Tasks Mapping with Quality of Service for Coarse Grain Parallel Applications

P. Pascal, S. Richard, B. Miegemolle, and T. Monteil
ppascal@laas.fr, srichard@laas.fr, bmiegemo@laas.fr, monteil@laas.fr

LAAS-CNRS, 7 Avenue du Colonel Roche 31077 Toulouse France

Abstract: ***A new mapping algorithm, dealing with the notion of quality of service is proposed. Four classes of service are treated: deadline applications, high priority applications with immediate execution, dedicated resources applications and best effort applications.***
The mapping problem inputs are the application to map, all the unfinished applications that have been previously mapped with their classes of service, the characteristics of the applications (application model), the state of the execution support load, the processor access model (round robin model) and the equations to predict the end of execution. According to all this information, the scheduler finally finds an on-line mapping which respects all the expressed constraints. Two criteria are used to find the solution: the release date of all processors and the memory space used. To finish, the validation of the algorithm is performed with real log files entries simulated with Simgrid.
Keywords: scheduling, quality of service, resource manager, grid, clusters

1 Introduction

In distributed systems, resource management is very important in order to optimize their utilization. The common policy used to map tasks on computers is best effort. To differentiate users, different queues are created: short jobs, long jobs, high parallel jobs, etc and each queue has a FIFO or a more elaborated policy. This limitation is due to historical reasons because batch schedulers have been created for parallel computers. Clusters are more complicated, they have multi-tasking scheduler and can be multi-user. They are connected with network on which different policies of quality of service are possible. For this reason, batch schedulers are not well suited to deal with the notion of quality of service. In this article, a scheduling algorithm used in distributed systems like clusters or aggregation of clusters (grid) which implements different classes of services, is presented. This algorithm is implemented in a tool called AROMA (scAlable ResOurces Manager and wAtcher) [4]. AROMA integrates a resource management system, an application launcher, a scheduler, a statistic module and an accounting system.
In the first section, a state of the art will be presented. In the second section, the context of the study, the notion of quality of service and AROMA will be detailed. Next, the optimization problem that has been solved will be explained. To conclude, first results validating the proposed algorithm will be given. The

originality of this work is to mix different processes from different classes of service on the same processor at the same time.

2 Related work

Batch and dynamic scheduling must solve a difficult problem because resources needed by an application may not be known. Resources are heterogeneous and their availability knowledge is incomplete. Moreover, the mapping algorithm must run quickly. Heuristics with good properties are used. Henri Casanova in [5] studies deadline scheduling on computational grid. His goal is to minimize the overall occurrences of deadline misses as well as their magnitude.
Rajkumar Buyya in [6] proposes a deadline and budget constrained cost-time optimization algorithm for scheduling on grids. The algorithm is called DBC (Deadline and Budget Constrained).
Mechanisms have been created to improve the mapping:

- Different types of resource reservation algorithms are studied in [1]. They evaluate the performance with or without preemption. The reservation insures that all the resources necessary to run the applications will be free.
- The gang-scheduling creates time slices. Those parts of time are allocated to the parallel and sequential applications [2]. With this solution, all applications progress simultaneously. Nevertheless, it could create a problem of overload and memory saturation.
- The backfilling [3] allows the insertion of jobs into scheduler queue. The insertion is possible, if it does not perturb the other jobs. It is a way to remove the holes in resources utilization.

This article proposes a way to mix jobs requirements with different quality of service (deadline, immediate execution, dedicated resources).

3 Mapping Algorithm

As the context of the study is ASP (Application Service Provider), the hypothesis that all the applications running on the sites are known is made. That is to say, hosts are considered dedicated to computation. All the jobs are submitted through AROMA and system tasks are neglected. AROMA daemons also know if an application previously mapped is still running. This is used to refresh estimated completion date of running jobs. A second hypothesis is that applications are supposed to be regular coarse grain parallel applications for which the time spent in communications is small and the execution time can be predicted. The problem is to propose a mapping of a new application with the knowledge of all the previously mapped applications that are not completed yet. This mapping must guarantee that quality of service is respected for all applications (running applications or applications to be scheduled).

