# Dynamic Meta-Scheduling Architecture based on Monitoring in Distributed Systems

**Florin Pop*, Ciprian Dobre, Corina Stratan, Alexandru Costan, Valentin Cristea**

Computer Science Department, Faculty of Automatic Control and Computer Science, University "Politehnica" of Bucharest, Romania
*E-mail*: {florin.pop, ciprian.dobre, corina.stratan}@cs.pub.ro, {alexandru.costan, valentin.cristea}@cs.pub.ro
*Corresponding author

**Abstract:** The Scheduling process in Large Scale Distributed Systems (LSDS) became more important due to increases in the number of users and applications. This paper presents a dynamic meta-scheduling architecture model for LSDS based on monitoring. The dynamic scheduling process tries to perform task allocation on the fly as the application executes. The monitoring has an important role in this process because it can offer a full view of nodes in distributed systems. The proposed architecture is an agent framework and contains a Grid Monitoring Service, an Execution Service and a Discovery Service. The performance of the used monitoring system, MonALISA, is very important for dynamic scheduling because it ensures the real-time process. The experimental results validate our architecture and scheduling model.

**Keywords:** Grid Scheduling, Monitoring, Resources Allocation, Distributed Systems.

**Biographical notes:**
**Florin Pop**, PhD, is assistant professor of the Computer Science Department of the University Politehnica of Bucharest. His research interests are oriented to: scheduling in Grid environments (his PhD research), distributed system, parallel computation, communication protocols and numerical methods. He received his PhD in Computer Science in 2008 with "Magna cum laudae" distinction. He is member of RoGrid consortium and participates in several research projects in these domains, in collaboration with other universities and research centers from Romania and in international project like EGEE and SEE-GRID. He has received an IBM PhD Assistantship in 2006 (top ranked 1st in CEMA out from 17 awarded students) and a PhD Excellency grant from Oracle in 2006-2008.
**Ciprian Dobre**, PhD, is assistant professor of the Computer Science Department of the University Politehnica of Bucharest (UPB). The main fields of expertise are Grid Computing, Monitoring and Control of Distributed Systems, Modeling and Simulation, Networking, Parallel and Distributed Algorithms. He is involved in a number of national and international projects (MonALISA, MONARC, VINCI, VNSim, EGEE, SEE-GRID, EU-NCIT). He is actively collaborating with

Oracle from which he received a PhD grant of excellence. His PhD thesis was oriented on Large Scale Distributed System Simulation. His research activities were awarded with the Innovations in Networking Award for Experimental Applications in 2008 by the Corporation for Education Network Initiatives (CENIC).

**Corina Stratan**, PhD, is a postdoctoral researcher in the Computer Systems Group at Vrije Universiteit Amsterdam, working on resource selection in large scale distributed systems. In 2008 she obtained a PhD in Computer Science from the University Politehnica of Bucharest, Romania; her PhD research was focused on monitoring and performance analysis in distributed systems. During the PhD studies she contributed to several national and international projects, and was a teaching assistant for courses like Parallel/Distributed Algorithms and Communication Protocols. She received IBM PhD Fellowship awards in 2006 and 2007 and worked as a summer intern at the IBM T.J. Watson Research Center.

**Alexandru Costan** is a PhD student and Teaching Assistant at the Computer Science department of the University Politehnica of Bucharest. His research interests include: Data Storage and Modeling, Grid Computing, P2P systems. He is actively involved in several research projects related to these domains, both national and international, from which it worth mentioning MonALISA (in collaboration with Caltech and CERN), MedioGRID, EGEE, P2P-NEXT. His PhD thesis is oriented on Data Storage, Representation and Interpretation in Grid Environments. He has received a PhD Excellency grant from Oracle in 2006 and was awarded an IBM PhD Fellowship in 2009.

**Valentin Cristea**, PhD, is a professor of the Computer Science and Engineering Department of the University Politehnica of Bucharest (UPB). He teaches several courses on Distributed Systems and Algorithms. The course Distributed Computing in Internet delivered to master degree students is close to the subject of the proposed book. Also, as a PhD supervisor he directs several thesis on Grids and Distributed Computing. The co-authors of this proposal are among his former or actual PhD students. Valentin Cristea is Director of the National Center for Information Technology of UPB and leads the laboratories of Collaborative High Performance Computing and eBusiness. He is coordinator of national and international projects in IT, member of program committees of several IT Conferences, reviewer of ACM. He directs R&D projects in collaboration with multinational IT Companies (IBM, Oracle, Microsoft, Sun) and national companies (RomSys, UTI).

## 1 Introduction

Over the last years, there has been an important shift in high performance computing from resources mainly architected around few devices with important processing power towards systems with large amounts of commodity components. This architectural shift from the few to the many raises numerous design issues and assumptions pertaining to scale, reliability, heterogeneity, manageability, and system evolution over time. Hence, the challenges faced by high performance distributed systems are scalable monitoring of system state and dynamic

scheduling of tasks. The aim of this paper is to presents a solution for dynamic scheduling based on monitoring information in distributed systems.

More applications are turning to Large Scale Distributed Systems (LSDS) computing to meet their computational and data storage needs. Single sites are simply no longer efficient for meeting the resource needs of high-end applications, and using distributed resources can give the application many benefits. Effective LSDS computing is possible, however, only if the resources are scheduled well.

Scheduling is a key concept for multitasking and multiprocessing design in large scale distributed systems and a most important aspect in real-time system design. Scheduling is the process of assigning tasks on compute resources according with a task policy and ordering communication between tasks. This assignment is carried out by software known as a scheduler. The scheduling process is also known as the allocation process of computation and communication in time. In manufacturing, the scope of scheduling is to minimize the production time and costs (in many case the cost is represented by money), by offering a production facility what to make, when, with which staff, and on which resources. Production scheduling aims to maximize the efficiency of the operation and reduce costs.

The paper presents the monitoring and dynamic sche-duling for LSDS and it is organizes as follows: first, it is presented the related work about approaches that are representative for the areas of dynamic meta-scheduling and monitoring in distributed systems. The 3rd section presents the scheduling process in LSDS, then the 4th section presents the important monitoring systems. The section 5 presents a grid scheduling architecture based on monitoring and describes each component. In the 6th section we analyze some experimental results for the chosen monitoring system and an important result obtained in scheduling process: estimation of execution time.

## 2 Related work

In this section we shall briefly present some approaches that are representative for the areas of dynamic meta-scheduling and monitoring in distributed systems.

Frumkin (2003) describes a methodology that uses monitoring and benchmarking to obtain information about the current performance of Grid resources in a dynamic way; the results are further used in meta-scheduling. The jobs are submitted with the aid of a broker (or navigator), together with their hardware and software requirements; the broker then consults the GridScape, which is a description of the Grid resources and of their current state. The GridScape is updated with the results obtained by running periodically the NAS Grid Benchmarks Frumkin (2002), and also with information provided by various monitoring tools (like the Globus Monitoring and Discovery Service or the Network Weather Service Swany (2002)). After the broker consults the GridScape, it assigns the job to a suitable resource by following a protocol named WARE (Waiting-Assigned-Running-Executed). The protocol specifies a set of steps to be taken in the broker's communication with the task management servers that are running on the resources; its purpose is to prevent both the congestion and the underuse of the Grid resources.

