Centro Euro-Mediterraneo
per i Cambiamenti Climatici

Research Papers
Issue 2011
December 2011

Scientific Computing and Operation (SCO)

Green Computing: Design of a power saving solution based on Mixed Integer Programming (MIP)

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 This report describes the design, implementation and test of a software able to monitor the load of a cluster IBM Power 6, acting on its nodes switching them on and off on the basis of their load (jobs), scheduled by LSF, taking into account a linear, integer variables optimization model which produces a feasible schedule, possibly optimal, of pending jobs, given running jobs. Optimizations happens in two phases: optimization of the energy cost function, associated to each cluster node; optimization of makespan in order to avoid holes in the schedule produced by the 1st optimization. Machines which are idle in the schedule can be powered off. The aim is to save energy, when possible, consumed by IBM Power 6 cluster. Moreover, turning off idle nodes will also imply less heat produced by the cluster, and additional savings on the cooling system.

INTRODUCTION

Regarding Green IT, the activities of the CMCC are aimed at reducing energy consumption due to the use of computational resources for high performance in production in the center. The reduction of such costs can be obtained in an optimized manner switching off the resources not used, while maintaining a certain quality of service, with the consequent lowering of the heat produced and thus a saving on the energy consumed by the cooling system.

The problem of minimizing the energy cost in the presence of constraints on resources (cluster nodes), is known in Operations Management as Resource Constrained Project Scheduling Problem (RCPSP). Therefore, the main goal of this work has been the design of a mathematical MIP (Mixed Integer Programming) model with the dual aims of reducing the energy consumption cost and simultaneously optimizing the makespan of the schedule of jobs on various nodes of the cluster.

The optimization approach tries to dynamically manage the power consumption ensuring the demand, in terms of resources, of the scheduled jobs on the cluster is met. Jobs are scheduled optimally, and nodes identified as idle can be turned off. The optimization model has been implemented using IBM ILOG CPLEX solver, v. 12.2, already installed on the Calypso cluster. The model was tested, considering different workloads in the cluster, scheduled by the LSF scheduler. The model implemented is a component of a larger system that is able to iteratively collect information on the load of the cluster (acquired from LSF), formulate and solve a new instance of the optimization problem, obtaining an optimal configuration of new nodes (power on, off or idle nodes), apply the changes in scheduling already planned to LSF (or other resource manager), to achieve a new optimal configuration for the next state of the system.

SCHEDULING ISSUES

Formally, a scheduling problem can be characterized by the following three sets:

- $T = \{T_1, T_2, \ldots, T_n\}$, set of n tasks (operations);
- $P = \{P_1, P_2, \ldots, P_m\}$, set pf m processors (machines);
- $R = \{R_1, R_2, \ldots, R_s\}$, set of $s$types of additional resources.

In this context the word "scheduling" means assigning machines and (if necessary) additional resources to the tasks in order to execute while respecting the constraints that in the classical theory of the scheduling are typically the following:

- Each task is run by a machine at a certain time;
- Each machine can perform at most one task at a time.

In general it is assumed that the tasks can be partitioned into subsets (chains in general) each of which is called a job. Therefore, a job $j = \{T_{1j}, T_{2j}, \ldots, T_{n_j j}\}$ and adjacent tasks are performed on different resources. With respect to methods of task execution, machines are divided into two types:

- Parallel: If these perform the same function. Moreover, parallel machines are further distinguished , on the basis of their speed of execution as:

- Identical: the machines all have the same processing speed v. The processing time of a task T_j depends only on the task and not on the machine P_i that runs it: $p_{ij} = p_j = l_j/v$ (l_j is the length of T_j);
- Uniform: the machines have different speeds v_i but invariant with respect to the task. The processing time of a task T_j is inversely proportional to the speed v_i of the machine P_i executing it: $p_{ij} = l_j/v_i$;
- Non-related: the velocities v_{ij} of the individual machines are different depending on the tasks.

- Dedicated: if these are specialized to perform particular tasks. In general, in the general job shop model, the number n_j of tasks of each job is arbitrary and there is a fixed order for their processing. In this context there are three sub-models:
  - Open-Shop: the number of tasks per job is the same ie $n_j = K \; \forall j$, and are not given precedence relationships between tasks of a job;
  - Flow-Shop: in addition to $n_j = K \; \forall j$, processing T_{i-1j} should precede T_{ij} $\forall i \; \forall j$;
  - Job-Shop: the number n_j of tasks of each job is arbitrary and there is a total ordering on the tasks of each job.

