Monitoring of Complex Applications Execution in Distributed Dependable Systems

Florin Pop, Alexandru Costan, Ciprian Dobre, Corina Stratan, Valentin Cristea
University "*Politehnica*" of Bucharest
Faculty of Automatic Control and Computer Science, Romania
E-mails: {florin.pop, alexandru.costan}@cs.pub.ro,
{ciprian.dobre, corina.stratan, valentin.cristea}@cs.pub.ro

Abstract

The execution of applications in dependable system requires a high level of instrumentation for automatic control. We present in this paper a monitoring solution for complex application execution. The monitoring solution is dynamic, offering real-time information about systems and applications. The complex applications are described using workflows. We show that the management process for application execution is improved using monitoring information. The environment is represented by distributed dependable systems that offer a flexible support for complex application execution. Our experimental results highlight the performance of the proposed monitoring tool, the MonALISA framework.

Keywords: Monitoring, Workflow Management, DAG Scheduling, Dependable System.

1. Introduction

Distributed dependable systems (DDS), and the Grid in particular, has been developing rapidly in the past few years, with the potential of increased applicability and number of users. As a result, Grid computing is still a very active research field.

One of the most important characteristics of distributed computing in depdendable systems is balancing and optimizing the resource utilization. In order for that to be accomplished, a distributed system is using different software components, such as management tools, schedulers or monitoring tools.

The status of resources in DDS is used in task scheduling with the purpose to meet every task requirements on the one hand, and to fully use the available resources. In the case of dependencies existing between the submitted tasks, it is possible that some of the tasks will have the start time in the future. A dynamic scheduling approach can be applied in this case, with the possibility of increasing the idle time of those tasks. The resource state predictions can be used in a static scheduling algorithm, with an immediate effect on load balancing and resource utilization balance. For complex applications, that are described using workflows, the monitoring offer an realistic overview in the execution process.

As society increasingly becomes strongly dependent of distributed systems (Grid, P2P, network-based), it is becoming more and more imperative to engineer solutions to achieve reasonable levels of dependability for such systems. Monitoring plays an important part in the building process of application for dependable distributed systems. Together with the extension of the complex application domains, new requirements have emerged for large scale distributed systems; among these requirements, reliability, safety, availability, security and maintainability, in other words dependability [1], are needed by more and more modern distributed applications, not only by the critical ones.

This paper presents a solution for monitoring of complex applications execution in distributed dependable systems. Section 2 presents the organization and the related work of monitoring in DDS. In Section 3, the workflow management of complex application is described. The monitoring architecture used for application instrumentation in DDS is presented in section 4. Experimental results of application monitoring are analyzed in Section 5. In section 6 is analyzed the impact of the proposed solution for one of SEE-GRID application. Section 7 presents the future work and the concluding remarks.

2. Related work

Monitoring systems are concerned with collecting information about resources and running applications in the Grid. This information includes status and characteristics of the resource of interest. This large amount of monitoring data is used for a variety of tasks such as performance analysis, prediction, scheduling, data replication, accounting or fault detection and diagnosis.

Taking into consideration the dynamic nature of the Grid and the increasing number of Grid resources, real-time monitoring is gaining in importance. Informations about availability and utilization are also important for the detection of faults and bottlenecks. Zanikolas in [17] proposed a four stage monitoring process:

1. Generation of monitoring data by a sensor (a monitoring process);
2. Processing of generated information such as filtering to some criteria, or summarising a group of data;
3. Distribution refers to the transmission of the monitoring data from the source to any interested parties;
4. Presentation typically involves some further processing.

According to the Global Grid Forum [13], the main components of a monitoring system are the following:

A producer is a process that generates monitoring data (events). Producers can provide access control to the event data, allowing different access to different classes of users, concording to access policies. Other services provided by a producer may include event filtering, caching, and intermediate processing of the raw data as requested by a consumer.

A consumer is a process that receives the events. Depending on his type, a consumer can aggregate and store event data in long-term storage, collect real time monitoring data for online analysis tools or collect events from several sources and use the combined information to make a decision.

A registry is a service that includes information publication and discovery. The registry stores metadata about the performance events and information about producers and consumers that accept requests.

A republisher has both producer and consumer functionality. This component implements functions as filtering, aggregation, summarization, broadcasting and caching monitoring data.

