Bi-criteria Scheduling Algorithm with Deployment in Cluster

Feryal-Kamila Moulaï[1],Grégory Mounié [1]
[1]LIG*, Grenoble University, MOAIS project
51 rue Jean Kuntzmann
38330 Montbonnot St. Martin / FRANCE
moulai, mounie@imag.fr

Abstract

Computational grids clusters, provide powerful computing resources for executing applications of large scale. In Grid (clusters) usually several applications run simultaneously.The originality of Grid'5000 is that each application has characterized by its own specific requirement such as operating system (OS) or library components. Deploying the adequate OS needs to reboot the processors on which the application is executed. It is time-consuming and moreover frequent reboots may damage machines. In this work we investigate how to minimize the number of deployments, while keeping the running time as short as possible. We present the multiprocssors scheduling with déployement problem and provides a lis scheduling algorithm. The analysis details are presented in the worst case performence of the algorithm.

[1]

Because

1 Introduction/Motivation

Computational clusters are more and more used to fulfill computing needs of medium and small size organization. Problems, such as electric consumption, price and generated heat, grow with the size of the cluster. In addition some users use only sporadically their clusters. The interconnection of clusters is a way to share both computer costs and to a comput problems of a large scale using the whole set of clusters.

Grid'5000 project aims at building a highly configurable, controllable experimental Grid platform gathering 9 sites geographically distributed in France featuring a total of 5000 CPUs [6]. Grid'5000 belongs to a "novel" category of research tools for Grid research, it is highly reconfigurable, controllable and monitorable real life time platform. The prototype feture that Grid'5000 introduces is its degree of reconfiguration, allowing researchers to deploy and install the exact software environment they need for every experiment.

The core of any automatic operating system deployment is the bootstrap stage. It consists of starting a minimal operating system being able to write on the hard disk and to acces the network. The bootstrap downloads the system image and writes it on the disk. One simple way to run the bootstrap is to load it from the network too, using one of the standard ways such as TFTP or PXE. Kadeploy [7] is the grid5000 software stack in charge of the operating system deployment. Kadeploy is a fast and scalable deployment system towards cluster and grid computing. It provides a set of tools, for cloning, configure (post installation) and manage a set of nodes. Currently it deploys succefully linux, *BSD, Windows, Solaris on x86 and 64 bits computers. Deploying multiple computing environment on clusters or grids cannot be easily handled with current tools that are often designed to deploy a single system on each node. Nodes are rebooted during each deployment two times at least. On hardware with gigabit of memory, rebooting is a very long process, taking few minutes. On Gigabit networks, downloading the operating system image is less than 10 seconds. Thus whatever the operating system to install, the deployment time is strongly dominated by the reboot time [1]. Consequently, the time of installation does not depend much on the numer of nodes. In our experiments installing an operating system on one node requires 200 seconds, while installing an operating system on 100 processors is 300 seconds long. In this paper, the study is focused on Grid'5000's clusters.

A recent study of the availability of desktop machines within a large industry network [10], which is a typical large scale virtual PC farms targeted for global computing platforms in industry and university demonstrates that from 5% to 10% of the machines become unreachable in a 24

[1]This research was funded by SJW, grant number 1234567.

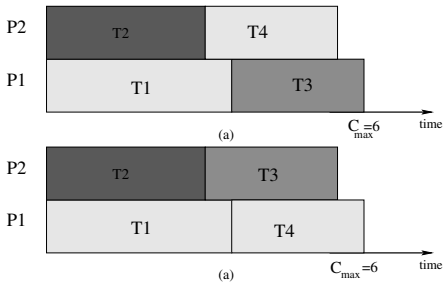

Figure 1. (a) Number of deployement $D_{sum} = 4$, (b) Number of deployement $D_{sum} = 3$

hour period. In the Grid'5000 the problem is similar. When the requests for deployment is important, a simultaneous interaction between the various servers can be a source of failure and the nodes become inaccessible for very long time.

Our goal is to achieve all together an efficient allocation of parallel tasks on the cluster and to maximize the avaibility of of the cluster by minimizing the number of environment deployment.