In the scheduling process is important to distinguish between the parameters that define the problem known a priori (in the deterministic case) and variables that describe the solution produced by the scheduling:

- Parameters: each job j is associated to the following data: r_j = ready time (or release date): indicates the arrival time in the shop or the minimum starting time related to the processing of the first operation of the job itself; d_j = due date: indicates the time limit within which job processing is expect to end. Normally, depending on the "due date" are defined some penalty functions; $p_j = \sum_{i=1}^{n_j} p_{ij}$ Processing time: indicates the time to execute all operations of the job, given the vector of processing times $[p_{1j}, p_{2j}, \ldots, p_{nj}]^T$ associated to all machines $P = \{P_1, P_2, \ldots, P_{n_j}\}$; $a_j = (d_j - r_j)$ allowance: indicates the total time available to the job in the shop.
- Variables: solving a scheduling problem involves determining the time at which each job must be processed and more precisely when each operation must be performed. This means for every operation determining the value of the following variable: W_{ij} = waiting time of the i-th step T_{ij} of job j, ie the waiting time in the shop after the $(i-1)$-th step before the start of the i-th step.

The total hold time of the job is: $W_j = \sum_{i=1}^{n_j} W_{ij}$
The scheduling of a particular problem is therefore completely specified by knowing the set $\{W_{ij}\}$. Any other variable introduced to evaluate the goodness of a schedule is a function of W_{ij}. One example is the "completion time" C_j of the job j, which indicates the instant at which the last operation $T_{n_j j}$ of the job j is completed: $C_j = r_j + W_{1j} + p_{1j} + \ldots + W_{n_j j} + p_{n_j j} = r_j + p_j + W_j$.

A criterion for scheduling is given by some functions that should be minimized:

Makespan $C_{max} = max_j\{C_j\}$. It Measures the completion time of the last job. Minimizing the makespan implies that many machines

are used and balances their use.

SCHEDULING ON PARALLEL RESOURCES

The processing of jobs on parallel machines is important both theoretically and practically. From a theoretical point of view this case is a generalization of the scheduling on a single machine, while from the practical point of view it is important because the existence of parallel resources in common occurs in reality. The main objectives are:

- Makespan;
- Total completion time;
- Maximum lateness.

The makespan in the case of a single machine is relevant only if the setup time depends on the sequence while in the case of parallel machines minimizing the makespan in any case ensures load balancing between the machines. Moreover, preemption which in the case of single machine is significant only in the presence of "ready time", becomes important in this case, even with concurrent jobs. Finally, even if in most cases these models give rise to excellent schedules that are non-delay, in the case of minimization of the "total completion time" without preemption and with parallel machines which are not related the optimal scheduling is not necessarily non-delay. The scheduling of parallel machines is a two step process:

- Allocation of jobs to machines;
- Sequencing of jobs on each machine.

If the objective is the makespan only the first step is significant. This model has a considerable practical interest because the minimization of the makespan also provides load balancing between the various machines. This problem is NP-hard even in the simplest case $P2||C_{max}$, which can be reduced to the well-known PARTITION problem, a classic NP-complete problem. That said, it follows that many heuristic algorithms have been developed for the general model. The Longest Processing Time first (LPT) rule, gives the time $t = 0$ the m longer job to the m machines. Later, when a machine is free it will be assigned a job not scheduled with the largest processing time.

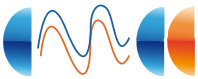
This heuristic tries to assign shortest jobs at the end of the scheduling where they can be used to balance the loads between the machines. For this heuristic a guaranteed approximation exists; the following relationship can, indeed, be proved: $\frac{C_{max}(LPT)}{C_{max}(OPT)} \leq \frac{4}{3} - \frac{1}{3m}$ where $C_{max}(LPT)$ indicates the makespan of the LPT scheduling while $C_{max}(OPT)$ the one related to the optimal scheduling. There are other heuristics for the problem $P2||C_{max}$ that are more sophisticated than the LPT rule and consequently provide better bounds.

DESIGN OF AN OPTIMIZATION MODEL

The problem is formulated taking into account a careful analysis of the information and data provided by the computer center. The authorized users submit jobs to perform scientific calculations, simulations, etc. Each job requires a certain number of cores for its execution, and is managed by LSF according to its submission request. At the time of submission a user specifies:

- Number of cores;
- Queue (optional);
- mode of submission (optional).

If the mode is not specified, then the cores are allocated, on one or more nodes, preferring less



loaded nodes. If the request is for exclusive use of nodes, then one or more nodes are allocated exclusively to that job (if the job does not use all the cores of the node, so that $\#core_{required} < \#core_{node}$ these cores are idle). If the number of cores per node has been specified, then it allocates the required number of cores on a number of nodes equal to $\left\lceil \frac{\#core_{required}}{\#core_{node}} \right\rceil$

Within the cluster, there are some nodes which must always remain turned on since they provide services critical to the operation of the cluster. These nodes are:

- GPFS Server nodes (I / O nodes) ;
- LOGIN nodes.

