A grid-based programming approach for distributed linear algebra applications

Lamine M. Aouad\textsuperscript{a,∗} and Serge G. Petiton\textsuperscript{b}

\textsuperscript{a}University College Dublin, School of Computer Science and Informatics, Belfield, Dublin 4, Ireland
\textsuperscript{b}University of Lille I, Villeneuve d’Ascq, France

\textbf{Abstract.} Grid technologies have emerged as an important area in distributed and parallel computing. An increasing amount of users among scientific communities are using grid facilities to share, manage and process large-scale datasets and applications. However, despite the increasing maturity of grid tools and middleware, the grid lacks well-adapted large-scale programming models. This paper proposes grid-based solutions for the deployment of basic numerical applications. We give an investigation of well-adapted scheduling schemes for such applications in these systems, and discuss the performance of block-based matrix-vector products and the Gauss-Jordan method for matrix inversion, using different testbeds. The proposed approach is based on an efficient data locality management, both locally and through the network. These include persistent data placement and explicit management of local memories on the computational nodes. Finally, we will discuss some constraints and limitations on the experimentation and underlying tools that make scalability and realistic expectations more difficult to achieve on the grid.

1. Introduction

Many applications in different areas have already been developed to take advantage of grid computing facilities and have achieved elementary success. However, inherent characteristics of the grid, such as the large-scale dynamicity and heterogeneity, make effective applications development much more difficult, and as the field grows so also do the related issues. Furthermore, the study of these systems and the behaviour of parallel and distributed algorithms is based on a variety of methodologies and tools: mathematical models, simulators or emulators, and experimental testbeds. In most cases, the grid can hardly be reduced to a model since traditional parallel metrics no longer apply. Algorithmic evaluations must then be done under a large variety of realistic conditions.

Large grid systems and the related constraints, either concerning the underlying middleware or tools or the characteristics of the grid itself, put pressure on the design and implementation of effective grid-based distributed applications. In addition to a range of existing parallel and distributed programming paradigms, such as message passing or remote method invocation, which are partly supported by some high-level grid tools, efficient algorithmic abstractions are required. During the last decade, interesting discussions have been conducted about the need for new abstract programming models and techniques for the grid, and dealing with the constraints and the distributed computing aspects of grid applications development.

\textsuperscript{∗}Corresponding author: Lamine M. Aouad, University College Dublin, School of Computer Science and Informatics, Belfield, Dublin 4, Ireland. Tel.: +353 8 7979 3963; Fax: +353 1269 7262; E-mail: lamine.aouad@ucd.ie.

ISSN 1574-1702/10/$27.50 © 2010 – IOS Press and the authors. All rights reserved
Numerical and matrix algebra are an important source of large datasets that are getting more and more common. They represent interesting candidates for grid platforms. Typically, these massive datasets cannot be completely loaded into the main memory even in dedicated computational nodes. Efficient data placement is also an important issue. Designing practical programs for a large set of computational nodes must take these factors into account. The general goal is to redesign parallel and distributed algorithms to run with minimal performance loss in the grid with effective data locality policies using persistent storage and out-of-core programming. In this paper we test these data placement techniques on basic block-based matrix algebra applications and propose a well adapted programming technique for the grid. We also test grid middleware scalability on widely distributed non-dedicated grid testbeds and on the Grid’5000 platform which provides a configurable and controllable grid infrastructure allowing monitoring and reproducing a variety of experimental conditions.

The paper is organised as follows. The next section reviews some related work in parallel and distributed matrix algebra. Section 3 gives the description and some definitions related to the proposed approach. Section 4 describes the block-based matrix algebra applications tested in this work. Section 5 describes the grid testbeds used in our experiments and the experimentation tool. Then, Section 6 shows and discusses the experimental results. Finally, concluding remarks are made in Section 7.

2. Related work

Matrix algebra is of fundamental importance in many grand challenge applications in scientific and engineering disciplines. Many research works have addressed this area over the last few decades. These works were specially dedicated to parallel implementations on high-performance computers using standard interfaces such as MPI or PVM, or parallel extensions of traditional languages (Fortran or C/C++) such as HPF or OpenMP. For many years, this research field was dominated by this tendency and many parallel libraries for linear algebra on multicomputers were proposed. These include many widely popular libraries for dense and sparse matrix algebra such as ScalAPACK [13,14], PLAPACK [2], PETSc [6], among others. These libraries are often quite specialised either targeting the problem to solve or the environment on which they run, or both. They also propose different data distribution schemes among the processes, such as the block cyclic distribution in ScalAPACK, or block row (default) or application-specific data distribution in PETSc. The user has more or less control over the data layout in these cases.

Furthermore, a large body of work has been carried out on automatic data distribution on distributed memory machines. Most of them are related to automatic array alignment to reduce communication through compilers optimisation. Different loop transformation techniques have been used to improve locality, such as in [19] or [26], among others. However, all programs are not amenable to loop transformations, and some work focused on the data space and transformations to attempt to change the data layouts so that better locality is obtained [15,3,12], etc. Results of many of these optimisations have been shown to achieve quite large speedup for several linear algebra programs on few parallel architectures, mainly distributed-shared-memory multiprocessors. We have to emphasise that both the problem of finding optimal data distributions and layouts are NP-Complete. Other research works focus on control flow or replication in the context of distributed linear algebra, or adaptive data placement on parallel architectures [18,11].

1http://gams.nist.gov/
On the other hand, only few research works have considered grid-based matrix algebra using heterogeneous computational sites, with different architectures, dynamic resources, etc. Most of them use the remote procedure call paradigm to build problem solving algorithms and environments using several tools such as NetSolve/GridSolve [24], DIET [9,10], OmniRPC [23], or Ninf-G [25]. Unlike these approaches, this paper concentrates on a well-adapted algorithmic scheme for suitable data management. Indeed, on the basis of previous works [4,5,21], in which we presented basic grid-based matrix algebra implementations on both non-dedicated and dedicated large grid platforms, we propose a programming approach based on an efficient data locality management, and explicit memory management of the computational nodes. The next section will introduce the basic features of the proposed technique.

3. Grid-based programming approach

The major issue in grid programming is the performance gap between processors and networks on the one hand, and memories and input/output systems on the other hand. The performance of processors and communication networks increase about ten times faster than memories and input/output (I/O) systems. This makes them the major bottleneck for large-scale applications in the grid. Furthermore, frequently moving large amounts of data through the network may lead to a bottleneck as well. To address these issues, new models and programming approaches for grid-based computation are needed. Before describing the approach itself, we first briefly discuss the concept of resource metrics used to characterise existing models of parallel and distributed computing. Based on these concepts, we present how we synthesised and developed efficient grid-based matrix algebra programs that have a block-based computational structure.

A resource metric is a measure of an architectural feature that affects the performance of a given distributed platform. A model is a set of metrics representing an abstraction of this platform that is used to design and analyse algorithms. Significant resource metrics for distributed computation can be summarised as follows: (1) processors-based, such as the number of processors, (2) network-based, such as the bandwidth, latencies, overheads, topology, etc., and (3) Memory-based, especially related to each node in the targeted environments.

The proposed approach is based on an efficient data distribution scheme that minimises both the communication and the I/O overheads. For efficient programs synthesis we need an I/O model describing the memory hierarchy, and computational and communication models that hide the architectural details. However, the grid has typically very different architectures, including different interconnection properties. Some approaches build complex parameterised models of distributed computation which make algorithms design very difficult because not abstract enough and amenable to analysis. We propose a method based on efficient data locality in all the computation hierarchy, i.e. at different levels of granularity, from the local memory to the communication network for accurate grid-based implementations of linear algebra applications. We consider the basic higher two levels memory hierarchy, i.e. main memory and disk, and different levels of block partitioning.

3.1. Persistent storage

There are different definitions of data persistency in literature. In Java for instance, a persistent object refers to information retrieval even after the end of an application. In OceanStore [16], the persistent

\footnote{Between 30% to 50% per year for processors and networks, versus 3% to 5% for memories and I/O systems.}
storage refers to the consistency, high-availability, and durability of storage through an infrastructure comprising of untrusted servers. Thus, this notion refers either to the reliability or services continuity in some systems, or ‘data survival’ in others.

Our definition of persistent storage is related to data locality and placement. This refers to migration anticipation, and data labelling (or data nailing). Our approach maximises the migration anticipation rate since the data locality scheme (and then the labelling scheme) is fixed for a given application. Thus, for each application, the persistent storage corresponds to some data distribution and placement optimisation in order to minimise communications. Tasks scheduling is then carried out according to this placement and the dataflow. The persistent storage scheme implementation can be done implicitly or explicitly. In the latter case, application programmers have to manage data locality explicitly, using handles or functions for instance. The implicit case can be related to a global file system or a persistent scheduler. In this paper, the persistent storage is simulated through block generation since the grid middleware does not manage data and jobs locality.

3.2. Out-of-core management

Many large problems that could not be solved on the largest supercomputer few decades ago, can now be solved on a standard laptop computer. However, data structures for data intensive applications, such as in linear algebra, remain too large to fit in the main memory and must be stored on disks. Out-of-core algorithms are designed to achieve acceptable performance when data is stored on disks. They access data in large contiguous blocks and reuse data that is stored in the main memory several times. In out-of-core algorithms the ordering of independent operations must be chosen so as to minimise I/O. All or most of the data in the main memory is used before it is evicted.

It is obvious that the implementation of large-scale applications on the grid would lead to many issues and that their execution time can become prohibitively large. It is also clear that in the absence of a substantial increase in the ratio of memory to processing power, it is natural to develop out-of-core solvers for large and data-intensive applications, and, a fortiori, if it is intended to non-dedicated platforms. Therefore, we use explicit I/O calls to redefine the underlying numerical algorithms. We then reduce the I/O accesses by restructuring the code and using better data locality. This avoids the use of the trivial virtual memory paging which has poor performance due to slow disk access, which is several times slower than memory access. We will design basic out-of-core tasks for the tested block-based algorithms. These versions use different levels of block partitioning. We will describe these algorithms in the following sections. The I/O characteristics of the computing nodes and the size of the available memory are used to derive the parameters for these implementations.

4. Matrix algebra applications

In this section we describe the two tested matrix algebra applications: the matrix-vector product and the Gauss-Jordan matrix inversion. In addition to the algorithmic foundations, this section also gives basic implementation schemes.

4.1. Block-based matrix-vector product

The matrix-vector product, $Ax = b$, is an elementary operation in matrix computing. It forms the core of many iterative methods. We use two block-based decompositions; two-dimensional block distribution
and unidimensional block distribution (Fig. 1). Basically, the first version needs two communication steps. Firstly, rows of blocks of the matrix $A$ are communicated horizontally with the vector blocks over $p^2$ nodes ($p$ is the partitioning size). Then, the result vector blocks, for a given row, are communicated over $p$ nodes for addition. In the second version, there is no communication needed between tasks and the blocks of the matrix $A$ and the vector $x$ are communicated vertically over $p^2$ nodes.

In iterative methods, the matrix $A$ remains unchanged. Thus, it is sufficient to schedule computations on the nodes where the blocks of the initial matrix are located. Only the updated blocks of the vector are communicated, i.e. the matrix blocks are referenced and only the vector blocks are sent.

### 4.1.1. Out-of-core MVP

The MVP operation can be scheduled out-of-core efficiently using a very simple technique that we describe here. We denote the size of main memory by $M$. We consider a $p - b y - N$ submatrix of $A$ with $p = (M - N)/(N + 1)$, $N$ is the order of $A$. The algorithm is made up of $N/p$ steps. At each step we perform the product of the submatrix $A_{(p-b-1)}$ and the vector $x$, which gives the subvector $b_{(p)}$. In this schedule, the words that are accessed in the inner iteration need to be read from disk only once because all of them fit within main memory and are written back once. Another level of block partitioning is then considered. The algorithm is as following:

```plaintext
out-of-core matrix-vector product
for i = 1 to N step p
    for j = 1 to N
        multiply(submatrix $A_{(p-b-1)}$, vector $x$, subvector $b_{(p)}$)
    end for
end for
```

### 4.2. Block-based Gauss-Jordan matrix inversion

The block-based Gauss-Jordan algorithm is one of the most traditional methods used for matrix inversion [20]. Let $A$ be a matrix of dimension $N$, partitioned into $p \times p$ blocks of dimension $n$. Let $B$ be the inverted matrix of $A$, progressively built. Each of the $p$ steps of the algorithm has 3 parts; the first part is to invert the pivot block, in the second part, we compute $2(p - 1)$ blocks product, and finally $(p - 1)^2$ blocks triadic are performed. Figure 2 shows the task graph for one given step (for $p = 3$). We use the element matrix inversion from LAPACK³ (Linear Algebra PACKage) to invert the pivot

³http://www.netlib.org/lapack/.
blocks at each step (DGETRF and DGETRI routines), and BLAS3\(^4\) (Basic Linear Algebra Subroutines) for matrix product and matrix triadic (DGEMM routine). The algorithm used is given below. The main insight of our algorithm is that the blocks of the inverted matrix are progressively built. This leads to less communication and storage needs.

Basically, the number of processors required for computation is \((p - 1)^2\), and each loop \textbf{‘for’} (in lines 3, 6 and 14 in Algorithm 1) is executed in parallel. It is necessary to take into account the dependencies. Figure 3 shows the intra-step and the inter-steps parallelism. At each step, the loops (1) and (2) (respectively in lines 3 and 6) depend on the computation of the inverse block \(B_{kk}\) and the loop (3) (in line 14) depends partially in (1) and (2). This means that at least all the blocks products in the loops (1) and (2) are independent tasks and are executed in parallel, and also matrix triadic in the loop (3). However, the loop (3) can start without the completion of (1) and (2). According to the inter-steps dependencies, step 3 can start without the completion of step 2 (computations represented by gray squares at the bottom of Fig. 3 are not finished yet) and small numbers in squares represent the

\(^4\)http://www.netlib.org/blas/.
ongoing computations for step 3. Only intra-step parallelism is exploited in our implementations.

In our implementation, using the persistent data placement scheme, only one block is sent in the loops (1), (2) and (3) because the data blocks are pre-deployed and referenced. This means that at each step, we anticipate the communication of data blocks on the computational nodes where they will be used. In the loop (1), the block \( A_{ki} \) is pre-deployed and as soon as the computation of the inverse of \( A_{kk} \) (i.e. \( B_{kk} \)) is made, it will be sent. In the loop (2), the blocks \( A_{ik} \) and \( B_{ki} \) are pre-deployed. In the loop (3), the blocks \( A_{kj} \) and \( B_{kj} \) are located on the nodes where they were initially computed and we anticipate the send of the block \( A_{ik} \). Recall that the persistent storage schemes are simulated by data block generation at the computational nodes since it is not possible to manage data locality and task scheduling under the used grid middleware.

