Distributed Analysis using ASAP

C Munro[1], J Andreeva[2], J Herrala[2], O Kodolova[2] and A Khan[1]

[1] Brunel University, Uxbridge, London UB8 3PH

[2] CERN, CH-1211 Genve 23, Switzerland

E-mail: craig.munro@brunel.ac.uk

Abstract. ASAP is a system for enabling distributed analysis for CMS physicists. It was created with the aim of simplifying the transition from a locally running application to one that is distributed across the Grid. The experience gained in operating the system for the past two years has been used to redevelop a more robust, performant and scalable version. ASAP consists of a client for job creation, control and monitoring and an optional server side component. Once jobs are delegated to the server it will submit, update, fetch or resubmit the job on behalf of the user. ASAP is able to make decisions on the success of the users job and will resubmit if either a grid or application failure is detected. An advanced mode allows running jobs to communicate directly with the server in order to request additional jobs and to set the status of the job directly. These features reduce the turnaround time experienced by the user and increase the likelihood of success.

1. Introduction

ASAP[1] is a system that enables CMS physicists to perform distributed analysis using the Grid. It consists of a client which prepares the jobs and a server which can process the jobs on behalf of the user. ASAP has been used successfully for the past three years. The version that is currently in production has been redesigned using the experience gained from earlier version to provide a modular, cohesive, fully functional framework which can be easily maintained. In operation ASAP interacts with many of the other components of the CMS system as shown in Figure 1.

This paper will discuss the client and server in more depth and introduce the agent mode which can reduce the turnaround time for the users jobs.

2. Client

Before using ASAP, users must already have a working CMSSW application and create a configuration file for ASAP containing parameters such as the dataset to use, how many events to process and where to store the output data. The client then performs all of the steps that are required to execute that application on the Grid. The users libraries are packaged along with any additional input data or files that are required on the Worker Node (WN). The configuration file the user uses for CMSSW is also copied to the WN and altered so that each job will have the correct input data and event range at runtime.

If the input data that the user requires is published ASAP contacts the CMS Dataset Bookkeeping Service (DBS)[2] to obtain metadata about the dataset and the CMS Dataset Location Service (DLS)[3] to obtain the location of each block of the dataset. Jobs are split

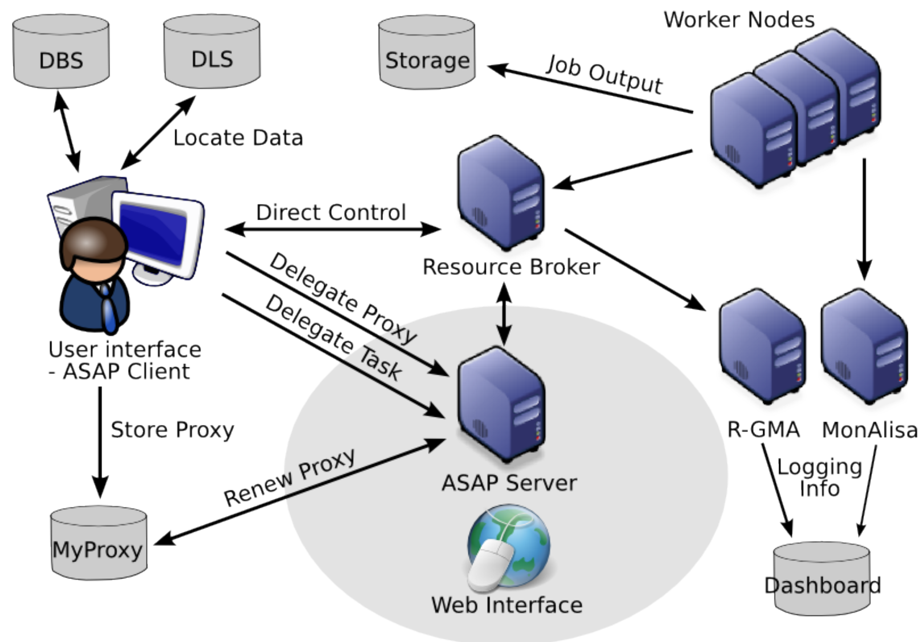

Figure 1. The major components and their interactions in the ASAP distributed system.

across this dataset according to its composition and the requirements of the user. Simulating data, where each job has a different seed, and private data, where data is copied with the job or from a Storage Element (SE), are also supported.

