Centro Euro-Mediterraneo per i Cambiamenti Climatici

Research Papers
Issue 2011
December 2011

Scientific Computing and Operation (SCO)

Green Computing: Design of a reactive monitoring solution for power saving

By **Osvaldo Marra**
CMCC
University of Salento, Italy
osvaldo.marra@unisalento.it

Maria Mirto
CMCC
maria.mirto@cmcc.it

Massimo Cafaro
CMCC
University of Salento, Italy
massimo.cafaro@unisalento.it

and **Giovanni Aloisio**
CMCC
University of Salento, Italy
giovanni.aloisio@unisalento.it

SUMMARY The report describes the design of a system that can monitor the load on the Calypso IBM Power 6 cluster and act on different nodes turning them on and off according to their load (in terms of jobs), scheduled by LSF. The system is able to collect and analyze information about the cluster (queues, jobs, and hosts) in order to decide its course of action. After data collection and analysis, the system can take one of the following actions: 1) turn off n nodes, 2) turn on n nodes, and 3) do nothing. The system then waits for a time interval defined by the user before checking again the current state. The entire logic consists, then, in a series of steps that are repeated periodically at regular intervals of time in a loop.

INTRODUCTION

The fast hardware evolution in recent decades has led to processors increasingly smaller and faster, with a strong increase in power dissipation for the calculation: while a 486 dissipated about 10 watts, the Pentium IV dissipated 120 watts, with an energy consumption increased by an order of magnitude. To get an idea of the energy consumed by IT systems it is sufficient to consider that modern blade servers consume about 1kW (as much as a home air conditioner running at full power). Consequently, a rack of blade servers, for example, consisting of 5 shelves with 8 units each, consumes 40 kW, the equivalent of a building. A medium-sized data center consumes about 250 kW, as a district, while a large data center, can consume even more than 10 MW, the equivalent of a small town. In modern data centers, one of the strategic objectives is to reduce energy consumption, by means of Green Computing. We have designed, built and tested a software that can monitor the load of the IBM Calypso cluster, and act on different nodes turning them on and off according to their load (in terms of jobs), scheduled by LSF. Our aim, if possible, is trying to reduce the cluster energy consumption, which amounts to about 2000 euros per day. Moreover, turning off the unallocated nodes also implies a lowering of the heat produced with consequent cost savings with regard to the cooling system. The architecture on which this work has been done is the Calypso IBM cluster, consisting of 30 IBM P575 nodes each of which is equipped with 16 dual cores Power 6 processors operating at a frequency of 4.7 GHz. Therefore, the number of physical cores amounts to 2 * 16 * 30 = 960 cores, capable of providing a computing power of about 18 TFLOPS. All this computing power is combined with a capacity of data storage of about 1.5 petabytes - or 1536 terabytes.

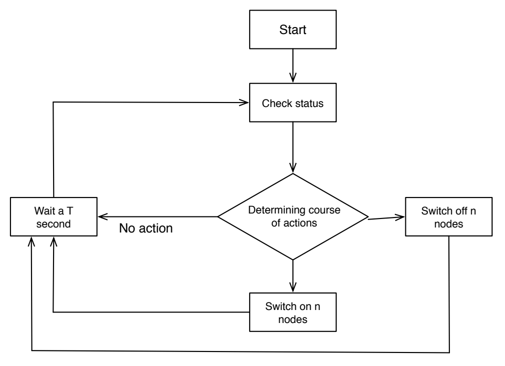

Figure 1:
Flow chart of the system

A REACTIVE MONITORING ALGORITHM

Analyzing the architecture of the IBM cluster and the local LSF scheduler, we identified the data to be monitored and the APIs required to extract this information. The design phase has produced a series of flow charts, providing an overview of the logic of the proposed algorithm. The goal of the algorithm is to act on the nodes (turning them on and off) as required, to save energy. The algorithm's logic to achieve this goal is summarized by the flow chart in Figure 1.

Checking the current state represents the first logical block, covering the collection and analysis of information to be used later to decide the course of action. The elements to be analyzed, in order, are:

1. Queues;
2. Jobs;
3. Hosts.

After collecting and analyzing the required information, the system can react with the following actions:

- Turn off n nodes;
- Turn on n nodes;
- Do nothing.

The system then waits for a time interval defined by the user before checking again the current state. The entire logic consists, then, in a series of steps that are repeated periodically at regular intervals of time in a loop. We have described how the algorithm works at an high level of abstraction; in the following we will provide details related to the different blocks of the flow chart, which will make clear the algorithm's internals. We begin by delving into the details of how the cluster's state is checked.