### 4.2.1. Out-of-core matrix inversion

For this task, we used the same block-based Gauss-Jordan algorithm (presented in Algorithm 1) using explicit I/O access. We assume that at least three matrix blocks are held in the main memory. Thus, for each computation step, the blocks are locked in memory until the computation’s completion. We consider a \( p \times p \) submatrix of \( A \) and we denote the size of the available main memory by \( M \). The main memory can store \( 3 \times p^2 \) elements (3 blocks of \( p \times p \) elements), \( p \) is then smaller than \( \sqrt{M/3} \).

As in the MVP operation, another level of block partitioning is considered. The summarised algorithm can be described as follows:

\begin{algorithm}
\caption{The block-based Gauss-Jordan matrix inversion}
\begin{algorithmic}
\Require \( A \) partitioned into \( p \times p \) blocks,
\Ensure \( B = A^{-1} \), progressively built,
\end{algorithmic}
\begin{algorithmic}
1. \textbf{for} \( k = 0 \) to \( p - 1 \) \textbf{do}
2. \( B_{kk} = A_{kk}^{-1} \)
3. \textbf{for} \( i = k + 1 \) to \( p - 1 \) \textbf{do}
4. \( A_{ki} = B_{kk} \times A_{ki} \)
5. \textbf{end for}
6. \textbf{for} \( i = 0 \) to \( p - 1 \) \textbf{do}
7. \textbf{if} \( (i \neq k) \) \textbf{then}
8. \( B_{ik} = -A_{ik} \times B_{kk} \)
9. \textbf{end if}
10. \textbf{if} \( (i < k) \) \textbf{then}
11. \( B_{ki} = B_{kk} \times B_{ki} \)
12. \textbf{end if}
13. \textbf{end for}
14. \textbf{for} \( i = 0 \) to \( p - 1 \) \textbf{do}
15. \textbf{if} \( (i \neq k) \) \textbf{then}
16. \textbf{for} \( j = k + 1 \) to \( p - 1 \) \textbf{do}
17. \( A_{ij} = A_{ij} - A_{ik} \times A_{kj} \)
18. \textbf{end for}
19. \textbf{for} \( j = 0 \) to \( k - 1 \) \textbf{do}
20. \( B_{ij} = B_{ij} - A_{ik} \times B_{kj} \)
21. \textbf{end for}
22. \textbf{end if}
23. \textbf{end for}
24. \textbf{end for}
\end{algorithmic}
\end{algorithm}
out-of-core matrix inversion
for $i = 1$ to $p$
inverse(submatrix $A_{(p-by-p)}$)
for $j = 1$ to $2(p-1)$
multiply(submatrix $A_{(p-by-p)}$, submatrix $B_{(p-by-p)}$)
for $j = 1$ to $(p-1)^2$
triadic(submatrix $C_{(p-by-p)}$, submatrix $A_{(p-by-p)}$, submatrix $B_{(p-by-p)}$)
end for
end for

The additional block partitioning for matrix multiplication and matrix triadic is shown in Fig. 4.

5. Experimentation testbeds and middleware

In this section we present the grid testbeds, and the underlying middleware used to manage the computational resources.

5.1. Testbeds

Firstly, a non-dedicated platform, composed of three computational sites, is used for our tests. It uses 248 CPUs distributed over three geographic sites in France (Lille and Orsay) and Japan (Tsukuba). The second testbed is the Grid’5000 platform [1,22]. Grid’5000 is a large-scale infrastructure for grid research. It aims to provide a reconfiguration and monitoring instrument to investigate grid issues under real and controllable conditions. It is not a production grid, but an instrument that can be configured to work as a real testbed at a nation wide scale. Grid’5000 is composed of nine geographically distributed clusters (with 100 to 1000 CPUs for each one) interconnected by the French national research network (RENA TER) through VLANs using MPLS at level 2. A set of control and monitoring tools allows users to make reservation, configuration, preparation, and run experiments. This includes OAR\(^5\) and KaDeploy\(^6\) for nodes reservation and the deployment of specific execution environments with a given stack of software and/or operating system. This platform is already used for a range of experiments.

\(^5\)http://oar.imag.fr.
\(^6\)http://ka-tools.imag.fr.
Table 1  
Non dedicated testbed configuration

<table>
<thead>
<tr>
<th>#nodes</th>
<th>CPU</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site I – Lille</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Pentium III (Katmai), 450 MHz</td>
<td>128 MB</td>
</tr>
<tr>
<td>28</td>
<td>Intel Celeron, 2.2 GHz</td>
<td>512 MB</td>
</tr>
<tr>
<td>23</td>
<td>Intel Celeron, 2.4 GHz</td>
<td>512 MB</td>
</tr>
<tr>
<td>15</td>
<td>AMD Duron, 750 MHz</td>
<td>256 MB</td>
</tr>
<tr>
<td>14</td>
<td>Celeron (Coppermine), 600 MHz</td>
<td>128 MB</td>
</tr>
<tr>
<td>8</td>
<td>Intel Celeron, 2 GHz</td>
<td>512 MB</td>
</tr>
<tr>
<td>8</td>
<td>Intel Celeron, 1.4 GHz</td>
<td>256 MB</td>
</tr>
<tr>
<td>4</td>
<td>Pentium 4, 2.4 GHz</td>
<td>512 MB</td>
</tr>
<tr>
<td>Site II – Orsay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>AMD Athlon(tm), 2.8 GHz</td>
<td>1 GB</td>
</tr>
<tr>
<td>6</td>
<td>AMD Athlon(tm), 1.8 GHz</td>
<td>1 GB</td>
</tr>
<tr>
<td>Site III – Tsukuba</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Bi-processor Xeon, 2.4 GHz</td>
<td>1 GB</td>
</tr>
<tr>
<td>10</td>
<td>Bi-processor AMD Athlon, 1.8 GHz</td>
<td>1.5 GB</td>
</tr>
<tr>
<td>8</td>
<td>Quadri-processor Pentium II (Deschutes), 500 MHz</td>
<td>2 GB</td>
</tr>
</tbody>
</table>

Table 2  
The Grid’5000 testbed configuration

<table>
<thead>
<tr>
<th># used per site</th>
<th>CPU/Memory</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>bi-proc AMD-64 opteron, 2 GHz/2 GB</td>
<td>Orsay</td>
</tr>
<tr>
<td>70</td>
<td>bi-proc AMD-64 opteron, 2 GHz/2 GB</td>
<td>Sophia</td>
</tr>
<tr>
<td>64</td>
<td>bi-proc AMD-64 opteron 248, 2.2 GHz/2 GB</td>
<td>Rennes</td>
</tr>
<tr>
<td>58</td>
<td>bi-proc AMD-64 opteron 248, 2.2 GHz/2 GB</td>
<td>Toulouse</td>
</tr>
<tr>
<td>46</td>
<td>bi-proc AMD-64 opteron 246, 2.2 GHz/2 GB</td>
<td>Nancy</td>
</tr>
</tbody>
</table>

addressing issues at different levels of the grid. This includes network protocols, operating system mechanisms, middleware, issues in performance, scheduling, fault tolerance and parallel/distributed programming and applications. Tables 1 and 2 give a description of the two testbeds.

5.2. XtremWeb

The experimentation and management platform used in our tests is the Xtrem-Web system [8,17]. XtremWeb is an experimental global computing project intended to distribute applications over dynamic resources according to their availability and implements its own security and fault tolerance policies.

XtremWeb manages tasks following the coordinator-worker paradigm. The coordinator controls the tasks management process, especially output management since there are, currently, no true notions of tasks scheduling, tasks are scheduled to workers according to their demand in FIFO mode. Workers are distributed volunteer entities which use part of their CPU time to compute tasks provided by the coordinator. They are not under the control of the coordinator and are volatile. Workers initiate each action and all connections (pull model). Every worker connection is registered by the coordinator, and it requests tasks to compute according to its own local policy. The workers download task software and all expected objects (input files and arguments in zipped format), store them and start computing. When a task is completed, the workers send back results to the coordinator.

Several fault tolerance mechanisms are used in XtremWeb to handle resources failures. The main purpose is to enable the system to restart properly after any failure (workers and coordinator). The coordinator manages its tasks using transactions and stores them in reliable media (disks) so that the full
system integrity is preserved even if the coordinator shuts down for any reason. At starting time, the coordinator reads the information stored on disk to set up its proper state. The client submits tasks and the worker fetches tasks using transactions; this ensures a consistent state when the coordinator restarts from a fault and the client/worker have not failed. Workers failures are detected by the alive signal, so if this signal is not received after a time out, the coordinator considers the worker as lost and reschedules the same task on another available worker. Also, to avoid redundant tasks and results overwriting, a worker can be brought to stop executing its current task if it has been disconnected for a long time. XtremWeb also ensures user authentication, workers integrity, application and results protection and user execution logging. It mainly relies on three mechanisms:

- A list of authorised users: after registration, the coordinator provides to each user a key to be used for each subsequent connection. All communications between the user and the coordinator are also encrypted using SSL,
- Authentication of the coordinator by workers and clients and,
- Sand-boxing system utility: if a malicious user succeeds on launching an aggressive application, XtremWeb workers still protect their host by implementing sand-boxing functionality. Workers run any task inside a sand-box which is customisable, from memory usage to file system operations. This can be disabled on secured systems.

6. Tests

In these tests, we consider a homogeneous data distribution among the computational nodes, and two manners for block transfer as quoted earlier. Firstly, all the blocks are sent across the network, then we implemented an efficient data placement scheme, well adapted to each application, based on pre-deployment, migration anticipation, and data labelling. Recall that the whole data placement process is referred to as *persistent storage*. Finally, we use explicit I/O disk accesses in the computational nodes to deal with memory constraints in relation to the local available memory, and according to the algorithmic out-of-core schedule presented before. Our goal is not to measure the performance on each node or of the network because this is a fairly useless set of data that is likely to be constantly out-of-date. Indeed, considering the dynamic nature of the grid, this information is not interpretable and is useful only for a scheduling in real time. However, we will give an interpretation of the final results in order to derive a general computing framework for numerical applications in such platforms.

In the first set of tests, presenting the execution of the classical versions and using persistent storage, various platform configurations, data sizes, and distributions were considered. Table 3 shows the average and minimum computing times for version 1 of the MVP using different sizes of blocks (the size of the matrix is 30000). The best average execution time is given by a block’s size of 3750 in the case of persistent storage management, and 3000 without using persistent storage. This highlights the importance of the choice of block sizes, especially in relation to the system overheads and their overlapping. Table 4 shows the computing times on the dedicated testbed. The persistent storage performs better by a factor of up to 7.7 for both executions. Tables 6 and 7 show the average and minimum execution times for the Gauss-Jordan matrix inversion. In this case, the version using the persistent storage scheme performs better by a factor of up to 9.8.

Furthermore, the results of using the Grid’5000 testbed show better performance by a factor of up to 25 for the versions managing all communications, and between 4 to 18 using persistent storage, compared to the non-dedicated platform. This was expected since the nodes are dedicated and the communications
are several orders of magnitude faster. Recall that the performance gain for the persistent storage version in this case was up to 9.8. Also, the difference factor is less important between the versions using data pre-deployment and labelling when comparing the two testbeds. This highlights the importance of the optimisation of communications and the data/tasks placement beyond the communication speed.

Indeed, in Table 5 we show that the computation/communication ratio for the MVP operation is very small. This strongly penalises the performance. Communications and task scheduling must then be fast. The persistent data placement reduces the communication needs and then increases the computation/communication ratio. These become $\frac{N}{p}$ for the first version and $\frac{N}{2p}$ for the second version. Furthermore, the results show that such data placement optimisations are still interesting even with a relatively high quality of interconnection. Nevertheless, we supposed that the anticipation rate is maximal and that the overheads related to the management of references is overlapped by calculation and communications since we generate the persistent state from the computational nodes. This can be the case with a good choice of blocks sizes and by implementing additional stages in the block partitioning and distribution which would allow subblocks to be communicated while making calculations on others.

In the versions using the out-of-core schedule, the basic matrix programs are redesigned to minimise I/O operations so that all or most of the data in the main memory is used before it is evicted. We introduced the notion of subblocks that are assimilated to the available memory on the computational nodes. Recall that we considered only two memory levels; the main memory and disks. The computing times for the Gauss-Jordan matrix inversion are shown in Tables 8 and 9. On the one hand, comparing these results
<table>
<thead>
<tr>
<th>Matrices sizes</th>
<th>Managing all communications</th>
<th>Using persistent storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>min.</td>
</tr>
<tr>
<td>3000</td>
<td>608 s</td>
<td>513 s</td>
</tr>
<tr>
<td>6000</td>
<td>3675 s</td>
<td>2765 s</td>
</tr>
<tr>
<td>9000</td>
<td>6770 s</td>
<td>4453 s</td>
</tr>
<tr>
<td>12000</td>
<td>26818 s</td>
<td>17555 s</td>
</tr>
</tbody>
</table>

Table 7
The GJ inversion on the Grid’5000 platform, blocks size = 1500

<table>
<thead>
<tr>
<th>Matrices sizes</th>
<th>Managing all communications</th>
<th>Using persistent storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>min.</td>
</tr>
<tr>
<td>3000</td>
<td>327 s</td>
<td>297 s</td>
</tr>
<tr>
<td>6000</td>
<td>1137 s</td>
<td>1107 s</td>
</tr>
<tr>
<td>9000</td>
<td>3860 s</td>
<td>3604 s</td>
</tr>
<tr>
<td>12000</td>
<td>10301 s</td>
<td>9087 s</td>
</tr>
</tbody>
</table>

with previous results, we see that the use of explicit I/O disk accesses locally gives better performance by a factor of 1.8 to 6 (for both implementations, with or without using the persistent storage scheme). Furthermore, the use of persistent storage increase the performance in all cases, by a factor of 1.3 to 3.5. On the other hand, we observe that the out-of-core programming is only interesting if the size of the matrix increases in the Grid’5000 testbed. Indeed, the classical versions perform better for the tested sizes when using persistent storage because the overheads related to the out-of-core schedule are more important than the potential gain in comparison to an initial thrashing effect (which in fact does not exist for the considered sizes). However, for the version which manages all communications, the computing times are better by a factor of up to 3.2. It is important to note here that the experimentation middleware does not allow to send bigger blocks of data, and thus to test bigger sizes. For the matrix-vector product, and using the Grid’5000 testbed, the computing times for version 2 (unidimensional distribution) using the out-of-core schedule is 39 seconds (by managing all communications) and 10 seconds (using the persistent storage scheme). This leads to the same observation as above, the classical version (Table 4) performs better and there is no need for I/O explicit management for these sizes. This was not the case in the non-dedicated computational platforms where the gain factor was up to 6.2 compared to the classical implementations.

However, the performance level reached is still relatively low. This is due to the low size of the tested matrices. Indeed, there are two limits related to the used middleware: the memory management in the client programs and the size of the data files. The client programming interface of XtremWeb is written in Java and memory management is made by the garbage collector, which cannot be managed explicitly and does not ensure that the client program will not be concerned by a lack of memory. It is possible to increase this size but the number of loops in the client program remains limited (and thus the choice of the partitioning and sizes of the matrices). The other issue is an upper bound of the size of the input data files. The scalability of the tested middleware is then primarily limited by the programming language, and also its design and the used protocols.