3.1 Quality of service and mapping problem

Applications are grouped into four application classes (in order of importance):

- **Deadline applications (class 1):** this class of service guarantees that the execution will finish before the deadline. Execution can be immediate or deferred.
- **High priority applications (class 2):** this class of service guarantees that the execution will be immediate.
- **Applications with dedicated resources (class 3):** this class of service guarantees that each application will be the only one to use resources during its execution. Execution can be immediate or deferred.
- **Applications without constraint (class 4):** this class of service corresponds to applications which will be executed as soon as possible with available resources. This class is also named "Best Effort".

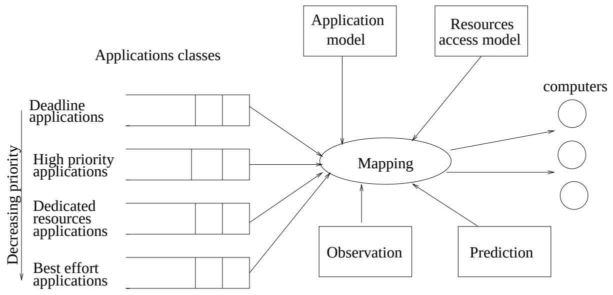

Fig. 1. The mapping problem

The mapping problem inputs are (figure 1):

- **Application models:**
  Applications models are necessary to make a good mapping. When an application is submitted, information is given to the scheduler, some are inputs of the algorithm others are constraints for the mapping. An estimation of the cpu time required by each task, the number of tasks and the size of exchanged data are computed. Those values can be given by the user or retrieve from a database containing information on previous runs for the same type of application. The application model can express additional constraints like all the tasks must begin at the same date or temporal relations between the tasks (classical description in graph theory).
  Software or specific hardware requirements add constraints on the execution host. Class 1 induces a constraint on the termination date of the application, class 2 induces a constraint on the starting date of the application and class 3 induces a constraint on the execution hosts.
  All the constraints are verified by the algorithm.

- **Resources access models:**
  According to the access policy to the resources, equations are deduced and used to predict the utilization time of the resources and the end of execution. Models developed in this article are deterministic, nevertheless models which take care of random perturbation (arrival of uncontrolled jobs, for example a direct login on the host) have been developed in [12]. The difficulty is to mix the different applications in different classes of service on the same host with the respect of the constraints: more than one task can be run at the same time on the same host.
- **Observations:**
  The processor and network load (percentage of processors utilization, bandwidth used), idle memory space and number of processes are monitored. They are used to update information for mapping.
- **Mapping:**
  Several queues exist, each corresponding to a priority level (figure 1). When several applications must be mapped, the jobs of the queue with the higher priority will be first mapped. If there is no solution that respects the constraint, three cases are studied:
  - the algorithm searches an application in a lower priority queue which has been plan to be run in the future, then it removes it and try to map it again after the current mapping.
  - if the previous case is impossible, the algorithm can stop a running application and try to find a new mapping for this application. In this case, new constraints are created to express that this application must continue later on the same host. There is no migration.
  - if the two previous cases are impossible, the mapping is rejected.

3.2 Mathematical expression of the problem

The variables

- t_0: initial date of the mapping research.
- $\mathrm{t}_b(a,p,m)$: starting date of execution of the task p of the application a on the host m.
- $\mathrm{t}_{fwn}(a,p,m)$: end of execution of the task p of the application a on the host m when the network is neglected.
- $\mathrm{t}_f(a,p,m)$: end of execution of the task p of the application a on the host m when an estimation of time spend on communication is done.
- $\mathrm{t}_c(a,p,m)$: processor time requested by the task p of the application a on the host m. Coarse grain applications are considered so the "small" communication time is taking into account and the synchronization time is neglected.
- $\mathrm{t}_c^k(a,p,m)$: remaining processor time for the task p of the application a after the event number k on the host m (events are : a beginning of a task or the end of a task).
- $\mathrm{D}_r(a,p)$ estimation of the size of data send and receive by the task p of the application a.