STATUS CHECK

To recover the required information we start analyzing the system-level queues available to LSF, listed below:

- poe_sys;
- poe_tests;
- poe_short;
- poe_medium;
- poe_long;
- poe_serial;
- poe_serial_highmem.

Each of these is characterized by different properties, and we collect and store the most relevant information. Indeed, only few data are essential for the subsequent phases in which a decision should be taken:

- number of pending job slots;
- maximum run-time set;
- list of available resources (hosts assigned to a queue);
- list of users authorized to run jobs on the queue.

After the queues we analyze job information collecting and storing the following data. For each job running:

- job ID;
- username of the user who submitted the job;
- number of job slots allocated;
- list of the hosts allocated;
- start time of the job;
- the queue on which it is running.

For each pending job:

- job ID;
- username of the user who submitted the job;
- number of required job slots;
- submission options (ptile = ...,-X - exclusive mode);
- resources (nodes and cores) requirements;
- the queue on which it has been submitted.

The last information required is related to the hosts. At first we must fetch the list of available hosts. Then, each host must be analyzed in order to retrieve this information: status of the host (available using the APIs or running from the command line bhosts-w):

- Not available (unavailable as a node is powered off or not reachable);
- Closed_Adm (the administrator closed it for maintenance);
- Closed_Excl (closed with a run in exclusive mode);
- Closed_full (closed because all of the cores have been allocated);
- No license for LSF;
- Operating with some or all of the resources available (status = ok);
- number of job slots available;
- number of job slots running.

The step in which the algorithm performs a check of the current state provides an instantaneous snapshot overview of the system through the data characterizing the cluster operation, that are stored for use in the next step (when the algorithm decides what to do next). Figure 2 represents a flow chart synthesizing the previous discussion.

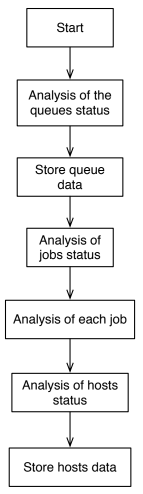

Figure 2:
Flow chart of the Status Check phase

ACTIONS BASED ON COLLECTED DATA

When a job is submitted through the *bsub* command, LSF checks the availability of resources by comparing them with the user's request and, if resources are available, it allocates on each host a number of job slots (cores) whose sum equals the total number of cores requested; it then actually start the job. Often, the number of required resources exceed those available and a job becomes *pending*. Other times the opposite situation occurs, the number of available resources exceed those required and some nodes are turned on but are not running any job. In both cases is required a software's intervention, taking into account the management policies, which could turn on (in the first case) or off (in the second case) cluster nodes. The action to be taken is determined by the data related to queues, jobs, and hosts previously collected. Among those affecting the queues is the number of pending job slots that can be 0 or greater than 0. We analyze both cases.

CASE 1: PEND = 0

In this case, the first thing to do is to check the status information (retrieved earlier) of individual nodes (hosts) to determine which ones have jobs running and which are turned on (online) but idle owing to job slots not allocated. The latter nodes, are candidates for shutdown (Figure 3) if these are not service nodes. For each node which is candidate to shutdown, we check that it persists in this state for more than half an hour (as required by the Calypso sys admin). If yes, we can then proceed to switch off the node.

We select a job from the list of pending jobs, and

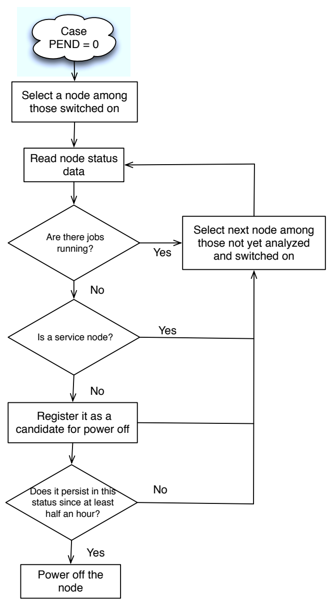
Case
PEND = 0
Select a node among those switched on
Read node status data
Are there jobs running?
Yes
Select next node among those not yet analyzed and switched on
No
Is a service node?
Yes
No
Register it as a candidate for power off
Does it persist in this status since at least half an hour?
No
Yes
Power off the node

Figure 3:
Case PEND = 0 flowchart

determine how many nodes are required for its execution. The information required includes:

1. Total number of cores required by the pending job;
2. Number of cores available on the cluster nodes currently turned on;
3. The command line options of the *bsub* command used to submit the job:
   - -X: This is a request for exclusive execution, i.e., no other job can be running on the same cores utilized by the job;
   - -R-span [ptile = k]: The execution requires, mandatorily, k cores per host (node); For example: -n 10 -R span[ptile=2] /home/user/application requires execution on 5 different hosts, using 2 cores per host.
4. Number of jobs that are running, which are deemed to complete their execution "at most" in an hour.

