DEVELOPMENT AND USE OF MONALISA HIGH LEVEL MONITORING SERVICES FOR THE STAR UNIFIED META-SCHEDULER

E. Efstathiadis*, L. Hajdu, J. Lauret, BNL, Upton, NY 11973, USA
I. Legrand, CalTech, Pasadena, CA 91125, USA

Abstract

We study and develop a kernel of tools that allow Meta-Schedulers to take advantage of a consistent set of shared information across Resource Management Systems, in both local and/or Grid scheduler systems. We demonstrate the usefulness of such tools within the MonALISA monitoring framework and the STAR Unified Meta-Scheduler. We will present and define the requirements and schema by which one can consistently provide queue attributes for the most common batch systems. We evaluate the best scalable and lightweight approach to access the monitored parameters from a client perspective and, in particular, the feasibility of accessing real-time and aggregate information. Client programs are envisioned to function in a non-centralized, fault tolerant fashion. Inherent delays as well as scalability issues of each approach (implementing it at a large number of sites) will be discussed. We believe that such developments could highly benefit Grid laboratory efforts such as the Grid3+ and the OpenScience Grid (OSG).

INTRODUCTION

While many success stories can be told as a product of the Grid middleware developments, most of the existing systems relying on workflow and job execution are based on integration of self-contained production systems interfacing with a given scheduling component or portal. This approach hardly satisfies the Grid concept demanding the use and re-use of inter-operable components. In fact, such systems are often considered non-grid enabled that use a specific Local Resource Management System (LRMS), or fully grid-enabled, but never a hybrid solution taking advantage of both worlds. This is often due to the lack of a consistent set of monitoring information usable and sharable across resource management systems allowing Meta-Schedulers to evaluate the resources and negotiate with heterogeneous workload execution software in order to utilize the right resources to fulfill a high level user described request.

THE STAR UNIFIED META-SCHEDULER

The STAR Unified Meta-Scheduler (SUMS) project provides to the users of the STAR collaboration at the Relativistic Heavy Ion Collider (RHIC) [1] (and beyond) the ability to submit jobs to a farm, to a site (with multiple pools or farms) or to the Grid without the need to know or adapt to the diversity of technologies and knowledge involved while using multiple LRMS and their specificities. Additionally, the strategy was adopted to shield the users against changes in technologies inherent to the emerging Grid infrastructure and developments.

In its simplest form, SUMS presents itself to the end user as a wrapper around evolving technologies sitting on top of one or multiple queue systems negotiating between available queues and pools. The client wrapper interprets as input an XML file describing the user's intend for accomplishing a task and its associated work-flow rather than a traditional ”shell script” code workflow. The XML meta-job description follows a schema defined by a high level User Job Description Language or U-JDL [2] and a single meta-job may be split into many jobs or sub-tasks which are then dispatched to diverse LRMS. The sub-division of a meta-job into sub-jobs depends entirely on the user's input and the available resources. Such job spliting decisions are based on adaptive scheduling policies which may take into consideration conditions such as: the computing cycles available across available resources (via delegation to the LRMS or based on a statically defined branching ratios for scheduling to sites RMS or via Condor-G), the availability and knowledge of statically populated datasets across distributed pool, the intended user access type and pattern for a given requested dataset: resources depend on available access methods for datasets from a given site, a given host or a given cluster etc.

While simplistic, the initial implementation of SUMS was planned to be enhanced by several additional modules aimed to provide information such as the ones related to the available LRMS (status, availability, priorities, queue length and latencies, etc.), available resources and predicted resource availability. In essence, this kind of information is provided to the Meta-Scheduler as input necessary to make high-level resource brokering decisions allowing, for example, a dynamic load balancing of jobs between and across available sites. The evolutionist architecture of the Meta-Scheduler allows for an easy way to take advantage of existing information provide plugins, such as the Monitoring and Discovery System (MDS) [3], Ganglia [4] and MonALISA [5]. Such information will be gathered along experiment specific information (FileCatalog, MetaData or dataset and data collections information) into scheduling policies that encapsulate the logic by which the request will be satisfied. Additionally, new policies may be developed in parallel of production-level policies without disruption of user's activities.

* stratos@bnl.gov

THE MONALISA PROJECT

The MonALISA (Monitoring Agents in A Large Integrated Services Architecture) system provides a distributed monitoring service. It is based on a scalable Dynamic Distributed Services Architecture (DDSA) that is implemented using JINI/JAVA and WSDL/SOAP technologies. The scalability of the system derives from the use of autonomous multi-threaded station servers 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 other services or clients that require such information, and the ability of all services and clients subscribing to a set of events (state changes) in the system to be notified automatically. The framework integrates several existing monitoring tools and procedures to collect parameters describing computational nodes, applications and network performance. It has built-in SNMP support and network-performance monitoring algorithms that enable it to monitor end-to-end network performance as well as the performance and state of site facilities in a Grid.