- C(m) : coefficient to take care of the heterogeneous processors.
- $t_c(a,p,m) = t_c(a,p)$ * C(m) : equivalence of processor time requested by the task p of the application a for the host m.
- B(i,j) : estimation of bandwidth of the network between host i and host j.
- M : number of hosts.
- A : number of applications to map.
- N^a : number of tasks of application a.
- $X_m(t)$: number of processes on the host m at time t.
- t_m^k : a beginning event or an ending event of a process on the host m. It is the event number k.
- P(a) : set of possible mapping for application a.
- $t_f(m,s)$: the end of all the tasks mapped on the machine m after the mapping s of application a with $s \in$ P(a).
- $M(t(a,p))$: the machine on which the task p of the application a is executed.
- $M_u(m)$: total memory used on host m.
- $M_t(m)$: total memory on host m.
- M_f: constant to modify the weight of the memory criteria.

The optimization criteria: The mapping problem is an optimization problem, criterion has to be chosen. Several criteria are well known ([7][8][9][10]) : makespan (minimizing the termination date of an application), sum-flow (minimizing the quantity of resources used), max-stretch (it expresses that a task has been slowed compared to what would have been its execution on an idle server). The objective is to optimize the use of the providers resources and to guarantee the level of quality of service required. So it has been chosen to liberate all the resources as soon as possible. Applications already mapped may be influenced by the mapping found. By consequence, all the applications (the currently mapped and the previously mapped) must finish as soon as possible.

$$\min_{s \in P(a)} (\max_m (t_f(m,s) + t_f(m,s) * M_u(m) * M_f / M_t(m))) \tag{1}$$

The date of the end of resources utilization is optimized. Moreover a second criteria consists to moderate with memory space used: hosts which have the most free memory space are privileged first. The choice has been made to introduce memory in criteria because the exact amount of memory requested by an application is often unknown by users. The problem is multi-criteria by using a linear combination of different criteria (1). The first part of the addition ($t_f(m,s)$) refers to the release date of the machine. The second part of the addition $t_f(m,s) * M_u(m) * M_f/M_t(m)$ penalizes machines which have less free memory. $M_u(m)/M_t(m)$ gives an idea of memory utilization on this host. The coefficient M_f influences the weight of this part of criterion. The multiplication with $t_f(m,s)$ puts the value into the same order of value as the first part of the criterion.

The mapping algorithm: A list algorithm for applications, tasks and machines is used to reduce the combinatorial. The mapping of an already studied task of an application a is revised only if it is impossible to find a mapping for this application. Moreover, in order to reduce the time used by the algorithm, for each host, release date is saved and hosts are ordered to study which of them will give the best mapping first.
The quality of service is already respected, all the constraints induced are verified by the algorithm. If it is impossible to find a mapping corresponding to the demand, the request is refused.
The computation of $t_b(a,p,m)$ and $t_f(a,p,m)$ will now be explained. The equations are found considering the processor access follows a round robin policy.

The starting date of a task : $t_b(a,p,m)$ is computed with an iterative algorithm (at the beginning it is t_0). If at this date no mapping can be found, another date is searched.

$t_b^0(a,p,m) = t_0$

$t_b^{k+1}(a,p,m) = \min t_f(i,j,m)$

with $i \in [1, a-1]$, $j \in [1, N^i]$ and $t_f(i,j,m) > t_b^k(a,p,m)$ (only the tasks which can end after the last t_b studied, are considered). The idea is to search a new starting date when the system is less loaded: when a job finishes.

