ASAP Distributed Analysis

Craig Munro, Julia Andreeva, and Akram Khan

***Abstract*—The Grid offers physicists far more resources than have previously been available by combining elements in computing centers around the world into a globally accessible resource. However, in order to harness this computing power many steps need to be performed for job creation and management and data discovery. ASAP (ARDA Support for CMS Analysis Process) was developed to make this step as simple and straightforward as possible for physicists working on the Compact Muon Solenoid (CMS) experiment at CERN. ASAP transfers a local application to the Grid by packaging the application, discovering the location of input data then creating, submitting and monitoring jobs. One of the main components of the system is that users can delegate responsibility for their tasks to the ASAP server which can take actions on behalf of the user to ensure that tasks are completed successfully. The system can operate in the traditional push model but can also be used by agents executing on the Grid to pull jobs to the worker node for execution.**

***Index Terms*—Distributed analysis, grid computing, high energy physics, software.**

I. Introduction

When the Large Hadron Collider [1] begins operation in 2007 each of the four major experiments at the LHC will use a Grid architecture to fulfil their demands for CPU and storage. While this approach is scalable the complexity increases the learning curve for new users trying to access these resources. An analysis that is executed on the Grid as opposed to a local computer is much more complex due to the number of interactions that are required with other systems and, as a consequence, the number of things that can go wrong.

The ARDA [2] project began in 2004 to address these issues by prototyping distributed analysis systems for each of the LHC experiments. The CMS component of ARDA developed ASAP (ARDA Support for CMS Analysis Process) an analysis framework which provides a layer of abstraction on top of the Grid middleware. Steps such as packaging code, locating data, application execution, monitoring and output retrieval are automated so that users can quickly move from a working local application to the Grid. Instead of managing individual jobs users can work at a higher level with a collection of jobs or a task. ASAP also provides a server side component to which users can delegate responsibility for their tasks. The ASAP server will perform job submission, monitor job status and retrieve job output on the users behalf. If a job fails or an error is detected in the jobs output the job will be automatically resubmitted. One of the most interesting features of this server side component is its ability to act as a server to agents operating on the Grid. Operating in this manner the turnaround time of the users tasks can be greatly reduced.

This paper discusses the ASAP design and presents the results of performance measurements.

Manuscript received February 18, 2007; revised May 29, 2007. This work was supported in part by the PPARC.

C. Munro is with Brunel University, Uxbridge, Middlesex UB8 3PH, U.K. and also with CERN, 1211 Geneva 23, Switzerland.

J. Andreeva is with CERN, 1211 Geneva 23, Switzerland.

A. Khan is with Brunel University, Uxbridge, Middlesex UB8 3PH, U.K. (e-mail: Akram.Khan@brunel.ac.uk).

Digital Object Identifier 10.1109/TNS.2007.905162

II. Background

ASAP began to be developed in late 2004 as a prototype used to evaluate the emerging gLite middleware [3]. From the start, development was user driven with physicists invited to test the prototype in order to assemble requirements and identify weaknesses with the new middleware. It quickly became evident that users needed to be insulated from the many steps that were required to prepare and execute jobs on the Grid and from the many changes and errors that could occur. Starting with an analysis application and a configuration file users generate a task which can be submitted using the LCG [4] or gLite [3] middleware. Each task consists of a single analysis application packaged with its libraries, a wrapper script and potentially some input data. The wrapper script establishes the correct environment on the Worker Node (WN), downloads input data if required, unpackages and executes the application and uploads the jobs output to a user defined Storage Element (SE). The script is also responsible for reporting job progress and errors to the CMS Dashboard so that a global view of resource usage and problems can be formed. Jobs are split across the selected dataset and sent to sites that contain the data required for analysis. Upon completion of the job the results are stored on a user-defined SE for later retrieval and analysis. A server side component was also introduced which would resubmit aborted jobs automatically. The ASAP system has evolved to support a variety of modes including public, private and simulated data. The experience gained in the past eighteen months has been used to redevelop ASAP to provide a cleaner, more flexible and robust design with a new higher performance server.