The Monitoring Service and the Data Collection Mechanism

The core of the MonALISA monitoring service is based on a multithreaded system (the monitoring service) that performs the many data collection tasks in parallel, independently. It is designed to easily integrate existing monitoring tools and procedures and to provide this information in a dynamic, self-describing way to other services or clients. MonALISA services are organized in groups and their group attribute is used for registration and discovery.

The modules used for collecting different sets of information, or interfacing with other monitoring tools, are dynamically loaded and executed in independent threads. A dedicated control thread is used to stop properly the threads in case of errors, and to reschedule those tasks that have not been finished. Monitoring modules are dynamically loaded units that execute procedures to collect sets of parameters (monitored values). They can be used for pulling data (and in this case it is necessary to execute them with a predefined frequency) or to install pushing scripts that send the monitoring values periodically back to the monitoring service. Monitoring modules can be loaded dynamically from a few centralized locations, as they are needed, making large monitoring system easy to update. The collected monitoring values are stored in a relational database, locally for each service. A normalized scheme is used to store the result objects provided by the monitoring modules in indexed tables, which are generated dynamically, as needed.

Registration and Discovery

Each MonALISA service registers with a set of JINI Lookup Discovery Services (LUSs), as a member of a group, and having a set of attributes. The LUSs are also JINI services and may be registered with other LUSs resulting in a distributed and reliable network for registration of services. Services also provide the code base for the proxies that other services or clients will need to instantiate for using it. The registration is based on a lease mechanism that is responsible to verify periodically that each service is alive. In case a service fails to renew its lease, it is removed from the LUSs and a notification is send to all the services and clients that subscribed for such events.

Clients, Proxy Services, Pseudo-Clients and Repositories

Any monitor client service is using the LUSs to find all the active MonALISA services running as members of one or several groups, ”communities”. It is possible to select services based on a set of matching attributes. The clients connect directly with each service it is interested in for receiving monitoring information. They first download the proxies for the service and they instantiate the necessary classes to communicate with it.

The architecture also provides a client proxy service (also a JINI service) which is used by clients to connect to different services. The mutual discovery between services and proxies is used to detect when a service runs behind a firewall. In this case the service initiates a connection to all available proxies in a community and registers itself with the LUSs. Any client can now interact with such services via the proxy services. At the same time, the proxy service does an intelligent multiplexing of subscribed data for multiple clients. Multiple proxy services offer redundancy and load balancing of clients.

Clients can get any real-time or historical data by using a predicate mechanism for requesting or subscribing to selected measured parameters. These predicates are based on regular expressions to match the attribute description of the measured parameter a client is interested in. In case of requests for historical data, the predicates are used to generate SQL queries into the local database. Subscription requests result in the creation of dedicated threads to serve the client. These threads are responsible to perform the matching for all the predicates submitted by the client with the measured parameters. The same threads send the selected results back to the client as compressed serialized objects. Requests for monitoring data are also possible using the WSDL/SOAP binding. The class description for predicates and the methods to be used are described in WSDL and any client can create dynamically and instantiate the objects it needs for the communication.

A generic framework for building pseudo-clients for the MonALISA services was developed. This has been used for creating dedicated web service repositories with selected information from specific groups of monitoring services. The pseudo-clients use the same LUSs approach to find all the active MonALISA services from a specified set of groups and subscribes to these services with a list of predicates and filters. These predicates or filters specify the information the pseudo-client wants to collect from all the services. Pseudo-clients store all the values received from the running services in a local MySQL database, and is using procedures written as Java threads to compress old data. A Tomcat based servlet engine is used to provide a flexible way to present global data and to construct on the fly graphical charts for current or customized historical values, on demand. Multiple Web Repositories can easily be created to globally describe the services running in a distributed environment.

PROVIDING MONALISA MONITORING DATA TO SUMS

Monitoring information is essential for developing the required higher level services and components of the Grid system that provide decision support (and eventually some sort of automated decisions) to help maintain and optimize workflow. As SUMS needs to reliably access monitoring information for complex decision making mechanisms, the MonALISA monitoring system appears to be a natural match.

We have currently deployed three separate MonALISA monitoring services for the Grid needs of the STAR collaboration: one for the resources at the RHIC Computing Facility (RCF) at BNL, one at the Information Technology Division (ITD) also at BNL, and a third one at PDSF, at NERSC/LBNL. All three monitoring services join a group that has been defined to automatically discover the monitoring services that provide relevant monitoring data. A Web Repository has also been set up to provide real time and historical aggregate monitoring data for our group of services.