The end of a task : $t_f(a,p,m)$ is computed with an iterative algorithm. Each date is studied when there is a creation of a job or a termination. t_m^0 corresponds to the arrival of the first process on the host.

```
for all tasks ($a$ *goes from 1 to* $A$ *and* $p$ *from 1 to* $N^a$, *on* $m$)
 if $X_m(t_m^k) > C$ (*is there more processes than processors ?*)
     $t_m^{k+1} = \min_{a,p} (t_b(a,p,m), t_c^k(a,p,m) * X_m(t_m^k) + t_m^k)$ (*next event corresponds to a creation or a death of a process*)
 else
     $t_m^{k+1} = \min_{a,p} (t_b(a,p,m), t_c^k(a,p,m) + t_m^k)$ (*next event corresponds to a creation or a death of a process*)
 if $t_b(a,p,m) > t_m^k$ and if $X_m(t_m^k) > C$
     $t_c^{k+1}(a,p,m) = t_c^k(a,p,m) - C * (t_m^{k+1} - t_m^k) / X_m(t_m^k)$ (*estimation of the new requested time of processor for this task after this short execution on processor: it is the time sharing policy*)
 if $t_b(a,p,m) = t_m^i$
     $t_c^i(a,p,m) = t_c(a,p,m)$
     (*it is case of an insertion of a new process in the recurrence*)
```

estimation of the finished date of process without the network:
  if $t_c^{k+1}(a,p,m) = 0$, $t_{fwn}(a,p,m) = t_m^{k+1}$

Time spend in communication are put inside the estimation of the end of process to advantage location of tasks on the same cluster or on the same site because the bandwidth will be better:

$$t_f(a,p,m) = t_{fwn}(a,p,m) + D_r(a,p)/\min_{i\in[1,a[} B(i,a)$$

The iterations continue until all the tasks of the host finish. In fact, the mapping algorithm quickly simulates the execution of jobs and can mix the different classes of services.

4 Validation

To validate the algorithm, Simgrid ([11]) simulator has been used. Real jobs submission log files have been used to estimate the behavior of the algorithm with Simgrid. Nevertheless, it is difficult to compare the algorithm to others because algorithms found do not define classes of service and do not execute processes of the different classes at the same time on the same processors.

4.1 Comparison with NQS

Feitelson logs (real logs) have been used. These logs give : the submission date of jobs, the required cpu time, the number of tasks. The log file *l_sdsc_sp2.swf* [13] is used. This Job Trace Repository is brought by the HPC Systems group of the San Diego Supercomputer Center (SDSC), which is the leading-edge site of the National Partnership for Advanced Computational Infrastructure (NPACI) [14], [15]. The real system has 128 nodes and is scheduled with NQS [16]. Jobs submissions are reproduced, the mapping research is done with the algorithms on 128 nodes and their execution is simulated with Simgrid. The mean waiting time given by the logs are compared to the mean waiting time of the algorithms with quality of service. So, 10000 and 35000 jobs are simulated. Each job was synchronized, this means that all the tasks of the same parallel application must begin at the same date and belong to the dedicated resources class. This class seems to be the nearest from NQS policy. The simulation of 10000 jobs is equivalent to an activity on the supercomputer during 112 days. The simulation of 35000 jobs is equivalent to an activity on the supercomputer during 249 days. There is no information about the communications, so they are neglected. The results show that, with the proposed algorithm, a better waiting time than NQS is obtained. In fact, with the proposed algorithm, an application can be mapped between two others because the processor time required is known and the prediction of the end of each task mapped can be done. This reduces significantly the waiting times. NQS sorts the applications into queues (based on required cpu time) and mixes the applications when a mapping is researched. This can increase the waiting time of short jobs which can be slowed by long jobs. This problem is avoided with the proposed algorithm because the applications are considered in the order of submission and classes. The second reasons is that the proposed algorithm uses more precise values for the requested time of CPU than NQS so the mapping is more accurate. The comparison would be more fair if NQS had as much queues than the different time of processor requested. The results are presented in the table 1. The times are given in seconds (s).

