Monitoring the efficiency of user jobs

James Casey, Daniel Rodrigues, Ulrich Schwickerath, Ricardo Silva
CERN IT, 1211 Geneve 23, Switzerland
E-mail:
James.Casey@cern.ch,Daniel.Rodrigues@cern.ch,Ulrich.Schwickerath@cern.ch,Ricardo.Silva@cern.ch

Abstract. Instrumentation of jobs throughout its life-cycle is not obvious, as they are quite independent after being submitted, crossing multiple environments and locations until landing on a worker node. In order to measure correctly the resources used at each step, and to compare it with the view from a Fabric Infrastructure, a solution is proposed using Messaging System for the Grids (MSG) for integrating information coming from different sources.

1. Introduction

Due to the limited available computing resources at computer centers both resource providers as well as the users benefit from an optimal use of these resources. While users are typically interested in a high job throughput in order to get their results in time, resource providers want to keep their CPUs busy. Some user applications are I/O bound, and naturally do not consume a lot of CPU time, for example because they need to wait for file recalls from tape[1]. Resource providers can attempt to improve the usage of such boxes by artificially adding CPU bound jobs which will run while the tape recall is taking place. This approach has recently been implemented at CERN [1].

Often job inefficiencies have different reasons though, like undetected bugs or inefficiently written code, or human errors. These kinds of problems can be difficult to detect. Resource providers can monitor the job efficiencies from the batch system accounting files, or implement an idle job detection for running jobs to warn users. Some experiments have implemented their own tools to monitor the efficiency of their software as well. In general it is difficult to match the view of the resource providers with the view of the experiments, while this would be necessary to identify sources of inefficiencies properly. This is specifically true for pilot job frameworks, which involve a large amount of job wrappers.

In this paper a method is proposed which can help to get around these limitations. Moreover, it is made to be unique across experiments, and portable across sites.

2. Messaging system for Grid, MSG

This method requires a system to send job state information from jobs as they executed on the computing resources. We have chosen to use message-oriented middleware [2], in particular MSG, an implementation used with the EGEE [3] Grid project as an integration platform for

[1] Within this paper we treat jobs waiting for tape recalls as I/O bound jobs. These jobs are effectively sleeping while other jobs in this category may be busy doing I/O operations locally or over the network.

operational tools [4]. MSG was chosen because it allows for both point-to-point transmission of messages(Queues) and broadcast of messages to multiple consumers (Topics).

Messages are sent by a publisher, received by a consumer, and routed between these by a broker. The routing of messages is made at a logical level rather than physical addressing: By using queues and topics, which are semantic implementations of point-to-point and publish-subscribe communication, we are able to decouple sender and receiver components. As a consequence, users of MSG achieve a great flexibility, mostly for dealing with event-driven systems and to cope with software distributed across many locations.

MSG uses a standard open-source messaging middleware implementation, Apache ActiveMQ, and defines standards for the format of the messages and protocols that clients of MSG should use to produce and consume messages. Different language bindings exist for the clients. Java, C++, Python and perl are supported.

Two client implementations are provided by MSG at the time of conference that allow us to simply produce and consume messages:

- msg-consume2oracle allows to put the data send in messages into an Oracle database. Its been used for the pilot described in this document.
- msg-simple-publish is a script used to send messages. As described below, it is used by a wrapper script called msg-tag to send o messages into the system.

MSG also supports different wire protocols, specifically Streaming Text Orientated Messaging (STOMP), Openwire(JMS), and HTTP, and provides reliable delivery of messages. One or more fail-over brokers ensure high availability of the service. Fig. 1 shows an overview over the architecture.

3. MSG for job monitoring

MSG can serve as a framework to implement a job instrumentation mechanism in which specifically prepared messages are sent at different job stages. The batch system view of user jobs can be reported asynchronously after the end of the jobs, in a bulk operation containing one message per job. This message should be as small as possible to reduce the stored data, and contain enough information to be able to match it with the experiment provided data and extract useful information. Experiments, on the other hand, should instrument their jobs to send messages at reasonable stages, containing enough information so that they can be uniquely matched with the messages send by the resource providers.

A prototype has been implemented which is in place and usable on all public batch nodes at CERN.