ASAP is not the only job creation tool used by the CMS community. CRAB [5] has many of the same features as ASAP but does not currently have a server side component which means users are responsible for monitoring and resubmitting their own tasks. The Atlas and LHCb experiments use Ganga [6] with LHCb also using DIRAC [7] which has agents which work in a similar manner. DIRAC and Panda [8] are also used for production activities and as such they support more complex workflows and data management operations. They also pre-submit pilot jobs which can execute production or analysis tasks.

III. ASAP Design

A multi-tiered architecture is used and the client and server components interact with a wide range of other Grid services in order to operate, see Figs. 1 and 2. Both the client and server

0018-9499/$25.00 © 2007 IEEE

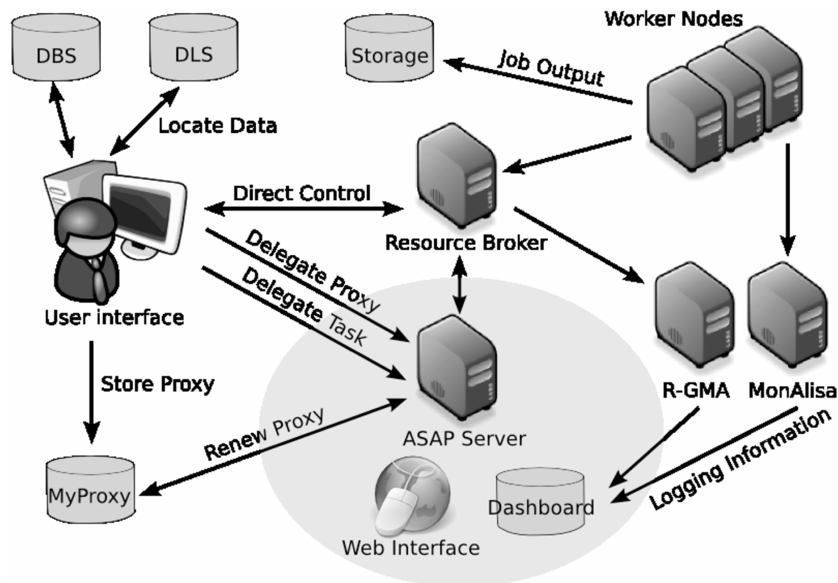

Fig. 1. The interaction of ASAP with other components in the CMS Distributed System.

Server
MyProxy
Server
VOMS
Server
DLS Server
Client
WebBrowser
Server
ASAP Delegation Server
gLite Proxy Renewal Server
Server
DBS Server
Client
CLI Interface
Apache Server
ASAP Server
Server
ResourceBroker
Feedback when using Agent mode
Database
MySQL
Server
ComputingElement
Server
WorkerNode

Fig. 2. Deployment model showing the major interactions for the ASAP client and server.

have been written with maintainability and flexibility in mind. The services that ASAP interacts with are frequently changed or updated and it needs to be possible to easily replace components without rewriting large parts of the code. As a consequence of this although the components are currently CMS specific it would be simple to add components to support different experiments.

The client is responsible for task creation and management including job submission and for interacting with the server. Jobs may be registered or unregistered from the server at any point so that the user can always have full control and flexibility. For example, the user can submit the jobs and then register the jobs with the server to take advantage of the monitoring and resubmission capabilities. The following sections discuss the client and server in more detail.