Every job must have a unique startup time. For each job we know the submission time (submit_time) and its startup time (start_time). For running jobs we can get an estimate for the maximum remaining time (processing) of each job. The following values characterize a queue: priority and maximum runtime. Jobs are classified as running or pending. If there are no jobs pending the model returns which nodes are idle and can therefore be turned off; if there are jobs pending the model determines the number of nodes that can be turned on (if there are one or more nodes turned off), or whether the ready time of some nodes is lower than the time at which these nodes have been turned on, so that it is worth waiting for these nodes. As an example, if a job has been running on the queue poe_long for 23h and 40 ' it can run for at most 20'; since we know that the time required to switch on a node is about 30', then the system should not switch on new node to manage pending jobs.

NOTATION

We now introduce the formal notation used to describe the mathematical model. For each entity of interest in the world of the problem, we use a mathematical variable to represent it properly. In order to obtain a modeling as general as possible, we consider a variable number of clusters, and therefore define the variable M, which represents the number of clusters that the model takes into account. A cluster consists of a set of nodes. For each cluster, then we can define a value N_i, which represents the number of nodes on the cluster i ($i = 1, \ldots, M$). Each node contains a set of cores. Normally the number of cores installed in a node of a cluster is homogeneous for all of the nodes in the cluster. However, in this work, we prefer to adopt a more general approach and let K_{in} be the number of cores installed on node n of cluster i ($i = 1, \ldots, M; n = 1, \ldots, N_i$). This generalization apparently seems to be too strict, but in our case it is instead strictly required. Indeed, despite the Calypso cluster nodes are homogeneous with regard to the number of cores, in practice for a few nodes all of the the available cores can not be used. This is because for service nodes, which perform essential tasks for cluster management etc, some cores are dedicated to run those tasks and therefore can not be used for scheduling users' jobs.

A set of hosts in the cluster is logically connected to form a submission queue. Let Q be the number of queues in the system; jobs scheduled on a queue will run only on the nodes associated with that queue. For each queue we define a parameter, denoted as STM^r_{MAX}, which is related to the quality of service and that we call System Queue Time Average of the queue r ($r = 1, \ldots, Q$). The logical association between cluster nodes and queues is handled by the mathematical model using a binary data vector ρ_{inr}, , in which an entry is 1 if and only if the node n-th is compatible with the queue r, and is 0 otherwise. We denote the number of pending jobs with J, and the values from 1 to J

will be associated with the corresponding jobs.

Finally, we adopt the "rolling horizon" approach, and therefore we define a point in time that determines the planning horizon denoted as T_{RH}. That said, for each job j ($j = 1, \ldots, J$) we must define a variable corresponding to the number of cores required for execution. Then we define n_{ij}, the number of cores required by the j-th job ($j = 1, \ldots, J$), if assigned to the cluster i ($i = 1, ldots, M$). Moreover, we need to associate each job j to its submission queue. This is controlled by the vector a_{jr} in which an entry is 1 if and only if the j-th job ($j = 1, ldots, J$) has been scheduled on the queue r.

Depending on the submission request, a job can be assigned to a specific number of nodes, which we denote by A_j and corresponds to the maximum number of nodes to which a job can be assigned. If this variable is null, then the allocation of the cores is left to the model solver, which obviously will opt to obtain the most advantageous solution in terms of energy saving, trying to saturate the nodes that are already active and with one or more available cores.

Each job has a different duration, which in theory is known only to the user when she submits the job choosing the most appropriate queue. This is because each queue is characterized by a temporal parameter, which determines the maximum number of seconds for which a job can be running on it. Therefore, for each job j ($j = 1, \ldots, J$) we define an estimated due date which is exactly the maximum run time of the queue on which the job has been scheduled, and denote it by T_j. We define also the submission time of job j-th ($j = 1, \ldots, J$) to be μ_j, while δ_{ij} denotes a vector whose entries are 1 if and only if the j-th job ($j = 1, \ldots, J$) can be executed on cluster i, 0 otherwise.

In order to obtain a realistic scheduling the mathematical model takes into account the jobs that are already running on the cluster. Running jobs represent a constraint with dual nature:

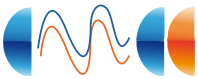
- the nodes hosting a running job can not be turned off at least until the job terminates its execution (assuming that no other job is using those cores);
- the cores used by a running job are themselves a constraint related to the scheduling of pending jobs.

It is therefore necessary to define a variable which takes into account when a certain set of cores of a given node can be reused, or when a node can become a candidate for shutdown, i.e., when all of the cores of a node are not allocated. Then, we define "Ready time of the core" and we denote it as r_{ink} the ready time of the k-th core ($k = 1, \ldots, K_{in}$) of the n-th node ($n = 1, \ldots, N_i$) belonging to cluster i th ($i = 1, \ldots, M$). Finally, we define the energy cost associated to each node in a cluster, denoted as c_{in} which is the energy cost of a single node n ($n = 1, \ldots, N_i$) of cluster i ($i = 1, \ldots, M$).