Finally, we emulated the inversion of very large matrices of size 90000 and 150000 by separating the computation process, the data transfer and the data compression processing. The blocks are of size 6000
Table 8
The GJ inversion on the non-dedicated testbed using local out-of-core schedule

<table>
<thead>
<tr>
<th>Matrices sizes</th>
<th>Managing all communications</th>
<th>Using persistent storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>min.</td>
</tr>
<tr>
<td>6000</td>
<td>2030 s</td>
<td>1793 s</td>
</tr>
<tr>
<td>9000</td>
<td>4111 s</td>
<td>3443 s</td>
</tr>
<tr>
<td>12000</td>
<td>5810 s</td>
<td>4910 s</td>
</tr>
<tr>
<td>15000</td>
<td>6991 s</td>
<td>5811 s</td>
</tr>
<tr>
<td>18000</td>
<td>14842 s</td>
<td>13020 s</td>
</tr>
</tbody>
</table>

Table 9
The GJ inversion on the Grid’5000 testbed using local out-of-core schedule

<table>
<thead>
<tr>
<th>Matrices sizes</th>
<th>Managing all communications</th>
<th>Using persistent storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>min.</td>
</tr>
<tr>
<td>6000</td>
<td>1112 s</td>
<td>1077 s</td>
</tr>
<tr>
<td>9000</td>
<td>1687 s</td>
<td>1611 s</td>
</tr>
<tr>
<td>12000</td>
<td>2320 s</td>
<td>2244 s</td>
</tr>
<tr>
<td>15000</td>
<td>2966 s</td>
<td>2838 s</td>
</tr>
<tr>
<td>18000</td>
<td>3544 s</td>
<td>3398 s</td>
</tr>
</tbody>
</table>

with subblocks of size 3000. The number of the computational nodes used is between 196 and 576. The average cost of the transfer and data compression processing of blocks of size 6000, by subblocks of size 3000, is 193 seconds under XtremWeb. Recall that the transfer by subblocks is due to the scalability issue related to this middleware. The average computing times (by using the persistent storage scheme) is $7.18 \times 10^4$ seconds for matrices of size 90000 and $1.23 \times 10^5$ seconds for matrices of size 150000. This gives $8.34 \times 10^4$ seconds and $1.34 \times 10^5$ seconds as computing times, respectively. The performance level reached in this case is up to 51GFlops. This specially shows that it is possible to reach interesting performance levels on the grid by the mean of well-adapted data management, and scalable management and experimentation tools.

Discussion

This work focuses on the problem of determining a suitable data management approach to derive a simple technique that can be applicable to multiple tasks in grid-based linear algebra. By using two types of data locality (persistent storage and out-of-core programming), we try to make the data accessible in an efficient way in local nodes, and optimise the data movement over the grid. The information of the placement and transfer of the data is of great relevance in the scheduling decisions. Indeed, we are currently implementing a workflow system which basically favours the moving of computation close to the data. The scheduler takes as input a set of calls of the computational jobs along with their dependencies and data requirements. It assigns the jobs to the processing resources based on the location of their parents in the task-graph. Then, it places the datasets accordingly, and controls the execution of the jobs and the data/job transfer. This especially allows deploying inputs that are not outputs of higher levels in the workflow. This is basically intended to maximise the migration anticipation rate. On the other hand, in order to fully support this programming approach in local nodes, an out-of-core library of basic linear algebra routines is needed.
In contrast to the large body of work on automatic data distribution on distributed memory machines, as well as on loop and data transformation discussed earlier, very few papers have considered grid-based approaches. The GrADS project [7] aims to adapt the existing distributed memory software libraries to fit into the grid. Basically, this project focuses on interfacing/integrating existing software, such as ScaLAPACK, to the Globus grid system. It employs the message-passing communication on top of the grid. Other systems, mentioned earlier, build linear algebra solving environments using the remote procedure call model (DIET, GridSolve, etc.). In comparison, we concentrate more on a general programming approach determining suitable layouts.

However, there are limits of the inherent parallelism in any application which incidentally limit the proposed technique. To address this and fully benefit from the proposed programming approach, future work will implement additional pipeline techniques in the data management hierarchy in order to overlap communications with computations and I/Os. In addition, the overheads related to different levels in the grid should be taken into account while considering the additional block partitioning stages. More analysis of the hierarchy of these overheads will also be considered to give more details about the effort that should be done at the data management level.

7. Conclusion

In this work we proposed a programming approach and performance evaluation of basic matrix algebra applications on the grid. We showed that the proposed optimisations for data locality management are effective. This reduces the communications needs and is still interesting, with significant factors, even with a relatively high quality of interconnection. We also showed that, in addition to well-adapted programming optimisations, the application developers need scalable tools, with large-scale and intensive properties, to reach a high level of performance and good scalability. Furthermore, the use of explicit I/O access is required in the case of large sizes and/or non-dedicated experimentation platforms.

Coding very large applications to achieve high performance and scaling on very large numbers of processors is still an important challenge. Constraints and limitations on existing models and related infrastructures make the path to scalability more difficult. Realistic expectations cannot be achieved as well. However, this work shows that we can adapt parallel algorithms of basic linear algebra, and in particular the block-based versions, for largely distributed environments by mean of well-adapted data (incidentally tasks) placement and explicit local memory management.

References


EU-IndiaGrid interoperability experiments using GridSeed tool

Balachandar R. Amarnath\textsuperscript{a,*}, Thamarai Selvi Somasundaram\textsuperscript{a}, Mohammad Kashif\textsuperscript{b}, Alberto Masoni\textsuperscript{c} and Stefano Cozzini\textsuperscript{d}

\textsuperscript{a}Department of Information Technology, MIT Campus, Anna University, Chennai, India
\textsuperscript{b}Department of Particle Physics, University of Oxford, Oxford OX1 3RH, UK
\textsuperscript{c}INFN, Sezione Di Cagliari, University of Cagliari, Monserrato, Italy
\textsuperscript{d}CNR/IOM Democritos c/o Sissa via Bonomea 265, 34136 Trieste, Italy

Abstract. This paper investigates the interoperability issues between European and Indian Grid infrastructure and implements methodologies to overcome some of them. The main objective of EU-India Grid interoperability project is to aggregate the computational resources available across Europe and India, and to use them effectively in solving computationally intensive scientific applications. As both European and Indian Grid is based on different middleware, the interoperability initiative leads to several issues with respect to job submission, resource management and monitoring, security, and data transfers. The European Grid is based on gLite middleware and Indian Grid Garuda is a combination of in-house software’s developed by C-DAC (Centre for Development of Advanced Computing), India, Globus toolkit and industry grade components. In this work, several interoperability issues have been investigated and a preliminary interoperable system to facilitate remote job submission, resource discovery between gLite and Garuda Grid infrastructure is proposed. An ExtendedUI is created to interoperate between the middleware and its prototype implementation is presented. This ExtendedUI consists of all the libraries and adapters for job submission across globus and gLite based Grid resources. GridSeed tool is used to setup experimental Grid infrastructure consists of resources based on both gLite and globus middleware and the extendedUI has been tested for job submission and information aggregation.

Keywords: Grid Computing, Globus Middleware, gLite middleware

1. Introduction

The emergence of scientific and industrial applications requiring large amount of computational power need seamless access to geographically distributed computational resources. This forces the respective institutions and organizations to harness idle computers on an international scale. However, there was no technology to support flexible and controlled sharing of various types of resources that are needed to solve computationally intensive applications. To address this issue, an extended distributed computing technology, termed as ‘Grid’, was coined by Ian Foster et al. that supports aggregation of distributed computational resources that spans beyond organizational boundaries, and their coordinated use to meet the requirements of advanced science and engineering [3–5]. Consequently, Grid middleware have been proposed, which performs basic authentication and authorization of the participants of the Grid and governs execution of job and monitoring the dynamic state of the participating resources. However,
depending on the demands of wide range of applications, research communities developed different middleware such as Globus, Unicore, gLite etc.

In addition to that, the emergence of standards related to Grid computing technology and associated tools enabled many research institutes and organizations to deploy Grid for their own applications such as Molecular Docking, Life Sciences etc. Currently most of the countries are providing more funds to encourage research and deployment of Grid infrastructure to solve high end applications such as Disaster Management, Drug Discovery and High Energy Physics. However, the middleware that power these Grids are different. They have different security infrastructure, different resource monitoring and job management mechanisms. Further, there was no issue of interconnecting a Grid with another one during the inception phase. Hence, before the idea of standardizing the middleware architecture was initiated, different middleware’s were developed and they have been widely used by various research institutes to develop their Grid Infrastructure based on their needs.

Recently, there are initiatives to interconnect different national Grids. In this paper, one such initiative which aims to interconnect European and Indian Grid infrastructure to aggregate huge computational resources spanning across these two Grids [2] is discussed. Funded by the European Commission, the Enabling Grids for E-sciencE (EGEE) is a seamless Grid infrastructure to integrate current national, regional and thematic Grid efforts to support scientific research. At present it consists of 250 sites in 48 countries and 68000 CPUs to cater 8000 users on 24X7 basis. The gLite middleware distribution developed by the Enabling Grids for E-sciencE (EGEE) project of Europe is now deployed by about 80 percent of the sites connected to the EGEE infrastructure, making it the main middleware distribution used in production. gLite makes use of components from other Grid middleware projects and is designed as a modular system to allow users to tailor the system to their specific needs by deploying the services they require, rather than having to use the system as a whole [6]. Garuda is one of the major Grid initiatives of Government of India to provide computing power and help research in the field of high energy physics, disaster management, astronomy and bioinformatics etc. It connects and aggregates high end computational resources from 45 research laboratories spanning over 17 cities across the nation [15]. Garuda middleware is built with in-house developed software’s, Globus Toolkit [7] and standard commercial products. Globus Toolkit is the base on which other components are implemented. Currently Garuda is using Globus 2.4 but they are planning to migrate to Globus 4.x version. Garuda Integrated Development Environment (G-IDE) and Paryavekshanam are softwares developed by C-DAC for Grid application development and monitoring of the Grid respectively. MOAB from Cluster Resource Inc. has been used for metascheduling.

The EU-India Grid interoperability initiative has lead to high level collaborative research activities between the participating organizations of Europe and India. In this case, there is a problem of interoperability which is to be solved between these two Grids as they were built using different middleware. To interoperate two Grid infrastructures each with different middleware, there can be two approaches; an adapter based approach that translate the specific design aspect from one Grid middleware to another and, a standard based approach in which the components of both the Grids follow specific international standard. Still there is no de-facto standard for most of the Grid services, and developing such standard is also a very long process. So the scope of the paper is restricted to short term interoperation using adapter approach.

In this paper, an architecture for interoperation between gLite and Globus based Garuda Grid infrastructure is proposed. An extended User Interface (UI) is developed for job submission across both the middleware. Suitable adapters have been implemented in the extendedUI for switching between middleware for job submission and resource management. Further, gLite and Globus based Grid infrastructures
have been created using GridSeed tool [9], a training toolkit developed by ICTP, Trieste, as part of EU-India Grid interoperability project. With this architecture, it is possible to achieve interoperation and interoperability between gLite, Globus 2.4 and Globus 4.0 middleware based Grid infrastructure.

The rest of the paper has been arranged as follows: – Section 2 highlights other research works that comes closer to the one described in this paper. Section 3 describes the proposed architecture in detail. Section 4 explains the experimental setup that implements the proposed architecture. Section 5 concludes the paper by highlighting further scope of this paper.

2. Related work

In this section, some of the research works carried out across the globe from which first blood to propose this idea was drawn is discussed.

Grid Interoperation Now (GIN) [16] Community Group (CG) of the Open Grid Forum (OGF) is a group of representatives from world-wide Grid, e-Science infrastructures and other national Grid initiatives. Its aim is to organize and manage interoperation efforts among different production Grid infrastructure. GIN-CG is working on different areas of Grid middleware through different working groups like GIN-INFO, GIN-DATA and GIN-AUTH etc and it has practically demonstrated Grid interoperation between Grid infrastructures like EGEE, OSG, NAREGI, DEISA etc.

OMII-Europe [14] project has been funded by European Union to explore interoperability among several heterogeneous Grid infrastructures. OMII-Europe follows the standard based approach to achieve interoperability. It has selected specific components like Basic Execution Service (BES), Accounting Service, Database service and VOMS services and makes them available across middleware platforms.

In EU-CHINA [1] project, adapter based approach is implemented to achieve interoperability between China and European Grid. China Grid is using Grid Operating System (GOS) middleware and European Grid is using gLite as its middleware. Different adapters have been used to interoperate various components of middleware, like JSDL to JDL and JDL to JSDL adapter for job submission.

The Australian Research Council has funded the GRIDS lab of the University of Melbourne towards Gridbus project. The GRIDS lab has developed Gridbus Broker which schedules jobs to Grid resources across middleware such as Globus 4.0, Globus 2.4 and also to local schedulers such as SGE, PBS etc [11].

The ‘InterGrid’ project focus on internetworking heterogeneous Grids across the globe by proposing InterGrid gateway peering arrangement at every Grid. Also, it deals with the formation and management of virtual organization across the Grids [12,13].

Gridway Metascheduler, developed by University of Madrid, achieves interoperability between gLite and Globus middleware. Initially, it was developed as a metascheduler for job scheduling across Globus Grids. Exploiting the Globus 2.4 characteristics of gLite, it is also possible to schedule jobs across gLite based computing elements but one has to bypass WMS component of gLite middleware [10].

Commodity Grid (CoG) Kits [8] enables Grid users, Grid application developers, and Grid administrators for rapid Grid application development. It provides the implementation of Java-based GSI, gridFTP, myProxy, GRAM client implementations and capable of submitting job across several versions of Globus middleware but not other middleware.

3. Proposed architecture

In this section, the main issues that affect seamless joint usage of the European and Indian Grid infrastructure are discussed and an architecture is proposed to overcome them. The major interoperability issues which have been foreseen are listed below:
– Mechanism for aggregation of resource information
– Issues related to Security infrastructure towards authentication and authorization and
– Job submission to the resources across two Grids.

In order to address these issues, an adapter based system should be implemented in a centrally located
user interface through which job submission to both European and Indian Grid is accomplished. This
system should consist of various adapters for interoperation of different middleware components of both
the Grids. The adapters proposed in our system are aimed to interoperate between gLite and Globus
2.4 and 4.x based Garuda infrastructure. This is because of the fact that Garuda Grid will migrate to
Globus 4.x in future. Further, the Globus 2.4 is developed based on Component based Model whereas
the Globus 4.0 version is based on Web Service framework. Hence, there is a need to develop different
adapters for Globus 2.4 and Globus 4.0 based Garuda middleware.

Since core components of Garuda, like information services and job management is based on Globus,
the characteristics of gLite, Globus 2.4 and Globus 4.0 middleware is compared with respect to interop-
erability issues mentioned above.