3.1. batch system view

At CERN, a bulk upload of messages to MSG is done once per night, containing one message per job run during that day. The information contained in these messages is described in table 1. A default consumer has been put in place which consumes the received messages, and puts them into an Oracle database.

The received data in the database can then be used for data mining. As an example, fig. 2 histograms the job efficiency[2] for all user jobs seen at the CERN batch farm, for a period of about 3 months. A large spike can be seen at around zero. This spike contains jobs which failed, for example because of user errors, but also very inefficient jobs. It also contains pilot jobs which arrive on the worker nodes and exit because there is no payload to be executed. From this figure it is not possible to resolve this peak any further. Jobs need to be categorized, so

[2] Within this paper, a job efficiency is defined by the ratio of the CPU time used and the wall clock time. All applications are assumed to be single threaded.

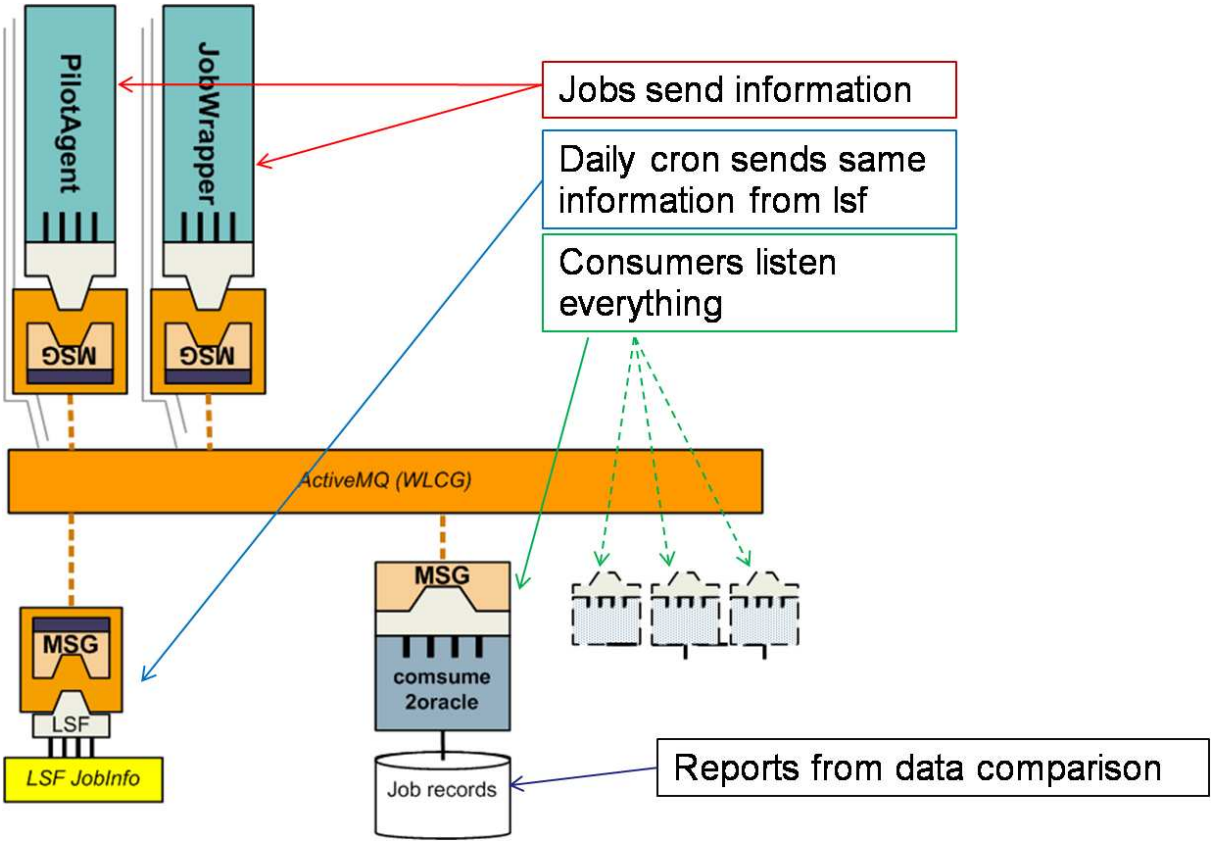

Figure 1. MSG architecture.