A scope-oriented taxonomy of monitoring systems is presented in [17]. The categories of monitoring systems are defined depending on the provision and characteristics of a system's producers.

Self-contained systems are monitoring systems that do not use a producer interface that would enable the distribution of information to other components. The monitoring data may flow from the sensors in an off-line fashion or in an on-line fashion, through a web interface that provides interactive access to measurement information. Examples of such systems are MapCenter and GridICE.

In producer-only systems, sensors are implemented and hosted at the same machine as the producer. An example of such systems is Autopilot, a framework that determines applications to dynamically adapt to changing environments.

Producer and replublisher systems contain at least one type of republisher that has a fixed functionality. The additional feature provided by this type of monitoring system is the distribution of the functionality among different machines. The republisher can be centralized (CODE-based monitoring system [12], GridRM - Grid Resource Monitoring [2]), distributed (HBM - GlobusHeartbeatMonitor [14], JAMM - Java Agents for Monitoring and Management [9], NetLogger - Network Application Logger Toolkit [5]), or distributed republisher with replication (NWS - Network Weather Service [11]).

Monitoring systems consisting of hierarchy of republishers are highly flexible. They provide configurable general purpose replublishers structured in an arbitrarily hierarchy. Each node in the hierarchy collects and processes monitoring data from lower level to provide it to higher-level of events in a customized view or preparation. Examples of such systems are Ganglia - a scalble cluster monitoring system [16], Globus MDS - Monitoring and Discovery Service [6], MonALISA - a Jini-based monitoring system for large distributed systems [3], Paradyn/MRNet - a performance analysis toolkit for distributed applications [15].

3. Applications Workflow Execution

Grid users submit complex applications to be executed on available resources provided by the Grid infrastructure, setting a number of restrictions like time (deadline), quality constrains, cost of the solution, etc. These applications are split into tasks with data dependencies.

Two types of workflows can be distinguished: *static* and *dynamic*. The process description of a *static workflow* is invariant in time. The process description of a *dynamic workflow* changes during the workflow enactment phase due to circumstances unforeseen at the process definition time. For dynamic workflow, the evolution of tasks execution is esential, so a monitoring component is needed.

The model often used to represent a Grid workflow is a DAG (Directed Acyclic Graph). The main problem raised by the workflow consists of submitting the scheduled tasks to Grid resources without violating the structure of the original workflow.

The DAGs can be separated into: coarse grained DAGs, where computation is dominant over communication, and fine grained DAGs, in which communication is dominant with respect to computation. In some cases, there must be a trade-off between computational cost and reducing the communication cost. The critical path is the weight of the longest path in the DAG and offers an upper limit for the scheduling cost. Algorithms based on "critical path" heuristics for task scheduling and execution produce the best results on average. They take into consideration the critical parh of the scheduled nodes at each step. However, these heuristics can sometimes result in a local optimum, failing to reach the optimal global solution. An workflow example is presented in Figure 1. Tha nodes' value represent the execution costs for tasks and the edges' value represent the transfer costs. For this example, the critical path is 23.

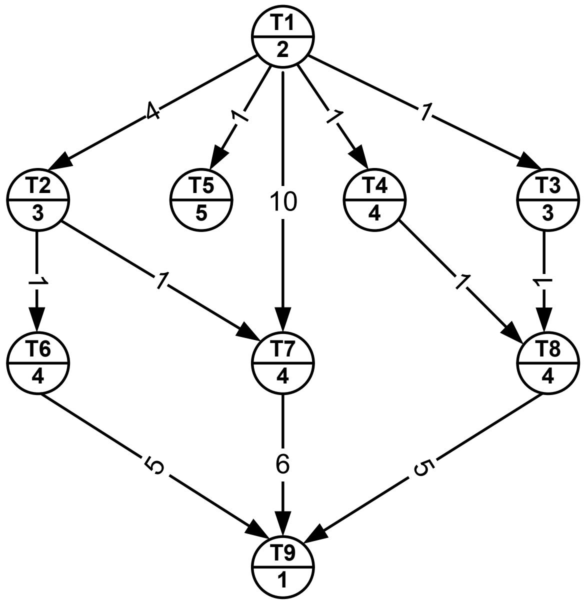

Figure 1. Workflow example

There are several Grid workflow systems that offer solutions for complex applications described using workflows. We present in the following two example of such systems and its capabilities.