One of the most well known Grid meta-schedulers is GridWay Poletti (2007), which acts as a job execution manager and resource broker on top of Globus services. GridWay works in a decentralized fashion, and is designed as a client-side tool; its purpose is to make job scheduling and submission transparent to the user. After the resource selection step, a set of Middleware Access Drivers (MADs) are used to access Grid services like job execution and file transfer. In order to adapt dynamically to the changes that often happen in the environment, GridWay uses mechanisms like fault recovery, migration on-request and opportunistic migration.

Another system that provides meta-scheduling functionalities is Condor-G Frey (2002), which can be used as a front-end to a computational Grid based on Globus middleware. Condor-G is targeted to end-users and help them manage large numbers of jobs, through various services like monitoring, logging, credential management and fault tolerance.

The EGEE Workload Management Systems, known as EGEE WMS Poletti (2007) adopts a more centralized approach to meta-scheduling. One of its main components is the Resource Broker, which is based on Condor-G; the jobs are submitted to a Computing Element, which acts as a front-end to a computing cluster made up of Worker Nodes. Other available components are the Logging and Bookkeeping Service, the Information System and the Data Management Service.

The Koala meta-scheduler Avizienis (2005) can provide processor and data co-allocation, which brings significant performance improvements when executing scientific applications - as they usually process large amounts of data. The metascheduler implements policies targeted to malleability - the ability to adapt the applications to variable amounts of resources. Other important aspects of Koala are the extensibility (new scheduling policies can be easily implemented and integrated into the scheduler) and the fault tolerance mechanisms.

Other Grid meta-scheduling projects are pre-sented in Buyya (2004), which introduces GridBus, an economy-based Grid service broker, and in Bobroff (2008), which describes a set of protocols for interoperability among different meta-schedulers.

In what concerns distributed systems monitoring, several architectural models and standards have been proposed in order to provide starting points for implementations and to facilitate the interoperability among various monitoring platforms. We mention here the Grid Monitoring Architecture (GMA) Tierney (2002), a high-level design model proposed by the Global Grid Forum. GMA is based on the interaction between three components: producers, consumers and a directory service; the latter allows the producers to publish the types of monitoring data that they provide, and the consumers to search for the producers that can supply the needed type of data.

Since the Grids are large and heterogeneous systems, an important requirement for the monitoring instruments is interoperability - i.e., the ability to exchange data with other monitoring instruments. In order to make this possible, a standardized schema for representing the monitoring information is necessary. The most widely used standard for representing monitoring information is GLUE Andreozzi (2003), which specifies sets of attributes for Grid sites and services. The GLUE schema is represented as a set of UML class diagrams, and mappings to technologies like LDAP, XML and relational data models are also provided.

A monitoring project evolved within the European EGEE project is GridIce Aiftimiei (2006), which has a two-level hierarchy for collecting data: local site collection and grid wide collection. GridIce provides several ways to access the monitoring information: a web-based interface with both textual and graphical representation, an XML representation to be used by applications and also a publish/subscribe system for event notifications.

Another distributed monitoring system with a hierarchical design is Ganglia Massie (2004). For intra-cluster monitoring, it uses a multicast-based protocol to distribute the information, and for aggregating information in a multi-cluster environment a tree of point-to-point connections is created among representative cluster nodes. Among the technologies used in Ganglia are XML for data representation, XDR for data transport and RRDtool for storing and visualizing the data.

The R-GMA (Relational Grid Monitoring Architecture) monitoring system Cooke (2004) is based on the GMA architecture, but has two particularities: there is no need to know about the directory service in order to supply or obtain information (because the communication with the service is handled by the producer and the consumer), and the system appears and can be queried as a relational database. The data in this virtual database is stored in a distributed fashion, and the database is defined by its schema (the set of table definitions), a list of data providers and a set of rules used for deciding which providers to contact for a given query.

## 3 Scheduling in Distributed Systems

Today, the Gird systems are characterized by **heterogeneity** and **autonomy**. Different systems have different resources types and optimal scheduler is one of the main challenges for grid middleware in all Grid systems. Autonomy and dynamism for Grid systems require different methods and policies for resource selection and computation-data separation.

At the highest level, a distinction is drawn between local and global scheduling. The **local scheduling** discipline determines how the processes resident on a single CPU are allocated and executed; a **global scheduling** policy uses information about the system to allocate processes to multiple processors to optimize a system-wide performance objective. Obviously, Grid scheduling falls into the global scheduling branch.

Under the global scheduling we can choice between static and dynamic scheduling. In the case of **static scheduling**, information regarding all resources in the Grid as well as all the tasks in an application is assumed to be available by the time the application is scheduled. By contrast, in the case of **dynamic scheduling**, the basic idea is to perform task allocation on the fly as the application executes. This is useful when it is impossible to determine the execution time, direction of branches and number of iterations in a loop as well as in the case where tasks arrive in a real-time mode. These variances introduce forms of non-determinism into the running program Ernemann (2002). Both static and dynamic scheduling is widely adopted in Grid computing. For example, static scheduling algorithms are studied in Braun (2001); Casanova (2000), and

in Takefusa (2001); Muthuvelu (2005); Kurowski (2004); Chen (2002), dynamic scheduling algorithms are presented.

In LSDS scheduling process involves three main phases described in Schopf (2004): first, resource discovery, which generates a list of potential resources; second, information gathering about those resources and selection of a best set of resources according with users requirements; third, task execution (system preparation and submission), phase which includes task staging and system cleanup. Because each of the main phases includes some steps, it produces many levels and it's difficult to implement all steps in a real environment.

For a *general purpose* a scheduling approach should make some assumptions about and have few restrictions to the types of applications that can be executed. Interactive tasks, distributed and parallel applications, as well as non-interactive batch tasks, should all be supported with good performance. This property is a straightforward one, but to some extent difficult to achieve. Because different kinds of tasks have different attributes, their requirements to the scheduler may contradict. For example, a real-time task, requiring short-time response, prefers space-sharing scheduling; a non-interactive batch task, requiring high-throughput, may prefer time-sharing scheduling. To achieve the general purpose, a tradeoff may have to be made. As it is mentioned above, the scheduling method focused on parallel tasks, while providing an acceptable performance to other kinds of tasks.

*Efficiency* has two meanings: one is that it should improve the performance of scheduled tasks as much as possible; the other is that the scheduling should incur reasonably low overhead so that it won't counterattack the benefits.

The *fairness* refers to sharing resources among users raises new challenges in guaranteeing that each user obtains his/her fair share when demand is heavy. In a distributed system, this problem could be exacerbated such that one user consumes the entire system. There are many mature strategies to achieve fairness on a single node.

The *dynamics* of the scheduling problem means that the allocation algorithms employed to decide where to process a task should respond to load changes, and exploit the full extent of the resources available.

For the *transparency* of scheduling process the behavior and result of a tasks execution should not be affected by the host(s) on which it executes. In particular, there should be no difference between local and remote execution. No user effort should be required in deciding where to execute a task or in initiating remote execution; a user should not even be aware of remote processing, except maybe better performance. Further, the applications should not be changed greatly. It is undesirable to have to modify the application programs in order to execute them in the system.