The *Multi-Processor Scheduling with Deployment* problem; *MPSD* is a bi-objective problem, which the objectives are to optimize both the completion time of tasks, i.e. the makespan C_{max}, and the number of deployment of operating systems D_{sum} necessary for the execution of the tasks.

Figure 1 details an example with 2 processors, 3 environments (represented by different geometrical patterns) and 4 tasks Each task must be executed on one processor,and needs an environment. Case (a) shows that if we do not take care about the number of deployment, the processor $P2$ makes two environments installation leading to the completion time $C_{max} = 6$. Case (b) shows that the makespan is the same when reducing the number of deployments i.e. $C_{max} = 6$ and the number of deployments is 3.

In some case, minimizing the number of deployment, can not change the makespan value or make this value larger or smaller.

Our goal is to find a strategy which makes it possible to minimize the makespan and the numbers of deployment.

In this paper we present the *Group List Scheduling* algorithm, *GLS*. Given an arbitrary parameter λ as a makespan, the algorithm can construct a schedule whose makespan is less than 4λ and a number of deployment less than $2D_\lambda$ where D_λ is the minimum number of deployments for given λ.

After a short recall of some fundamental results in rigid multiprocessors scheduling in related work section 3, we define the scheduling problem with deployment in section 4. We present in section 5 a bi-criteria algorithm for scheduling with deployement for multiprocessors rigid tasks and we prove its guarantee before concluding.

2 The Multiprocessors Scheduling Problem with Deployment

We consider a cluster architecture composed of m identical processors and a set of environments of work $E = \{e(T_i) = j \backslash j = 1, \cdots, s\}$. The system executes the set $\Gamma = \{T_1, T_2, .., T_n\}$ of independents tasks.

Each multiprocessor task T_i needs to be run on q_i processors with the environment $e(T_i) = j/j = 1, \cdots, s$. The processing time of T_i is p_i and the amount of work of each task is $p_i q_i = w_i$.

The operation of the system is assumed to be non-preemptive. A task must start on all the processors in the same time.

Before a processor proceeds to execute the task, the required environment must be installed.

The finishing time C_{max} of the scheduling is defined to be the time at which all tasks have been completed:

$$C_{max} = \max_{T_i \in \Gamma} C_i$$

where C_i is the completion time of task T_i for the scheduling. Let a deployment function $D : E \mapsto \mathbb{N}$, counts for each environment e the number of processor's deployment. Then the total number of deployment for scheduling is :

$$D_{sum} = \sum_{i=1}^{n} D(e(T_i))$$

The problem of scheduling computation on processors to minimize the total execution time of a program which consists of a set of independents multiprocessor tasks has been widely studied [9, 8]. Given m processors, the Multiprocessors Scheduling Problem with Deployment is to determine when and where every task is executed while optimizing the makespan and the number of deployment. As for most scheduling problem, optimal solutions are generally difficult to obtain unless $P = NP$ [3].

3 Related Work

We tried to model our problem on graphe representation. Such as, the tasks are represented by the vertex and the deployment by the edge. We realized, that graph representation can not take care about the allocation of the multiprocessors tasks. The particular case of multiprocessors tasks (malleables tasks) can be represented like a DAG, the interested reader can find more information on [8].

Like the majority of scheduling problems to optimize the makespan, our problem includes the traditional problem $P2||C_{max}$, which consists in scheduling sequential tasks on 2 processors and is equivalent to the well-know NP-Hard PARTITION problem. When the number of processor is

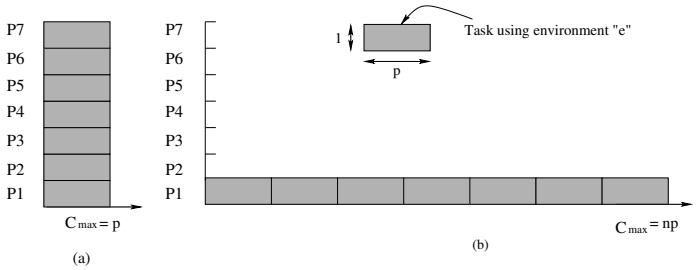

Figure 2. (a) Minimizing the makespan, (b) Minimizing the number of deployment

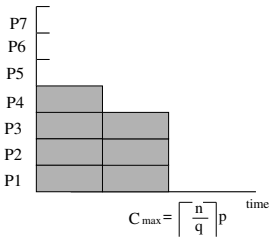

Figure 3. Goal : Minimizing both the makespan and the number of deployment

limited, to $m = 2$ or 3, this problem can be solved in a pseudo-polynomial time but for $m \geq 5$ it becomes strongly NP-Hard [2].