DECISIONAL VARIABLES

Suppose that the job processing and ready time are all multiples of a certain temporal "quantum" (for example $\Delta t = 4h$). Under this assumption we can assume that time is discrete and adopt a "rolling horizon" approach in which the discrete time variable t is such that $t \in \{0, 1, \ldots, t_{RH}\}$.

- y_{int} is a binary variable whose value is 1 (0 otherwise) if and only if the n-th node ($n = 1, \ldots, N_i$) of cluster i-th ($i = 1, \ldots, M$) is turned on in the t-th time slot $t > r_{ink}$);
- z_{inkj} is a binary variable whose value is 1 (0 otherwise) if and only if the j-th job ($j \in J$) uses the k-th core ($k = 1, \ldots, K_{in}$) of n-th node ($n = 1, \ldots, N_i$) of i-th cluster ($i = 1, \ldots, M$);



- x_{inj} is a binary variable whose value is 1 (0 otherwise) if and only if the j-th job ($j \in J$) is assigned to the n-th node ($n = 1, \ldots, N_i$) of i-th cluster ($i = 1, \ldots, M$);
- $p_{jj'ink}$is a binary variable whose value is 1 (0 otherwise) if and only if the j-th job ($j \in J$) precedes job j' on the the k-th core of node n-th ($n = 1, \ldots, N_i$) of i-th cluster ($i = 1, \ldots, M$);
- s_{jt} is a binary variable whose value is 1 (0 otherwise) if and only if the j-th job ($j \in J$) begins at time t;
- w_{ij} is a binary variable whose value is 1 (0 otherwise) if and only if the j-th job ($j \in J$) is assigned to the i-th cluster ($i = 1, \ldots, M$).

OBJECTIVE FUNCTION AND CONSTRAINTS

Referring to the energy optimization model, we examine in this subsection, the objective function and constraints associated to the problem. The objective function can be expressed as minimization of the cost function energy. Mathematically we have:

$$Minf = min \sum_{i=1}^{M} \sum_{n=1}^{N_i} \sum_{r_{ink}}^{T_{RH}} c_{in} y_{int} \qquad (1)$$

This function has the aim to minimize the energy expenditure of a set of clusters. In our case study we will restrict ourselves only to Calypso, and then this function will minimize the energy expenditure of Calypso, providing a scheduling reaching this goal. Of course, as already discussed, there are a number of constraints that must be applied to obtain the schedule. Later, we will also analyze the objective function required to minimize the makespan. Ultimately we will have two cascading models; the second one takes as input the result obtained from the first model, i.e., the minimum energy cost and provide as output the final schedule. The first constraint we examine is related to the uniqueness of the processing start time. This must happen for each pending job. Therefore, it must be:

$$\sum_{t=0}^{T_{RH}-T_j} s_{jt} = 1 \qquad (2)$$

Moreover, It is also required that for each job to be scheduled its start time is higher (at most equal) of the ready time of all the cores that it should use. This is obvious, since if we consider cores with a ready time greater than the start time of a job, we would not be taking into account that some resources are already committed to that node in the run of a previously scheduled job. That said, we add the following temporal constraint on decision variables with regard to decision variables for the use of cores:

$$\sum_{t=0}^{T_{RH}} ts_{jt} \geq r_{ink} z_{inkj} \qquad (3)$$

$$\forall j \in J; \forall i = 1, \ldots M; \forall n = 1, \ldots N_i; \forall k = 1, \ldots, K_{in}$$

In order to submit a job, we said that the user must specify a queue in which the job will be run (a default queue is used otherwise). This corresponds to a constraint on nodes that the job will be able to use. The association between queues and nodes is handled by the vector ρ_{inr} while the association between a queue and the job is handled by the vector a_{jr}. It is then clear that there must be a constraint on decision variables x_{inj} that determines on which nodes a job can be scheduled:

$$a_{jr} x_{inj} \leq \rho_{inr} \qquad (4)$$

$$\forall j \in J; \forall r = 1, \ldots Q; \forall i = 1, \ldots M; \forall n = 1, \ldots, N_i$$

A job must be assigned to only one cluster; this constraint is expressed by acting on the decision variablesw_{ij}, by requiring that only a value of w_{ij} is set to 1. In our case study, since our attention is focused only on Calypso, these variables are all equal to 1. The constraint is the following one:

$$\sum_{i=1}^{M} \delta_{ij} w_{ij} = 1 \qquad (5)$$
$$\forall j \in J; \forall i = 1, \dots M$$