In the case that all information regarding the state of resources and the tasks is known, an **optimal assignment** could be made based on some criterion function, such as minimum makespan and maximum resource utilization. But due to the NP-Complete nature of scheduling algorithms and the difficulty in Grid scenarios to make reasonable assumptions which are usually required to prove the optimality of an algorithm, current research tries to find **suboptimal** solutions, which can be further divided into the following two general categories.

If a distributed scheduling algorithm is adopted, one issue that should be considered is whether the nodes involved in task scheduling are working

**cooperatively** or independently (**non-cooperatively**). This is an important issue for agents' framework architectures for computational intensive and data intensive tasks. In high energy physics, bioinformatics, and other disciplines, there are applications involving numerous, parallel tasks that both access and generate large data sets, sometimes in the petabyte range. Data sets in this scale require specialized storage and management systems and data Grid projects are carried out to harness geographically distributed resources for such data-intensive problems by providing remote data set storage, access management, replication services, and data transfer protocols. Further, assigning a task to the machine that gives its best execution time may result in poor performance due to the cost of retrieving the required input data from data repositories. In Park (2003), Park et al classify models of cost measured in makespan into five categories, namely: Local Data and Local Execution (here, local means where a task is submitted), Local Data and Remote Execution, Remote Data and Local Execution, Remote Data and Same Remote Execution, Remote Data and Different Remote Execution.

There also exist non-traditional approaches for Grid task scheduling like **Grid economic model**, scheduling methods inspired by nature's laws (e.g. using **Genetic Algorithms**), scheduling under **QoS constraints**, and using strategies treating dynamic resource performance. Each method can be implemented using different schedulers' architecture and different policies for execution decisions.

Many models and algorithms for Grid scheduling are developed using classic algorithms for traditional systems. Three heuristics are used for scheduling of tasks with precedence orders in heterogeneous parallel and distributed systems: list heuristics, duplicated heuristics and clustering heuristics. Almost all algorithms in the current literature refer to list algorithms. The ideas behind the latter two categories have many advantages in the Grid scenario. Since all of these heuristics consider complex application models, where tasks can be fine granular and with data and control dependency, there is great potential for using these heuristics in Grid computing.

The dynamism in the Grid requires the assumptions approximation algorithms optimizing. To deal with performance variation, resource information and prediction are recently used. As the techniques in this field develop, better performance knowledge prior to the task scheduling stage can be expected. Current scheduling algorithms consider a snapshot value of the prediction when they make the estimate, and assume that value is static during the task execution period. This might be a waste of the prediction efforts which can actually provide continuous variation information about the system. So, heuristics that can exploit multiple stage prediction information should be designed.

Another issue is to reestablish approximating for make-span optimization based on performance predictions. For example, if we know the range of performance fluctuation is bounded, we can find a bound for the ratio of real make-span to optimal finish time accordingly.

The problem with current rescheduling algorithms is high cost and lack of consideration of dependent tasks. For tasks whose make-spans are large, rescheduling for the original static decisions can improve the performance dramatically. However, rescheduling is usually costly, especially in DAGs where there are extra data dependencies among tasks compared to independent applications.

In addition, many other problems also exist, for example when the rescheduling mechanisms should be invoked, what measurable parameters should decide whether a rescheduling is profitable, and where tasks should be migrated. Current research on DAG rescheduling leaves a wide open field for future work.

QoS is the concern of many Grid applications. Most current research concentrates on how to guarantee the QoS requirements of the applications like Huedo (2004), but few of them study how the QoS requirements affect the resources assignment and then the performance of the other parts of the applications. Scheduling algorithms in traditional computing paradigms barely consider the data transfer problem during mapping computational tasks, and this neglect will be costly in the Grid scenario. Only a handful of current research efforts consider the simultaneous optimization of computation and data transfer scheduling, which brings opportunities for future studies.

Although the Grid have the characteristics of heterogeneity and dynamics, these features are not flatly distributed in resources, but are rather distributed hierarchically and locally in many cases, due to the composition of the resources. Current resources are usually distributed in a clustered fashion. Resources in the same cluster usually belong to the same organization and are relatively more homogeneous and less dynamic in a given period. Inside a cluster, communication cost is usually low and the number of applications running at the same time is usually small. These distribution properties might bring another possibility for new algorithms to deal with the challenges. For example, by taking multiphase or multilevel strategies, a scheduler can first find a coarse scheduling in the global and then a fine schedule in a local cluster.

This type of strategy has the following advantages: At the higher level, where fine resource information is harder to obtain, the global scheduling can use coarse information (such as load balancing, communication delay of WAN links) to provide decentralized load balancing mechanisms. At the lower level, it is easy for local scheduling to utilize more specific information (such as information from a local forecaster) to make adaptive decisions.

In distributed systems, various applications require dynamic scheduling for optimized assignment of consisting tasks. A number of different scheduling model and algorithms was presented here. The performance metrics can be taken into account in order to design a feasible scheduling algorithm. Those performance metrics can also represent optimization criteria and are based on various constraints such as deadline restrictions, guaranteed completion time, average service time, start and end time, etc. Some of the metrics that can be used to measure the performances of a Grid scheduling algorithm are: the global task success rate (percentage of co-allocated tasks that were started successfully before their deadline), the local task kill rate (the percentage of local tasks that have been killed), the total load (the average percentage of busy processors over the entire system), the global load (the percentage of the total computing power that is used for computing the global tasks), the processor wasted time (the percentage of the total computing power that is wasted because of claiming processors before the actual deadlines of tasks), max-span (the total execution time of tasks in the system, and is practically equal to the largest processing time over all processors), average processor utilization (a measure of the average times of utilization of processors, relative to the maximum execution time) and load-

balancing (measure of the uniformity of the tasks disposal on the processors, with the purpose to obtain similar execution times on processors, and reduce idle times and overloading).

All this criteria and metrics represent the measure effective computing power that the scheduler has been able to get from the distributed system and managed for tasks execution.

In distributed system middleware a large number of tools is available for scheduling. For cluster scheduling we have PBS, Condor, Sun Grid Engine, and LSF. These tools are included in the centralized scheduling class. Inter-cluster scheduling, known as meta-scheduling are studied, so a number of meta-scheduling research projects are under development, like GridWay (that is an incubator project in Globus), Globus CSF. Still there is no meta-scheduler used on a large scale. A problem that must to be solved for this type of scheduling is scalability. It is an aspect more important in the context of heterogeneous systems (that require a simultaneous management of multiple clusters) and middleware tools.

Genetic Algorithms are used for searching large solution spaces. Multiple possible mappings are computed, which are considered chromosomes in the population. Each chromosome has a fitness value, which is the result of an objective function designed in accordance with the performance criteria of the problem (for example maxpan). At each iteration, all of the chromosomes in the population are evaluated based on their fitness value, and only the best of them survive in the next population, where new allocations are generated based on crossover and mutation operators. The algorithm usually stops after a predefined number of steps, or when no noticeable improvements are foreseen.

Genetic algorithms have been largely used for the task allocation problem. The successful results obtained by means of GAs have proved their robustness and efficiency in the field. Research has been done recently, particularly in the area of hybrid algorithms, which use problem-specific knowledge to speed up the search or lead to a better solution. The optimization of scheduling via GAs using the load balancing performance metric has been a key concern for genetic research.