A wrapper script is created which will be executed on the WN to perform middleware discovery, copy input data, create and run the application and store the output. The script also checks for any errors and reports progress and problems to the CMS Dashboard[4].

Once everything has been created the task is stored and the user is given a task ID which can be used to refer to that task. The user can then use the client to interact directly with the Grid by matching, submitting, updating, cancelling or fetching output from the Grid. Operations can be performed on the entire task or on a subset.

However, interacting directly with the Grid is a time-consuming process. Instead, most users choose to delegate responsibility to the ASAP server which can then process and monitor jobs on the users behalf.

3. Server

The ASAP server consists of four loosely coupled components: the interface, the delegation service, the server itself and the database. For scalability each component can be located on a separate machine provided there is a shared filesystem between them.

An Apache server is used as the interface for all client interactions due to its flexibility, performance and security. There is an XML-RPC interface for client-server interaction and web pages for additional user and administrator interaction. For example, users can inspect the logs from completed jobs via their browser and then choose to resubmit them. All communication between client and server is authenticated using the users Grid certificate. Client-server operations are performed in bulk where possible to improve performance where large numbers of jobs are being manipulated. The interface interacts with a MySQL database to register, unregister and update jobs. Input and output files (that are not stored on a SE) are transferred using HTTPS PUT and GET using the Gridsite[5] Apache module. Transfer directories are protected by Grid Access Control Lists so only the owner of each task (or an administrator) can

read or write a directory.

In order to operate on the users behalf the server requires a copy of the users proxy. This is acquired through the delegation service, a SOAP server which implements the Delegation Interface. Before registering tasks with the server the client delegates a copy of the users proxy to a MyProxy service and another to the delegation service. The delegation service then registers the users proxy with the gLite Proxy Renewal Service which runs on the same machine. The proxy renewal service can then contact MyProxy to ensure that the server maintains a valid copy of the users proxy. VOMS proxies are supported by each component.

The main component is the ASAP server itself. Interacting with the Grid is a major bottleneck with each command taking several seconds, some level of concurrency is therefore very desirable. The server is implemented in Python using the Twisted[6] library. The server selects jobs from the database and launches the appropriate command for each job. Twisted does not block on commands, instead it adds a callback which is executed when the command is completed. This callback can then process the output of the command and take any further action that is necessary. Processing multiple commands in this way provided a simple, performant method of interacting with the Grid.

If the server detects a failure in a job that is submitted, either in the Grid itself or in the application it can perform resubmission a configurable number of times. If the error is terminal then the job will not be resubmitted any more. If it is due to a problem at a certain site then that site will be avoided the next time the job is resubmitted.

4. Agents

Each job within a task only differs from another by the arguments that are given to the wrapper on the WN. For example, input files and number of events. It is therefore very simple for one job within a task to run another given the new arguments. A script is executed on the WN which controls the execution of the jobs and communicates with the ASAP server via secure XML-RPC calls. The script requests the arguments for any job in the task 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 without contact from the Agent the job will be resubmitted in the normal manner. Output data from each job is stored on a SE as before. The agent can then request 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 user who still interacts with their jobs in the same manner as before via either the command line client or the monitoring web page.

Figure 2 compares the turnaround time for a users task using the traditional 'push' model and the new 'pull' model. The main advantage of the pull 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. Figure 2 compares the time from job registration for 100 jobs simulating 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. Figure 2(a) illustrates the problem when a small proportion of the jobs delay the completion of the entire task. That

