology generator which provides bandwidth and latency information based on well-accepted models. Topology generators are widely used to model large-scale network environments [25], such as the Internet. BRITE was considered due to its flexibility, adaptability and interoperability [25]. The BRITE output file contains communication links with their associated bandwidths and latencies which are used, by OptorSim, to model the grid environment.

OptorSim requires the definition of a job list to be associated to a set of files. Jobs may access a subset of those files, according to an access pattern. The simulator also has four algorithms to take job scheduling decisions: `Random`, `QueueLength`, `AccessCost` and `QueueAccessCost`. `Random` randomly assigns a job to a CE, similarly to Legion scheduling [9]. `QueueLength` assigns jobs to the shortest-queue CE, what may represents the idler resource. `AccessCost` estimates the access time for all files required by a job, then, the job is assigned to the CE with the lowest estimated cost. The last algorithm operates similarly to the latter, but it also considers queue length (hybrid approach).

In addition to the scheduling algorithm, OptorSim provides five access optimization approaches: 1) In `SimpleOptimiser`, no replication is performed, files are remotely accessed; 2) `DeleteOldestFileOptimiser` replicates files when jobs need them, removing the least-recently-used (LRU) replicas; 3) `DeleteLeastAccessedFileOptimiser` replicates files when jobs need them, removing the least-frequently-used (LFU) replicas; 4) `EcoModelOptimiserBinomial` considers an economic model to determine replications. In this approach, file replicas are removed according to a Binomial estimation function; 5) `EcoModelOptimiserZipf` considers an economic model to replicate files. Replicas are removed according to a Zipf estimation function. We employed the approaches LRU, LFU and `EcoModelOptimiserZipf` in the experiments (Section 5.4) in order to compare them with the heuristic proposed in Section 4.3.

OptorSim provides performance metrics for its entities. For grid sites, OptorSim provides: number of remote reads; number of local reads; percentage of time that every CE has been active; number of file transfers that were routed through a site; and the total time (in seconds) consumed to run all jobs submitted to a site. For storage elements, OptorSim provides: capacity and usage, in MB, of the SE. OptorSim also provides the following metrics for computing elements: job execution time, in milliseconds; job execution time added to the queuing time; number of remote file reads of a CE; number of jobs currently executed by a CE; number of local file reads of a CE; percentage of time that a CE was running jobs; list of the files accessed by jobs running on a specific CE; and, finally, the total time (in seconds) to execute all jobs.

5.1 OptorSim Extensions

Some extensions were necessary to adapt OptorSim to meet our needs. Those extensions provide support for:

1) trace files; 2) write operations, and 3) adaptive sliding window of historical operations.

OptorSim provides different file access pattern approaches: 1) `SequentialAccessGenerator`, where files are sequentially accessed; 2) In `RandomAccessGenerator`, files are accessed using a uniform distribution; 3) In `RandomWalkUnitaryAccessGenerator`, files are accessed using a unitary random walk distribution; 4) `RandomWalkGaussianAccessGenerator`, where files are accessed using a Gaussian random walk distribution; and 5) In `RandomZipfAccessGenerator`, files are accessed using a Zipf random distribution. Those access patterns dynamically generate the job behavior, what did not meet our needs to have historical operations. That motivated us to develop a new approach: 6) a trace-based file access approach, where events represent process operations (behavior). Those events provide the following information for every operation: process identification (`pid`), interval in between consecutive access operations (`time`), file identification (`inode`), operation type (`op`) and volume of data (`amt`). Table 5 presents an example of a trace file.

Moreover, the original OptorSim provides only read operations, thus, we also needed to extend it to incorporate write operations. This extension was based on the cost (Equation 3) to write data and propagate it to file replicas, as presented in Algorithms 3 and 4.

Finally, we developed an adaptive sliding window strategy to provide future events (operations) based on historical information, as previously presented in Equation 9. Experiments have confirmed that the adaptive sliding window heavily impacts the heuristic performance (Section 5.4).