## 4 Monitoring in Distributed Systems

Operating a successful grid, network or computing facility requires vast amounts of monitoring information. Projects and organizations worldwide that need to track resource usage, network traffic, task distribution and many other quantities rely on monitoring systems to collect the information and present it in a way that allows them to make effective decisions. The systems also have to automatically troubleshoot and optimize very large grid and network systems.

While the initial target field of these applications were networks and Grid systems supporting data processing and analysis for global high energy and nuclear physics collaborations, monitoring tools are broadly applicable to many fields of "data intensive" science, and to the monitoring and management of major research and education networks.

An essential part of managing a global Data Grid is a monitoring system that is able to monitor and track the many site facilities, networks, and the many tasks in progress, in real time. The monitoring information gathered also is essential for

developing the required higher level services, and components of the Grid system that provide decision support, and eventually some degree of automated decisions, to help maintain and optimize workflow through the Grid. The relevant efforts invested in this domain are gathered in some major projects.

Traditionally, high performance computing has focused on scalability as the primary design challenge. The architectural shift towards increasingly distributed and loosely coupled systems, however, has raised an additional set of challenges. These new challenges arise as a result of several factors: increased physical distribution, long running distributed services, and scaling and evolution of systems over time. Increased physical distribution implies multiple, independently failing and unreliable components. This, in turn, requires designing applications whose management overheads scale slowly with the number of nodes. Long running distributed services imply the need to be highly available to clients of the service. This reflects on the need for applications to be robust to a variety of different types of failures. Finally, the scaling and evolution of systems over time implies that hardware and software will change. This, in turn, requires addressing issues of extensibility and portability. Hence, there are a number of fundamental challenges associated with monitoring of distributed systems, which we review in brief: no single point of observation, diversity of hardware and software systems, different policies and decision making mechanisms, scalability support for concurrent and distributed correlated events, frequent updates, stochastic performance information, monitoring intrusiveness issues.

Monitoring large scale distributed systems typically includes four stages as presented in Mansouri (1993):

- (i) generation of events, that is, sensors enquiring entities and encoding the measurements according to a given schema;
- (ii) processing of generated events is application-specific and may take place during any stage of the monitoring process, typical examples include filtering according to some predefined criteria, or summarizing a group of events (i.e., computing the average);
- (iii) distribution refers to the transmission of the events from the source to any interested parties;
- (iv) finally, presentation typically involves some further processing so that the overwhelming number of received events will be provided in a series of abstractions in order to enable an end-user to draw conclusions about the operation of the monitored system.

A presentation, typically provided by a GUI application making use of visualization techniques, may either use a real-time stream of events or a recorded trace usually retrieved from an archive. However, in the context of grids, we generalize the last stage as consumption since the users of the monitoring information are not necessarily humans and therefore visualization may not be involved.

Several architectural models for monitoring systems were developed over the last years. The de facto standard is the Grid Monitoring Architecture (GMA) Aydt (2001) put together by the Global Grid Forum to encourage

implementations (i.e., it is not a standard) and further applications use. A relational model was further developed starting from this standard using the database paradigms for the monitoring process. Besides the layered architecture models, a new model converges from recent research, centering on a hierarchy of components and services. This hierarchical approach is best suited for efficient data collection, federation and aggregation. The model is based on creating a tree of monitoring connections while keeping the monitored system's load at minimum and transmitting changes in monitoring levels to the local monitors. The hierarchical connection structure allows for efficient data aggregation, aiming to reduce the load of the monitoring system.

A generalized monitoring system may need to perform all these activities in various places and in different orders to meet the specific requirements. For instance, processing and dissemination may be omitted and information displayed directly. Therefore a more convenient technique for modeling the architecture of a general monitoring system is to abstract from these activities the functional components: Sensors, Collecting / Publication Interfaces, Discovery, Storage, Analysis / Processing of data, Visualization. These components support various implementations and may be omitted according to different specifications. Their functions are a core set of reference models used by any generic monitoring system. We survey in the following some of the most important monitoring frameworks used for large scale distributed systems and argue the choice for a particular solution for our purposes.

**GridICE** is a distributed monitoring tool designed for Grid systems. It promotes the adoption of de-facto standard Grid Information Service interfaces, protocols and data models. Further, different aggregations and partitions of monitoring data are provided based on the specific needs of different users categories each of them dealing with a different abstraction level of a Grid: the Virtual Organization level, the Grid Operation Center level, the Site Administration level and the End-User level. Being able to start from summary views and to drill down to details, it is possible to verify the composition of virtual pools or to sketch the sources of problems. A complete history of monitoring data is also maintained to deal with the need for retrospective analysis. GridICE integrates with local monitoring systems and offers a standard interface for publishing monitoring data at the Grid level (the default fabric monitoring tool is Lemon from CERN). The set of attributes that are measured is an extension of the GLUE Schema v.1.1. The distribution of monitoring data follows a two-level hierarchy (local site collection, grid-wide collection). The global monitoring information can be accessed in different ways: web-based interface offering both textual an graphical representation, XML representation over HTTP for application consumption and publish/subscribe for the notification of events of interest. GridICE was started in 2003 during the European DataTAG project and is being evolved in the framework of the EGEE project, used in INFN grid.

**R-GMA**. The Grid Monitoring Architecture consists of three components: Consumers, Producers and a directory service, (which we prefer to call a Registry). In the GMA Producers register themselves with the Registry and describe the type and structure of information they want to make available to the Grid. Consumers can query the Registry to find out what type of information is available and locate Producers that provide such information. Once this information is known

the Consumer can contact the Producer directly to obtain the relevant data. The Registry communication is shown by a dotted line and the main flow of data by a solid line. The GMA architecture was devised for monitoring but it also makes an excellent basis for a combined information and monitoring system. The Relational Grid monitoring Architecture (R-GMA) is an implementation of GMA, with two special properties. Anyone supplying or obtaining information from R-GMA does not need to know about the Registry, the Consumer and Producer handle the registry behind the scenes. The Information and monitoring system appears like one large relational database, and can be queried as such. APIs are available in various languages for interaction with R-GMA. Currently APIs are available in Java, C, C++, and Python.

R-GMA is currently being developed as part of the 'Enabling Grids for E-science in Europe' project. Previously it was developed as part of the European DataGrid Project

**Ganglia** is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on over 500 clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes.

Ganglia is based on a hierarchical design targeted at federations of clusters It relies on a multicast-based listen/announce protocol to monitor state within clusters and uses a tree of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state. Within each cluster, Ganglia uses heartbeat messages on a well-known multicast address as the basis for a membership protocol. Membership is maintained by using the reception of a heartbeat as a sign that a node is available and the non-reception of a heartbeat over a small multiple of a periodic announcement interval as a sign that a node is unavailable.

Each node monitors its local resources and sends multicast packets containing monitoring data on a well-known multicast address whenever significant updates occur. Applications may also send on the same multicast address in order to monitor their own application-specific metrics. Ganglia distinguishes between built-in metrics and application-specific metrics through a field in the multicast monitoring packets being sent. All nodes listen for both types of metrics on the well-known multicast address and collect and maintain monitoring data for all other nodes. Thus, all nodes always have an approximate view of the entire cluster's state and this state is easily reconstructed after a crash. Ganglia is an open-source project that grew out of the University of California, Berkeley Millennium Project which was initially funded in large part by the National Partnership for Advanced Computational Infrastructure (NPACI) and National Science Foundation.