3.1. Globus vs gLite – Information service

The Information Service in both Globus 2.4 and gLite middleware is built with MDS component
of Globus middleware. The MDS in Globus 2.4 is implemented with two services namely Grid Index
Information Service (GIIS), which provides an aggregate directory of lower-level data, and Grid Resource
Information Service (GRIS), which runs on a resource and acts as a modular content gateway for the
resource. A GRIS registers with a GIIS, and one GIIS may register with another using a soft-state protocol
that allows dynamic cleaning of dead resources. However, for scalability reason in gLite middleware,
a new component called Berkeley Database Information Index (BDII) is used to store and publish data
from the local GRISes. Although, both of the middleware uses LDAP for storing low level resource
information, they use different schemas; publish almost same information but in different formats. Data
in the MDS in gLite conforms to the LDAP implementation of GLUE schema, whereas, in Globus,
it follows its own schema called as MDS schema. So by querying LDAP database, it is not possible
to obtain resource information of both the middleware. Further, in Globus 4.0, the MDS component
has been implemented as a web service called as WS-MDS. It has its own web service interface for
interacting and aggregating resource information from the underlying middleware.

Hence, separate adapters have to be developed to query the information sources from these middleware.

3.2. Globus vs gLite – Execution management service

In gLite middleware, Workload Management System (WMS) component governs the job submission
and execution. In addition to this, it can also discover suitable Grid resources across the Grid that matches
the application requirements with the help of Condor Matchmaking system implemented in WMS. The
Execution Management service in Globus is implemented in GRAM component. GRAM simplifies the
use of remote resources by providing a single standard interface for requesting and using them for the
execution of jobs. GRAM is designed to provide a single common protocol, API for requesting and
using remote resources, a uniform, flexible interface for local job scheduling. Jobs are described using
Resource Specification Language (RSL). In Globus 4.0 version, this component is implemented as a web
service called as WS-GRAM. Hence, it has web service interface for job submission and also RSL is
used to describe the job is in XML format. With this diverse nature of components, there is a need for an
adapter to negotiate with both the middleware for job execution on behalf of the user from a single UI.
3.3. Globus vs gLite – Grid security infrastructure

The Grid Security Infrastructure of both gLite and Globus enable secure authentication and communication over an open network. It is based on public key encryption X.509 certificates, and the secure sockets layer communication protocol, with extensions on single sign-on and delegation.

However, in gLite, an additional service called Virtual Organization Management Service (VOMS) is included which provides information about the roles and privileges of users within the VO. In gLite, X.509 certificates are created with VOMS extension. So, there is a need for a component that can map the certificate of Globus users to access gLite based Grid resources and vice versa.

Our proposed architecture, shown in Fig. 1, includes necessary components such as User Interface, Job Submission Adapter and Information Service Adapter which have been described in detail in the following sections.

The User Interface component contains necessary functionalities for interacting with Grid environment transparent to the underlying middleware. It includes Globus client libraries as well as gLite based User Interface libraries for job submission across both the Grids. It enables the user to query the top BDII of the gLite Grid and WS-MDS of Globus based Garuda Grid for obtaining resource information. It can also interact with WMS of gLite and WS-GRAM component of Globus for job submission. It allows the user to submit the job described using JDL as well as RSL so that they can run their job either in gLite or Garuda resources.

The Information Service Adapter aggregates the resource information across Globus and gLite Grid middleware. Since, the information schema used for publishing resource information in these middleware is different, an adapter is needed to contact the respective middleware for collecting resource information. In gLite middleware, the MDS collect the resource information and stores in LDAP server. This resource site information is then stored in BDII. The Information service adapter implements a script that contact the BDII for retrieving resource information across gLite Grid. Similarly, in Globus based Grid, the resource information is obtained through LDAP server and registered with GIIS. In this case, the adapter uses a script to contact GIIS for resource information retrieval in Globus based Grid. This information shall be effectively used for discovering suitable resources depending on the requirements of the job submitted by the user.

The most important component in the architecture, the Job Submission Adapter (JSA), contains appropriate software to submit jobs to suitable resources present across both Globus and gLite Grid.
infrastructure. The gLite middleware provides inbuilt matchmaking mechanism through its WMS component. However, in Globus middleware, the matchmaking component is not provided and it needs to be developed externally. Hence, in Garuda Grid, a semantic based resource discovery mechanism described in [17] is implemented which performs matchmaking of resource request against the available resources. It uses ontology representation of available Grid resources thereby forming a resource ontology knowledge base. Algernon inference engine is used to retrieve resource information from the knowledge base. The Grid Resource information is populated onto the ontology by interacting with information database maintained in Garuda Grid. As soon as, a suitable resource is discovered, depending on the middleware installed in it the adapter prepares the job and schedules it to that resource. For instance, if an user submits his job described in JDL, it contact the WMS component of gLite Grid for finding suitable resource. If a resource that matches the requirements is found in gLite Grid, then the job submission adapter sends the job to WMS of gLite for execution. If the suitable resource is not available in gLite Grid, it then invokes trce information service adapter. Conversely, if a resource is available in Garuda Grid, job submission adapter converts the JDL description of the job into RSL and schedules the job to Globus using the Globus client libraries. Similarly, if the user submits the job described in RSL, it contacts the matchmaking mechanism of Garuda Grid, which requests WS-MDS for suitable resource. If found, it schedules the job to that resources through GRAM component of Globus middleware. If not, it then contact information service adapter, which query the top BDII of gLite for resource information. If resource is available, the job submission adapter converts the RSL description to JDL and submit the job to WMS of gLite Grid which inturn discovers suitable resources and submit the job to it.

In this architecture, there is a need for certification authority to establish secure communication channels which enables mutual authentication and authorization between the Grid resources. The complexity in this case is that the additional attribute in the form of VOMS in gLite X509 certificate in gLite Grid. In this architecture, it is proposed to use Grid proxies without VOMS extension when submitting job to Globus resources and create VOMS proxies when submitting job to gLite resources.

4. Experimentation with GridSeed

The proposed architecture has been integrated and tested in a controlled Grid environment and only after a careful testing and analysis could be installed in production environment. To create such a controlled environment, GridSeed (GS), a tool developed by the EU-India Grid project that simplifies the setting up of a fully fledged Grid infrastructure based on the gLite middleware by means of VMware machines is used.

GridSeed consists of a set of VMware virtual machines, each of them dedicated to specific Grid element. In order to install, configure and work with a gLite Grid, the basic hardware requirement is just to have enough physical machines connected to local area network with the VMware player and/or VMware server installed. VMware based GridSeed machines are then booted in a specific order: all central services (CA/VOMS/TopBDII/WMS etc) must be up and running before a Grid site can operate. The configuration of each service proceeds in parallel with its installation and hence there is therefore a precise ordering for the configuration and booting of a full gLite Grid. The first machine to bootup is CA. It consists of an Apache web server, a DNS, an implementation of NTP server and a Certification Authority based on scripts. For other services, a script is provided in /etc/init.d, so that, when a particular service is started on new virtual machine, it takes host certificate automatically from CA using client-server model, where CA is the server. All the virtual machines are synchronized through NTP server provided in CA. Different scripts are available to configure services depending on the infrastructure. For
example, if two CE’s were booted, then a script will add this information in BDII’s bdii-update.conf file so that it can fetch information from CE. GridSeed encapsulates this knowledge and leverages on it when automatically configuring a Grid on the fly. GridSeed is designed to create and configure a gLite Grid made up of 1 WMS, 1 Top BDII, 1 LFC, 1 VOMS, up to 99 UI, up to 255 Sites each one consisting of 1 CE + 1 SE + up to 252 WN. More information about GridSeed can be found in https://euindia.ictp.it/Grid-seed.

In our experimental setup for interoperability experiments, GridSeed is used to create gLite based Grid infrastructure first. The gLite UI machine allows users to submit job to the gLite Grid. A script is provided in this machine to obtain user certificate for every user from CA. The other central services were booted and up for proper operation of Grid followed by a Computing Element and a set of Worker Nodes. This infrastructure has been then tested for proper operation such as resource discovery, job submission through WMS.

Once GridSeed Grid infrastructure is operating, a separate VMware based Virtual Machine was booted and Globus middleware version 4.0.0 was installed. Host certificate and User certificate for this machine have been obtained from GridSeed CA machine after properly modifying the CA script provided in GridSeed. Also, this machine is time synchronized with gLite Grid by editing the NTP script present in GridSeed CA machine.

In our adapter based approach, the user can submit their job described either with JDL or RSL. If suitable resources are not found in one Grid infrastructure, the adapter can prepare the job for the other
middleware and submit. The proposed adapter components must be installed in a machine in which both gLite based UI and Globus client libraries are available. In the GridSeed based experimental setup, GT4 client in gLiteUI machine is installed for cross job submission between Globus and gLite middleware. It has been termed as extendedUI because all the proposed adapters have been implemented in this UI as shown in the Fig. 2.

A software has been written in UI machine which can accept job from the users that are described either by JDL or RSL and is sent to switch module. If JDL is submitted, the switch, would try to submit the jobs to gLite resources. The gLite resource discovery module queries the topBDII through WMS for finding out suitable resources matching the requirements specified in the JDL. This module has been implemented using the user interface command provided by gLite middleware to query information services. However, if the resources are not available in gLite Grid, the switch component will convert the JDL into RSL and sends the job to Globus job adapter. The Globus job adapter is implemented using GT4 client libraries and hence the job can be scheduled to Globus 2.4 as well as Globus 4.0 middleware. Similarly, if RSL job is submitted, the adapter would try to submit the jobs to Globus resources. If the resources are not available in Globus Grid, the switch component will convert the RSL into JDL and sends the job to gLite job adapter. The gLite job adapter is implemented using gLite based UI libraries with which the job can be scheduled to gLite WMS middleware. This entire flow of operation is depicted in the Fig. 3.

5. Conclusion

In this paper, an architecture based on extendedUI is proposed to achieve interoperation and interoperability between gLite and Garuda Grid. The centrally located extendedUI can also be implemented
as a web service acting as Grid portal through which users can submit job. This architecture does not require any piece of software to be installed in the participating Grid resources and hence preserving their autonomy. The extendedUI can be developed as an independent Grid component which can be downloaded and installed by the users who wish to use both gLite and Garuda resources. For experimentation purpose, the resource discovery mechanism is implemented with a static file containing resource characteristic and their hostnames, with which matching is done based on keywords.

Further investigation shall be done to extend this implementation in a larger scale which requires collaborative efforts from many research institutes across Europe and India Grid infrastructure. We believe the active contribution of Grid community and flexible resource usage policies will lead to success of this interoperability project.

**Acknowledgement**

The authors take immense pleasure to thanks Centre for Advanced Computing Research and Education of Anna University, India for providing moral support to take up this work. The authors also thank Ministry of Italian University Research for providing financial support to carry out collaborative research activity to investigate the interoperability issues in EU-India grid project.

**References**


Extending self-healing in grid environment by pulse monitoring

Inderveer Chana and Seema Bawa
Department of Computer Science and Engineering, Thapar University, Patiala, India
E-mail: inderveer@thapar.edu, seema@thapar.edu

Abstract. The increasing complexity of Grid services and systems demands larger human effort for system configuration and performance management. The physical management of these systems is not only tedious but also error-prone. Inherently, effective monitoring is essential for Grid management, which can further facilitate self-management of the Grid as advocated by Autonomic computing i.e. shifting the burden of managing systems from people to technologies. This paper proposes a pulse monitoring based Grid Monitoring System that passes the monitoring data to Fault Discovery and Healing System, which has been designed for facilitating self-healing in Grid. The Grid monitoring system demonstrates that through efficient monitoring, self-healing can be achieved in a Grid environment and Grid can work autonomously, thus minimizing the physical management and service-downtime.

Keywords: Grid computing, self-management, autonomic computing, grid monitoring system, fault discovery and healing

1. Introduction

Grid is a collection of heterogeneous resources, which are open, shared and often geographically distributed and aim to achieve high computational performance collaboratively [1,2]. The increasing complexity of grid services and systems and further its dynamic nature demands correspondingly larger human effort for system configuration and performance management. In Grids, there are many failure possibilities, including not only independent failures of each element, but also those resulting from interactions between them [12]. Detection and diagnosis of failure in a very complex and heterogeneous environment such as a computational grid is an intricate job. To solve this problem self-healing principle of Autonomic Computing provides a requisite solution and advocates autonomic functioning of a grid [3]. This automated operation is not only necessary for reducing human administration effort, but also to limit possible sources of errors and to further reduce service downtime.

Self-healing can be achieved by firstly monitoring the actual state of hardware and software components, correlating the sampled data with the goal state and further automatically devising actions for repairing or updating the affected components of a system. Till-date self-healing has been quite popular in energy infrastructure systems. Modern operating systems like Solaris 10 too have self-healing features as has been discussed in Queue [23] but not many self-healing systems are available for Grids. The prevalent monitoring systems like Hawkeye [30], Ganglia [10], RGMA [35], when used independently, report only about the occurrence of failures but further, debugging, fault isolation and restoration is still a tedious and physical job done by the administrators manually. Different monitoring systems and tools have been used in Eurogrid, BIO Grid and GRIA but none of these Grids are based on pulse
monitoring [15] and the monitoring information has also not been put to efficient use for self-healing in the Grid environment.

These considerations motivate this work which concerns how self-healing can be achieved on the basis of an efficient monitoring system for a Grid environment. We describe the design of both a pulse monitoring based Grid monitoring system and a Fault Discovery and Healing System (FDHS) for self-healing of a Grid. This work demonstrates that monitoring information can be used effectively for self-healing that can further improve the level of automation and self-management capability to a great extent in a Grid environment. This can further ensure the development of Autonomic Grid and thus lower the administrative burden of managing multiple heterogeneous hardware and software components at a Grid computing site. The organization of the paper is as follows: Section 2 discusses the background of Autonomic Computing, goals of self-healing and related work, Section 3 describes Pratham, a pulse-monitoring based grid monitoring system; its architecture, components and implementation, Section 4 discusses how self-healing can be achieved through fault discovery and healing system, Section 5 presents a case study of a campus wide Grid, demonstrating the inclusion of monitoring system and fault discovery and healing system into its architecture, Section 6 finally concludes the paper.

2. Background

IBM introduced Autonomic Computing in 2001, with the aim to develop self-managing systems [4, 5]. Autonomic Computing helps to address complexity by using technology to manage technology. An autonomic system senses its operating environment, models its behavior in that environment, and takes action to change the environment or its own behavior. It has the following four properties (as shown in Fig. 1): Self-healing, Self-protecting, Self-configuring and Self-optimization.

2.1. Autonomic framework

Figure 2 shows a Grid environment. Here the resources, which collaborate to form the Virtual Organization (VO) [1], form the base layer. Their respective resource managers manage these resources and constitute the second layer. These resource managers further utilize the Autonomic capabilities.
extended by the Autonomic managers (Layer 3 and 4) and finally Grid manager manages the VO of the Grid and forms the highest layer.