It is also required that each job running on the k-th core ($k = 1, \dots, K_{in}$) of the n-th node ($n = 1, \dots, N_i$) of cluster i-th ($i = 1, \dots, M$), must also be running on the n-th node ($n = 1, \dots, N_i$) of the i-th cluster ($i = 1, \dots, M$). From this assumption we derive the next two constraints:

$$z_{inkj} \leq x_{inj} \qquad (6)$$
$$\forall j \in J; \forall i = 1, \dots M; \forall n = 1, \dots N_i; \forall k = 1, \dots, K_{in}$$

$$x_{inj} \leq \sum_{k=1}^{K_{in}} z_{inkj} \qquad (7)$$
$$\forall j \in J; \forall i = 1, \dots M; \forall n = 1, \dots N_i$$

Given a cluster, we must model the nodes belonging to it as not being associated to other clusters. This can be expressed binding the variables x and w by the following relationship:

$$\sum_{i=1}^{M} x_{inj} \leq w_{ij} \qquad (8)$$
$$\forall j \in J; \forall i = 1, \dots M; \forall n = 1, \dots N_i$$

For each job j it is necessary to allocate a specific number of cores on a single cluster. This is expressed binding the variable z, which represent the cores used by job j, to the variables w. The key role is played by vector n, which shows the number of cores required by job j:

$$\sum_{n=1}^{N_i} \sum_{k=1}^{K_{in}} z_{inkj} = n_{ij} w_{ij} \qquad (9)$$
$$\forall j \in J; \forall i = 1, \dots M$$

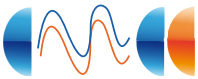
In the process of scheduling with resource constraints we must introduce time constraints. In our case, it must hold that the execution of different jobs on the same core may not cause overlap. Therefore, f the job j and j' must use the k-th core ($k = 1, \dots, K_{in}$) of n-th node ($n = 1, \dots, N_i$) of i-th cluster ($i = 1, \dots, M$), then the execution of job j will precede job j' or vice-versa. Mathematically, this constraint impacts on the start processing decision variables s and on the decision variables that govern the precedence relations between jobs. The above results in a series of three constraints:

$$\sum_{t'=0}^{T_{RH}-T_{j'}} t' \cdot s_{j't'} \geq \sum_{t=0}^{T_{RH}-T_j} t \cdot s_{jt} + T_{ij} \qquad (10)$$
$$p_{jj'ink} - t_{RH}\,(1 - p_{jj'ink})$$
$$\forall j \in J; \forall j' \in J - j; \forall i = 1, \dots M; \forall n = 1, \dots N_i;$$
$$\forall k = 1, \dots, K_{in}; \forall t = 0, \dots, T_{RH}; \forall t' = 0, \dots, T_{RH};$$

$$p_{jj'ink} + p_{j'jink} \geq z_{inkj} + z_{inkj'} - 1 \qquad (11)$$
$$\forall j \in J; \forall j' \in J - j; \forall i = 1, \dots M;$$
$$\forall n = 1, \dots N_i; \forall k = 1, \dots, K_{in}$$

$$p_{jj'ink} + p_{j'jink} \leq 1 \qquad (12)$$
$$\forall j \in J; \forall j' \in J - j; \forall i = 1, \dots M;$$
$$\forall n = 1, \dots N_i; \forall k = 1, \dots, K_{in}$$

One of the goals of our model also is also a



guarantee related to a certain quality of service, on the basis on the time μ_j spent by a job j in the system. We define the average system time in the following as the first member of the inequality. The second member is the value of the vector STM^r_{MAX} of queue r. Mathematically this is expressed as:

$$\begin{aligned} STM_r &\triangleq \frac{\sum_{j\in J} a_{jr}\left[\left(ts_{jt}+\sum_{i=1}^{M} T_j\delta_{ij}w_{ij}\right)-\mu_j\right]}{Q} \\ &\leq STM^r_{MAX} \qquad (13) \\ &\forall r=1,\ldots,Q;\forall t=0,\ldots,T_{RH} \end{aligned}$$

where the quantity

$$ts_{jt}+\sum_{i=1}^{M}\sum_{n=1}^{N_i} T_j\delta_{ij}w_{ij}$$

represents the estimated time of execution. We come now to the most important constraint. Fixing a node in the cluster, it must be turned on at a certain point in time if there is at least ajob that uses its cores. This constraint binds the variable y, which tells whether a node is turned on in a certain time slot, the variable s and start processing x to identify on which node a certain job is running.

$$y_{int} \geq \left[\left(\sum_{t'=max(0,t-T_{ij}+1)}^{t} s_{jt'}\right)+x_{inj}-1\right] \qquad (14)$$

$$\forall j\in J;\forall t=0,\ldots T_{RH};\forall i=1,\ldots M;\forall n=1,\ldots N_i;$$

