Requirements for Job Monitoring

REX (Contact Adam Lyon for comments)
22 September 2010

In the course of discovering requirements for Grid Monitoring, it is important to think about what users and operators do when they run jobs or monitor the system. Let's do this without worrying about implementation, or even if an implementation is possible.

1 Use cases for Users:

Before a user submits a set of jobs to the system, there are several things they want to discover:

- Whether the system as a whole is operational and working well.
- What types of resources are available on the batch/cluster system.
- Whether these resources will match their job's needs [though in principle this should be handled automatically by the submission system].
- How available those resources are, as measured by the time before the user's jobs will be run.

The answers to these questions go into the decision regarding whether to submit jobs to a particular system or to search for a more appropriate resource (e.g. if the users jobs require large memory footprints, but the proposed slots in a queue have available system memory to close to this limit to allow the jobs to run efficiently, then it may be advantageous to seek out a "high memory" queue rather than submit to the current one.) Whether to submit their jobs at that time or defer submission to a time when the system is in fully operational state (e.g. if the system shows signs that it is clearly broken, or has a large number of failed resources, the user may view the risk of their jobs failing as a greater liability than delaying submission.) How many jobs or which jobs to submit (e.g. if the system is extremely busy, the user may choose to submit only a small fraction of their jobs or their highest priority jobs and save the bulk of their submissions for a time when the system is not as heavily loaded.)

Once the jobs are submitted to the system, the user requires information corresponding the state of their jobs and specific events related to the run cycle of their jobs, as well as periodic updates that allow them to track their progress. This information includes:

- The system "state" of their job.
  For batch/grid systems this can correspond to states such as:
  - Queued, Running, Completed

  or errors states such as:
  - Failed, Unable to start, Zombied, Mysteriously disappeared
- What their job's position is in the queue or fairshare, and what is the estimated time that the job will start

- Once jobs begin running, the user needs information that allows them to determine if they are progressing normally, or if they are not, the reason why. (Are they stuck waiting for data? Are they churning the CPU without advancing to the next file in a timely manner?)
- When jobs stop running, the users needs to know the manner by which they finished. Did jobs that recently completed do so successfully, or did a failure cause them to exit? Was the failure a system one, or one that was caused by their jobs (e.g. did a bug in their own code cause their job to seg fault out, or did the job violate site rules, or did the system/grid/queue crash?) Did the output data get stored back? How long did the jobs run for? How much data did they read? Write?
- Users may also want summary information of the jobs they submitted together (e.g. a transcript of the group submission made available via e-mailed to them when the set of jobs complete).

Furthermore, it is also useful for users to look at other people's jobs:

- Are jobs at this site working or having problems like mine?
- Is a different user hogging resources at a site?
- Is a different experiment hogging resources at a site?

2 Use cases for Operators:

Here, operators are *not* site-admins nor system-admins. Rather they are personnel who monitor and troubleshoot aggregate data handling and job handling systems for the experiments.
The basic job of an operator is to determine whether or not a system is working properly and troubleshoot if not. Furthermore, operations management may also like to know the usage level of a system by the experiments (such people are included here under "operators"). A list of things that operators need to use monitoring to discover are,

- Is the system working? Are jobs running successfully? How many are stalled? Is data flowing to jobs in a timely manner? For certain jobs that have problems, it may be important to view system information for the job's machine (perhaps virtual machine); e.g. load average, iowait, ...
- If jobs are failing.... How many jobs (users) are impacted? What is the distribution of errors (is there one error everyone is getting or are there lots of different errors). Are the errors happening at one site or multiple sites? Are errors occurring at the same time?
- If data handling is failing... [Same questions as above]
- Are resources being utilized effectively? Are lots of slots free? Are lots of slots running but using zero cpu?
- Are some users hogging resources?
- What is the distribution of resource utilization by experiment?
- Operations will also want to see much of this information historically.

3 Requirements

Below are requirements for a job monitoring system, given the use cases above.

3.1 Overall system monitoring

Overall system monitoring is extremely important for operators, but also useful for users to get a sense of the state of the systems.

For lots of this information, a viewer of the monitoring may want to sort, break up by a factor (user, experiment, site), or aggregate. Such options are explained below.

Information very specific to a site. A viewer may want to sum the information over all sites for a total. It would also be useful to have historical information (viewable as a strip chart). A few day's worth of data should be sufficient for history.