The number of jobs that are running, which are deemed to complete their execution "at most" in an hour, is very important, being part of the policies adopted to compute the number of nodes to be turned on. Each queue has a runtime limit parameter that represents the maximum time a job can be running on the queue; once the runtime limit has elapsed for a job, if it is still running LSF kills it abruptly. This information is used to determine the number of nodes to be switched on, since we take into account the cores that will soon be available. The analysis continues with the next pending job in the list. If the list is empty, we have collected all of the required data.

CASE 2: PEND > 0

Described in Figure 4.

We have already said that idle nodes are not immediately turned off; instead, they are simply candidates to shutdown and actually turned off if they persist being idle for at least half an hour. Here is why. Suppose that, having collected and processed the data, our algorithm decides to turn off 3 nodes which are immediately shut down. After a few minutes, a user submits a job that can not be immediately scheduled because the number of cores requested is not available, so that the job state is *pending*. During the next status check the algorithm analyzes the state of the job and (if possible) it turns on a certain number of nodes to allow the job transition to the *running* state. After the job execution, if there are no other jobs requiring the resources of the nodes that have been turned on, these nodes are again turned off. The scenario described could occur several times resulting in continuous alternating phases

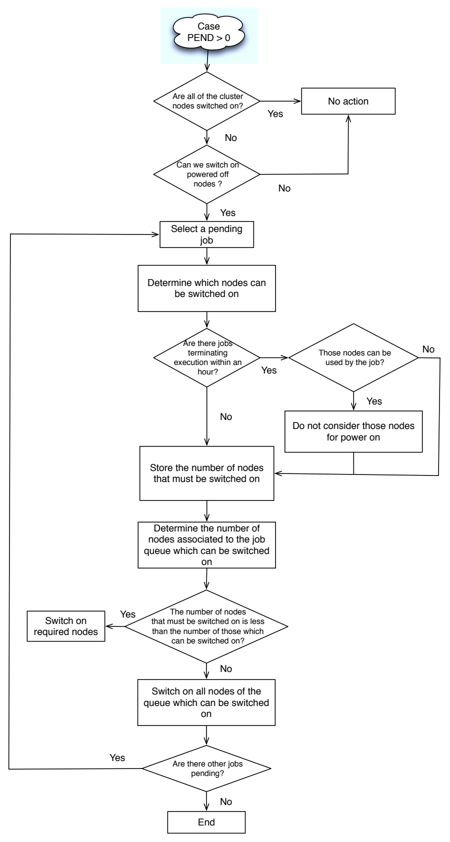
Case
PEND > 0
Are all of the cluster nodes switched on?
Yes
No action
No
Can we switch on powered off nodes ?
No
Yes
Select a pending job
Determine which nodes can be switched on
Are there jobs terminating execution within an hour?
Yes
Those nodes can be used by the job?
No
Yes
Do not consider those nodes for power on
No
Store the number of nodes that must be switched on
Determine the number of nodes associated to the job queue which can be switched on
The number of nodes that must be switched on is less than the number of those which can be switched on?
Yes
Switch on required nodes
No
Switch on all nodes of the queue which can be switched on
Yes
Are there other jobs pending?
No
End

Figure 4:
Case PEND > 0 flowchart

in which nodes are turned on and off with consequent delays related to the execution of jobs and overhead caused by the hardware checks performed upon turning on a node to verify that the node is fully operational. Indeed, switching on a node takes a relatively long time on average when compared to a normal desktop computer starting up. The factor that justifies such a long time lies in the hardware, which is very sophisticated and complex, requiring the execution of many checks at startup in addition to the system routines and LSF procedures allowing the node to be inserted in the current list of active nodes managed by the cluster.

To cope with the problem, the algorithm proceeds to switch off a node when it has been idle for at least 30 minutes. This temporal parameter is chosen by the sys admin.

IMPLEMENTATION

The software runs in background, implemented as a daemon. Retrieving the information required to the algorithm is down through the LSF C APIs, provided by a set of developer libraries to interact with the scheduler. We have implemented our own APIs as a set of library to simplify information retrieval and to support the decisional process.

DATABASE DESIGN

The data we collect must be stored in order to allow faster retrieval in the many different phases of the algorithm, and for bookkeeping purposes. The database design must take into consideration the observations we made previously when defining the data to be retrieved. Furthermore, the database must be able to handle temporal data, in order to support the creation of logs, the access to historical data which will be subject to statistical analysis and will also allow monitoring the daemon to control the correctness of its operations.