The **MonALISA** system is designed as an ensemble of autonomous multi-threaded, self-describing agent-based subsystems which are registered as dynamic

services, and are able to collaborate and cooperate in performing a wide range of information gathering and processing tasks. These agents can analyze and process the information, in a distributed way, to provide optimization decisions in large scale distributed applications. An agent-based architecture provides the ability to invest the system with increasing degrees of intelligence, to reduce complexity and make global systems manageable in real time. The scalability of the system derives from the use of multithreaded execution engine to host a variety of loosely coupled self-describing dynamic services or agents and the ability of each service to register itself and then to be discovered and used by any other services, or clients that require such information. The system is designed to integrate existing monitoring tools and procedures and to provide this information in a dynamic, customized, self describing way to any other services or clients.

The scalability of the system derives from the use of a multi threaded engine to host a variety of loosely coupled self-describing dynamic services, the ability of each service to register itself and then to be discovered and used by any other services, or clients that require such information. The framework integrates many existing monitoring tools and procedures to collect parameters describing computational nodes, applications and network performance. Specialized mobile agents are used in the MonALISA framework to perform global optimization tasks or help and improve the operation of large distributed system by performing supervising tasks for different applications or real time parameters.

MonALISA is currently running around the clock monitoring several Grids and distributed applications on around 350 sites. It has been developed over the last four years by Caltech and its partners with the support of the U.S. CMS software and computing program.

**TeraGrid**. As an example of an MDS4 deployment, we can mention the TeraGrid project, providing both "static" and "dynamic" data (e.g., queue lengths, architecture types) relevant to selecting the "best" resource(s) to use for a particular task. End-users (via a web interface), metascheduling systems, and other applications can use this deployment to find the resource(s) that best meet their needs. A picture of the planned TeraGrid deployment is in the next figure.

Information relevant to dynamic resource selection is available from cluster monitoring, resource management, and scheduling systems. While there is no agreement on the data needed to make resource selection decisions, based on previous work with scheduling system and analysis of several common schedulers used with queued platforms such as TeraGrid, it has been decided to use:
- *14 queuing attributes*: LRMSType, LRMSVersion, DefaultGRAMVersion and port and host, other GRAM versions and hosts and ports, TotalCPUs, Status (up/down), TotalJobs (in the queue), RunningJobs, WaitingJobs, FreeCPUs, MaxWallClockTime, MaxCPUTime, MaxTotalJobs, MaxRunningJobs
- *10 cluster description attributes*: Unique Cluster ID, Cluster Benchmark, ProcessorType, MainMemory size, OperatingSystem, StorageDevice, Architecture, Number of nodes in a cluster/subcluster, TG-specific Node properties, and Node-Queue.

We chose MonALISA as our reference monitoring system due to its scalability and wide range of collected parameters needed by our scheduling component. We needed a system which makes it relatively easy to monitor a large number of heterogeneous nodes with different response times, and at the same time to handle

monitored units which are down or not responding, without affecting the other measurements. Moreover, the monitoing system offers a module based approach which allows us to develop specific monitoring modules for our application. A Monitoring Module is a dynamic loadable unit which executes a procedure (or runs a script / program or performs some requests) to collect a set of parameters (monitored values) by properly parsing the output of the procedure. In general a monitoring module is a simple class, which is using a certain procedure to obtain a set of parameters and report them in a simple, standard format. Monitoring Modules can be used for pulling data and in this case it is necessary to execute them with a predefined frequency (i.e. a pull module which queries an webservice) or to "install" (has to run only once) pushing scripts (programs) which are sending the monitoring results (via SNMP, UDP or TCP/IP) periodically back to the Monitoring Service. Allowing to dynamically load these modules from a (few) centralized sites when they are needed makes much easier to keep large monitoring systems updated and to provide new functionalities dynamically.

## 5 Grid Scheduling based on Monitoring

In order to use the shared resources provided in a distributed environment and to enable high-performance computing, the monitoring service plays an essential role in a scheduling system.

The Grid Monitoring Service has the specific purpose to collect information regarding the status of the Grid global system and obtain real-time information about the various site facilities, networks, and about state of the current activities performed in the system. The monitoring information gathered is used by our method with the purpose to develop, based on the genetic algorithm, automated decisions that maintain and optimize the assignation of tasks on the resources of the computational Grid. We are using MonALISA Newman (2003) distributed service system, which provides reliable real-time information, such as systems information for computer nodes and clusters, network information (traffic, connectivity, topology, etc.) for WAN and LAN, monitoring information about the performance of applications, tasks or services. MonALISA has been used in several large scale distributed systems in which it proved its reliability and scalability.

A schematic view of the DIOGENES (Distributed near-Optimal GENEtic algorithm for grid applications Scheduling) system is presented in Figure 1. Users submit Scheduling requests. A near-optimal schedule is computed by the Scheduler based on the Scheduling requests and the Monitoring data provided by the Grid Monitoring Service (MonALISA). The schedule is then sent as a Request for task execution to the Execution Service. The user receives feedback related to the solution determined by the scheduler, as well as to the status of the executed tasks in the form of the Schedule and task information. Furthermore, the system can integrate new hosts in the scheduling process, or overcome failure situations by means of the Discovery Service. The Scheduling request contains a description (in XML) of the tasks to be scheduled. This way, a user may ask for the scheduling of more than one task at a time. Various parameters have been taken into account for task description: resource requirements (CPU Power, Free Memory, Free Swap), restrictions (deadlines) and priorities.

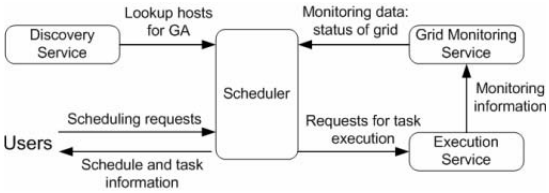


**Figure 1** The DIOGENES Architecture

The assignment of a task on a given computing node is conditioned by meeting the resource requirements.

**Grid Monitoring Service.** The Grid Monitoring Service has the specific purpose to obtain real-time information in a heterogeneous and dynamic environment such as a Grid. It uses the MonALISA distributed service system in conjunction with ApMON, which is a library that can be used to send any status information in the form of UDP datagrams to MonALISA services. MonALISA provides system information for computer nodes and clusters, network information for WAN and LAN, monitoring information about the performance of applications, tasks or services. It proved its reliability and scalability in several large scale distributed systems. We have deployed the existing implementation of the MonALISA Web Service Client to connect to the monitoring service via proxy servers and obtain data for the genetic algorithm. Task monitoring is achieved by means of a daemon application based on ApMON. This daemon provides information regarding task status parameters on each node (amount of memory, disk and CPU time used by the tasks). The up-to-date information offered by the Grid Monitoring Service leads to realistic execution times for assigned tasks.

The monitoring information about computers in the cluster is used for fitness computation in the genetic algorithm. We have enriched the existing implementation of the MonALISA Web Service Client to connect to the monitoring service via proxy servers and obtain data for the genetic algorithm. The up-to-date information leads to realistic computed execution times for assigned schedules, as proved by the experimental results.