5.2 Environment Parametrization

A 128-site grid was considered in experiments. Every grid site was composed of one `SE` where the capacity was modeled by an exponential probability function with average 100GB (parameter Ω described in Section 3). Communication links were modeled by using the BRITE topology generator which considers the Barabasi model for Autonomous System [25]. In addition, BRITE adopts a Heavy-Tail distribution function to define the addition of nodes and bandwidth (the minimum bandwidth was 10 Mbits/s and the maximum $1,024$ Mbits/s). An exponential distribution function was considered to model communication delays (with average 0.5 seconds).

Experiments were configured with 128 jobs (one per grid site). More jobs may be considered, however when a set of jobs runs in a specific grid site, the behavior of each job contributes to the data access behavior issued by that grid site. Then, by having one job per site, we can indeed represent the data access patterns of several individual jobs allocated into the same grid site.

A uniform distribution function (average 10) was used to determine the number of files accessed by every job (i.e. the set of files accessed by a job). Files in the OptorSim have fixed size and can be model by following the chunk representation in the GFS [17]. The behavior of every job is assigned to the simulator by using a configuration file, which may be synthetic or obtained by using an interception approach (Section 4.1 and Table 5). All jobs have the same probability to run in every `CE` of the environment. An exponential distribution function was adopted to model the job inter-arrival time (average $1,500$ ms).

Processing capacities were obtained by using the benchmark SPEC CINT 2000, which is a standard way of measuring CPU performance. It generates a performance measurement based on a reference machine (Sun Ultra 10) [2]. For example, a Pentium IV has a performance of 0.5 CINT 2000. In this work, every `CE` is homogeneous and has a processing power of 1.0k CINT 2000 (or $1,000$ CINT 2000). The CPU homogeneity does not influence in the data access operations which are the main focus of this paper.

We consider the same job scheduling algorithm in every experimental scenario, which is the `QueueAccessCost` (Section 5) and the same trace file to describe the data access pattern of files (Section 5.1).

5.3 Intrusion Analysis when Capturing Information

We evaluated costs involved in acquiring the necessary application knowledge (operations, also called events) to generate trace files and, therefore, assessed the proposed approach. The same cost involved in obtaining such information would be necessary in a real environment. Two acquiring approaches were considered: Ptrace [32] and the DLSym [21] under two benchmarks: `Nbench` [24] and `Bonnie` [6]. `Nbench` probes the CPU capacity in terms of float-point operations as well as the memory subsystem. `Bonnie` performs read-and-write operations on files. Experiments were executed 30 times on a `Intel Core i7 CPU 2.67GHz, 8GB RAM and 250GB HD`. We have evaluated the execution time of such benchmarks with and without acquiring mechanisms. Table 6 presents results which confirm that the intrusion is lower than 12% in the worst case of Ptrace and 1.5% for DLSym. Each value in Table 6 corresponds to the benchmark

execution time and its standard deviation, respectively. This confirms that the second approach, this is DLSym, would better suit on a real environment, that is why we selected it.

Table 6: Information capture approach evaluation

Bench	Time(s)	`Ptrace`	`DLSym`
Nbench	272.81	309.69	276.78
	±5.99	±20.26	±5.46
Bonnie	10.41	11.97	10.47
	±0.03	±0.07	±0.04
Nbench Impact		11.91%	**1.43%**
Bonnie Impact		13.03%	**0.57%**

5.4 Results

This section presents experimental results which are organized according to the previously described environment: 128-site grid. Charts present bars which correspond to the average of 30 executions (this number of executions is based on the Central Limit Theorem [29], which supports the obtainment of a significative statistical measurement).

Experiments are categorized according to the percentage of read-and-write operations. For example, Figure 9 presents an environment with 5% of the writing operations, i.e., the trace file has 5% of events classified as **w** (Table 5).

We evaluated three optimization techniques: LRU, LFU and the Economic Model (ECO). All those techniques are available in the OptorSim simulator. We also compared the results of such techniques against the heuristic proposed is this paper. The DAP heuristic is evaluated under different values of parameter β: $\{0.05, 0.10, 0.15\}$.

In Figure 8, the simulated environment has 100% of reading operations and 10 files are accessed by jobs. The x-axis corresponds to parameter β considered by the heuristic and the y-axis represents the average execution time of jobs (in seconds). As shown in Figure 8, the heuristic was capable of reducing the job execution time in one order of magnitude, when compared to other data replication techniques.