DAGMan. The Directed Acyclic Graph Manager (DAGMan) is a service provided by Condor for executing multiple jobs with dependencies [7]. DAGMan does not support automatic intermediate data movement, so users have to specify data movement transfer through pre-processing and post-processing commands associated with processing job. The DAGMan meta-scheduler processes the DAG dynamically, by sending to the condor scheduler the jobs as soon as their dependencies are satisfied and they become ready to execute. DAGMan can also help with the resubmission of uncompleted portions of a DAG when one or more nodes resulted in failure. If any node in the DAG fails, the remainder of the DAG is continued until no more forward progress can be made based on the DAG's dependencies [10].

Pegasus. Pegasus, which stands for Planning for Execution in Grids, was developed as part of the GriPhyN project [4]. Pegasus is a configurable system that can map and execute complex workflows on the Grid. Currently, Pegasus relies on a full-ahead-planning to map the workflows. Pegasus was first integrated with the GriPhyN Chimera system. In that configuration, Pegasus receives an abstract workflow description from Chimera, produces a concrete workflow, and submits it to DAGMan for execution. The workflows are represented as Directed Acyclic Graphs (DAGs). Pegasus contains a "Concrete planner" which consults a Transformation Catalog to determine a location where the job can be executed. If there is more than one possible location, a location is chosen randomly. Pegasus also contains a Virtual Data Language generator that can populate the Chimera catalog with newly constructed derivations. In this configuration, Pegasus similarly generates the necessary submit files and sends the concrete workflow to DAGMan for execution. Once the resources are identified for each task, Pegasus generate s the submit file for Condor-G. The resulting concrete DAG is sent to DAGMan for execution [8].

4. Monitoring architecture

The proposed architecture aims at end-to-end application monitoring across complex environments with total application oversight and the ability to easily, test, restructure and automatically control processes. However, in a large scale application correlated events are generated concurrently and could be distributed in various locations in the applications environment which complicates the management decisions process and thereby makes monitoring of complex systems an intricate task. Therefore, our approach implements a scalable high-performance monitoring architecture for such systems using an agent based mechanism to detect and classify interesting local and global events and disseminate the monitoring information to the corresponding end-points management applications. In order to achieve complex monitoring of resource and application parameters, cooperation of many, and possibly heterogeneous, monitoring data collectors distributed over a wide-area network must be accomplished. In such an environment, the processing and correlation of the data gathered at each collector gives a broader perspective of the state of the monitored resources, in which related events become easier to identify.

Our monitoring framework is able to collect and store the relevant monitoring information, using it to present significant perspectives and synthetic views of how the large distributed system performs. The monitoring information gathered is further used for developing the required higher level services and components of the complex systems that provides decision support, and a degree of automated decisions, in order to help maintaining and optimizing application work-flows. We detail in the following the main components of the proposed monitoring architecture model.

The **MonALISA Service** monitors and tracks site computing farms and network links, routers, and it dynamically loads modules that make it capable of interfacing existing monitoring applications and tools (e.g. Ganglia, MRTG, LSF, PBS, Hawkeye.). The core of the monitoring service is based on a multi-threaded system used to perform the many data collection tasks in parallel, independently. The modules used for collecting different sets of information, or interfacing with other monitoring tools, are dynamically loaded and executed in independent threads. A Monitoring Module is a dynamic loadable unit which executes a procedure (or runs a script / program or performs SNMP request) 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 pre-defined frequency (i.e. a pull module which queries a web-service) 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; users can also implement easily any new dedicated modules and use it in the MonALISA framework. This architectural model of the Service 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. We use the Service to collect key parameters (ex: Load5, FreeMemory, DiskSpace) which are further used for accurate predictions.

The **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.

The Repository is capable to dynamically plot the parameters of interest (load, free memory, free disk space, job resources, cpu time, error codes, run-time 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. We enhanced the Repository System with fault tolerance capabilities in order to achieve high availability. Hence, we used 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 several instances of the repository are deployed, 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.