A. Client

In order to use ASAP to create tasks users must create a configuration file which specifies, among other things, which dataset they wish to use and how they wish to split this dataset across jobs. ASAP uses environment variables specific to the CMS analysis framework to discover the users application and packages any user specific code and libraries that need to be sent with the task. A wrapper script is created for the worker node which unpacks any files that need to be sent with the job, recreate the users environment, executes the application and copies input and output data as required. Data discovery is performed by querying the Dataset Bookeeping System (DBS) [9] for the dataset specified by the user. The DBS resolves the dataset into multiple blocks which are in turn composed of multiple files containing a certain number of events in each. The Dataset Location Service (DLS) [10] then provides the location for each block. The user specifies the total number of events they wish to analyse and how many events they wish to analyse with each job. ASAP uses all of this information to create jobs which are sent to sites which contain the input files that are required to satisfy the requests. In addition to using this official, published data users may also specify their own private data (or no data in the case of simulation) and specify their own job splitting. Private data can be on the local filesystem in which case it will be sent with the job or ASAP can download it from a SE once the job starts.

Once the task has been created the user can perform certain actions depending on each job depending on the status of each job. Actions such as match, submit, update, cancel, fetch, register and unregister can be performed on the entire task or any subset of that task. After the job has been submitted and begins running on a WN the wrapper script will control each stage of execution and report progress or errors to a MonAlisa [11] server from where it will be retrieved by Dashboard. The user can monitor the progress of each job in a specific task on the command line. When the application is finished the wrapper analyses the output to obtain the number of events that were processed and the exit code of the application. These values are compared with the expected values when the client fetches the output and a warning given if the results are unexpected. Output is stored on a SE and recorded in a file catalogue if the output is too large for the output sandbox or if the user explicitly requested the output be stored at a certain SE. The client can then be used to retrieve the output and/or logs from the SE.

While this method automates many of the steps for users they are still required to submit and monitor their tasks and make their own resubmission decisions. A more efficient method is to delegate the task to the remote ASAP TaskManager.

B. Server

The server consists of an Apache server which is used as a frontend for all client-server interactions, the ASAP server itself which processes the jobs, the ASAP Delegation server which accepts delegated proxies and registers them with the gLite Proxy Renewal service.

All communication between the client and the server is performed over HTTPS using GET, POST and PUT operations with Apache. This solution was chosen as Apache has a proven record of being secure, scalable and performant and when used in combination with gridsite, php and SSL is a very flexible solution. The ASAP client communicates with the MySQL database via Apache using secure XML-RPC requests. Clients can be restricted by Virtual Organisation or by the Distinguished Name of their grid certificate and non-admin users are only able to view tasks they have registered.

Once a task has been created by the client it can be registered and subsequently unregistered at will at any point in the tasks lifecyle. If desired users can use the server for job submission, monitor the jobs manually and then use the server to fetch and analyse the jobs output. The input and output files for each job are transferred using Gridsite's `htcp` command which authenticates using HTTPS then transfers the files using HTTP GET and PUT. Performance is not a real issue here since only comparatively small input and output files can be transferred due to the size of the input and output sandboxes. Larger files that are stored on a SE will not pass through the service, instead they are fetched from the SE when the job begins to run or are retrieved directly by the command line client once the task is unregistered. Users can monitor information such as the current status, exit code and stdout from the website where they can also choose to cancel or resubmit jobs.

The main component of the server is an asynchronous server which selects jobs from the database based on their status. Interacting with the Grid is a major bottleneck with each command taking several seconds to complete due to the security overheads. A task with thousands of jobs can therefore take an excessive amount of time to submit, query or fetch. The original version of the server simply looped over each task belonging to each user which did not scale well. As the problem is completely I/O bound a design which allows some level of concurrency is very desirable. A traditional threaded server and an asynchronous, event based design using the Twisted [12] framework were both prototyped. They offered similar maximum performance but the asynchronous design offered a simpler programming model than co-ordinating multiple threads.