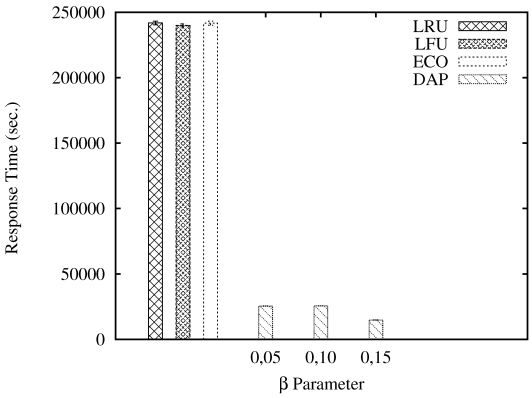

Figure 8: Environment with 100% of reading operations

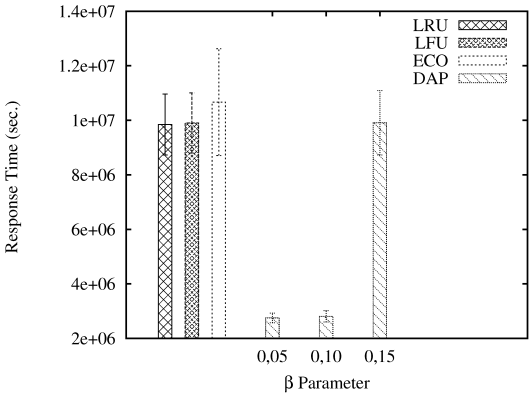

Figure 9: Environment with 5% of writing operations

In Figure 9, the environments deal with 5% of writing and 95% of reading operations and 10 files are accessed by jobs. The heuristic reduced the execution time in one order of magnitude, but, when $\beta = 0.15$, the results are similar to the other techniques.

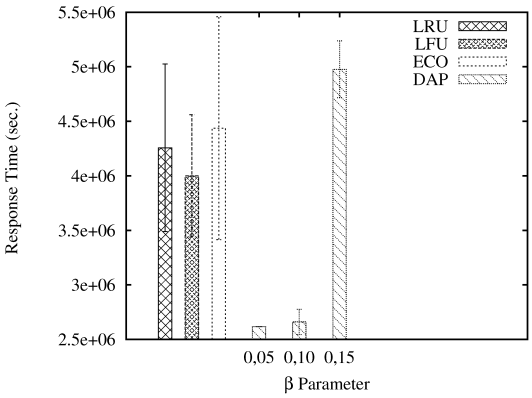

Figure 10: Environment with 95% of writing operations

In Figure 10, the simulated environments present 95% of writing and 5% of reading operations and 10 files are accessed by jobs. The heuristic has reduced job execution times in around 40%. When $\beta = 0.15$, the heuristic results were worse than other replication techniques.

We also compared the results of the adaptive heuristic to the ones obtained in the initial paper [19]. From that, we concluded that this heuristic presents similar execution times to the environments under 100% of reading operations, and this new approach was 10% better for the environment under 5% of writing operations and 15% better for the environment under 95% of writing operations, what confirms the need for an adaptive window.

These results confirm improvements in application performance and suggest the adoption of the adaptive sliding window length. We observed that the adaptation of the sliding window is better when considering $\beta \in [0.05, 0.10]$. In the considered scenario, we experimentally confirmed that $\beta \in [0.05, 0.10]$ gives more relevance to the current window, W_t, according to Equation 9, and when increasing β, more relevance is given to Op. This fact, allow us to conclude that as β gets lower, the adjustment of the window length is improved.

The reduction of access costs was obtained due to the adaptation of the sliding window length (i.e., the window now follows the dynamic behavior of processes), which updates the number of future events considered when taking decisions on data replication, migration and consistency. We also observed that the greater the differences in access patterns (different types of operations – read, write and idle – interposed), the unstabler the application performance is.