We further developed a Management Application which uses the monitoring information received from the Service and the Repository to implement high level control through automated actions. Actions are a special type of Filters which can run as an independent application or on top of the Repository. They can register for the data produced by the monitoring modules, but can also take action 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 towards 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 (in the MonALISA service) where simple actions can be taken like restarting a dead service and globally, in a MonALISA client, where the logic for triggering the action can be more complicated, as it can depend on several flows of data as depicted in the Figure 2.

The Actions framework is a key component of monitoring complex distributed applications. Apart from monitoring the state of the various application components and alerting the appropriate administrators of any problems that occur during the operation, this framework is also used to automate the processes. One such automation is called LPM and takes care of the production of Monte Carlo data in the AliEn Grid infrastructure used in the ALICE experiment from CERN. ALICE collaboration produces large data sets which are using the Grid and execute jobs that are very CPU-intensive (approx. 8 SI2K CPU hours) and produce files of about 1GB. Such jobs can fail from various causes: networking, local machine, storage or central services problems are among the most frequent ones. The Actions framework is used to continuously monitor the central task queue

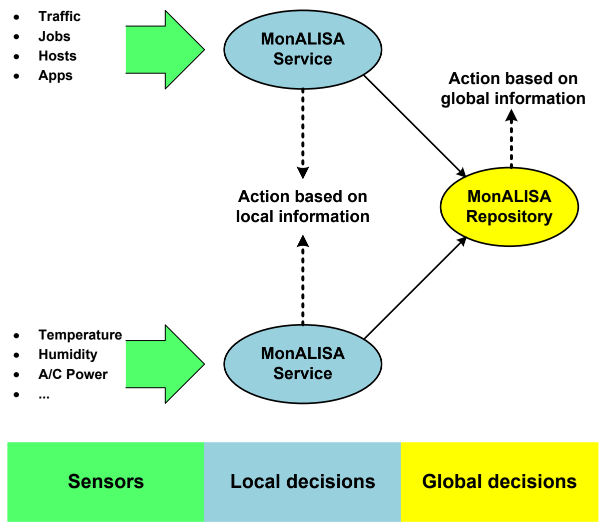

Figure 2. Collecting parameters in MonALISA Services and Repository

for production jobs and take action when the number of waiting jobs goes below the preset threshold (400 jobs at the moment). The action in this case is not to send an alert but first to resubmit the jobs that ended up in an error state then, if the number of waiting jobs is still low, to submit a new production master job that currently splits into 1000 production jobs.

The same framework is used to automatically test the FTD/FTS components. ALICE Grid has to be fully prepared for the data taking with all the components, including the capacity to transport in real time all the data that is acquired from the detectors to one of the ALICE T1 centers. To test the communication and storage infrastructures the Actions framework was set to monitor the transfer queues to each T1 site and submit new transfers to be executed when there are too few waiting in each queue. Combining the continuous testing with error reporting proved to be a very efficient tool in debugging the system. This approach has the advantage to be non-intrusive, whenever there are other transfers to be executed the actions are not triggered, making this testing a background operation that doesn't affect the real transfers.

The **Web Service Client** is used to access data from the Repository via its published Web Services interface. We have implemented a Web Service client that periodically requests monitoring data from the repository. The client selects the parameters depending on the farm, cluster, node and the parameter name. The number of values that the user is interested in is specified with two time parameters, which describe the amount of time between the return results. Each returned values will have a time-stamp associated with it, which will be later used to compute the predicted value time-stamp.

Monitoring is based on the instrumentation with **ApMon** - a lightweight library of APIs that can be used to send any information to MonALISA services. The monitoring data is sent as UDP datagrams to one or more hosts running MonALISA services. Applications can periodically report any type of information the user wants to collect, monitor or use in the MonALISA framework to trigger alarms or activate decision agents. We use this tool to inject the monitored information back into the system for further analysis and comparison. ApMon is available in several programming languages and it was designed to be easily added to applications in order to provide monitoring support, both for application specific data and process and host information.

5. Experimental Results

We used our framework to monitor the The Open Science Grid (OSG) is a consortium formed in 2004, as a continuation of the Grid3 project, with the purpose of enabling multiple scientists communities to access a common Grid infrastructure. One of the main project domains in OSG is nuclear physics, as many of the current OSG applications regard the physics experiments at the Large Hadron Collider from CERN. Other projects developed within OSG are in astrophysics, biology and gravitational-wave science. The OSG includes an Integration Grid, used for testing of new technologies and applications, and Production Grid, which is a stable environment for executing applications. The OSG middleware is packaged with the Virtual Data Toolkit (VDT), including Globus and Condor as main components.