A job at the time of submission may require a number of well-defined nodes. This value is stored in the vector A_j. If$A_j>0$, then a job requires the allocation of its cores on A_j nodes:

$$\sum_{k=1}^{K_{in}} z_{inkj} = \left\lceil \frac{n_{ij}}{A_j} \right\rceil x_{inj} \qquad (15)$$

$$\forall j\in J;\forall i=1,\ldots M;\forall n=1,\ldots N_i;$$

$$\sum_{i=1}^{M}\sum_{n=1}^{N_i} x_{inj} = A_j \qquad (16)$$

$$\forall j\in J;$$

When describing the Calypso cluster we said that some of its nodes are of key importance for the cluster, owing to the fact that service nodes must always be turned on in each time slot; then, for each service node:

$$\sum_{t=0}^{T_{RH}} y_{int} = T_{RH}+1 \qquad (17)$$

$$\forall i=1,\ldots M;\forall n\in\{service \quad nodes\}$$

We now turn our attention to makespan minimization. Solving the energy optimization problem provides us with a job scheduling such that the use of computing resources (nodes) of the cluster is optimum. However, this solution does not address the makespan of the schedule, i.e., the time it takes to schedule all of the jobs until the last job terminates its execution. A solution of the first problem could be:

- Turn on the node n in the time slot 0;
- Shut down the node n in the time slot 1;
- Turn on the node n in the time slot 2.

Through a second optimization model, we try to remove the middle period, as the first model suggests to turn off the node, and instead we try to move the processing done during the second period in the first one. This is precisely the aim of minimizing the makespan. In this way we obtain a hierarchical family of optimization problems that solve a multi-objective problem. The formulation starts with the definition of a job duration, which is given by the following relationship:

$$ts_{jt} + \sum_{i=1}^{M} T_j \delta_{ij} w_{ij} \tag{18}$$

$$\forall j \in J; \forall t = 0, \ldots T_{RH};$$

This problem will have the same decision variables and the same parameters of the energy optimization problem; we add the following new parameters:

- c_{min}, which represents the minimum energy cost, obtained by the first model.
- $ms \geq 0$, which represents the makespan to be minimized.

Finally, we add a new constraint:

- when optimizing the makespan, the energy cost due to the cluster nodes turned on must be equal to the minimum energy cost obtained by solving the energy optimization problem:

$$\sum_{i=1}^{M} \sum_{n=1}^{N_i} \sum_{r_{ink}}^{T_{RH}} c_{in} y_{int} = c_{min} \tag{19}$$

- the makespan must be greater than or equal to the completion of a job:

$$ts_{jt} + \sum_{i=1}^{M} T_j \delta_{ij} w_{ij} \leq ms \tag{20}$$

The makespan objective function therefore is

$$Minf = min\,(ms) \tag{21}$$

IMPLEMENTATION

The system was implemented using the C programming language and the following software and libraries (Figure 1):

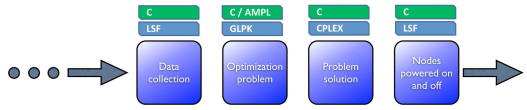
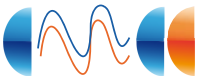


Figure 1:
System Implementation

- MPL (A Mathematical Programming Language) is a software that can translate models written in the Mathematical Programming Language AMPL in a format understandable to a general solver. AMPL is a high-level language for describing models of Mathematical Programming;
- GLPK (GNU Linear Programming Kit) is a software library written in ANSI C; it can be used to solve linear programming problems, both continuous (LP) and mixed integer (MILP). The library implements the simplex method and a method based on interior points for solving linear problems.
- CPLEX (solver): ILOG CPLEX optimizer is the world's most advanced and used solver for Mathematical Programming; it represents the de facto standard in the field of Operations Research. It improves the efficiency of decision making, allows rapid implementation of various strategies, and increases their profitability.
- LSF API can be used to retrieve the information managed by the LSF scheduler.

VALIDATION AND TEST

The models must be validated. For their validation some instances have been built and tested to verify the correctness of the proposed scheduling provided as output.



TEST ONE

The first test involved a problem with a cluster containing two nodes and five cores per node. Two jobs were submitted, with a request for three cores for the first job and two cores for the second one. The planning horizon for this first check was $t = 5$ time slots. We expect a single node turned on to run both jobs, since these require a total of 5 cores. The steps followed for this test are the following ones:

- Creation of data files coded in AMPL language;
- Generation of the problem file in LP format, to be used as input for the CPLEX solver;
- Creation of the corresponding instance in CPLEX;
- Collection of results.