The manual and autonomic manager layers can obtain and share knowledge via knowledge sources. Managers are responsible for determining which tasks can best benefit from autonomic self-managing technologies and they develop the policies that direct the autonomic function. A policy often starts as a technical implementation of a defined manual process. A self-managing autonomic system requires [3]:

1. A collection of self-managing resources that implement the desired function.
2. Autonomic and manual managers that implement system functions that enable the required system-level behaviors.
3. Knowledge Base that contains management data such as fault and remedy information.
4. Design patterns that offer models for the structure and arrangement of components for system self-management.

2.2. Self-healing

Self-healing implies that the system automatically detects, diagnoses and repairs localized software and hardware problems [6,7]. In a Grid the main aim is to keep VO free from the burden of manually maintaining resources, thereby allowing them to concentrate on the higher-level organizational tasks in the Grid. The main goals of self-healing are:

1. To predict problems and take necessary actions.
2. To make the system more fault tolerant

Primarily, self-healing can be achieved through monitoring [8,20]. Therefore, to facilitate self-healing of a Grid, monitoring data about different Grid components must firstly be collected and then it must
be stored and made available for analysis by fault detection systems [17,18]. To achieve this, the major components of a monitoring system must include:

- Information collectors (sensors, benchmarks etc.)
- Support Services (Collection, archiving, management etc.)
- Interfaces (GUIs or APIs)

Along with these components, within a monitoring system, a schema must be defined to express the data collected. With a well-defined schema, different monitoring implementations can share same data and understand one another’s data sources.

2.3. Related work

Monitoring systems have been identified and studied in various information system disciplines. One of such classifications of existing grid monitoring systems has been presented in a report by Serafeim Zanikolas et al. [9]. Similarly a general monitoring system has been developed at the DIII-D National Fusion Facility [29] that combines code run status, data analysis tracking, logfile access, complex error detection, and expert system capabilities. This system allows for rapid detection of discrepancies in diagnostic measurements or the results from physics analysis codes. A report by Salima Benbernou et al. also summarizes the existing monitoring systems in their report of S-Cube [27]. Another monitoring system LEMON [32], used in GridICE [31], uses sensors. On every monitored node, a monitoring agent launches sensors for retrieving monitoring information; it communicates with them using a push/pull protocol. It is useful in performance monitoring and helps system administrators in debugging problems but does not exhibit exact cause of failure. To monitor the behavior and performance of WORLDGRID testbed and spot problems as soon as they arise, the Work Package 4 of DataTAG project has developed the EDTMonitor tool based on the Nagios [34] package that allows for Virtual Organization centric views of the Grid through dynamic geographical maps.

In their paper, Sherif A. et al. [25] focus on how the self-healing autonomic attribute can be implemented and applied using Web Services. In this work, the decision-making engine uses a symptom database, which is an XML file containing symptoms and recovery actions. CoreGrid Technical report on monitoring framework for self-healing [28] presents a fine-grain monitoring framework architecture based on aspect-oriented technology. Aspect oriented programming offers the possibility to inject monitoring code dynamically, and even activate or deactivate monitoring code at runtime, allowing the collection of metrics at the appropriate granularity. In Unity [26], a decentralized architecture for autonomic computing based on multiple interacting agents called autonomic elements, an autonomic element within each application environment computes a resource level utility function based on information specified in that application’s service-level utility function. These utility functions from multiple application environments are sent to a resource arbiter element, which computes a globally optimal allocation of servers across the applications. In this work, we present Pratham Monitoring System; a pulse monitoring based grid-monitoring system, which helps in self-healing of the grid through efficient use of monitoring information, generated over-time. Here self-healing has been implemented in a single element, the Fault-tolerance rules.

3. Pratham monitoring system

Pratham is an automated agent-based architecture, which helps in monitoring the Grid resources based on pulse monitoring technique [15]. Pulse monitoring is a hybrid approach for the autonomic environment. It employs the necessity concept of the beacon monitor [14] to turn the heartbeat monitor [13]
into a pulse monitor. In pulse monitor instead of just checking the presence of a beat, the rate is also measured. The pulse monitor provides an indicator of the health of the system instead of simply checking presence or absence of liveliness of a component.

### 3.1. Architecture of monitoring system

Architecture of Pratham Monitoring System is shown in Fig. 3. Assuming that a Grid environment may have different middleware installed, resource information needs to be gathered from them and stored in a Resource Information Collector. This information is transformed into corresponding OWL and sent to the Semantic Repository [24]. The main aim is to provide a uniform schema of resource descriptions for the monitoring system. This resource information is being used by the Autonomic Manager, which manages all the managed components (resources) inside a Grid environment. Every Managed component has an agent, which sends out pulses to the Autonomic Manager at defined frequency. The Autonomic Manager consists of a self-monitor, pulse monitor and an actuator. The pulses are intercepted by the pulse monitor and sent to self-monitor which analyzes the data received and executes the corresponding action by the Actuator.

### 3.2. Components of monitoring system

#### 3.2.1. Resource information

This component collects information from the repository or index service of the Grid middleware, assuming that there can be more than one middleware running in the Grid environment.

#### 3.2.2. Resource Information Collector

This repository stores the information of resources collected from different middleware inside a DBMS like MySQL.
Table 1

<table>
<thead>
<tr>
<th>Bit (2)</th>
<th>Bit (1)</th>
<th>Bit (0)</th>
<th>Meaning of pulse</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>All components are working normal</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Memory approaching max limit</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>CPU Load approaching max limit</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Disk Usage approaching max limit</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Major component has failed</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Network traffic exceeding the limit</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reserved (Can be used in future)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Complete failure, needs immediate attention</td>
</tr>
</tbody>
</table>

3.2.3. Transformation Process and Semantic Repository

Different middleware represent and store their resources differently, therefore it becomes difficult to share the common data in a VO. To avoid this and for better interoperability the information is converted into Resource Description Framework (RDF) using Web Ontology Language (OWL) and stored along with their ontologies in a Semantic Repository. This repository can be queried for the states of different resources, their performance data etc. on uniform basis.

3.2.4. Self-Monitor

Self-monitor gathers the information about the resources and for each resource checks the corresponding pulse data generated by the pulse monitor on the basis of rules. If some deviation from normal behavior is assessed, the actuator is activated. The alterations in components for self-healing of the Grid are processed in this component. (It houses FDHS discussed in Section 4.1)

3.2.5. Pulse Monitor

Pulse monitor intercepts pulses sent by each resource. As per the pulse rules, the meaning is passed on to self-monitor which utilizes it for adjustment information.

3.2.6. Actuator

Actuator takes the corresponding action as directed by the self-monitor. Input is given by self-monitor and corresponding to this input, it generates the output for the managed component.

3.2.7. Resource (Managed Component)

Each individual resource in VO gets registered with the VO Manager and the Manager installs an agent on the resource to make it a Managed Component (MC). The Managed Component is created using agent based technology as agents [21] can perform a number of tasks. They can securely start monitoring programs on any host and manage their resulting data. They can provide a standard interface to host monitoring sensors for CPU load, interrupt rate, TCP retransmissions, TCP window size, and so on.

In case of Pratham Monitoring System, the agent residing on each of the resource takes resource’s input information from Ganglia [10], a monitoring tool and passes it on to the pulse monitor.

3.3. Implementation of monitoring system

Pratham Monitoring System has been implemented through agents. A sending-agent is loaded on each resource at the time of its registration with the middleware manager. One listening-agent resides on the manager. The pulses are sent in the form of packets, which are deciphered at the listening-agent for inferring the meaning regarding the health of each resource. Packet details, being sent by sending-agent have been designed as shown in Table 1.
The listening-agent (part of pulse monitor) assesses the packet details, bit by bit. For example, data 000 (0) means, all components are working normal and no action needs to be taken, data 010 (2) means, the resource would be approaching its load limit and the corresponding action would be to inform the manager so that no more jobs are directed towards this resource. Similarly, if complete failure occurs, job migration might be required.

A standardized pulse monitor (shown as part of Autonomic Manager) can provide a convenient design of fault handling and provide efficient information for self-healing. Agent residing on the managed resource sends out a steady beat of pulses at defined frequency. When pulse monitor detects these pulses, it indicates that the autonomic element is alive and when missing, it indicates an operational problem, like network failure or complete failure of the element itself. Pulse monitor further passes this information to the self-monitor (which comprises of FDHS) discussed in the next section.

4. Self-healing

To achieve self-healing goals, primarily error and fault prediction is required. Failures in Grid can be due to many reasons similar to configuration problem, independent failures of each element, failures resulting from interactions between components etc [16,19]. The existing solutions for these failures are mainly application dependent. To facilitate the self-healing of the Grid, the fault discovery and healing system uses monitoring data that has been generated by the Pratham Monitoring System.

4.1. Architecture of the fault discovery and healing system

The Fault Discovery and Healing System (FDHS) is rule based. Each rule compares the data received from the monitoring system against the permissible limits. If the condition of a rule holds, the corresponding action is performed. Figure 4 shows the components of FDHS. FDHS resides in the ‘self-monitor’ shown in Fig. 3.
4.2. Components of the fault discovery and healing system

The main components of the FDHS are:

4.2.1. Inference engine

The Inference Engine is the heart of FDHS and is rule-based engine. It operates by the method of forward chaining. In order to execute a rule-based Inference engine using the method of forward chaining there is a need to fire (or execute) actions whenever they appear on the action list of a rule whose conditions are true. This involves assigning values to attributes, evaluating conditions, and checking to see if all of the conditions in a rule are satisfied. This inference engine reads the resource information, which is stored in the Resource Repository along with the pulse data, which is being transmitted at defined frequency by the sending agent residing on each individual resource. This data is then compared with the Fault Tolerance Rules, which have been predefined. If resource is performing within the configured limits no action is taken else if adjustments are required the corresponding action is executed by launching the Actuator. This process works in a control loop and the corrections are made till the resource performs again within the permissible limits.

4.2.2. Fault tolerance rules

Fault Tolerance Rules have been predefined corresponding to the pulse data shown in Table 1. For example, for pulse data ‘010’, the scheduler automatically migrates the job to the next available and eligible system. In total, eight rules have been defined as per requirement. These rules have been kept simple to avoid situations where different rules conflict with each other. Currently only three bits are being sent, in future, if some more inferences need to be drawn then more bits can also be introduced.

4.2.3. Resource repository

Repository stores the details of the resources along with the current details of Jobs being executed. It stores the details of the resources’ uptime, current state, last down time, downtime duration etc. It also stores the past performance data of the resource so that future performance predictions can be made and past actions which have emerged successful may be used again by the inference engine.

4.2.4. Actuator

Actuator executes automatic recovery actions. For example, resource or service restart may be required in case of some fault, automatic backup may be required or alert messages may be mandatory to be passed on to the middleware managers etc. The action can be executed by the Actuator on the individual resource or on the manager of the middleware as per the requirement.

5. Incorporating self-healing into a campus-wide grid: A case study

This section discusses the case study of a campus-wide Grid for Thapar University (TU). Different departments at TU, such as Computer Science and Engineering, Electronics and Communication Engineering, Instrumentation Engineering, Applied Sciences, etc, have individual computational resources. A compute-intensive research project, may be restricted by the limited resources available within the local administrative domain due to the lack of resource co-operation or occupy and overload all the local resources, while those in other departments may be lying idle. Therefore, this campus Grid setup is
motivated by the need to allow easy access to researchers to computational resources across the departments, anywhere within the University. TU Grid comprises of several clusters of systems managed by different middleware, which interoperate through Grid Bus Broker [33]. Users can have an access to Grid for application submission via the TU Grid Portal. TU Grid has a layered architecture as shown in Fig. 5. It comprises of following layers:

**Fabric Layer** – Consists of the set of all available resources forming the part of Virtual Organization. Individual computers, Condor pools, file systems, archives, metadata catalogs, networks, sensors, etc., etc.

**Grid Middleware** – Consists of the middleware layer and has SUN N1, Condor, Globus and Alchemi as the middleware for the said architecture. These middleware have their local resource Managers for the resource management tasks.

**Broker Layer** – A Broker layer is required for the interoperability amongst the middleware and for job scheduling. Grid Bus Broker has been chosen for interoperability. This layer also houses the self-monitor and the pulse-monitor (components of Autonomic Manager). As this is a multi-Grid middleware environment the actuator is connected to the scheduler, which schedules jobs on different resources through their respective resource managers. FDHS resides in the self-monitor unit of Autonomic Manager.

**Application Interface** – This interface is used for submitting the user jobs. Here it is the (Thapar University Portal) TU Grid Portal.

In the campus Grid, the lowest level, that is at the cluster level resource management is being done by the local resource managers and at the Grid level, broker takes care of the management part. Implementation
of these self-management properties has been done at the Broker layer. For making the Grid autonomic, the self-healing framework requires a monitoring system to monitor the Grid in totality and fault detection and recovery system for detecting the faults and automatically recovering from them, making the Grid self-healing.

5.1. Incorporating self-healing in TU Grid

Self-healing in TU Grid has been accomplished by incorporating the Pratham Monitoring System and corresponding FDHS into the TU Grid. As shown in Fig. 5, for TU Grid, the Portal sends the jobs to the Chief Manager, which handles the resource brokering and job scheduling part. At the time of registration of a new resource in the VO, the Grid middleware manager registers the resource and then this information is passed on to the Chief Manager (Broker Layer). The Chief Manager subsequently installs the sending-agent on the resource for making it a Managed Component. The Job Monitor collects the information of different resources in the campus through the managers of the Grid Middleware and stores it into the repository. The Autonomic Manager (shown in Fig. 3) comprises of the Self-monitor and the Pulse monitor, who perform their respective roles as discussed. In case of Pratham Monitoring System, the sending agent takes resource information input from Ganglia and passes it on to the pulse monitor. Pulse monitor further passes this information to self-monitor (comprising of FDHS). If some adjustment is required the information is passed on to the Scheduler, which activates the Actuator. The Actuator then sends an agent for executing the adjustment on the actual Managed Component i.e. the Resource. This process is conducted in the form of a control loop and thus helps in self-healing of the components individually and the Grid in totality.

5.2. Experimental results

A Grid environment has been setup in the Centre of Excellence (CoE) for Grid Computing, computer laboratories in different departments etc. in TU by installing various middleware on different systems. For experimentation, the test bed consisting of two middleware, namely Globus and Alchemi has been chosen. Globus Toolkit 4.0 has been installed on six systems having Fedora core 6 operating system at CoE for Grid Computing at Computer Science and Engineering Department. Alchemi 1.06 has been installed on twenty systems within the CoE for Grid Computing, CCCT lab and Software Engineering. Grid bus broker along with grid-monitoring tool Ganglia has been installed for generating the monitoring information.

For Grid monitoring, the resource information is desired. This information is collated in the base tables of the middleware and has been extracted and collected in a central repository. This resource information is converted into OWL and stored in semantic repository (shown as Repository). On each resource Gmond component of Ganglia (a multi-threaded daemon, which runs on each cluster node to be monitored) has been installed along with the sending agent. Ganglia creates graphs for each individual node similar to the one shown in Fig. 6.