In practice, instances with large amount of data cannot be solved optimally in a reasonable time unless $P = NP$. That is why heuristics technics were designed. Among the first heuristic with performance guarante, Graham proposed List Scheduling [4, 5].

List scheduling [4] is a class of low complexity scheduling algorithms in which tasks are placed in a list, ordered as decreasing priority. The selection of tasks to be immediately processed is done on the basis of priority with the higher priority tasks executable being assigned processors first. [5] showed that for an instance using S resources, any list scheduling algorithm provides a solution more than $S + 1$ of the optimal solution. Our problem corresponds to the case using only one resource ($S = 1$), the processors.

4 The Idea of Trade-Offs

The problem of minimizing the number of deployments is an easy problem. To obtain the optimal solution it is sufficent to sort task by environment and schedule them sequentially. However, it does not minimize the makespan. In Figure 2, the instance is composed of n single-processors tasks, each one requiring p processing time and all tasks used the same environment. Minimizing the number of deployments is achieved by scheduling all tasks of a particular environment on a single processor. The makespan in the worst case is as worse n times the optimal makespan. The optimal makespan is achieved by scheduling all tasks of a particular environement on n processors.

The two criteria are antagonistic, the approche is to find a trade-offs between the criteria, such as find a "good" makespan for a "good" number of deployement, figure 3.

This article presents a scheduling algorithms which provide efficient trade-offs between the makespan and the number of deployments

4.1 Lower bounds

We denote by the parameter λ the makespan value which is taken as reference. A feasible scheduling is enough if it is larger than the size of the greatest task and the average of execution time of all tasks. $\lambda \geq C^*_{max} \geq \max_{1 \leq i \leq n} \left(\frac{\sum_{i=1}^{n} W_i}{m}, p_i \right)$

Let us denote D_λ our lower bound on the number of deployment. D_λ is a function of a given makespan. If we choose an arbitrary value λ for the makespan, for all environments, D_λ fulfills two properties:

- The task T_i of a particular environment e using the largest number of processors q^e_{max} must be scheduled, thus at least q^e_{max} deployments of environment e are done in any solution.
- All tasks of environment e must be scheduled before λ. Thus enough processors need to be provided in any solution with a makespan lower than λ.

Thus,

$$D_\lambda = \sum_{e \in E} \max \left(q^e_{max}, \left\lceil \sum_{T_i \in e} \frac{W^e_i}{\lambda} \right\rceil \right)$$

5 Bi-Criteria Algorithm

There is a trade-off between the number of deployments and the makespan, the greater the makespan, the smaller the number of deployments. The idea is to build a rectangle with length λ and width m and then schedule the tasks inside the rectangle trying to minimize the number of deployments.

List Scheduling Group, algorithm (LSG) is made of two phases. The first phase consists of building the groups. During the second phase those groups are scheduled. The tasks are partitioned according to their environment.

Two classes of groups are defined :

- The canonical groups whose class will be noted G^c
- The layered group whose class will be noted G^l

Each group g use q_g number of processors during p_g time units.

5.0.1 Layered Group

The tasks belonging to a layered group are scheduled sequentially in the order of decreasing number of processors. Each task of the group uses strictly more than half of the processors used by the first task of the group (cf Figure 4, part (a) and (b)).

If there is a layered group with processing time higher than 2λ, i.e. $\sum_{T_i \in g} p_i > 2\lambda$ then it is changed to canonical group.