field name	description
context	identifier, set to LSOF for LSF
state	state of the job, set to JOBEND
localjobid	the local job ID
ownerdn	job owner, the local user name in this case
wnhostname	the worker node name
cpufactor	CPU factor of the worker node, for normalization
cpuUsage	used CPU time in seconds (not normalized)
walltime	Wall clock time for the job (not normalized)
memoryUsage	memory usage seen by the batch system
exitcode	exit status of the job, zero if successful
submittime	submit time in UNIX time
finishtime:	finish time in UNIX time

Table 1. Structure of messages sent in a bulk operation once per night for CERN.

that it is possible to quantize the number of well behaving jobs, and improve on the rest. Such a categorization would also allow to find out the reason for the second peak of jobs with low efficiencies. This peak may be due to I/O bound jobs. Such an ear-marking of jobs can only be done by the users themselves. It cannot be provided by the service providers.

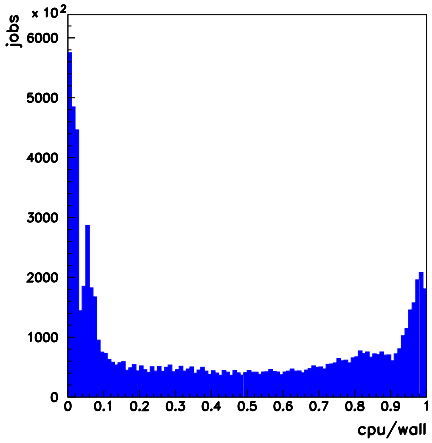

Figure 2. Job efficiencies for all users, as seen by the batch system at the CERN batch farm. The large spike on the left contains also failed and exited jobs.

argument	type	description
context	string	string identifying the type of activity
state	string	state of the job
cputime	integer	(raw) CPU time in seconds used so far
walltime	integer	(optional) run time in seconds so far
vojobid	string	(optional) VO internal job identifier
vosite	string	(optional) site name in VO language
memoryusage	integer	(optional)peak memory usage at this stage
cpufactor	number	(optional) CPU factor (per CPU core for this WN

Table 2. Available arguments for msg-tag. At CERN, the CPU factor is filled automatically

3.2. msg-tag

While the piece of code sending the asynchronous messages to the MSG system containing the batch system view can be batch system specific, site specific settings have to be hidden from the users whereever possible. For this reason, a wrapper has been implemented for the prototype which hides site specific settings from the users and experiments. Experiments can use a wrapper to upload messages to the system, which hides all site specific settings from them. However, for technical reasons they need to keep track of the CPU consumption of their payload processes themselves, and provide this number as an argument to *msg-tag*.

Note that the context and the cpu time numbers have to be provided. *msg-tag* will automatically generate an MSG message from the provided data, and upload it to the system. The user does not need to know where he is running or where the MSG servers are. Executing *msg-tag* with the required information as arguments is enough.

4. Sample instrumentation: DELPHI experiment code

As a real world proof of concept, DELPHI [5] experiment code has been instrumented with the proposed mechanism, for two typical use cases which in principle also exist at LHC: MonteCarlo production, and a typical data analysis job [6].

4.1. Classifying jobs

Individual users tag their jobs using the context field in the messages. *msg-tag* adds enough information so that these additional messages can be matched with the nightly messages sent from the batch system accounting records. Doing this, efficiency and timing figures can be

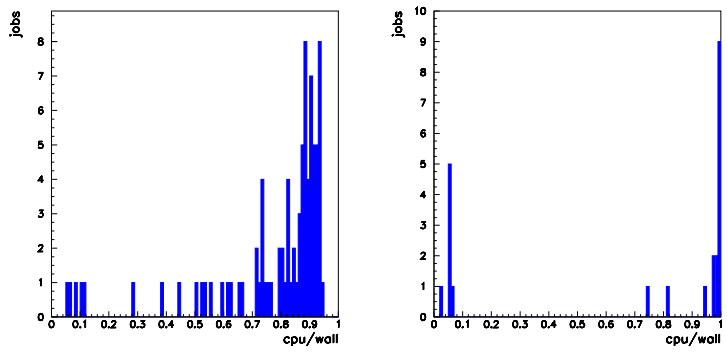

Figure 3. Job efficiencies for a single user who is running two different classes of jobs: A production and an analysis use case. The user earmarked his jobs so that the jobs can be separated and matches with the batch system view.

extracted from the Oracle database. This way, in fig. 3 the efficiencies for the two different use cases are shown. The analysis code tries to pre-stage data which it has to read back from tape. This way, in the majority of cases the data is available on disk when the job starts up on the worker node. It reads over the file sequentially, and analyzes its contents event by event. This results in a distribution as seen on the left hand side of fig 3, with a peak above 80% and a tail towards lower efficiencies.