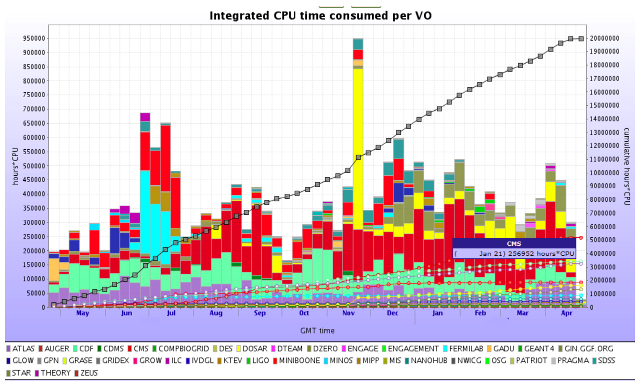

Figure 3. Integrated CPU time statistic from the MonALISA repository

We have been monitoring the OSG group using a global repository for, tracking more than 200 000 parameters from 60 deployed MonALISA services on 26300 nodes. The number of monitored finished jobs this was 10 000 000

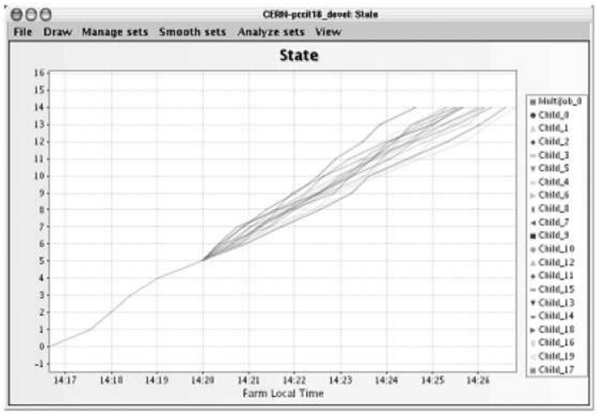

Figure 5. Monitoring the states of an ARDA job and of its sub-jobs

in 50 Virtual Organizations. The repository has been serving a large number of requests at an average rate of 120 requests/hour with peaks of 1500 requests/hour. The average collection rate is 2 300 results/minute. It is very important to understand that it's unfeasible to collect all this information in a central database, both from network and storage point of views. Therefore, detailed information is available for a short period of time by querying directly the MonALISA services running on sites, while aggregated data representing most common parameters is kept in the central repository. This is in fact the entry point for the monitoring of the entire large scale distributed system.

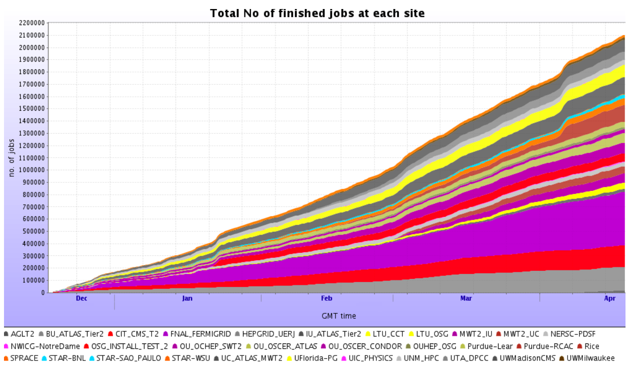

Figure 4. Finished jobs statistic from the MonALISA repository

Figure 3 shows an example of accounting resource usage in OSG group with our proposed monitoring arhictecture. The chart presents the total integrated CPU time consumed at each site in the last week, measured as hours (of CPU time) x number of CPUs. The user can select from the interface the farms which present interest, the time interval for the plot (with predefined periods - last hour, day, week, month, year etc. or specific periods), the representation model (stacked area, series etc.) and image size. The chart also has a description and annotations and is available for download in different formats: CSV, HTML.

Another example of accounting resource usage is shown in Figure 4 where a two axes plot presents the number of finished jobs in each Virtual Organization for the last year as well as the cumulative number of finished jobs in each VO, for the last year.