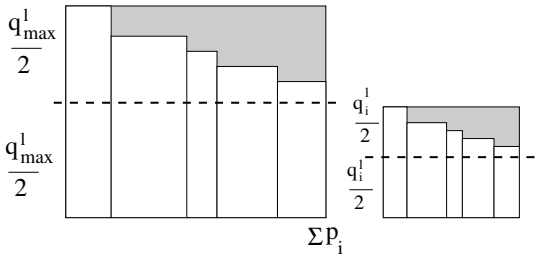

Figure 4. Layered Group

5.0.2 Canonical Group

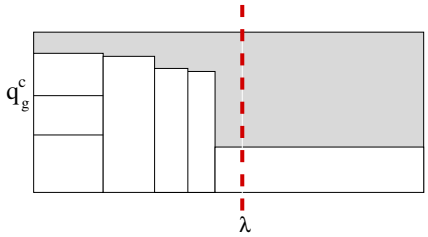

Figure 5. Canonical Group

We call a *canonical number of processors* (q_g^c) the number of processors necessary to execute the amount of work of a set of tasks in time lower than λ.

$$q_g^c = \left\lceil \frac{\sum_{E(T_k)=e} W_k}{\lambda} \right\rceil$$

A group is *canonical* if all the tasks of this group use less then q_g^c processors.

Then tasks can thus be scheduled on q_g^c processors by any list algorithm in time lower than 2λ [4].

It should be noted that any task using less q_g^c processors can be added to this canonical group, thus defining a new value for q_g^c.

5.1 The Group's Properties and Scheduling

Any group belonging to G^c or G^l has the following properties:

- **Property 1** : $\sum_{T_i \in g} W_i \geq \frac{p_g q_g}{2}$
- **Property 2** : $p_g \leq 2\lambda$ and $q_g \leq m$

The first property shows that the total amount of tasks's work requires more than half of the area of the each group. In the second, the processing time of each group has to be less than 2λ and the number of processors used by each group must be less m.

We seek to build the layered groups with different size those which use the majority of processors. Remaining tasks are gathered in the maximum canonical group. If the processing time of a layered group is longer than 2λ, it is converted to a canonical group.

Any layered group using fewer processors than a canonical group it is merged into a larger canonical group. By construction there cannot be more $\log_2 m$ layered groups by environment. Finaly, for each environment, at most one canonical group remains and all layered groups use more processors than the canonical group.
Indeed, each group can be seen like a rigid parallel Meta-task excuting the tasks of the group. Then, the algorithm schedules meta-tasks using the list scheduling algorithm on m processors.

Theorem 5.1 GLS *Algorithm constructs a schedule having a makespan less than* 4λ *time and the number of deployment less than* $2D_\lambda$.

5.1.1 Makespan' s approximation ratio

We will recall one of the [4] result for our case in order to highlight its properties which we used it to design our algorithm.

Theorem 5.2 *Any list algorithm, schedules a succession of independent rigid parallel tasks with* 2 *guarantee of performance for the makespan*

Proof 5.3 *Using the Graham's results in algorithm's construction, both of group respect the basic properties :*

Data: Set of Tasks $\Gamma = \{T_1 = (e(T_1), p(T_1), q(T_1)), T_2 = (e(T_2), p(T_2), q(T_2)), ..., T_n = (e(T_n), p(T_n), q(T_n))\}$
E_j Environments such as $j = 1, .., s$
m Number of processors
Result: Scheduling of tasks on m processors
begin