Gmond also generates resource information in XML format as shown in Fig. 7. This information is parsed and deciphered by the sending agent and sent to pulse monitor in the form of pulse data (part of Autonomic Manager), which is further sent to FDHS. In case the values exceed the threshold, an alert is generated and stored in the resource repository and the administrator can directly find out not only the failure but also the reason of fault and may take further steps to fix-up the error. Incase more than one problems surface for a resource say for example, network traffic exceeding limit and disk usage approaching max but the CPU load is low (which might be reported by more than one simultaneous
pulses), would indicate a data throughput problem and not a computational one. Thus, decision-making support regarding fault diagnosis is also provided to the administrator.

A particular scenario—‘Job migration due to processor overloading’—has been shown in Fig. 8. The scenario shows the steps that would be followed for self-healing incase of processor overloading.

Information regarding all the resources in VO is stored in resource repository and alert is generated for resources having problem. Consequently, the debugging effort is saved. This is particularly advantageous in large-scale Grids where identifying the reasons of faults is quite tedious. Thus, monitoring information has been put to effective use. Further, automatic self-healing has been achieved through Job migration [38].

6. Test results

- Corresponding to Pulse data 000 no action is desired, as it indicates normal operation.
- For pulse data 001, 010, 011 and 101 (as shown in Table 1) the self-monitor informs the scheduler that no jobs should be directed to these systems and stores alert flag in repository. At the time of resource discovery when scheduler looks for available resources these are not shown in the list of available resources.
- For pulse data 100 and 111 Job migration is required. Job migration means re-allocation of jobs from one system to another system or in other words is a process of moving a normal, check pointed job from one host to another. In situations, where load balancing is required, job migration is the only efficient way to guarantee that the submitted jobs are completed as per user requirements. In overloading situations check pointed jobs or jobs that require to be started again are migrated from overloaded system to under loaded system with the help of GridFTP [37]. GridFTP is service that transfers the large amounts of data faster, more securely, and more reliably than FTP alone.
In the current implementation job migration through GridFTP is being done manually which is being automated in near future. Thus self-management is accomplished through self-healing incase of TU Grid.

6.1. Comparisons

Figure 9 shows time spent in hours in isolating and managing the fault incase of physical management of the TU Grid and after the usage of Pratham monitoring system. The sample data was collected for five different categories of faults and time spent in fault isolation, manually and through Pratham. It can be inferred from the graph that the administrator can save much time in isolating and locating the root cause of the fault with the help of Pratham. The only time spent would be in sending the pulse data to the inference engine and storing the inferences in the resource repository which would be much less as compared to doing this activity manually.
**Resource Monitoring**

**Scenario** Job migration due to processor overloading

![Diagram of Resource Monitoring System]

**Fig. 8.** Steps followed for processor overloading scenario.

---

**Table 2** compares Pratham Monitoring System with the state-of-the-art, MDS4 and Beacon Monitor. The Monitoring and Discovery System (MDS4) [11] of the Globus Toolkit 4 is a suite of components for monitoring and discovering Grid resources and services. MDS4 is compliant with WSRF and WS-Notification specifications and implements a standard Web Services interface to a variety of local monitoring tools and other information sources. It was specifically designed to address the needs of a Grid monitoring system – one that publishes data that is available to multiple people at multiple sites. Currently, it is not an event handling system, like NetLogger [36] or a cluster monitor on its own, but can act as an interface to more detailed monitoring systems and archives and can thus be used for publishing summary data using standard interfaces.

The Beacon Monitor was first introduced in the Deep Space 1 mission of NASA [22]. The spacecraft sends a beacon signal to the ground indicating how urgent it is to track the spacecraft for telemetry. Beacon signal is a tone, used to indicate the degree of urgency involved in this monitoring. Five different
I. Chana and S. Bawa / Extending self-healing in grid environment by pulse monitoring

Table 2
Comparison of Pratham with existing monitoring systems

<table>
<thead>
<tr>
<th>Parameters/monitoring system</th>
<th>Globus MDS4</th>
<th>NASA beacon monitor</th>
<th>Pratham monitoring system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique used</td>
<td>Heartbeat Monitoring</td>
<td>Beacon Monitoring</td>
<td>Pulse Monitoring</td>
</tr>
<tr>
<td>Implementation</td>
<td>Through a signal</td>
<td>Through a tone</td>
<td>Through a packet sent by agents</td>
</tr>
<tr>
<td>Correctness</td>
<td>Signal indicates only the presence or absence of fault</td>
<td>Level of tone indicates urgency</td>
<td>Pulse data indicates fault as well as cause of fault</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Normal</td>
<td>More efficient than heartbeat monitor</td>
<td>More efficient then beacon monitor</td>
</tr>
<tr>
<td>Robustness</td>
<td>High</td>
<td>Long term benefits can be achieved through supplements only</td>
<td>High</td>
</tr>
<tr>
<td>Scalability</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 3
Comparison of Pratham FDHS with existing systems

<table>
<thead>
<tr>
<th>Parameters/systems</th>
<th>CoreGrid</th>
<th>Unity</th>
<th>GridICE</th>
<th>Pratham FDHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique used</td>
<td>Aspects Oriented Technology</td>
<td>Interacting agents called Autonomous Elements (AE)</td>
<td>Sensors based on Push/Pull protocol</td>
<td>Monitoring agent based</td>
</tr>
<tr>
<td>Implementation</td>
<td>Injects monitoring code dynamically for collecting resource metrics</td>
<td>AE compute resource level utility functions based on applications' service-level utility functions</td>
<td>Monitoring agent launches sensor on each node for information retrieval</td>
<td>Monitoring information is analyzed on the basis of Fault Tolerance rules</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Able to gather monitoring information at the desired level</td>
<td>Able to compute globally optimal allocation servers across applications</td>
<td>Helps in debugging only but does not reveal exact cause of failure</td>
<td>Monitoring information has been put to effective use in detecting the exact cause of failure</td>
</tr>
<tr>
<td>Self-healing</td>
<td>Does not support</td>
<td>Does not fully support</td>
<td>Does not support</td>
<td>Supports self-healing</td>
</tr>
</tbody>
</table>

states of tones are used for this purpose. At times, the spacecraft uses both the beacon signal and, when necessary, direct communications to downlink telemetry information for analysis at mission control.

Table 2 and the related work mentioned in Section 2.3 clearly show that the prevalent systems utilize different techniques of monitoring but no system is directly based on pulse monitoring which has been the basis of Pratham monitoring system. In a large and complex environment like Grid, Pratham Monitoring System successfully extends self-healing to a large extent as it provides correct and more efficient information as compared to MDS4 and Beacon.

Table 3 compares Pratham FDHS with the existing systems namely CoreGrid, Unity and GridICE.

7. Conclusion

This paper presents Pratham Monitoring System, a Grid monitoring system based on pulse monitoring. It demonstrates how the monitoring information generated by a monitoring tool can further be utilized by fault discovery and healing system for inferences of the occurrence as well as reasons of faults. In a large
cluster or fabric one faulty node can cause serious problems for the entire Grid, as Grids are more prone to failures than traditional computing platforms. The existing monitoring systems notify the presence of fault but not the exact cause of the fault and in various situations inferences need to be drawn further manually for finding out the root cause of the fault. The case study presented in this paper presents a prototype implementation of the Pratham Monitoring System and demonstrates that self-healing can successfully address this problem by isolating the fault along with its cause based on the monitoring information. Current implementation of Pratham monitoring system generates alerts after construing the pulse inputs and inferences generated by the inference engine, thus detailed information is generated for the administrator. But further the administrator needs to physically solve the problem, for example if a CPU is overloaded, job migration might be required. The alert for this problem would be generated for the administrator who would then physically carry out the job migration task. The future work would focus on the entire automation through GridFTP, so that as soon as the alert is generated the Actuator itself takes the action automatically as per the requirement.

References

I. Chana and S. Bawa / Extending self-healing in grid environment by pulse monitoring


[33] Grid service broker developed as part of Gridbus project, www.gridbus.org/broker/.


[38] N. Vasudevan, Prasanna Venkatesh Design and Implementation of a Process Migration System for the Linux Environment, 3rd International Conference on Neural, Parallel and Scientific Computations August 9–12, 2006 Morehouse College, IFNA Atlanta, GA, USA.
Grid jobs scheduling: The Alienated Ant Algorithm solution

M. Bandieramonte, A. Di Stefano and G. Morana*

Department of Computers Science and Telecommunication Engineering, Catania University, Italy

Abstract. Grid is a manifold and highly dynamic scenario. One of the fundamental issues in this environment is related to jobs scheduling; since jobs allocation to resources is a combinatorial optimization problem, NP-hard in most cases, several heuristics have been proposed to achieve near-optimal solutions.

This paper proposes a job scheduling policy based on the Alienated Ant Algorithm (AAA), a new metaheuristic strategy freely inspired by the ants’ self-organization ability. As it will be shown, AAA is able to find near-optimal solutions adapting its decisions to changing resources states and submitted workload. The experimental results show that the use of the proposed algorithm satisfies expectations.

Keywords: Scheduling algorithms, grid computing; simulation; ant colony optimization; alienated ant algorithm

1. Introduction

Grid computing is today one of the most popular and applied solutions for e-science applications and it is also becoming an important support for e-business. In recent years, considerable effort has been made to create Grid middlewares [18,20,39] capable of guaranteeing the safety and security of resources management, maintaining the shared resources under the strict control of the organizations. Moreover, also realizing Grid services adequately safeguarding the correct utilization of resources (like CPU load, memory usage, and available network bandwidth), critical issues of Grids are related to an effective exploitation of them. One of the most important and challenging problems for Grids continues to be the design of a valid resource scheduling policy to support the execution of jobs submitted to the system; jobs allocation made by a centralized broker, at a collective level, plays a crucial role and influences the utilization of the whole system. Scheduling an application in a Grid is significantly complex because of the heterogeneous nature of administrative domains taking part in Virtual Organizations (VO) composing the Grid [6,26] and the potential fluctuations in availability of resources; thus scheduling problem in general is an NP-hard problem and it is far from trivial in Grids. The approach followed in this paper solves the distributed job allocation in a Grid; it is based on an heuristic algorithm freely inspired by the Ant Colony Optimization (ACO) strategy. The latter emulates the ants natural intelligent behavior in looking for food by following pheromone traces of other ants. Indeed, the solution proposed here changes the search strategy followed by ants: our ants are alienated in that they stay away from others and prefer to follow the paths that were covered by no one else or, at most, by few other ants. Alienated Ant Algorithm

*Corresponding author: G. Morana, V.le A. Doria 6, 95125 Catania, Italy. Tel.: +39 095 738 2365; Fax: +39 095 738 2397; E-mail: gmorana@diit.unict.it.
Fig. 1. A typical Grid organization.

(AAA) can be very useful to solve those problems where changes on the state of condition bounds are required to reach quickly suitable alternative solutions. The algorithm proposed has been rigorously evaluated through extensive simulations on a real Grid infrastructure, currently shared between a number of Sicilians Research Organizations participating in the PI2S2 project [10]. The experimental results show that it satisfies expectations. The paper is organized as follows. The next subsection presents the scheduling challenges in Grid computing. Section 2 describes some related works. Section 3 introduces the Ant Colony Optimization theory. Then the Alienated Ant Algorithm is presented. Section 4 presents the simulation results. Finally, Section 5 briefly concludes the work.

1.1. The scheduling challenges in grid computing

Scheduling is a classical issue in the distributed systems design that has been investigated in depth [16, 24, 41, 49, 57] as the key element for an effective resources utilization. Scheduling involves a set of tasks, a set of networked machines, an adequate specification of the environment and other constraints, such as conditions on resources usage and jobs requirements. The goal of a scheduling algorithm is to find a mapping between jobs and resources, optimizing some criteria as load balancing or system throughput. This paper proposes an evolutionary algorithm[7], the Alienated Ant Algorithm (AAA), to address the jobs scheduling in Grid, analyzing how this algorithm is able to overcome the difficulties to gain a global view of the present state of resources utilization, adapting to its dynamic evolution.

Grid environments are composed by a wide quantity of different types of processing nodes, storage units, scientific instrumentation and information, belonging to different research organizations, exploited by users of a Virtual Organization. Each time a VO user submits a job in the Grid, it has to refer to the scheduler (that in a Grid is called Resource Broker – RB – ). The aim of a RB is to receive all of the job submission requests, to analyze job requirements and features, to map jobs to resources required for their execution and, finally, to dispatch them to the most suitable Computing Element (CE). Each CE
handles the Job Management System (JMS) of underlying cluster of Worker Nodes (WN). The aim of WNs is to execute jobs and to return their results to the VO user. Figure 1 shows a typical example of a Grid deployment. In this type of hierarchical organization three scheduling layers can be identified. The first one is related to the algorithm used by RB to distribute jobs among CEs; this algorithm strongly influences the performance of the whole system. The second scheduling layer manages job allocation from every CE to its underlying WNs. Finally the lowest scheduling activity is related to the mechanism used by the operating system (OS) of each WN to schedule jobs on its CPU. Here only the first two levels of scheduling will be taken into account: it has been assumed that the OS scheduling consists of non preemptive FIFO schedulers that link only one job per CPU at time, as used in High Performance Computing (HPC) systems, like the one considered in the simulations illustrated hereafter.

In a complex scenario as the one described, knowledge of resources utilization state and their availability is a fundamental support for the creation of a good scheduling algorithm.

Unfortunately, the Grid hierarchical structure makes it difficult to have complete and consistent information about the state of the underlying resources or about their actual workload.

This lack of knowledge, especially at higher levels, does not allow to assess the real cost of executing a task, to model resources availability, to characterize the same resources and to assess current system load. Due to this consideration, all algorithms that exploit global information about resources state for their correct functioning are not adequate for use in a Grid environment.

In order to overcome this lack of global knowledge, the proposed algorithm is distributed on each grid scheduling layers, both in RB and CEs, and takes its decision basing on local information, that are easily accessed and are always available. Coordinating each single part of the Alienated Ant Algorithm, it is possible to obtain a global scheduling policy that is aware of the state of resources on the whole grid.

Moreover, the ability to be continuously updated in regard to resource availability, which changed, renders the proposed algorithm able to adapt its decision at run time.