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. The described 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. As shown in Figure 5, 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.

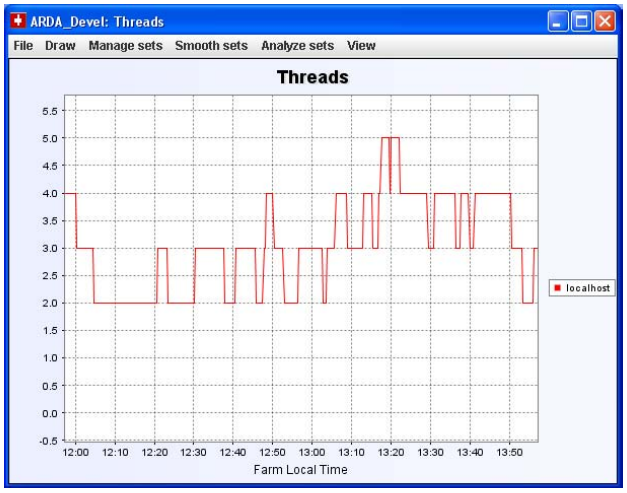

Figure 6. Monitoring the active threads of an ARDA job

For applications running on ARDA system (a Monte Carlo simulation), monitored parameters are: TotalEvents, EventsPerSecond, TotalMemory, ResidentMemory, SamplingInterval, ErrorPerSececond. In example from Figure

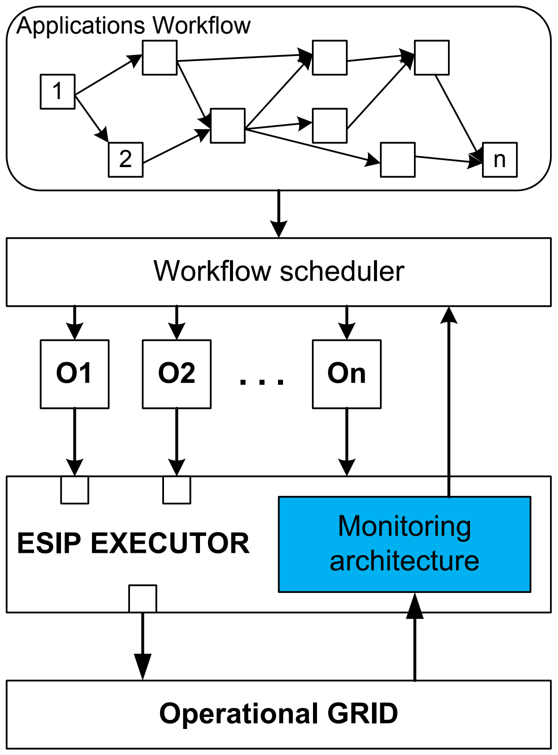

Figure 7. The role of monitoring in Workflow execution on ESIP framework

6 is related the job's threads. 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.

6. The impact of proposed solution

The proposed monitoring architecture will be used in ESIP application (see Figure 7). The Environment oriented Satellite Data Processing Platform (ESIP) is developed through the SEE-GRID-SCI (SEE-GRID eInfrastructure for regional eScience) project. ESIP provides a Grid based software platform for satellite image processing and development of environmental applications. The services and interactive tools are available by Web applications. ESIP will be extended to a larger Grid infrastructure by processing real data for SEE regions.

ESIP is based on execution of workflows. The execution toolset supports the flexible description, instantiation, scheduling and execution of the Grid processing. ESIP provides the user with the possibility to explore the optimal solutions for Grid processing and information searching in the multispectral bands of the satellite images.

The ESIP platform is a collection of Grid services and tools. The basic functionalities are: visual manipulation based interactive description of the Grid based satellite image processing by pattern workflow; development of hypergraphs as a composition of basic operators, services, and subgraphs; pattern workflow instantiation for particular satellite image; satellite data management, access and visualization; workflow based Grid execution. The optimal processing is achieved in terms of code optimization, total execution time, and data communication costs over the Grid. The monitoring solution will be integrated with ESIP executor. In this way, the workflow execution will be monitored and automatic controled. The Monitoring component offers the Grid resource status to the Workflow scheduler.