$\lambda \geq \max_{1 \leq i \leq m} \left(\frac{\sum^{n} W_i}{m}, p_i \right)$
Let $W_i = p(T_i)q(T_i)$ Amount of work using by the task T_i
$G^c \longleftarrow \emptyset$
$G^l \longleftarrow \emptyset$
for *j = 1 to s* **do**
  $\Gamma_{E_j} = \{T_i / e(T_i) = j\}$
for *each* Γ_{E_j} **do**
  Sort tasks according to decreasing number of processors:
  $\Gamma_{E_j} = \{T_1, T_2, \cdots, T_r\}$ such as $q(T_1) > q(T_2) > \cdots > q(T_r)$
  $BeginLayered \longleftarrow T_1$
  $EndLayered \longleftarrow T_1$
  $LayeredGroup = [BeginLayered, EndLayered]$
  $p_{Layered} = p(T_1)$
  $G_j^l \longleftarrow \emptyset$
  while $EndLayered \neq T_r$ **do**
    if $p_{Layer} \leq 2\lambda$ **then**
      if $q(EndLayered + 1) > \frac{1}{2} q(BeginLayered)$ **then**
        $EndLayered++$
        $p_{Layered} = p_{Layered} + p(EndLayered)$
      else
        $G_j^l \longleftarrow G_j^l \cup \{[BeginLayered, EndLayered]\}$
        $BeginLayered \longleftarrow EndLayered + 1$
        $EndLayered \longleftarrow BeginLayered$
        $p_{layered} = p(BeginLayered)$
    else
      $G_j^c \longleftarrow \{[BeginLayered, T_r]\}$
      $q_{canonical} = \left\lceil \frac{\sum_{T_i \in G_j^c} W_i}{\lambda} \right\rceil$
      for *All* $LayeredGroup \in G_j^l$ *in reverse order* **do**
        if $q_{canonical} > q(BeginLayered)$ **then**
          Add all tasks of $LayeredGroup$ in G_j^c
          $G_j^l \longleftarrow G_j^l \backslash LayeredGroup$
          $q_{canonical} = \left\lceil \frac{\sum_{T_i \in G_j^c} W_i}{\lambda} \right\rceil$
        List Scheduling of tasks in G_j^c using $q_{canonical}$ processors
    Let G_j^l set of MetaTasks such that $MetaTask = ([BeginLayered, EndLayered], p_{layered}, q_{Layered})$;
  $G^c \longleftarrow G^c \cup G_j^c$
  $G^l \longleftarrow G^l \cup G_j^l$
  List Scheduling of $MetaTask$ in G^l, G^c using m processors
end

Algorithm 1. GLS Algorithm

- $\max p_{g_i} \leq 2\lambda$
- $\sum_{T_i \in g} \frac{W_i}{m} \leq 2\lambda$

And $C^*_{max} \leq \lambda \leq 2C^*_{max}$ Let the optimal makespan for an instance with Meta-task $MetaC^*_{max} = 2C^*_{max} \leq 2\lambda$. And now consider the new scheduling problem of a rigid multiprocessors Meta-tasks i.e. groups.

We will show this result by contradiction. Let $q(t)$ amount of processors used by the Meta-tasks at time t, one of the consequences of list algorithms is that

$$\forall t_1, t_2 \in [1, MetaC_{max}) \quad t_1 \leq t_2 - MetaC^*_{max}$$

$$q(t_1) + q(t_2) > m \tag{1}$$

It should initially be noted that the processing time of all the meta-tasks are smaller than $MetaC_{max}$, if not, optimal scheduling could not schedule them on time $MetaC^*_{max}$. If the equation 1 were false, the Meta-tasks scheduled at the moment t_2 could all be scheduled earlier at the moment t_1.

Let us suppose that the makespan of a meta-tasks is $MetaC_{max} > 2MetaC^*_{max}$.

The optimal having scheduled the tasks on m processors in a time equal to $MetaC^*_{max}$, we can write:

$$\int_0^{MetaC_{max}} q(t)dt \leq mMetaC^*_{max}$$

what is equivalent to

$$\int_0^{2MetaC^*_{max}} q(t)dt + \int_{2MetaC^*_{max}}^{MetaC_{max}} q(t)dt \leq mMetaC^*_{max}$$

$$\int_0^{MetaC^*_{max}} \underbrace{(q(t) + q(t+1))}_{>m} dt + \int_{2MetaC^*_{max}}^{MetaC_{max}} \underbrace{q(t)}_{>0} dt \leq mMetaC^*_{max}$$

The second term is strictly positive and according to the equation 1 the first term is strictly higher than m which is a contradiction. Thus $MetaC_{max} \leq 2MetaC^*_{max} \leq 4\lambda$

5.1.2 Number of deployement's approximation ratio

Suppose we use $D_{sum} = \sum_{e \in E} D^e_{sum}$ to denote the number of deployments of a scheduling given by the GLS *algorithm and D_λ to denote the minimum number of deployment for given λ.*

First, if only one group remains i.e. canonical group for each environment then :

$$D_{sum} = \sum_{e \in E} \sum_{T_i \in e} \left\lceil \frac{W^e_i}{\lambda} \right\rceil = \sum_{e \in E} q^e_c$$