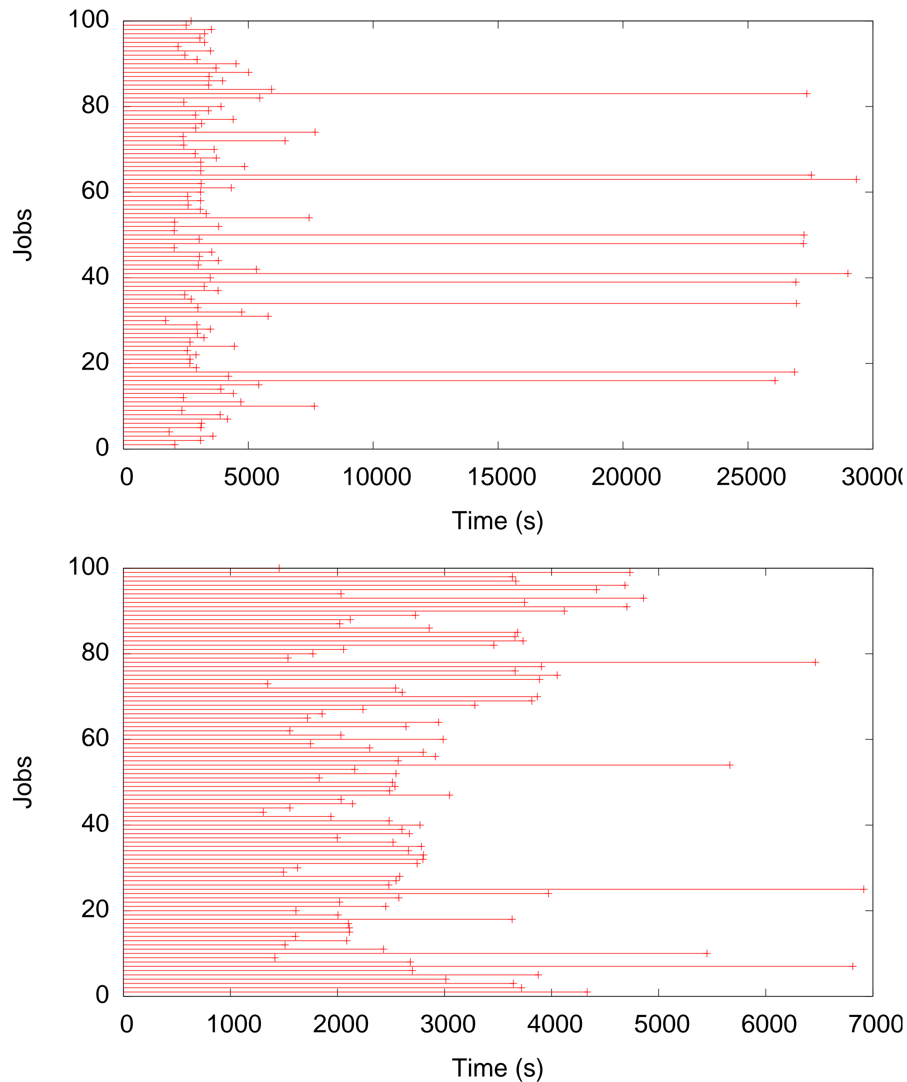

Figure 2. Figure caption for first of two sided figures.

behaviour is eliminated when using the agents.

5. Conclusion

ASAP has been successfully used by CMS physicists for several years. The client allows users to package their applications, locate data and create jobs. Users can choose to use the client to interact directly with the Grid or to delegate responsibility for their tasks to the ASAP server. The server can process and monitor the users jobs and perform resubmissions in the case of failure. The agent mode can significantly reduce the turnaround time for the user task by avoiding the overheads that are traditionally present in the Grid.

Acknowledgments

Thanks to everyone who has used ASAP or contributed support or ideas. Craig Munro would also like to thank the Science and Technologies Research Council for his funding.

References

[1] J. Andreeva, C. Cirstoiu, J. Herrala, M. Lamanna, T.S. Chen, S.C. Chiu, A. Berejnoi, O. Kodolova, and C. Munro. CMS/ARDA Activity within the CMS Distributed System. In *Proceedings of Computing in High Energy Physics (CHEP 2006)*, 2006.

[2] L. Lueking. The CMS Dataset Bookkeeping Service. In *Proceedings of Computing in High Energy Physics (CHEP 2007)*, 2007.

[3] A. Fanfani. Distributed Data Management in CMS. In *Proceedings of Computing in High Energy Physics (CHEP 2006)*, 2006.

[4] J. Andreeva. Grid Monitoring from the VO/User perspective. Dashboard for the LHC experiments. In *Proceedings of Computing in High Energy Physics (CHEP 2007)*, 2007.

[5] A. McNab. The gridsite security architecture. In *Proceedings of Computing in High Energy Physics (CHEP 2007)*, 2007.

[6] Twisted. http://www.twistedmatrix.com.