- How many slots are available to the experiment (this may be impossible to determine for sites used opportunistically)? [Site specific attribute]
- How many slots are available with certain attributes (e.g. long running, high memory, fast CPU, ...)? [Site specific attribute]
- Published downtime/problem information from the site

Information about jobs. This information should be viewable in a very flexible way. First, let's indicate the information that should be available:

- What jobs are running?
- What jobs are queued?
- What jobs are "stuck" (jobs in a running state, but taking no CPU time)?
- What jobs have low efficiency (jobs are have taking long wall clock time but little CPU time - perhaps make the threshold for low efficiency settable)
- What jobs have started within a given time interval (3hrs, 6 hrs, 24 hrs, 72 hrs, custom)?
- What jobs have finished within a given time interval?
    - What are the exit/problem codes of those jobs?
    - What is the efficiency of those jobs?

The display options should be extremely flexible. We imagine displaying in both table form and historical strip chart form.

Requirements on the table:

- When desired, individual job identifiers should be listed (e.g. a list of the jobs that are running for site A by user B). In such lists, the jobs should be somehow "clickable" to bring up a display of detail information about that job.
- When desired, sums of jobs should be displayed instead of individual job identifiers (e.g. display the number of jobs queued for experiment A summed over all sites).
- Information can be restricted by factors (factors are site, experiment, job type [SAM, MC production, ...], group [defined by the experiment?], user, and the bundle of jobs submitted together by a particular user). E.g. Display only the running jobs from user A on site B;
- Information can be categorized by factor. For example, display a table of sums of the information categorized by site (break out the sites in sections of the table). For example, display sums of the information categorized by site and user (so sums are

listed by site + user).
- The viewer of the monitoring should be able to sort the contents of the table in many ways. e.g. Have a table with column headings of Site, User, # of stuck jobs. Sorting by Site sorts the table by site name. Then do a sub sort by the # of stuck jobs. That way it will be easily apparent the user that has the most stuck jobs per site.
- The configuration of the table should be savable by the viewer for easy recall.

Requirements on the strip-chart:
- Detailed history should be kept for some reasonable amount of time (a week?)
- Charts should be drawn given factors (e.g. show history of # of stuck jobs for a site, for a user, ...)
- Information should be overlay-able on a chart. For example, show history of job starts for different sites super-imposed on the same chart for easy comparison.

Some charts and tables should be the default on the main monitoring page so that a quick glance would indicate if the system is running well or is having a problem.

3.2 Overall data handling monitoring

Information here may be shown along with the job information above, as they are many times tightly coupled.

The following is required both as tables and historical strip charts:
- Tape requests (by robot, and, perhaps, by tape)
- File requests (source, destination, size, file type)
- Deliveries (time of deliveries, source, destination, size, file type, duration)
- Errors (type, site)

3.3 Job level monitoring

Here, we assume the job runs within a VM (for the sites where jobs run on the bare metal, then quantities below will be for the whole machine).

In association with item 3.1 above, there needs to be a way to select job(s) to view. Users should be able to select jobs by looking at the list of jobs they submitted to the system, subdivided by site and state. Furthermore, operators should be able to select jobs by listing jobs that match criteria (e.g. jobs with low efficiency that have completed today; jobs from user A running on site B that have been stuck for more than n minutes; jobs on machines with high iowait).

For each job selected to view...
1. List basic information (job owner, job ID(s), site, state [queued, running, stuck, completed], time in that state
2. If the job is complete, then the completion information (exit status, error messages)
3. A log of the various state transitions (e.g. when did the job go from queued to running? How long was it in each state?) The log should also indicate how long the job was waiting for each file and busy working on each file

4. Log of files that were read in (name, when, size, source)
5. Log of files that were written out (name, when, size, destination)
6. If waiting on a file, how long has it been waiting
7. If working on a file, what file is being read (name, size, source)
8. VM information (load average, CPU time, iowait, memory usage, disk space) as a history plot over the life of the job
9. Indication of the job efficiency
10. Access to log files the job is producing while it is running

The monitoring system should also alert users (via e-mail or some other communication mechanism) when a set of jobs have completed (e.g. jobs from the same submission). The message should contain basic statistics on the job (e.g. exit status, efficiency, ...).

3.4 Access requirements

The information should be presented on web pages accessible from outside of the laboratory. Though use of VPN may be required by FNAL security rules.

4 Current examples

There are two examples of monitoring that come to mind.

4.1 CERN Dashboard

Job and file transfer monitoring for CERN experiments live at http://dashboard.cern.ch/ .
CMS seems to offer a rich set of monitoring http://dashboard.cern.ch/cms/index.html.
The interactive and historical job monitoring is especially relevant to this document.

4.2 CDF CAF Monitoring

CDF CAF monitoring at http://cdfcaf.fnal.gov/caf/cdfgrid/ is an example of monitoring that is especially comprehensive and easy to view.