The solution obtained through the solver is as follows:

```
Variable Name Solution Value
y (1,2,0) 1
y (1,2,1) 1
s (1.0) 1
s (2.0) 1
x (1,2,1) 1
x (1,2,2) 1
w (1.1) 1
w (1,2) 1
z (1,2,1,1) 1
z (1,2,2,1) 1
z (1,2,3,1) 1
z (1,2,4,2) 1
z (1,2,5,2) 1
```

This solution is interpreted as follows. Given a discrete time window, which we call time-slot in the real case we consider, a slot lasts for $4h$), the variables $y(i, n, t)$ are vectors of dimension 3 where each dimension indicates:

- i = cluster;
- n = node;
- t = time slot.

As stated in the formulation of the model, these binary variables assume the value 1 if and only if the n-th node of cluster i-th is turned on during the t-th time slot. In the solution only one node is identified as active, that is, the node corresponding to $n = 2$, which takes care of the execution of both Jobs.

The variable s, which is a vector in which the first and second index respectively account for job and time slot, indicates in which time slot the job starts. From the solution we can see that both jobs are scheduled in slot 0 (i.e., they start immediately). This result was expected, given that these jobs had no special requirements in terms of nodes or cores per node.

The node $n = 2$ remains active for two time slots, since in the data file the parameter corresponding to the job duration (time parameter $job\_estimated\_time$) is equal to 2 for both jobs. The energy cost a is associated to each node; therefore, since a single node is in operation for 2 time slots, the result of the objective function is 4. The time required to produce the solution of the energy optimization model was 0.04 seconds. The value of the objective function obtained is then inserted in the data file of the makespan optimization model. The solution of this second model is the following:

```
Variable Name Solution Value
ms 2
s (1.0) 1
s (2.0) 1
```

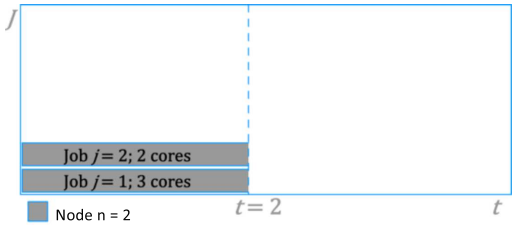


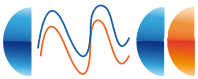
Figure 2:
Gantt Chart - Verify 1

```
x (1,2,1) 1
x (1,2,2) 1
w (1.1) 1
w (1,2) 1
z (1,2,1,2) 1
z (1,2,2,2) 1
z (1,2,3,1) 1
z (1,2,4,1) 1
z (1,2,5,1) 1
y (1,2,0) 1
y (1,2,1) 1
```

We note immediately that the solution of this second model is identical to the first; analogous considerations apply to this case. The fact that the two solutions are identical is not surprising and is an expected result, since the minimization of the makespan is precisely to minimize the overall time spent executing the jobs. The ms variable in this case is equal to 2, and given the simplicity of the problem is immediate to realize that the whole project ends in 2 time slots, since both jobs last 2 time slots, and can be executed in parallel on 5 cores of node $n = 2$.

As shown in the Gantt chart (Figure 2), it is evident that there is only one active node for two time slots. The model is planning to switch d off the node for the next 3 slots. The node $n = 1$ will always be turned off. Below is the collection of data characterizing this first test:

```
param M: = 1;
param N: = 2;
param K: =
1 1 5
1 2 5
;
param Q: = 2;
param STM_MAX: =
1 6
        2 6;

param j: = 2;

param a: =
1 1 1
1 2 0
2 1 0
2 2 1
;

param A: =
1 0
2 0;

param n: =
1 1 3
1 2 2;

param tempo_stim_job: =
1 2
2 2
;

param tempo_sott: =
1 1
2 1;

param map_queue_node: =
1 1 1 1
1 1 2 1
1 2 1 1
1 2 1 2
;
```




```
param delta: =
1 1 1
1 2 1
;

param ready_time: =
1 1 0 1
1 1 2 0
1 1 3 0
1 1 4 0
1 1 5 0
1 2 1 0
1 2 2 0
0 1 2 3
1 2 4 0
1 2 5 0
;

param costo_energetico: =
1 1 2
1 2 2
;

param T: = 5;
```

TEST TWO

The second test is similar to the first one, but in this case the job for $j = 1$ requires 5 cores. The duration of each job is equal to 2 time slots, and the configuration parameters of the cluster are the same. The planning horizon is set at $t = 5$ time slots. This time we do not expect the jobs to be placed both on a single node, given the request associated to the first job. The solution to the the energy optimization is the following:

```
Variable Name Solution Value
y (1,1,0) a
y (1,1,1) a
y (1,1,3) a
y (1,1,4) a
s (1.0) 1
s (2.3) 1
x (1,1,1) a
x (1,1,2) 1
w (1.1) a
w (1,2) 1
z (1,1,1,1) 1
z (1,1,1,2) 1
z (1,1,2,1) 1
z (1,1,3,1) 1
z (1,1,4,1) 1
z (1,1,4,2) 1
z (1,1,5,1) 1
p (2,1,1,1,1) 1
p (2,1,1,1,4) 1
```