The MonteCarlo production jobs are expected to be CPU intensive, and do little I/O. As expected, the right hand side plot of fig 3 shows a distribution with large peak at high efficiencies. Some jobs went wrong due to user errors. These end up at the lower end as they immediately exited without using CPU time.

4.2. Job instrumentation

The analysis use case is certainly more interesting to look into than the MonteCarlo production which is very efficient. The sample user sent several messages during the live time of his jobs, at different stages of the job, making use of the state field. In this case however, the CPU usage measurements at each step have to be done by the user himself.

Fig. 4 histograms the step efficiency as a function of the time spent in the step, for each job of the analysis class. Shown are the compilation, the linking, the running and the storing step. As expected, the bulk of the CPU time is spend in the running phase, which tends to be highly efficient. Two jobs show low efficiency, possibly because the input data was not available on disk when the job started up.

5. Summary

The MSG framework has been exploited for job instrumentation. A sample implementation is in place at CERN, and has been tested with real physics application code. At CERN, messages are sent containing information from the batch system perspective, which can be used by the experiments for efficiency considerations of their jobs. The flexible design of MSG allows the experiments to write their own message consumers, and use the provided information in their own existing monitoring tools. A unification of the context field in the messages across experiments for the same (or similar) use cases would also allow for a comparison between experiments. All LHC experiments are invited to use the framework.

6. References

[1] Batch efficiency, *Dr. Ulrich Schwickerath, Ricardo Silva, Christian Uria*, Proceedings CHEP 2009, to be published

[2] Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, *Hohpe G., Wolff B*, Addison-Wesley, 2003.

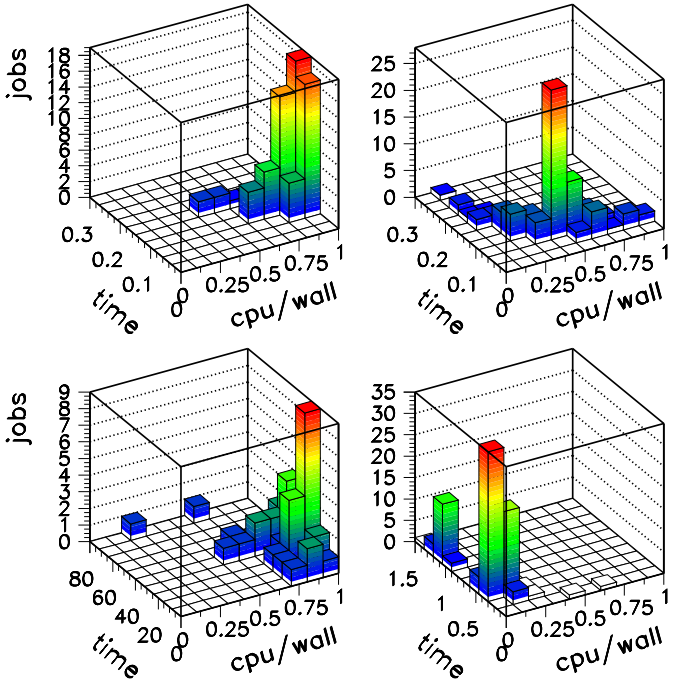

Figure 4. A detailed instrumentation of the analysis job allows to monitor the different steps inside a single job. The plots show the CPU/Wall time ratio as a function of the time spent in the following steps: compiling (upper,left), linking (upper right), running (lower left) and storing of the results (lower right).

[3] *http://www.eu-egee.org/*
[4] EGEE-III Operations Automation Strategy. *Casey J., et al.* https://edms.cern.ch/document/927171
[5] The DELPHI experiment at LEP, *http://delphiwww.cern.ch*
[6] Final results from DELPHI on the searches for SM and MSSM Neutral Higgs bosons, *DELPHI collaboration Eur. Phys. J.* C**32 (2004)** 145-183