The existing Web Service, "MLWebService", is integrated with the MonALISA service, as well as with the MonALISA Repository. The existing implementation was adapted to meet the necessities of the genetic algorithm. Each of the nodes executing the Genetic Algorithm runs a client application requesting and receiving the parameters needed in order to monitor data. Gathering monitoring data from the MonALISA service can be possible using a feature based on the WSDL/SOAP technology of the Web Service (see Figure 2). **Execution Service.** Given its capability to dynamically load modules that interface existing task monitoring with batch queuing applications and tools (e.g. Condor, PBS, SGE, LSF), the Execution Service can send execution requests to an already installed batch queuing system on the computing node to which a particular group of tasks was assigned. Sets of tasks are dynamically sent on computing nodes in the form of a specific command. The time ordering policy established in the genetic algorithm for tasks assigned on the same processor is preserved at execution time.

**Discovery Service.** Lookup processes are triggered by the Discovery Service and determine the possibility of achieving a decentralized schedule by increasing

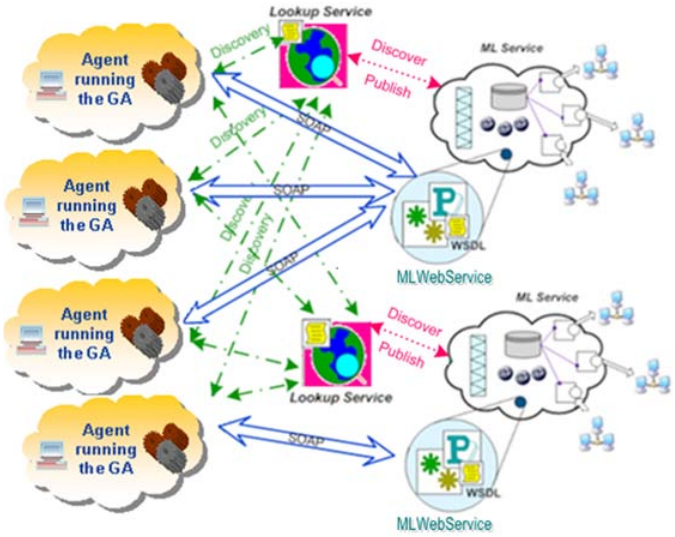


**Figure 2** DIOGENES Agents and MonALISA

the number of hosts involved in the genetic scheduling. The apparition or dysfunction of agents in the system can easily be intercepted, resulting in a scalable and highly reliable optimization tool. If one agent ceases to function, the system as a whole is not prejudiced, but the probability of reaching a less optimal solution for the same number of generations increases. The functionality of the scheduler relies on a communication model in which two different entities are involved: Brokers and Agents. The Brokers collect user requests for task allocations (Task Description Files), parse them and create an object, batch of tasks, which contains the tasks to be scheduled. Then the Brokers forward the requests to Agents. Each Agent executes the scheduling algorithm.

## 6 Experimental Results

### *6.1 MonALISA Performances*

The **Mon*ALISA* Repositories** are special types of clients used for long periods storage and further processing of monitoring data. They subscribe to a set of parameters or filter agents to receive selected information from all the Services. This offers the possibility to present global views from the dynamic set of services running in the distributed network environment to higher level services. The received values are further stored locally into a relational database, optimized for space and time. The collected monitoring information is further used to present a synthetic view of how the global system performs. The system targets developing the required higher level services and components of the network management system that provide decision support, and eventually some degree of automated decisions and control.

For scheduling could be developed a special Repository in our architectural model. It can be used to store for long periods of time the actual monitoring data

for further analysis, fine tuning and eventually some degree of automated decisions and control to higher level services.

The Repository is capable to dynamically plot the parameters we focused on (load, free memory, free disk space, task resources etc.) into a large variety of graphical charts, statistics tables, and interactive map views, following the configuration files describing the needed views, and thus offering customized global or specific perspectives.

The Repository offers several views for comparing our prediction models and also different charts to test the accuracy of our monitoring showing the real data that can be used in scheduling.

The Repository has an automated management systems that could be used in the scheduling process. It has special types of data filters, called Actions. They can register for the data produced by the monitoring modules and also for the predictions produces by the Prediction tool, and can take specific actions when some configurable condition is met. This way, when a given threshold is reached, an alert e-mail can be sent, or a program can be run, or an instant message can be issued. Actions represent the first step toward the automation of the decisions that can be taken based on the monitoring information. It is important to note that Actions can be used in two key points: locally, close to the data source (where monitored data / predictions are produces) where simple actions can be taken like restarting a dead service and globally, in the Repository, where the logic for triggering the action can be more complicated, as it can depend on several flows of data. This automated management framework becomes a key component of any Grid. Apart from monitoring and predicting the state of the various Grid components and alerting the appropriate people of any problems that occur during the operation, this framework can also be used to automate the processes.

The Repository System has fault tolerance capabilities in order to achieve high availability. We can use replication of the Repository Service aiming for a warm standby configuration: in case one repository fails, one or more replicas are ready to take over clients' queries in a transparent way. In this respect we deployed three instances of the repository, located at distinct locations so that local network failures wouldn't affect the system. Deployed repository replicas are permanently aware of each other's current state and after recovery from failure an instance synchronizes its state with the other running replicas ensuring consistency of monitored data.

The Grid Monitoring Service component of our proposed architecture proved its reliability as the MonALISA service on which it is based is used in several Grids, Virtual Organizations and high speed scientific networks. All these infrastructures are architected and operated to guarantee 24x7x365 network and resource availability and full performance, supporting both large-scale data transfers and real-time traffic therefore one of their main concerns is monitoring. These operational centers rely on our architecture for extensive monitoring of the links, the equipment, resources, tasks, users and the end hosts. For this purpose the system we developed within the MonALISA framework is able to meet the high and strict requirements of these infrastructures and to cope with vast amounts of transferred data and high speeds.

Indeed, the MonALISA framework is currently deployed and used to monitor grid sites in the U.S., Europe, Asia and Australia. We have therefore developed

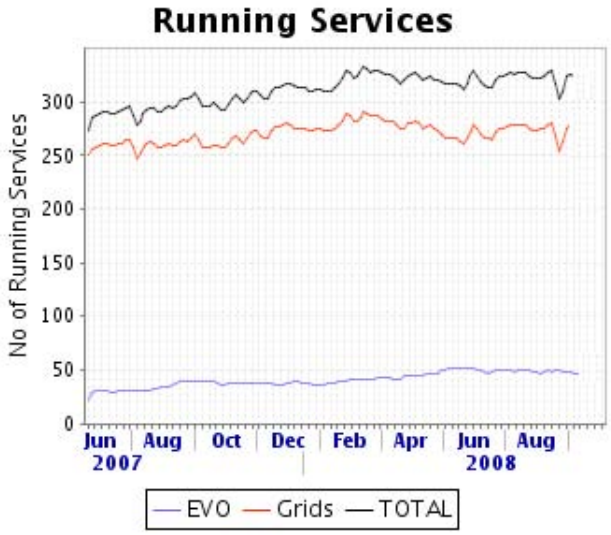


**Figure 3** Running services (total number)

a complementary monitoring tool, the Looking Glass, used for "tracking" of our distributed monitoring system. This system reports on the total number of deployed services, their collection rates, network bandwidth, used / available memory, disk space, number of requests, etc; hence it allows us to manage a global view of how our distributed monitoring system performs. We used the Looking Glass to analyze the overall performance of the deployed monitoring services over long periods of time.