The asynchronous design performs all operations in a single thread of control but makes those operations non-blocking. For our purposes this means that we can perform many operations without waiting for the outcome. As each operation (match,

submit, update, cancel and fetch) is executed a callback is attached which will be executed when the operation is finished. Before jobs are submitted a job-match is performed to ensure that there are available resources in the selected grid. If resources are available the job is submitted, if not the server tries again periodically until resources become available. When submission is completed the Grid JobID that is returned will be inserted into the database by a callback. The main difficulty is ensuring that only a certain number of operations are pending at any one time. If there are too many the machine will become overloaded and timeouts will begin to occur. Too few and performance will suffer. The server is aware of how many jobs are outstanding and is capable of adjusting the rate to avoid problems. If the status is checked or the output is retrieved and the job has not successfully completed the job will be resubmitted a set number of times. If there is a choice between sites to resubmit to ASAP will avoid resubmitting to sites where any job in the task has previously failed.

In order to interact with the Grid on the users behalf ASAP must have access to a copy of their Grid Proxy. To facilitate this before registering a task with the service users must store a long lived copy of their proxy in the MyProxy [13] repository and delegate a copy to the ASAP server. The user executes a command line client which stores the proxy in MyProxy and delegates the users proxy to a server running on the same machine as the ASAP server. The ASAP Delegation server receives this proxy and registers it with the gLite Proxy Renewal service which is responsible for renewing the proxy. This method is secure and ensures that the delegated proxy can then be renewed from the MyProxy service. The whole procedure is compatible with Virtual Organisation Membership Service (VOMS) [14] proxies. When the Delegation Server registers the users proxy it also creates a directory for storing the input and output from a users task. This directory can be accessed via Apache and is proctected by a Grid Access Control List [15] which limits access to the user with a Grid Certificate with a matching Distinguished Name (DN).

C. Agent Model

Jobs within a task differ only by the arguments that are passed to the job wrapper (eg. number of events, input files) and perhaps the site the data is located at. It is therefore very simple for one job within a task to run another given the new arguments. A script is executed on the worker node which controls the execution of the jobs and communicates with the ASAP server via secure XML-RPC calls. The script requests the arguments for a job which has not yet successfully finished. Once the agent starts executing a call is made to change the status of the job so that the another agent or the server no longer selects it. Two threads are created; one where the job executes and another which sends a periodic 'heartbeat' back to the server so that we know that the job is still alive. If the executable finished successfully the output logs are compressed and sent back back to the server for analysis. If the job is not considered a success the status of the job is changed so that another agent can execute it. If a certain time limit is passed the job will be resubmitted in the normal manner. Output from each job is stored on a SE as before. The agent then requests another job from the server for the same task at the same site (in the case where input data is required) and the process is repeated. If no more jobs are available the agent periodically polls the server until it receives a job or a certain time limit is reached. This process is completely transparent to the users who still interact with their jobs in the same manner as before via either the command line client or the monitoring web page.

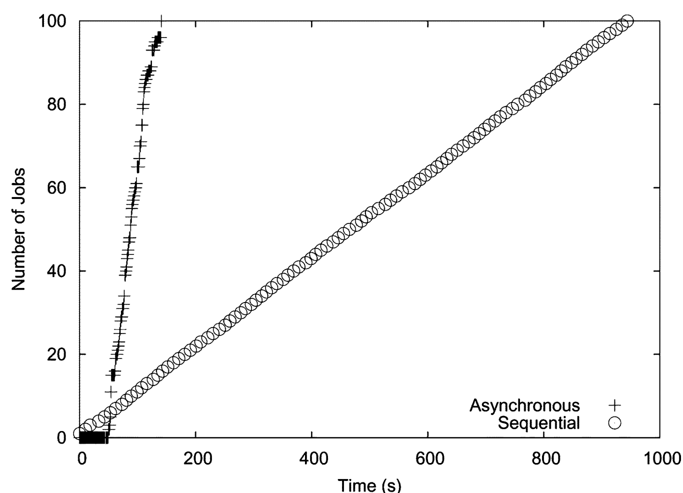

Fig. 3. Comparison of sequential and asynchronous submission.