This solution involves only one node (the one for $n = 1$) turned on for 4 time slots, because of the request for the first job. Job precedence constraints must hold, as shown by the values of the p variable indicating the order of precedence for cores 1 and 4 of node 1 (the vector p is 5-dimensional and each index is related to: job, job, cluster, node, core). The value of the objective function in this case is equal to 4. This value is passed to the makespan optimization model, obtaining the following solution:

```
Variable Name Solution Value
ms 2
s (1.0) 1
s (2.0) 1
x (1,1,1) 1
y (1,1,0) 1
x (1,2,1) 1
x (1,2,2) 1
w (1.1) 1
w (1,2) 1
z (1,1,1,1) 1
z (1,1,2,1) 1
z (1,1,3,1) 1
z (1,2,2,1) 1
```

```
z (1,2,3,1) 1
z (1,2,4,2) 1
z (1,2,5,2) 1
y (1,2,0) 1
y (1,1,1) 1
y (1,2,1) 1
```

In this case the makespan optimization took into account the minimum cost value calculated from the first model, but both nodes re turned on in order to minimize the completion of the project, which as in the first case study is $ms = 2$ (while the makespan after the first optimization is 4 since only one node is active node and carries over the whole schedule). Both jobs start in the same time slot, i.e., at the instant $t = 0$. With regard to the previous Gantt char, the situation remains unchanged, except that in this case the job $j = 1$ uses both nodes for its execution. The set of data of this second case study is not reported being almost equal to the first one (the only changes are related to the values of the vector n).

CASE STUDY

The case studies drawn from real job submissions are analyzed using the mathematical models and software described. We run the software designed and implemented as a job directly on the Calypso cluster. Both the model and data files are dynamically built using information extracted from the LSF scheduler. The instances that are generated are therefore much greater in size with regard to to the previous validation tests. The first instance of actual test was the scheduling of 5 pending jobs. Nodes in the model correspond to these physical hosts:

```
MODEL NODE ID REAL NODE ID
CORES AVAILABLE
1 8 64
2 13 60
3 23 60
4 27 64
5 1 60
6 10 64
7 28 64
8 2 60
9 21 60
10 29 64
11 3 64
12 12 60
```

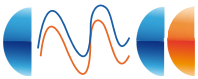
The jobs considered were 5 job pending, each requiring 256 cores. The energy optimization model found a solution after 175.72 seconds with objective function value 456 However, the makespan optimization model did not terminate its execution, owing to the fact that LSF killed the corresponding job after exceeding the the time limit associated with the run queue on which it was running.

Another instance involved the schedule created by 1 pending job. Nodes in the model correspond to these physical hosts:

```
MODEL NODE ID REAL NODE ID
CORES AVAILABLE
1 17 64
2 27 64
3 28 64
4 2 60
5 29 64
6 12 60
7 16 64
```

The energy optimization model found a solution after 0.11 seconds with objective function value 248 The makespan optimization model terminated after 0.09 seconds. We report here only a significant part of the solution found:

```
11 ms
```




```
s (1.5) 1
x (1,1,1) 1
x (1,3,1) 1
x (1,5,1) 1
x (1,7,1) 1
```

The planning returned a time horizon equal to 18 time slots. The solution found suggests that the job must start after 5 time slots (a slot is a time window of 4 hours). This is consistent with the status of nodes identified, which are nodes 3 and 5; the ready time of these nodes is equal to 5 time slots. We recall here that the ready time is the parameter that tells how much we must wait before a given node can be used. The conclusions we can draw about these experiments are fairly obvious. The model involves computation times that increase exponentially as the number of jobs increases. Therefore, in spite of the formal correctness of both models, we highlight the need to develop heuristics for the problem, since determining an optimal solution through the CPLEX solver is not a feasible alternative. One of the basic requirements is in fact the need to obtain a solution within a time window of no more than 30 minutes.

CONCLUSIONS

Scheduling is, in general, NP-Complete. This implies that, for large instances, the time required for an optimal solution increases exponentially. Because the instances we want to analyze are extremely large, a further development that can lead to faster solutions obtained in reasonable times is the aggregation of some variables. For example, instead of considering the total number of cores, we can consider packages of cores as if they were a single entity. Using this approach, we can follow two distinct directions:

- heuristic solutions: if the packet size is arbitrary;
- optimal solutions: if the packet size is obtained as the greatest common divisor (MCD_k) of the number of cores required by the jobs.

A similar simplification can also be done to correctly size the time slot. Indeed, considering the duration of all jobs, we can set the length of the slot as MCD_t of all job durations, still obtaining an optimal solution.