6 Conclusions

This paper has presented a history-based data access optimization approach for grid computing environments. Our main objective is to minimize the application execution time by optimizing data accesses and, therefore, improve decisions on replication, migration and consistency. From that, we proposed an adaptive sliding window length which aims at providing the dynamic behavior of application operations to our data access optimization heuristic.

The proposed approach also considers concepts of monitoring and intercepting system calls to capture applications operations and compose processes histories (i.e. the trace files obtained by intercepting calls). We kept such histories due to we believe that it is very important to understand, estimate and/or predict processes behavior as a way to optimize read-and-write operations, which was cleared confirmed. In addition, we defined an analytical optimization model for the Data Access Problem (DAP), which was considered to study approaches for minimizing the overall application execution times.

Simulations were conducted to study the efficiency of our heuristic using the adaptive sliding window length, under a wide range of environments and system configurations (frequency of read-and-write operations). Experimental results confirm that our approach outperforms other commonly considered ones (e.g., LRU, LFU and Economic Model) in approximately 50% when dealing with grid environments. Besides using historical information, the results motivate further work in designing and implementing on-line prediction mechanisms to take autonomic decisions on data replication, migration and consistency.

Acknowledgments

This paper is supported by CNPq-Universal (National Counsel of Technological and Scientific Development), under grant no. 470739/2008-8, CAPES (Coordination of Improvement of Higher Education) and FUNDECT (Foundation to Support the Education, Science and Technology Development of Mato Grosso do Sul) no. 23/200.402/2008 from Brazil. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of CNPq, CAPES or FUNDECT.

References

[1] The Earth System Grid–ESG Project. www.earthsystemgrid.org, 2005.

[2] SPEC's CPU Benchmark. See the definition of CINT2000 at http://www.spec.org/, Mar 2010.

[3] Abramovici, A., Althouse, W. E., Drever, R. W. P., Gürsel, Y., Kawamura, S., Raab, F. J., Shoemaker, D., Sievers, L., Spero, R. E., Thorne, K. S., Vogt, R. E., Weiss, R., Whitcomb, S. E., and Zucker, M. E. LIGO: The Laser Interferometer Gravitational-Wave Observatory. *Science*, 256(5055):325–333, 1992.

[4] AL-Mistarihi, H. H. E. and Yong, C. H. On fairness, optimizing replica selection in data grids. *IEEE Trans. Parallel Distrib. Syst.*, 20(8):1102–1111, 2009.

[5] Bell, W. H., Cameron, D. G., Millar, A. P., Capozza, L., Stockinger, K., and Zini, F. Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4):403–416, 2003.

[6] Bray, T. Bonnie benchmark. http://www.textuality.com/bonnie/, Jun 2009.

[7] Buyya, R. *High Performance Cluster Computing – Architecture and Systems*, volume 1. Prentice Hall, 1999.

[8] Chang, R.-S. and Chang, J.-S. Adaptable replica consistency service for data grids. In *Proc.* 3^{th} *Int. Conf. on Information Technology: New Generations*, pages 646–651, April 2006.

[9] Chapin, S. J., Katramatos, D., Karpovich, J., and Grimshaw, A. S. The Legion resource management system. In Feitelson, D. G. and Rudolph, L., editors, *Job Scheduling Strategies for Parallel Processing*, pages 162–178. Springer Verlag, 1999.

[10] Chervenak, A. L., Schuler, R., Ripeanu, M., Amer, M. A., Bharathi, S., Foster, I., Iamnitchi, A., and Kesselman, C. The globus replica location service: Design and experience. *IEEE Trans. Parallel Distrib. Syst.*, 20(9):1260–1272, 2009.

[11] Dang, N. N. and Lim, S. B. Combination of replication and scheduling in data grids. *International Journal of Computer Science and Network Security*, 7(3):304–308, Mar 2007.

[12] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., and Koranda, S. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, 1(1):25–39, March 2003.

[13] Dorigo, M. and Di Caro, G. The ant colony optimization meta-heuristic. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[14] Elghirani, A., Subrata, R., and Zomaya, A. Y. Intelligent scheduling and replication in datagrids: a synergistic approach. In *Proc.* 7^{th} *IEEE Int.Symposium on Cluster Computing and the Grid*, pages 179–182, Washington, DC, USA, 2007.