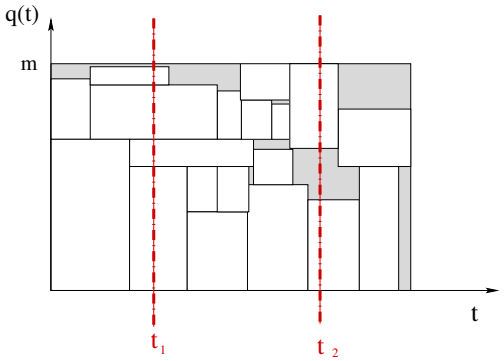

Figure 6. Graham resources constraints

and second,

$$D_{sum} = \sum_{e \in E} q^e_{max}$$

since this lower bound represents the best possible situation in which for each environment at the maximum, we deploye a number of processors used by the largest task.
In the general case, the single canonical group uses fewer processors than the smallest layered group, then :

$$D^e_{sum} \leq q_{max} + \frac{q_{max}}{2} + \frac{q_{max}}{4} + \ldots + \frac{q_{max}}{2^i} + q^e_c$$

$$D^e_{sum} \leq q_{max} + \frac{q_{max}}{2} + \frac{q_{max}}{4} + \ldots + \frac{q_{max}}{2^i} + \frac{q_{max}}{2^{i+1}}$$

$$D^e_{sum} \leq 2q^e_{max}$$

$$\sum_{\forall e} D^e_{sum} \leq 2 \sum_{e \in E} q^e_{max}$$

$$D_{sum} \leq 2D_\lambda$$

6 Conclusion and perspectives

We presented in this article the problem of scheduling Bi-criterion of independent, rigid and multiprocessors tasks. Before execution, each task required an environment deployment on their dedicated processors. We considered two criteria to optimize, the date of total termination i.e. makespan and the number of deployments (OS reboot).

We presented a polynomial algorithm GLS in two phases: The first phase consists of building the groups. During the second phase those groups are scheduled. The tasks are partitioned according to their environment. We built

groups of tasks which check Graham's properties [4]. We schedule these meta-tasks i.e. groups using the list scheduling on m processors.

Given an arbitrary parameter λ as a makespan, the algorithm can construct a schedule whose makespan is less than 4λ and a number of deployment less than $2D_\lambda$ where D_λ is the minimum number of deployments for a given λ.

One of the important issues for this work consists to design the "on-line" scheduling in order to be able to establish this type of scheduler in Grid 5000. It would also be necessary to evaluate its performances on average with realistic models of tasks. The first experiments of deployment in Grid 5000 being held for a few months on a large scale, it has seemed possible to design an approximate model of the tenders.

Our execution model supposes that one installs only one system at the same time. On the Grid 5000 platform it is possible to install several systems simultaneously. That takes a little more time, but saves phases of reboot. From this point of view of Nodes having several states simultaneously, it is the number of reboot which becomes probably the good criterion.

Lastly, it will be probably possible to consider also another criteria : the average time of termination of each task and to build an algorithm tri-criterion thus.

References

[1] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Vicat-Blanc-Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, B. Quetier, and O. Richard. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing CD*, pages 99–106, Seattle, Washington, USA, nov 2005. IEEE/ACM.

[2] J. Du and J.-T. Leung. Complexity of scheduling parallel task systems. *SIJDM: SIAM Journal on Discrete Mathematics*, 2, 1989.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.

[4] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal (BSTJ)*, 45:1563–1581, 1966.

[5] R. L. Graham. Bounds on certain multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, mars 1969.

[6] Grid5000. Grid5000. http://www.grid5000.fr, 2006.

[7] Kadeploy. Kadeploy. http://kadeploy.imag.fr, 2006.

[8] J. Y.-T. Leung. *Handbook of Scheduling*, chapter 25,26. CRC Computer and Information Science Series. Chapman and Hall, 2004.

[9] G. M. and R. L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SICOMP: SIAM Journal on Computing*, 4, 1975.

[10] W.J.Bolosky, J.R.Douceur, D.Ely, and M.Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. *In Proceedings of the ACM international conference on Measurement and modeling of computer systems SIGMETRICS*, 2000.