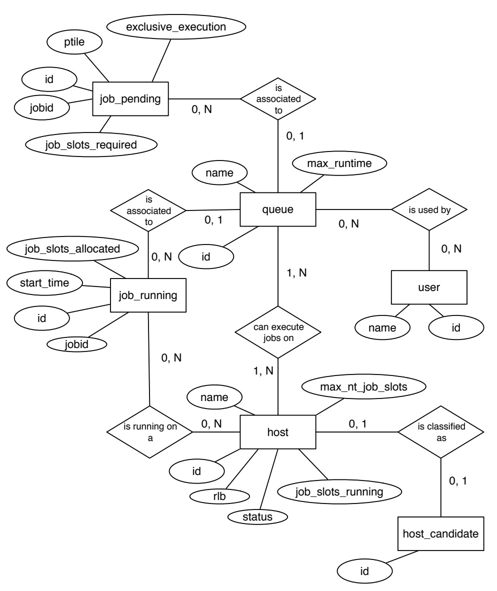

Figure 5:
Entity-Relationship Model

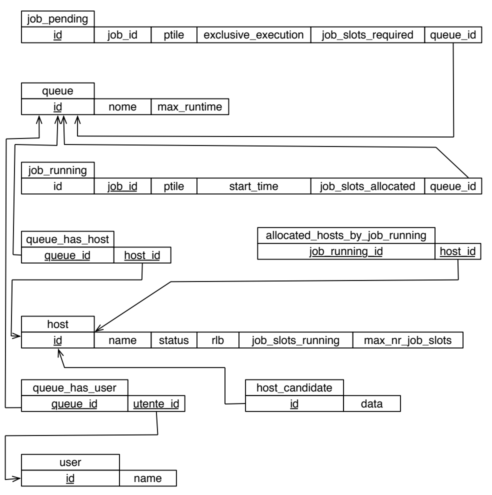

Figure 6:
Database Logic Model

Time can be managed in different ways; current state of the art approaches for managing temporal data include audit trails and snapshots. In our case, we choose the snapshot approach which is simpler to implement and well suited for our problem. The snapshot approach simply require cloning the database tables of interest and adding to cloned tables a field of type timestamp or date. During the execution of INSERT queries, data is inserted both in the tables of interest and in the clones, properly setting the date field. Figure 5 shows the conceptual model, while Figure 6 described the logical model of the database.

The database was implemented using SQLite owing to the following:

- it is a standalone, embedded database solution (no external dependencies) with small memory footprint;
- it does not require additional libraries;
- provides support for fully referential integrity;
- it is much faster than MySQL or Postgresql with regard to our needs (this is especially important, given that our software performs hundreds of queries every 10 minutes).

TEST AND EXPERIMENTAL RESULTS

We have tested the software thoroughly, leaving the daemon running uninterrupted for two weeks. A careful inspection of the log allows inferring the correct operation of the software. We now analyze the results obtained. The first snapshot in the cluster log shows the situation on December 4, 2010. After checking the state the daemon determines the nodes which are candidate to shutdown. There were no nodes candidate to shutdown (no nodes idles for at least 30 minutes). This situation occurs twice, for example, January 5, 2011 with the node n8-csm which has been analyzed (no jobs are running on the node, so that it is a candidate for shutdown). For this reason this node has been marked as a candidate for shutdown, as seen from the next snapshot, half an hour later. Let us analyze again the snapshot of December 4, 2010, this time half an hour later delay. We note that the daemon reported the following situation: All nodes of the cluster are turned on. That moment there were pending jobs on the cluster (case pending > 0), so that the software started computing how many and what nodes must be turned on. Unfortunately, all nodes in the cluster (30) were already switched on. As a consequence, pending job persisted in the pending state until resources were available. The last log we analyzed is related to January 12, 2011. There were pending jobs. The software verified the presence of nodes switched off and started analyzing the jobs pending and the resources available for each node in the cluster. The job whose ID is 128467 required 64 cores with the *ptile* option that specifies 64 cores per node. In other words, since each cluster node is equipped with 64 cores (32 physical cores, 64 virtual cores taking into account that Simultaneous Multi Threading is enabled allowing two threads per core) the user requested all of the cores of exactly one node. The daemon analyzed the available resources of all the nodes until it determined that node n4-csm had 64 cores available, and marked it as a node to be turned on. Indeed, at the end of the log, the daemon reports turning on node n4-csm. It is worth recalling here that the software would have turned on node n4-csm even if this node alone were not enough to grant the required resources to job 128467, according to the algorithm's design.

CONCLUSIONS

Energy saving is one of the most important issues concerning the entire ICT sector. This work is focused on these issues and it addresses the problem by proposing a solution for our specific use case, laying the foundations for future improvements. We plan to develop new advanced algorithms that can provide better choices according to the state of the system.