7. Conclusions

The execution process in Large Scale Distributed System became more important due of increases of users and applications. This paper presents a dynamic monitoring architecture for complex application execution. Dynamic scheduling process tries to perform task allocation on the fly as the application executes.

In Grids, complex applications, described using workflow, require dynamic scheduling for optimizing the execution process. Application monitoring has a significant role in Grid environments, both for providing the user real-time information regarding applications and for gathering statistical data that can be used to analyze the performance of the computing activities. The monitoring is also important in this process because can offer a full view of nodes in distributed systems.

Therefore, within the MonALISA framework, we developed a distributed monitoring architecture, able to collect relevant information, present global views from the dynamic set of services running in the distributed environment to higher level services and to further make accurate analysis of the monitored resources' behaviour.

The novelty of our architectural model consists of the combined used of Sensors (MonALISA Services), Web Service Clients (for monitoring data access), Repository (for global control) and ApMon (used for injecting the application monitoring information into the distributed monitoring system). The experimental results validate our monitoring architecture.

References

[1] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability. Technical report, Research Report No 1145, LAAS-CNRS, 2001.

[2] W.-C. Chung and R.-S. Chang. A new mechanism for resource monitoring in grid computing. *Future Gener. Comput. Syst.*, 25(1):1–7, 2009.

[3] C. Cirstoiu, C. Grigoras, L. Betev, A. Costan, and I. C. Legrand. Monitoring, accounting and automated decision support for the alice experiment based on the monalisa framework. In *GMW '07: Proceedings of the 2007 workshop on Grid monitoring*, pages 39–44, New York, NY, USA, 2007. ACM.

[4] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow management in griphyn. pages 99–116, 2004.

[5] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee. Netlogger: A toolkit for distributed system performance analysis. In *MASCOTS '00: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 267, Washington, DC, USA, 2000. IEEE Computer Society.

[6] H. N. L. C. Keung, J. R. D. Dyson, S. A. Jarvis, and G. R. Nudd. Predicting the performance of globus monitoring and discovery service (mds-2) queries. In *GRID '03: Proceedings of the 4th International Workshop on Grid Computing*, page 176, Washington, DC, USA, 2003. IEEE Computer Society.

[7] G. Malewicz, I. T. Foster, A. L. Rosenberg, and M. Wilde. A tool for prioritizing dagman jobs and its evaluation. *J. Grid Comput.*, 5(2):197–212, 2007.

[8] M.-C. Shan. Pegasus architecture and design principles. *SIGMOD Rec.*, 22(2):422–425, 1993.

[9] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P. Chan. Jam: Java agents for meta-learning over distributed databases. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, AAAI Press, Newport Beach, CA*, pages 74–81, 1997.

[10] C. Stratan, A. Iosup, and D. H. Epema. A performance study of grid workflow engines. In *Grid Computing, 2008 9th IEEE/ACM International Conference on*, pages 25–32, 2008.

[11] M. Swany and R. Wolski. Multivariate resource performance forecasting in the network weather service. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–10, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[12] M. Taufer and T. Stricker. Accurate performance evaluation, modelling and prediction of a message passing simulation code based on middleware. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–14, Washington, DC, USA, 1998. IEEE Computer Society.

[13] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A grid monitoring architecture. Technical report, GGF Performance Working Group, 2002.

[14] G. Toolkit. Globus heartbeat monitor. Communications of the ACM, http://www.globus.org/hbm/, 1997.

[15] R. Valisetty, P. Cheung, and R. Namburu. Scalable coupling of multi-scale aeh and paradyn impact analyses. In *DOD_UGC '04: Proceedings of the 2004 Users Group Conference*, pages 238–242, Washington, DC, USA, 2004. IEEE Computer Society.

[16] C.-T. Yang, T.-T. Chen, and S.-Y. Chen. Implementation of monitoring and information service using ganglia and nws for grid resource brokers. In *APSCC '07: Proceedings of the The 2nd IEEE Asia-Pacific Service Computing Conference*, pages 356–363, Washington, DC, USA, 2007. IEEE Computer Society.

[17] S. Zanikolas and R. Sakellariou. A taxonomy of grid monitoring systems. *Future Gener. Comput. Syst.*, 21(1):163–188, 2005.