2. Related work

Several studies investigate the ACO theory, both producing theoretical formulation on this issue and proposing specific solutions to face many and various types of scientific problems. The “ants” perform very well when used to solve combinatorial optimization problems like the Quadratic Assignment Problem (QAP) [50], Travelling salesman problem (TSP [35]) or sequential ordering problem (SOP [33]) but they have also demonstrated the capacity to face complex real world applicative scenarios like internet and telecommunication routing [32,44], traffic management techniques [37] and, in general, a wide series of assignment problems. The ACO approach has been applied with success also to scheduling problem, in many different applicative scenarios; for example, to the single machine weighted tardiness (SMWT) problem [14], to the flow shop scheduling problem [52] and to the group scheduling problem [9]. ACO based solutions have also been proposed for the grid job scheduling problem. An original application of the ACO algorithms in Grid computing is introduced by Fidanova et Durchova in [48]. In this paper authors handle job-machine issue using a graph-based solution. The originality of this solution consists in the use of multiple nodes as a single machine instead of using the one-to-one standard representation (also used in our solution). This allows an increase in the dimension of search space, providing opportunities to explore a greater number of solutions. This original approach is studied in depth and is improved in some noteworthy recent works [29–31,43]; in particular the last one proposes an ant algorithm that is combined with local search, producing good quality scheduling, refining mechanisms of pheromone update, taking into consideration job execution time and resources free time. The goal of a single ant is
to schedule the pool of jobs pushing it through graph model nodes trying to minimize the makespan. A number of ants smaller or equal to the number of jobs is used. At the end of the iterations the mechanism of local search allows to select, among ants, the winner, that is the ant with the lowest total makespan. Another interesting work related to the use of Ant Colony Optimization strategy for adaptive resource allocation in dynamic Grid environment is presented in [2]. It proposes a distributed ant-based system for the efficient placement of running services on the most suitable servers.

In [43] the authors propose a Balanced Ant Colony Optimization (BACO) algorithm for job scheduling in Grid; the objective of this algorithm is to balance the system load while trying to minimize the makespan of a given set of jobs. The ant system is mapped to the grid in the same manner of proposed algorithm, but pheromone values are calculated for every couple ant-route (job-resource) at each iteration. Pheromone density, from time to time, is changed by applying update functions at two levels: local and global. At the local level, updates involves only the resource selected to execute the job, at the global level, instead, it implicates the update of all entries of the PI (Pheromone Indicator) matrix, one for every couple resource-unscheduled job. Therefore, even though it offers good results, this algorithm is computationally expensive. In [25] an algorithm is proposed that maps the ACO solution on Grid in a similar way to the one we propose in this paper; this algorithm takes into account both the traffic over the resources and the characteristics of computational resources; furthermore, it uses an innovative pheromone releasing mechanism: it is laid down only if job executes successfully. Thus, in addition to the information on resources and jobs, this algorithm also requires feedback on the correct execution of job; it can became computationally expensive when the quantity of jobs becomes high (a frequent occurrence in a Grid environment).

The solution proposed in this paper, although it is based on the same resource informations, avoids using any report mechanism for updating: this makes decisions quicker and improves the ability to adapt itself to job workload changes.

A common feature of all the above the ACO based solutions for Grid is that they are based on a static representation of Grid resources: graph used for scheduling decisions, in fact, is set up just once, considering a fixed number of machines. This kind of approach shows its limit in real scenarios, since a fundamental characteristic of grid is its resources dynamism.

Moreover, all approaches seen before, and in general, the ACO algorithms existing in literature work in batch mode. Jobs, therefore, are not scheduled when submitted to the system but when the pool became full. On the one hand this allows to apply scheduling policies that use informations not restricted to the single job, but inclusive of the entire set, on the other hand introduces considerable scheduling delays. The novelty of the Alienated Ant Algorithm is that it succeeds in scheduling in on-line mode adapting its behavior to system changes, assuring to each job a short queue waiting time and providing a good load balancing over all available resources.

### 3. The ant colony optimization and the alienated ant algorithm

In 1992 [4,11,32,34] Dorigo et al. have proposed a new stochastic-based metaheuristic [3] strategy called Ant Colony Optimization (ACO), widely applied to solve complex combinatorial and multi-constraint optimization problems. It is inspired by the ability of real ants to cooperate in order to solve the issues related to colony's survival, especially those related to food search and its storage.

Ants, although blind, are able to coordinate their movements and decisions, exchanging knowledge on the path from the food source they find and their anthill. This is possible because each ant in looking for food lays down a quantity of a chemical substance (called pheromone), in a measure proportional
to quantity, quality and proximity of food, marking the path it takes. An isolated ant moves essentially randomly but, when it smells a previously laid pheromone trail it can decide to follow that new path reinforcing the trail with an additional quantity of pheromone.

The probability that an ant will follow a path is proportional to pheromone released on the path by other ants. This condition causes an auto-catalytic mechanism: the higher the number of ants that follows a trail, the more attractive that trail becomes for others; after a certain number of iterations, all ants will follow the path with highest concentration of pheromone, identifying it as the shortest path from colony to food.

ACO algorithms class represents theoretical formalization of the above-mentioned ant’s behavior. In general, each ACO algorithm bases its functioning on activities of some intelligent entities (that play the role of ants) that, in parallel and iteratively, explore the search space in looking for solutions to the considered problem. Movements of these entities are driven by a probabilistic evaluation of values stored in some vectors that represent the tracks of pheromone laid down along the path segment. Updating and checking vectors each ant influences, and is influenced by movements of other ones.

In general, for each algorithm iteration, four different phases can be picked out:

- **Solution construction phase**: ants build a set of feasible solutions for the given problem.
- **Solution selection phase**: each ant probabilistically extracts a solution from this set, basing on a specific policy related to application context.
- **Pheromone trail update phase**: each ant updates the pheromone trails vector, releasing on the related selected path (vector entry) a certain quantity of pheromone to mark its choice.
- **Pheromone trails evaporation phase**: the value of each entry of the pheromone trails vector is decreased by a certain quantity in order to simulate real trail evaporation.

Running these phases iteratively, a near-optimal solution for a given problem can be found. Two other phases have to be added to the previous iterative ones to complete the algorithm:

- **Initialization phase**: before starting the algorithm, all the entries of pheromone trails vector have to be initialized and the terminating condition set up. The terminating condition depends on the considered problem.
- **Solution evaluation phase**: the terminating condition has to be checked before the beginning of each cycle: if objective has been reached, i.e a good solution has been found, the algorithm can be considered successfully completed. Otherwise the iteration has to continue.

Every ant works in a constructive way developing, progressively, a part of the solution. Partial solutions are treated as states: each ant moves from a state i to another one j, producing a more complete partial solution. This process is iterated until the ant ends in a final state corresponding to a complete problem solution. In the next subsection we describe the AAA, explaining differences and similarities with the traditional ACO class algorithms.

### 3.1. The alienated ant algorithm

A key aspect of the ACO strategy is the concept of cooperation based on **indirect** communication through pheromone trails. Each ant is able to coordinate its decisions and movements in order to help other ants to accomplish some tasks related to colony survival.
As said above, thanks to the pheromone, ants are able to find shortest path from the nest to food; this peculiarity was exploited from the scientific community to solve many types of problems, as it was shown in the literature review presented in section 2. The essential characteristic of an ACO algorithm is to make it flow together, after a certain number of iterations, all ants in the same direction; at a certain point, indeed, the algorithm is able to find a sub-optimal solution, and from that moment all ants will follow that route. For this reason the ACO algorithms perform very well when solving some problematics like routing in telecommunication networks; in Grid scheduling case, however, the targets to reach are multiple and different from each other: it has to select that resource more capable to execute a job, but it has to balance the load too, avoiding to schedule all jobs on the same resource. Thus, instead of an algorithm where all ants flow together in the same direction (all jobs to the same resource) one it’s required that is able to divide ants across various sub-optimal routes. Based on these considerations, the authors have asked: ants use pheromone as an attractive substance: what would happen if pheromone were to be used as a repulsive substance? Is it possible to take advantage of the behavior of an ant that instead of using pheromone to coordinate its actions with others, uses pheromone trail to stay away from them?

In this paper the authors try to answer these questions proposing a solution, alternative to the above mentioned ACO algorithms, that takes into account the behavior of an ideal alienated ant. As mentioned in Section 1.1, the proposed solution will be used to build a scheduling algorithm for Grid environments.

An ideal alienated ant is an ant that stays away from the others and prefers to follow the paths that were covered by no one else or, at most, by few other ants. In order to achieve this, when it has to choose between two or more paths, it smells pheromone trails and takes the path with the lighter trail instead of following these where the pheromone is stronger (as other normal ants). The alienated ant, in fact, interprets a tenuous pheromone trail on a path either as if only few ants have taken that path or a long time has elapsed since the last use of the path.

In the design of our algorithm we consider a population of alienated ants that decide to cross a path instead of another one basing on the criteria explained before. The parallelism between alienated ant system and grid at this point is immediate: considering a job as an alienated ant, the Resource Broker as anthill and the Worker Nodes as different food sources, it’s possible to transform the behavior of the alienated ant in a grid job scheduling algorithm.

This scheduling algorithm, referring to phases describing the functioning of generic ACO algorithm depicted in previous section, can be summarized as follows:

### 3.1.1. Solution construction

As described above, each alienated ant has to build a solution of the given problem. In the Grid scheduling problem, a solution consists in the creation of the path linking the Resource Broker, to the Worker Node passing through a Computing Element. Each scheduling path is composed by two sub-paths: each one is related to one of scheduling layers mentioned in 1.1, from RB to CEs and from CE to WNs respectively.

Considering \( r_{bi}, c_{ej} \) the path linking RB to \( ith \) CE and \( c_{ej}, w_{nk} \) the one linking the \( jth \) CE with the underlying \( kth \) WN, the generic solution is represented by the couple \( (x, y) \):

\[
(x, y)|x \in \{r_{bi}\}, y \in \{c_{ej}, w_{nk}\}, i = j, i \in [1, n], j \in [1, n], k \in [1, m_j]\] (1)

### 3.1.2. Solution selection

This mechanism is based, as is the standard ACO algorithm, on pheromone trails evaluation. The AAA maintains a trail for each sub-path in some vectors (one for the RB and one for each CE): the value
of each trail is used to assign a certain probability to be chosen to the related path. For the $x_i$ sub-path ($x \in \{rb_{ce_i}\}$) starting from RB pheromone trails vector, considering $\phi_{x_i}$ its pheromone trail value, the probability $\Psi_{x_i}$ is calculated as:

$$\Psi_{x_i} = 1 - \frac{\phi_{x_i}}{\sum_{i=0}^{n} \phi_{x_i} | x \in \{rb_{ce_i}\} }$$

(2)

For the $y_{jk}$ sub-path ($y \in \{ce_{j,wn_{jk}}\}$) linked to CE$_j$ pheromone trails vector, considering $\phi_{y_{jk}}$ its pheromone trail value, the probability $\Psi_{y_{jk}}$, is calculated as:

$$\Psi_{y_{jk}} = \Psi_{x_i} \ast \left(1 - \frac{\phi_{y_{jk}}}{\sum_{j=0}^{m} \phi_{y_{jk}}} \right)$$

(3)

It should be note that $\Psi_{y_{jk}}$ is a conditional probability in that selection of the $y_{jk}$th path is conditional on the selection of $x_i$th path, with $i = j$.

The usage of these probability functions differentiates the behavior of an alienated ant from another “normal” ant. As opposed to the standard ACO approach, the alienated ant has high probability of choosing the path with lightest trail instead the strongest one.

3.1.3. Pheromone trails updating

After it has chosen its sub-path, the alienated ant marks it and updates the value of the related pheromone trail vector entry. On the sub-path our ant will release a quantity of pheromone related to some characteristics of the relevant job. Depending on the target proposed in this paper, an estimation of job execution time, expressed in minutes, will be taken into account. This estimation is an important problem: it can be obtained either from scheduler job profiling [17,5,47,56] or from user indication. If $jet_i$ is the job execution time estimation for the $i$th job, then the update value for trail representing $j$th sub-path ($\Delta \phi_{sp_j}$) is:

$$\Delta \phi_{sp_j} = jet_i$$

(4)

3.1.4. Pheromone evaporation

In order to simulate pheromone evaporation, at each scheduling cycle there is an update of trail vector entries value. Each pheromone trail vector entry is updated according to:

- computational power of relevant resource
- elapsed time

the first parameter assures that pheromone evaporation speed is proportional to computational power of the resource, increasing the probability that the more efficient resource can be frequently selected. The second parameter is used in the evaporation mechanism to have a time reference (from a fixed “start time”) in order to be able to perform asynchronous pheromone updating. Without this time reference, in order to maintain a consistence among pheromone trails values, it’s necessary to perform a synchronous update, that can be very expensive in terms of computational power. In this paper, the formula taken into account for the evaporation mechanism is:

$$\Delta \phi_{sp_j} = -(t \ast cmp_{pwr_i})$$

(5)

where $\Delta \phi_{sp_j}$ represents the updating value of the $j$th pheromone trail, $t$ is the time (in minutes) elapsed since the last update operation and $cmp_{pwr_i}$ is a value related to the computational power of $i$th resource.
It noted that $\Delta \phi_{sp}$ value can not be less than zero. If $\Delta \phi_{sp}$, after updating process has a negative value, it is rounded to zero.

The pseudo-code below summarizes the Alienated Ant Algorithm:

\textit{alienated Ant - C style pseudo code}

\begin{verbatim}
trailInit(pherVector[]);

void Alienated Ant(pherVector[], taskTimeEval)
{
    // evaporation step
    for (i=0; i<pherVectorLength; i++)
    {pherVector[i] = pherVector[i] - evap(elapsedTime, resNumber);
    }
    // solution creation->resource selection step
    res = evalPher(pherVector);
    // pheromone trail update step
    pherVector[res] = pherUpdate(TaskTimeEval);
    return;
}
\end{verbatim}

4. Simulations

This section will discuss all the issues related to the simulation campaign. It will analyze the metrics and the targets taken into account during the simulations and the model of the Grid considered. Moreover, comparisons between the proposed algorithm and other solutions are briefly introduced and, finally, the results of simulation will be illustrated and discussed. The performance evaluations and the comparisons with other solutions have been made using the SimGrid [46] simulator, a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. SimGrid is used to develop and evaluate scheduling algorithms on distributed resources infrastructures defined by the user.

4.1. Performance metrics

Several common performance metrics are frequently used in theoretical analysis. Since individual user’s and system administrators often have different and possibly conflicting demands, no single measure can comprehensively capture the overall Grid performance. From a users perspective, key measures that must be carried out include the Average Response Time and the Average Queue Waiting Time. A system administrator, instead, is more interested in maximizing the utilization of the available computational resources; metrics used in this case are the Job Throughput and the Load Balancing. Here, in order to consider the both different points of view, it has been decided to assess the Average Queue Waiting Time and the Load Balancing.

- **Average Queue Waiting Time.** This parameter (henceforth named $\mu_{qwt}$) measures the average time spent by each job in idle after being submitted and before being executed. For each algorithm, the $\mu_{qwt}$ is calculated as the average value of the $\mu_{qwt}$ measured on each CE:

\[ \mu_{qwt}^{tot} = \frac{\sum_{i=1}^{N_{CE}} \mu_{qwt_i}}{N_{CE}} \]  

(6)
The $\mu qwt$ related to the $i_{th}$ CE is defined as:

$$\mu qwt_i = \sum_{j=1}^{n_{jobs_i}} \frac{(T_{tot_j} - T_{ex_j})}{n_{jobs_i}}$$ \hspace{1cm} (7)

In Eq. (7), $T_{tot_j}$ is the time elapsed between the submission of a job and its termination, $T_{ex_j}$ is the execution time of the $j_{th}$ job, and $n_{jobs_i}$ is the number of jobs scheduled on the $i_{th}$ CE.