[15] Feng, J. and Humphrey, M. Eliminating Replica Selection - Using Multiple Replicas to Accelerate Data Transfer on Grids. In *Proc.* 10^{th} *Int. Conf. of Parallel and Distributed Systems*, page 359, Washington, DC, USA, 2004. IEEE Computer Society.

[16] Garey, M. R. and Johnson, D. S. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[17] Ghemawat, S., Gobioff, H., and Leung, S.-T. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[18] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[19] Ishii, R. P. and de Mello, R. F. A history-based heuristic to optimize data access in distributed environments. In *21st IASTED International Conference Parallell and Distributed Computing and Systems (PDCS2009)*, Cambridge, MA, Nov. 2-4 2009.

[20] Jacobson, V., Leres, C., and McCanne, S. TCPDump Man Pages. Available at: Linux Systems, calling the man command for tcpdump, Mar 2010.

[21] Jung, C., Woo, D.-K., Kim, K., and Lim, S.-S. Performance characterization of prelinking and preloadingfor embedded systems. In *Proc.* 7^{th} *ACM & IEEE Int. Conf. on Embedded Software*, pages 213–220, New York, NY, USA, 2007. ACM.

[22] Kim, J., Chandra, A., and Weissman, J. B. Using data accessibility for resource selection in large-scale distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 20(6):788–801, 2009.

[23] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[24] Mayer, U. F. Linux/unix nbench. http://www.tux.org/˜mayer/linux/bmark.html, Mar 2010.

[25] Medina, A., Lakhina, A., Matta, I., and Byers, J. Brite: an approach to universal topology generation. In *Proc.* 9^{th} *Int. Symposium on*

Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pages 346–353, 2001.

[26] Oldfield, R. and Kotz, D. Improving data access for computational grid applications. *Cluster Computing*, 9(1):79–99, January 2006.

[27] Oliker, L., Biswas, R., Shan, H., and Smith, W. Scheduling in heterogeneous grid environments: The effects of data migration. In *Proc.* 12^{th} *Int. Conf. on Advances in Computing & Communications*, Ahmedabad, INDIA, Jan 2004.

[28] Rahman, R. M., Barker, K., and Alhajj, R. Replica selection in grid environment: a data-mining approach. In *Proc. Symposium on Applied Computing*, pages 695–700, New York, NY, USA, 2005. ACM.

[29] Schefler, W. C., editor. *Statistics: concepts and applications*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1988.

[30] Semenov, M. A. and Terkel, D. A. Analysis of convergence of an evolutionary algorithm with self-adaptation using a stochastic lyapunov function. *Evol. Comput.*, 11(4):363–379, 2003.

[31] SMART. Self-Monitoring, Analysis and Reporting Technology. Available at: http://en.wikipedia.org.wiki/s.m.a.r.t., 2010.

[32] Spillane, R. P., Wright, C. P., Sivathanu, G., and Zadok, E. Rapid file system development using ptrace. In *Proc. Workshop on Experimental Computer Science*, page 22, New York, NY, USA, 2007. ACM.

[33] Stockinger, H. Defining the Grid: a snapshot on the current view. *The Journal of Supercomputing*, 42:3–17, 2007.

[34] Sun, Y. and Xu, Z. Grid replication coherence protocol. In *Proc.* 18^{th} *Int. Symposium on Parallel and Distributed Processing*, pages 232–239, April 2004.

[35] Voß, S. Meta-heuristics: The state of the art. In *ECAI '00: Proceedings of the Workshop on Local Search for Planning and Scheduling-Revised Papers*, pages 1–23, London, UK, 2001. Springer-Verlag.

[36] Wang, C.-M., Hsu, C.-C., Chen, H.-M., and Wu, J.-J. Efficient multi-source data transfer in data grids. In *Proc.* 6^{th} *IEEE Int. Symposium on Cluster Computing and the Grid*, pages 421–424, Washington, DC, USA, 2006. IEEE Computer Society.

[37] Ware, H. and Frederick, F. VMstat Man Pages. Available at: Linux Systems, calling the man command for vmstat, Mar 2010.