The Queue Monitoring Module

Custom monitoring modules have been developed to serve our monitoring needs, always focusing on the extensibility and reusability of such work. The queue monitoring module provides aggregate status information for the most popular queuing systems (CONDOR [6], LSF [7], PBS [8]) using the same attributes, making the use of such information easy and self-contained into a defined schema. Initially, we make the provided information compatible with the Grid Laboratory for a Uniform Environment (GLUE) [9] schema. In particular, the Computing Element (CE) of the GLUE Schema represents the entry point to a queue, with one CE per queue. The attributes of the Status object of the CE Element are the most relevant: the number of currently running jobs (RunningJobs), the number of jobs that are in a state other than running (WaitingJobs), total number of jobs (TotalJobs), states a queue can be in (Status), worst time between job submission till the job starts its execution (WorstResponceTime), estimated time between job submission till when the jobs starts its execution (EstimatedResponseTime), and number of free CPUs available to the Scheduler (FreeCPUs). In developing the queue monitoring module similar work [10] has been taken into account.

Using WSDL/SOAP Bindings

In order to provide easy access to monitoring data, the MonALISA system has included WSDL/SOAP bindings for all the distributed objects. A Web Service Client has already been integrated with the SUMS code and a policy has been implemented that uses the retrieved monitoring data in decision mechanisms. Predicates are used to select the parameters that are relevant to the Meta-Scheduler policy specified by the end user. Accessing, though, monitoring data that are stored locally at each individual remote monitoring service in a sequential order has a number of disadvantages: the direct invocation of individual remote web service functions completely bypasses all the advantages the JINI technology offers including the mechanism to automatically discover new monitoring services that have joined our ”community”, there is no fail-over policy in case the connection to a remote services is lost, the data retrieved using WSDL/SOAP were delayed or historical, and finally, there is an extra overhead to translate the web service methods invoked into SQL queries. Regardless of the above disadvantages, using this simplistic mechanism of accessing monitoring data from each individual remote site turned out to be a good debugging tool when comparing data retrieved using different mechanisms.

The same WSDL/SOAP bindings that are available at each individual monitoring service are also available at the Web Repository. Accessing monitoring data via the Web Repository eliminates many of the above disadvantages. Monitoring services that are part of a group are automatically discovered via the LUSs provided by the JINI technology. New web service methods were provided by the MonALISA project developers to access data directly from the data cache of the Repository, instead of translating such requests into SQL queries, returning the latest available monitoring data. Only one invocation of the web service client is necessary to retrieve the monitoring data from all services in the group, instead of the many required to access each individual service. Currently, the only disadvantage of this method is that the web repository may be a single point of failure. We are looking however into having replicas of the same repository with fail-over capabilities.

In order to provide SUMS with lightweight access to monitoring data we have deployed a pseudo-client locally. Pseudo-clients discover and access each monitoring service via one of the four available proxy services with fail-over capabilities. This way monitoring data from all services in the group are collected into a local data cache instead of the remote one at the repository.

Initial Testing

The implemented monitoring policy in SUMS accesses the locally cached queue monitoring data and uses them to select the most appropriate queue for job submission. The

Response Time is the parameter that is used in ordering all the discovered queues. This parameter takes into account the number of pending jobs, the average job pending time, the number of running jobs and the average job run time. The queue with the smallest Response time is selected. We are currently testing this mechanism using only two local LSF queues on separate clusters. A passive policy is used to submit jobs alternating between the two queues, while the monitoring policy uses the queue monitoring data to select a queue with the minimum response time. Preliminary results indicate that the monitoring policy was very effective in choosing the appropriate queue for the job submission reducing significantly the turn around time. However, there were situation in which our queues were being saturated by jobs submitted by another batch system submitting jobs that preempted our testing jobs. Since our policy did not have any information about the second batch system, the monitoring policy became ineffective. Although our first testing results are incouraging, we clearly need to continue polishing our monitoring policy.

CONCLUSIONS

We work on the development and deployment of a set of high-level services and solutions aimed to enhance scheduling capabilities of Resource Brokers, Grid enabled schedulers or Meta-Schedulers. We demonstrated that the MonALISA monitoring framework provides consistent monitoring information that has been successfully implemented in the STAR Unified Meta-Scheduler.

REFERENCES

[1] Nucl. Inst. and Meth. A499 (2003), 624-813

[2] http://www.star.bnl.ogv/STAR/comp/Grid/scheduler/rdl/

[3] The Monitoring and Discovery System (MDS) is the information services component of the globus toolkit: http://www.globus.org/mds/

[4] For a description of the Ganglia project see http://ganglia.sourceforge.net/

[5] H.B. Newman *et al.*, Proceedings of the Computing in High Energy Physics (CHEP) 2003 conference, La Jolla, CA, March 25-28, 2003.

[6] http://www.cs.wisc.edu/condor/

[7] http://www.platform.com/products/LSF/

[8] http://www.openpbs.org/

[9] For more information on the GLUE Schema, see: http://www.hicb.org/glue/glue-schema/schema.htm

[10] The European Data Grid (EDG) project has developed an MDS Information Provider that reports similar queue information.