IV. Performance

As previously mentioned the design of the original ASAP server was a limiting factor on its performance. Jobs were processed serially which introduced a serious bottleneck into the system and limited the overall throughput. As the new server has been designed to overcome these issues it is important to perform some performance tests to establish if these goals have been met.

Fig. 3 compares the maximum rate of submission for inserting 100 jobs in a normal loop and 100 jobs using the new asynchronous server. The jobs are very simple hello world jobs that require no input and have no requirements. Jobs were submitted using the CMS production Resource Broker. The rate of sequential insertion illustrates the problem we are trying to solve. To submit 100 simple jobs takes nearly 944 seconds or 9.4 seconds a job. The ability to submit multiple jobs in parallel is clearly imperative. The asynchronous server is able to insert 100 jobs to the same Resource Broker in 141 seconds.

The main advantage of the agent model is that it reduces the turnaround time for the users task. Tasks are frequently delayed by several jobs which do not begin to run for a disproportionally long time. The agents are able to complete these jobs without waiting for the original jobs to start. Additional savings are made by marking jobs as complete as soon as the output is available. Fig. 4 compares the time from job registration for 100 jobs simlulating 5 events using (a) the original method of job submission and (b) the agent method. In each case jobs are matched then submitted with any failing jobs resubmitted. The average run time is 5913 and 2895 seconds for the original and agent model, respectively. Fig. 4(a) illustrates the problem when a small proportion of the jobs delay the completion of the entire task. That behaviour is eliminated when using the agents.

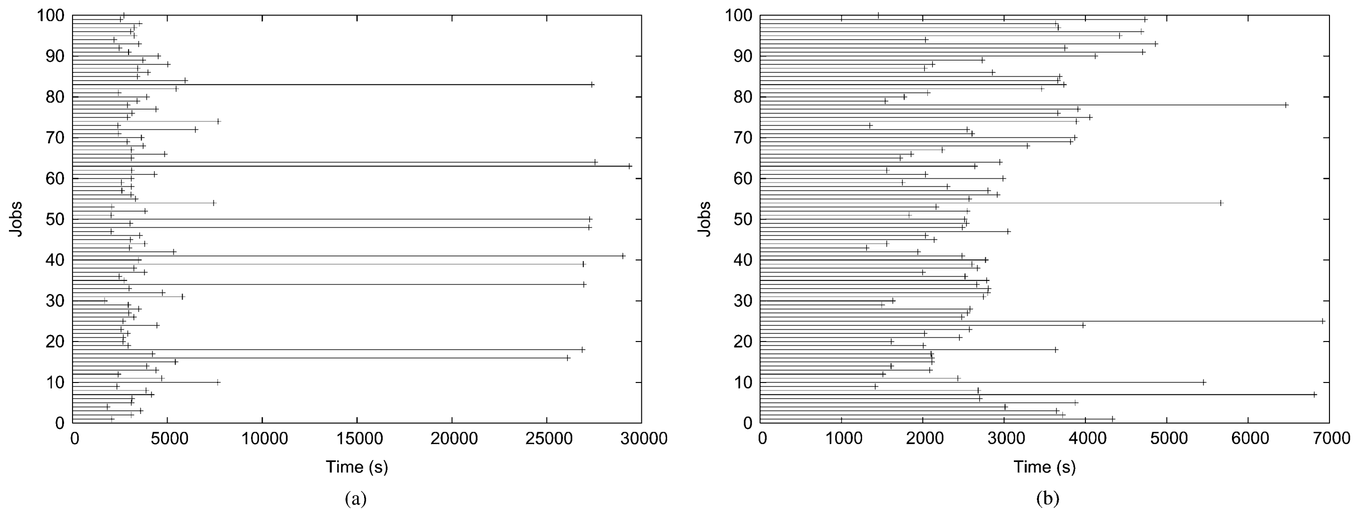

Fig. 4. Time from job registration to completion for (a) Original Model and (b) Agent Model.