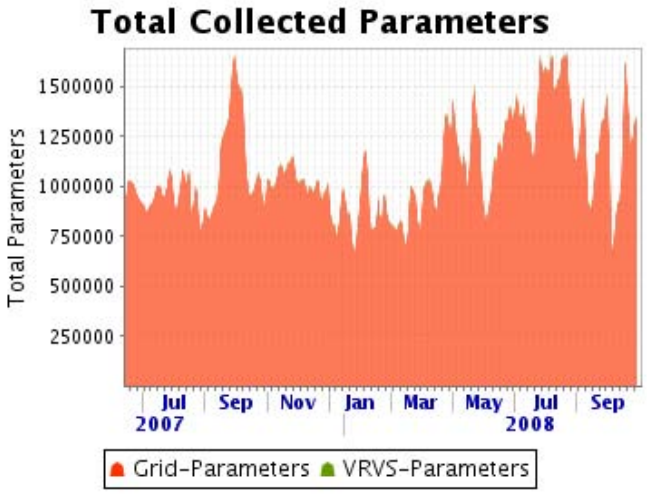


**Figure 4** Collected Parameters (total number)

Figure 3 presents the evolution of the total number of services over the last 2 years, both Grids and EVO farms. Currently there are more than 350 sites running MonALISA and the chart shows a constant evolution of the number of used instances. The inflexions which differentiate the series from an ideal linear increase are caused either by unavoidable network connectivity faults or by intentional stop / restart of the services. However, the series are not interrupted, indicating the continuity of the provided monitoring service; this is achieved in the absence of a single point of failure and using the implemented fault tolerance and replication policies.

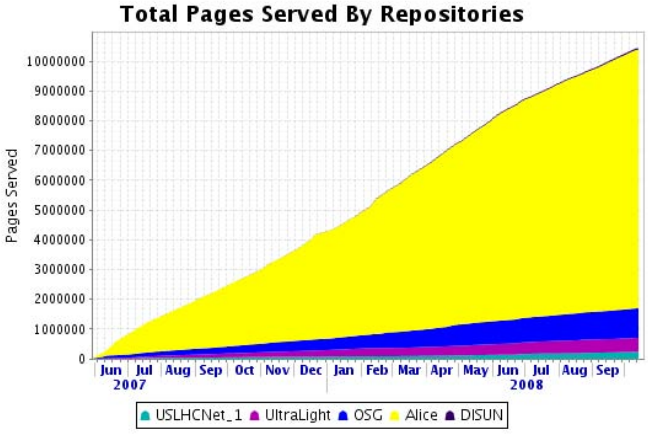


**Figure 5** Served monitoring requests (total number)

In Figure 4, we depict the total number of collected parameters from all the running MonALISA services. The plot shows an average of 1,500,000 parameters over the last months with peaks of 1,600,000. This data collection is achieved in near real-time with a rate of 32,000 updated parameters per minute, collecting data related to approximately 70,000 nodes and thousands of tasks running concurrently in Grid systems. Most of these collected parameters may be further used for scheduling policies as presented in the previous section.

Figure 5 gives an indication of the global load supported by the distributed monitoring system by presenting the total number or served requests from clients resolved over the last year. The data corresponds to a serving rate of 30 requests / minute with peaks of 200-300 requests / minute. As depicted in the above figure, some major Grid communities rely on MonALISA, among which are OSG, CMS, ALICE, D0, STAR, VRVS, LCG Russia, SE Europe GRID, APAC Grid, UNAM Grid, ABILENE, ULTRALIGHT, GLORIAD, LHC Net, and RoEduNet.

### *6.2 Estimated Versus Real Execution Times*

The quality of estimated times is essential for the quality of the schedule. We want the computed processing times to closely approximate the real execution times. That is especially important in hard real-time scheduling problems, in which missing deadlines is extremely problematic. The monotoring system has

an important role in execution time estimations, offering information from grid systems in real-time.

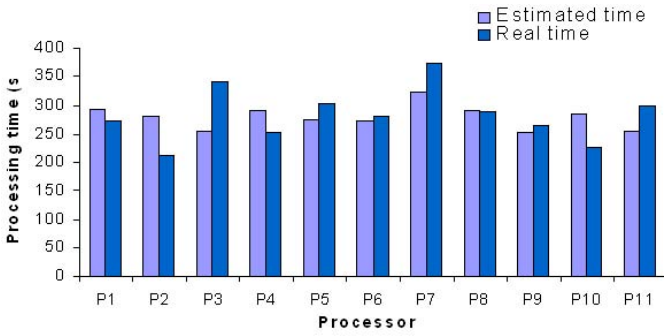


**Figure 6** Estimated and real processing time

Figure 6 provides a comparison between estimated time and real processing time achieved during the experiment. A configuration of the algorithm with 200 generations was used for computation of estimated times. Task running was pursued with PBS (Torque) and task monitoring information was achieved by means of MonALISA Service and its extensions (ApMon and MonALISA Client). The error of approximation is computed according to the method subsequently described.

The relative deviation of the estimated time from the real time for processor $i$ is noted $\delta_i$ and determined as: $\delta_i = \frac{t_i^r - t_i^e}{t_i^r}$.
where $t_i^r$ represents the real processing time obtained by running the tasks in the execution system, and $t_i^e$ is the estimated processing time provided by the genetic algorithm. The approximation error is further determined as $\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\delta_i^2}$.
$0 \leq \epsilon \leq 1$, where $n$ represents the number of tasks.

We obtained an average error value over 10 runs of approximately. The scheduling policy employing real monitoring data from the grid environment is therefore a viable one, providing very good estimation results, with an accuracy of about 84%.

Processing the data obtained from physics experiments is a complex task, involving high performance processing power and very large amounts of storage. It is important for the physicists to be able to trace the execution of their data processing jobs in order to ensure that the results are obtained correctly. One of the described monitoring tool, ApMon, is used for this purpose at CERN, within the ARDA project, whose objective is to coordinate different activities in the development of distributed analysis systems of the LHC experiments, which will be based on the new service-oriented Grid middleware and infrastructure. In this case, the goal is the real-time monitoring of the stages of execution for the processing jobs, the stages being associated with a finite number of states. Job states, from "submitted" to "done", are represented through small integer numbers. By using ApMon the user is able to see precisely what the state of a job was at a certain moment, to make correlations between multiple jobs' state, splitter jobs etc. For applications running on ARDA system (a Monte Carlo simulation), monitored

parameters are: TotalEvents, EventsPerSecond, TotalMemory, ResidentMemory, SamplingInterval, ErrorPerSececond. This is important to see the job's evolution (detect dead-loack, live-lock, etc).

These experimental examples demonstrate that the proposed monitoring component could be successfully used in many different projects, monitoring different complex applications. The flexibility, the diversity of provided results and the correctness of the monitoring process demonstrate that this is a reliable monitoring platform for Grid systems and complex applications.

## 7 Conclusion and Future work

Over the last years, there has been an important shift in high performance computing from resources mainly architected around few devices with important processing power towards systems with large amounts of commodity components. This architectural shift from the few to the many raises numerous design issues and assumptions pertaining to scale, reliability, heterogeneity, manageability, and system evolution over time. Hence, the challenges faced by high performance distributed systems are scalable monitoring of system state and dynamic scheduling of tasks. The aim of this chapter was to presents a solution for dynamic scheduling based on monitoring information in distributed systems.