The $\mu qwt$ value is inversely proportional to the Grid response time: a lower value means a faster response time of the system.

– **Load Balancing Capability.** For each scheduling algorithm it will show the trends of job distribution among the CEs in terms of standard deviation ($\sigma$, named $\sigma lb$):

$$\sigma lb = \sqrt{\frac{\sum_{i=1}^{n_{CE}} (x_{CE_i} - (n_{WN_i} \times x_m))^2}{n_{CE}}}$$ \hspace{1cm} (8)

In Eq. (8), $x_{CE_i}$ represents the workload submitted to $i_{th}$ CE, $x_m$ is the average workload that should be submitted to each WN, and $n_{WN_i}$ is the number of WN of the $i_{th}$ CE. The performance of the algorithm will be better when the value of $\sigma$ is small (i.e. $\sigma lb = 0$ means an equal job distribution on each CE, $\sigma lb >> 0$ means lower load balancing capabilities).

The above two parameters will be evaluated under different conditions. In particular, the behavior of the compared algorithms will be evaluated:

– when the system is stressed under different workload conditions varying the number of submitted jobs and the composition of jobs type (i.e. proportion among short, medium and long jobs)
– when the number of available grid resources change in an unpredictable way.
– when there are wrong estimations of the jobs execution times.

The first two points are very important in that they characterize the typical grid scenarios. They, in fact, are indispensable to evaluate the ability of the algorithms to adapt themselves to the dynamism of grids in terms of number and type of jobs and, as underlined in the section 1, in terms of resources availability.

Each evaluation makes reference to: the real Grid infrastructure, deployed within the PI2S2 [42] project, involving different cluster of worker nodes belonging to different research organization of the Sicily. For each simulation, measures refer to steady state condition.

4.2. Simulated scenario

The simulated scenario (schematically represented in Fig. 2) reproduces the real Grid infrastructure of the Cometa’s PI2S2 project. Cometa [10] is a multi-entity consortium for the promotion and adoption of advanced computing technologies. This Grid infrastructure involves 3 sites (in the cities of Catania, Palermo and Messina, including Universities and Research Centres) where a variable number of Worker Nodes is managed by 7 relevant Computing Elements, related to the various research organizations taking part in the consortium: in the figure each CE is named making reference to research organization that manages it (for example COMETA-UNICT-DIIT is the Computing Element handled by the Department of Information and Telecommunication Engineering – DIIT – of Catania University). The number of Worker Nodes managed changes from one to another: the figure highlights this number for each CE (for example COMETA-UNIPA manages actually 84 Worker Nodes). A user who wants to submit a job, reaches to User Interface; this last provides an interface between users and job scheduling system; in the
case study taken into consideration there are three User Interfaces (UI_1, UI_2, UI_3) displaced in three different campus (Catania, Messina and Palermo). As it can be evinced in Fig. 2 the architecture consists of 69 computation units distributed on 7 CEs under control of a single Resource Broker. Computational units are AMD 64bit quad-core processors (in all 276 CPUs), units are linked by 4 Gb infiniband. The Grid infrastructure and the jobs flow model have been modeled considering information related to (i) types and distribution of resources in the real PI2S2 Grid infrastructure and to (ii) dimension of waiting queues of different types of jobs typically submitted by the users, actually observed.

In the PI2S2 architecture workload is divided in three classes of jobs, distinguished according to their execution time and named (i) “short” if their duration varies in range between 1 and 10 minutes, (ii) “medium” if it varies between 10 and 100 minutes and (iii) “long” if their duration is more than 100 minutes. A typical jobs flow model in PI2S2 is so composed: short(10%):medium(75%):long(15%). In the simulations this proportion is maintained because this is the typical workload composition of a generic Grid scenario, but also other (and more general) percentages are considered to test the behavior of our algorithm.

4.3. Algorithm comparisons

The algorithms chosen for a comparison with the Alienated Ant Algorithm (AAA) are briefly recalled below. It is very important to note that all considered solutions are on-line scheduling algorithms, i.e algorithms that can be applied with a previous knowledge of the jobs workload composition. This is a fundamental characteristic of the proposed algorithm that makes itself different from the other heuristic based solutions, as well as the ACO ones, that needs to know a priori the jobs in order to make the best decision for jobs distribution over the available grid resources. This difference is also the main reason which forced as force the authors, in this paper, to avoid the comparison between the Alienated Ant Algorithm and other type of meta-heuristic solutions.

Fig. 2. PI2S2 Grid infrastructure.
4.3.1. Random-based scheduling

This algorithm (henceforth RbS) is used as a benchmark: it is suitable, in fact, only to show that the adoption of a scheduling policy is a necessary condition to obtain good performance.

4.3.2. Multi-queue RoundRobin scheduling

This algorithm (henceforth MRR) has been chosen because its good load balancing capability (in terms of “number of submitted jobs”) makes it an important reference point to evaluate the performance of the AAA in terms of $\sigma_{lb}$. It aims to be fair in distributing jobs between CEs and WNs. Used at both the RB and CE level, this algorithm ensures an equal distribution in terms of number of jobs submitted on underlying resources. This algorithm is very simple: the scheduler (RB or CE) keeps (i) a list of all the suitable resources (CE or WN) and (ii) the position of the last resource scheduled (initially the first resource in the list); every time a new job has to be scheduled, the scheduler (RB or CE) takes from the list the resource positioned immediately after the last one used. This algorithm, that does not require the knowledge of specific information on the structure of Grid, is often used in grids that do not

4.3.3. Scheduling based on resources availability

This algorithm (henceforth RA), used at RB level, has been chosen because represent an enhanced version of the MRR algorithm. It ensures that the jobs are submitted to the various CEs based on the number of the available underlying resources (WNs). This means that, unlike the previous two algorithms, the RB is aware of the overall structure of the network, i.e. the RB has to know how many WNs there are on each CE, and gives more jobs, in a proportionate manner, to the CE with more available resources.

While the MRR algorithm assigns $\frac{m}{n_{CEs}}$ jobs to each CE ($m$ is the number of jobs and $n_{CEs}$ is number of CEs), this one assigns to the $i_{th}$ CE a number of jobs equal to:

$$n_{WN_i} \times \frac{m}{n_{WN_{Total}}}$$

where $m$ is the number of jobs, $n_{WN_i}$ are the WNs of ith CE and $n_{WN_{Total}}$ is the number of all WNs in the considered system. This ensures a fair distribution of jobs at the CE level and tends to reduce the queue waiting times if compared to the MRR algorithm. In practice the RB assigns to each CE a probability of being chosen by the scheduling mechanism that is proportional to the number of underlying WNs. At CE scheduling level, the jobs are distributed on underlying WNs in a round robin way.

4.3.4. WN\textsubscript{dedicated} scheduling

This algorithm is built based on the real algorithm currently used in the PI2S2 grid infrastructure where each CE presents a queue for each specific job type. It (WND) assigns a number of different resources to different types of jobs: in particular, it maps specific types of jobs to specific WNs, used exclusively to execute these types of jobs. In practice, each CE reserves a certain number of WN for specific types of jobs based on load type expectation; in the simulated configuration, every CE reserves 50% of its WN to long jobs, 40% to medium jobs and the remaining 10% to short jobs. The proportion job-WNs is chosen based on the observation of the deployment of COMETA resource from November '06 to November '07. At the RB scheduling level, the jobs are distributed onto CEs using the round robin algorithm.

4.4. Experimental results

This section briefly discusses the results of the measures obtained from the simulations to evaluate the performance of the Alienated Ant Algorithm (AAA).
4.4.1. Behavior when changing workload in terms of quantity of submitted jobs

The Fig. 3 and the Fig. 4 show, respectively, the average queue waiting time and the standard deviation of load balancing when the number of jobs submitted to the Grid grows. The measures have been made respectively from 2000 to 10000 jobs (2000 jobs per step), maintaining constant the job rate (jobs/s) and the proportion between job types (short(10%):medium(75%):long(15%)).

In this simulation, all the grid resources are available for the jobs submission.

As can be observed, all the considered algorithms, in terms of $\mu_{qwt}$, present a very similar behavior. The value of $\mu_{qwt}$ for each algorithm, in fact, remains almost constant with the changing number of submitted jobs. This means that the performance of each algorithm does not depend on the number of submitted jobs.

The proposed algorithm, Alienated Ant, presents the best trend for all considered scenarios.

In terms of $\sigma_{lb}$, instead, the algorithm’s trend worsens with increasing workload size. These behaviors can be easily explained considering the formula (8) chosen for the evaluation. In fact, a greater number of submitted jobs implies a greater difference in term of jobs submitted on each CE and, as a consequence, a greater increment of the $\sigma_{lb}$ value. In this case, the best behavior is presented by the RA algorithm: this result can be easily foreseen because of the good load balancing capabilities is the strong point of this algorithm. The Alienated Ant Algorithm shows a very good load balancing capability if it is compared with the others. It should be noted that both results are not in contrast: the performance in terms of $\sigma_{lb}$, in fact, depends on the number of submitted jobs while the value of $\mu_{qwt}$ depends on the order of submission. The Random algorithm gives the worst result in both cases, demonstrating the importance of a good scheduling algorithm.
4.4.2. Behavior when changing proportion between job types

The Fig. 5 and the Fig. 6 show, respectively, the average queue waiting time and the standard deviation of load balancing when the proportions between short, medium and long jobs change. The measures have been made with 6000 jobs shared out respectively (short:medium:long) in (1)4200: 1200: 600, (2)3600: 1200: 1200, (3)2000: 2000: 2000, (4)1200: 1200: 3600, (5)600: 1200: 4200.

Also, in this simulation, all the grid resources are available for the jobs submission.

The results confirm what has been observed previously: the AAA presents the best behavior in terms of $\mu_{qwt}$ and the RA in terms of $\sigma_{lb}$. In this case, however, the trend of $\mu_{qwt}$ value for the AAA are different from the other algorithms. In fact, while the others are strongly influenced by the number of medium jobs (i.e the $\mu_{qwt}$ of each algorithm increases when increases the number of medium jobs), the AAA maintains a linear trend that increases proportionally with the number of long jobs.

4.4.3. Behavior when changing the number of available grid resources

The Fig. 7 and the Fig. 8 show, respectively, the average queue waiting time and the standard deviation of load balancing when the number of available grid resources decreases in an unpredictable way. This simulation has been done in order to evaluate the ability of each algorithm to adapt itself to the resources dynamism typical of grids scenarios. In particular, these measures have been made respectively when the number of resources that go down is respectively 0, 5, 10, 20 and 25% of the total available resources. In this simulation the job rate (jobs/s) and the proportion between job types (short(10%):medium(75%):long(15%)) are maintained constant.

As foreseeable, the performance of all the considered algorithms, both in terms of $\mu_{qwt}$ and $\sigma_{lb}$, decrease when the percentage of unavailable grid resources increase. Although this decrement is linear
Fig. 5. Average queue waiting time ($\mu_{qwt}$) when changing proportion between jobs types.

Fig. 6. Standard deviation of load balancing ($\sigma_{lb}$) when changing ratio between jobs types.
Fig. 7. Average queue waiting time ($\mu_{qwt}$) when changing the number of available resources.

Fig. 8. Standard deviation of load balancing ($\sigma_{lb}$) when changing the number of available resources.
for each algorithm, the AAA results to be more robust in respect with the others in that its decrement is small. This demonstrates the ability of the proposed algorithm to better adapt itself to the typical dynamism of grid resources. Moreover, the performance of AAA in terms of $\sigma_{lb}$ became better than the performance of RA when the number of resources that go down is greater that 15%. This happens because the proportion between jobs and resources in the RA changes, becoming unbalanced.

### 4.4.4. Behavior in presence of wrong jobs execution time estimation

The Fig. 9 shows the average queue waiting time when there are errors in the estimation of job execution time. Measures have been made when the estimation error is respectively 0 (correct estimation), 5, 10, 20, 25 and 30% of total execution time. The figure shows only the trends related to AAA and WND algorithms since their behavior is related to job size estimation: other algorithms are insensitive to these types of errors. the simulation has demonstrated that both algorithms present a high error tolerance since their performance remains approximately constant for the error percentages considered. This happens because, although the estimation could be wrong, the three types of jobs have such different ranges that it is very difficult to make a mistake in enqueuing a job.

### 5. Conclusion

In this paper the Alienated Ant Algorithm, a metaheuristic technique freely inspired by the Ant Colony Optimization theory, was proposed. The aim of this algorithm is to reduce the jobs average queue waiting
time and maintain a good load balancing capability in environments characterized by a great dynamism as the grids. The main characteristics of the proposed algorithm, that makes it different from the other ACO approaches, is an inverse interpretation of pheromone trails that allows the identification of the least loaded computational resource among the ones available in a Grid environment.

Also, a comparison with other scheduling algorithms performance is made in order to illustrate the Alienated Ant Algorithm performance when it is used to face the job scheduling issues in a Grid environment. The simulation campaign results show the benefits of the AAA for the allocation of a huge quantity of resources in the Grid and its capability to adapt itself to the dynamic resources state.

In fact, if a comparison has done with the other solutions within the considered grid, the AAA results as the one that gives the best performance in terms of average queue waiting time for all of the simulated conditions (different numbers of jobs, different number of workloads, different degrees of availability at the nodes), also presenting good performance in terms of load balancing capabilities. One of the most important aspects of the proposed algorithm results being its capacity of adapting to the dynamism of the nodes which represents one of the fundamental characteristics for a scheduling algorithm in the grid environment. The authors are evaluating new techniques to update pheromone trails and evaporation mechanism and the use of the algorithm in a multi broker Grid environment.

References

M. Bandieramonte et al. / Grid jobs scheduling: The Alienated Ant Algorithm solution

[42] PIZ2S http://www.piz2s.it.


Authors’ Bios

**Antonella Di Stefano** received the Laurea degree in Electrical Engineering from the University of Catania (Italy) in 1983. Her research interests have been mainly in the area of parallel and distributed systems, including applicative scenarios, models, software design, architectures and protocols. Currently, she is a full professor of operating systems in the Engineering Faculty of the University of Catania. She is currently working on multiagent systems, Grid and P2P, with special emphasis given to distributed scheduling, QoS and WS.

**Giovanni Morana** is a temporary research fellow at Dept. of Information and Telecommunication Engineering at University of Catania, where he took Ph.D. (2009) and degree in Information Engineering (2005). His research interests are mainly in the area of parallel and distributed systems, including applicative scenarios, models, architectures and communication protocols, with particular reference to Grid Computing and P2P systems.

**Marilena Bandieramonte** took the Laurea degree in Computer Science Engineering from the University of Catania (Italy) in 2007. Her research activity focuses on meta-heuristics and scheduling algorithm in Grid Computing.