V. Conclusion

In order to make distributed analysis a realistic prospect for most users an intermediate layer is required to perform common tasks and insulate users from the complexities and failures that are common when dealing with the Grid. ASAP takes care of job generation and manipulation on the Grid either with or without the users involvement. Ultimately this reduces the effort required from the user to obtain successful output.

The combination of client and server tools has received positive support from users as they are spared the time consuming, error prone steps of creating jobs for the Grid. The ability to delegate tasks to the server decreases the effort the user has to expend and increases the success. The performance tests illustrate that the server is capable of a much higher throughput than the previous model and will be able to sustain higher rates of job submission.

With the infrastructure in place, extending the server to support agents was straightforward. The results of the performance tests illustrate that this method of executing jobs can significantly reduce the turnaround time for users.

Acknowledgment

The authors would like to thank the LCG and EGEE projects for their support and useful discussion, in particular the CERN IT/GD group and the gLite team. In addition, a special thank you is due to the entire IT/PSS/ED section for fruitful collaboration and stimulating discussion. Additionally, the authors would like to thank the users of ASAP for their patience, feedback, and suggestions.

References

[1] LHC [Online]. Available: http://cern.ch/lhc

[2] M. Lamana, "ARDA experience in collaborating with the LHC experiments," in *Proc. Computing in High Energy and Nuclear Physics Conf.*, Mumbai, India, Feb. 2006.

[3] gLite Middleware [Online]. Available: http://cern.ch/glite

[4] LHC Computing Grid [Online]. Available: http://cern.ch/lcg

[5] F. Fanzago, S. Lacaprara, D. Spiga, M. Corvo, A. Fanfani, N. Defilippis, S. Argiro, G. Ciraolo, and N. Smirnov, "CRAB: A tool to enable cms distributed analysis," in *Proc. Computing in High Energy and Nuclear Physics Conf.*, Mumbai, India, Feb. 2006.

[6] K. Harrison, C. L. Tan, D. Liko, A. Maier, J. Moscicki, U. Egede, R. W. L. Jones, A. Soroko, and G. N. Patrick, "Ganga: A grid user interface," in *Proc. Computing in High Energy and Nuclear Physics Conf.*, Mumbai, India, Feb. 2006.

[7] S. Paterson, "DIRAC infrastructure for distributed analysis," in *Proc. Computing in High Energy and Nuclear Physics Conf.*, Mumbai, India, Feb. 2006.

[8] K. Harrison, R. W. L. Jones, D. Liko, and C. L. Tan, "Distributed analysis in the ATLAS experiment," in *Proc. AHM Conf.*, 2006.

[9] A. Afaq, G. Graham, L. Lueking, S. Veseli, and V. Sekhri, "Schema independent application server development paradigm," in *Proc. Computing in High Energy and Nuclear Physics Conf.*, Mumbai, India, Feb. 2006.

[10] A. Fanfani, "Distributed data management in CMS," in *Proc. Computing in High Energy and Nuclear Physics Conf.*, Mumbai, India, Feb. 2006.

[11] I. C. Legrand, H. B. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, M. Toarta, and C. Dobre, "Monalisa: An agent based, dynamic service system to monitor, control and optimize grid based applications," in *Proc. Computing in High Energy and Nuclear Physics Conf.*, 2004.

[12] Twisted [Online]. Available: http://www.twistedmatrix.com

[13] D. Kouril and J. Basney, "A credential renewal service for long-running jobs," in *Proc. 6th IEEE/ACM Int. Workshop Grid Computing*, 2005.

[14] Virtual Organization Membership Service [Online]. Available: http://infnforge.cnaf.infn.it/voms

[15] A. McNab, "Web servers for bulk file transfer and storage," in *Proc. Computing in High Energy and Nuclear Physics Conf.*, Mumbai, India, Feb. 2006.