In Grid environments, various real-time applications require dynamic scheduling for optimized assignment of tasks. This paper describes who monitoring information could be used for scheduling, and offers a solution for decentralized strategy for the problem of task allocation. The use of MonALISA monitoring system facilitates rapid integration of new sites that appear in the Grids.

More applications are turning to Large Scale Distributed Systems (LSDS) computing to meet their computational and data storage needs. Single sites are simply no longer efficient for meeting the resource needs of high-end applications, and using distributed resources can give the application many benefits. Effective LSDS computing is possible, however, only if the resources are scheduled well.

Scheduling is a key concept for multitasking and multiprocessing design in large scale distributed systems and a most important aspect in real-time system design. Scheduling is the process of assigning tasks on compute resources according with a task policy and ordering communication between tasks. This assignment is carried out by software known as a scheduler. The scheduling process is also known as the allocation process of computation and communication in time. In manufacturing, the scope of scheduling is to minimize the production time and costs (in many case the cost is represented by money), by offering a production facility what to make, when, with which staff, and on which resources. Production scheduling aims to maximize the efficiency of the operation and reduce costs.

Furthermore, we have validated our scheduling model in real-time environments, by means of existing monitoring and task execution systems. The experiments show a high accuracy of the results obtained.

A direction for future works is to measure the noise of this modification and also the scalability has to be tested in spite of the fact the scalability is guaranteed by MonALISA. A Benchmark with real applications, such as NAS, can

be interesting with a comparison between this and a classical approach without taking into account the monitoring parameters to show the improvement.

We intend to extend our research in the future towards to the backup and recovery from error. In the case of failures among the computers in the Execution Group, this system component would provide the means to recovery data (the tasks already allocated on the specific execution resource and their inputs), and to re-schedule the tasks to a different execution resource that is up and running. In this case, the monitoring system will be necessary.

## Acknowledgements

The research presented in this paper is supported by national project "DEPSYS - Models and Techniques for ensuring reliability, safety, availability and security of Large Scale Distributes Systems", Project "CNCSIS-IDEI" ID: 1710.

## References

C. Aiftimiei, S. Andreozzi, G. Cuscela, N. De Bortoli, G. Donvito, E. Fattibene, G. Misurelli, A. Pierro, G. L. Rubini, and G. Tortone. Gridice: Requirements, Architecture and Experience of a Monitoring Tool for Grid Systems. In *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP2006), Mumbai, India*, Feb 2006.

S. Andreozzi, M. Sgaravatto, and C. Vistoli. Sharing a Conceptual Model of Grid Resources and Services. In *Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP 2003), La Jolla, CA, USA*, Mar 2003.

R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

R. Buyya and S. Venugopal. The gridbus toolkit for service oriented grid and utility computing: An overview and status report. *CoRR*, cs.DC/0404027, 2004.

H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid en-vi-ron-ments. In *Proceedings of the 9th hetero-geneous Computing Workshop (HCW'00), Cancun, Mexico*, pages 349–363, 2000.

H. Chen and M. Maheswaran. Distributed dynamic scheduling of composite tasks on grid computing systems. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), Fort Lauderdale, Florida, USA*, pages 88–97, 2002.

C. Ernemann, V. Hamscher, and R. Yahyapour. Economic scheduling in grid computing. In *Proceedings of 8th Workshop on Job Scheduling Strategies for*

*Parallel Processing, in conjunction with HPDC/GGF 5, Edinburgh, UK*, pages 128–152, 2002.

J. Frey, T. Tan-nen-baum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

M. Frumkin and R. Hood. Using grid benchmarks for dynamic scheduling of grid applications. NAS Technical Report, Oct 2003.

M. A. Frumkin and R. F. V. der Wijngaart. Nas grid benchmarks: A tool for grid space exploration. *Cluster Computing*, 5(3):247–255, 2002.

K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak, and J. Pukacki. Improving grid level throughput using job migration and rescheduling. *Scientific Programming*, 12(4):263–273, 2004.

M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(5-6):817–840, 2004.

H. H. Mohamed and D. H. J. Epema. Experiences with the koala co-allocating scheduler in multiclusters. In *CCGRID*, pages 784–791. IEEE Computer Society, 2005.

N. Muthuvelu, J. Liu, N. L. Soe, S. Venugopal, A. Sulistio, and R. Buyya. A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. In *Proceedings of the 3rd Australasian Workshop on Grid Computing and eResearch (AusGrid 2005), Newcastle, Australia*, 2005.

H. Newman, I. Legrand, R. Voicu, and C. Cirstoiu. Monalisa: A distributed monitoring service. *CHEP*, 1, 2003.

S. Park and J. Kim. Chameleon: a resource scheduler in a data grid environment. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03), Tokyo, Japan*, pages 253–260, 2003.

D. M. Swany and R. Wolski. Representing dynamic performance information in grid environments with the network weather service. In *CCGRID*, pages 48–56. IEEE Computer Society, 2002.

A. Takefusa, S. Matsuoka, H. Casanova, and F. Berman. A study of deadline scheduling for client-server systems on the computational grid. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), San Francisco, California*, pages 406–415, 2001.

B. Tierney, R. Aydt, D. Gunter, M. Swany, and R. Wolski. A grid monitoring architecture, January 2002.

J. L. Vázquez-Poletti, E. Huedo, R. S. Montero, and I. M. Llorente. A comparison between two grid scheduling philosophies: Egee wms and grid way. *Multiagent and Grid Systems*, 3(4):429–439, 2007.

A. W. Cooke, A. J. G. Gray, W. Nutt, J. Magowan, M. Oev-ers, P. Taylor, R. Cordenonsi, R. Byrom, L. Cornwall, A. Djaoui, L. Field, S. Fisher, S. Hicks, J. Leake, R. Middleton, A. J. Wilson, X. Zhu, N. Podhorszki, B. A. Coghlan, S. Kenny, D. O'Callaghan, and J. Ryan. The relational grid monitoring architecture: Mediating information about the grid. *J. Grid Comput.*, 2(4):323–339, 2004.

N. Bobroff, L. Fong, S. Kalayci, Y. Liu, J. C. Martinez, I. Rodero, S. M. Sadjadi, and D. Villegas. Enabling interoperability among meta-schedulers. In *CCGRID*, pages 306–315. IEEE Computer Society, 2008.

M. Mansouri-Samani, M. Sloman, Monitoring distributed systems, IEEE Network 7 (6) (1993) 20-30.

R. Aydt, W. Smith, M. Swany, V. Taylor, B. Tierney, R. Wolski. A Grid Monitoring Architecture, July 2001

J.M. Schopf. Ten actions when grid scheduling: the user as a grid scheduler. In *In Grid Resource Management: State of the Art and Future Trends, J. Nabrzyski, J.M. Schopf, and J. Weglarz, Eds. Kluwer Academic Publishers, Norwell, MA*, pages 15–23, 2004.

E. Huedo, R.S. Montero, and I.M. Llorente. Experiences on adaptive grid scheduling of parameter sweep applications. In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04)*, pages 28–33, 2004.