	10000 events	35000 events
mean waiting time with NQS(s)	10796	8979
mean waiting time with the algorithm(s)	4008	4202

Table 1. Comparison of the waiting time between NQS and the proposed algorithm

4.2 Influence of quality of service

The same log file as previously has been used. In this log file, it is specified that there are four queues (low, normal, high, express), and for each job the queue of submission is known. So this information is used to make an arbitrary correspondence with the proposed classes of applications. The logs are used only to have an approximation of a realistic incoming rate of applications and a good sample of applications requested cpu time. So queue *low* corresponds to best effort class (class 4), queue *normal* corresponds to deadline class (class 1), queue *high* corresponds to dedicated resources class (class 3) and queue *express* corresponds to high priority class (class 2). For deadline applications (class 1) the deadline is : submission date + 5*cpu time required. For high priority applications (class 2) the starting date that must be respected is : submission date + 5 seconds. Four cases have been simulated:

- case 1 (10000 events using the four classes) : the waiting time for each class, the global waiting time, the mapping time for each class and the global mapping time are computed.
- case 2 (10000 events in the best effort class) : the global waiting time and the global mapping time are computed.
- case 3 (35000 events using the four classes) : the waiting time for each class, the global waiting time, the mapping time for each class and the global mapping time are given.
- case 4 (35000 events in the best effort class) : the global waiting time and the global mapping time are evaluated.

Doing so, case 1 can be compared with case 2 and then case 3 can be compared with case 4 to see the influence of the quality of service on the mapping performances. The results are presented in the table 2. The waiting times are given in seconds (s) and the mapping times in milliseconds (ms).

The introduction of the quality of service increases the mapping times and waiting times but the algorithm still has good performances (mapping time is

	Case 1	Case 2	Case 3	Case 4
waiting time class 1 (s)	30.51		30.7	
waiting time class 2 (s)	0.0036		0.003	
waiting time class 3 (s)	4845		5182	
waiting time class 4 (s)	4.55		30.5	
global waiting time (s)	710.5	64.68	396	70.3
mapping time class 1 (ms)	20.22		21.1	
mapping time class 2 (ms)	3.6		3	
mapping time class 3 (ms)	269.3		334.4	
mapping time class 4 (ms)	7.2		28.6	
global mapping time (ms)	50.3	13	49.1	76.4

Table 2. Influence of the quality of service on the performances of the algorithm

inferior to 80 ms). The mapping time increases because there are more constraints to verify. It takes many iterations before finding the good t_b. Globally the increase of waiting time is due to class 3 applications (dedicated resources applications) which have to wait a long time before being alone on the machines. The economical or political criteria, which define important jobs, could depreciate the global utilization of resources. In case 1, there are 21 rejects of applications belonging to class 1 (deadline). In case 3, there are 154 rejects of applications belonging to class 1 (deadline). In those cases the machines are full. Because of the constraints of deadline, the tasks can not start later like in classical batch schedulers. Applications of class 2 can be refused but a mapping has always been found. Applications of class 3 and 4 can never be refused, they will only be slowed down.

5 Conclusion

This article presents a scheduling algorithm with quality of service usable in distributed systems like clusters or grids. It is implemented in AROMA : a resource management system used in ASP model. The validation shows that the proposed algorithm is better than NQS, nevertheless, the comparison is difficult because NQS does not implement classes of service and has not access to the same information.
The algorithm always respects the quality of service required for accepted applications. When different applications are mixed, the mapping and the waiting time are more important than when there are only best effort applications. The performances of the algorithm are still very good (mapping time inferior to 80 ms). The communication weight are introduced to favor the execution of an application in the same area network.

Future work will be to improve the notion of communication model and its use into the application models. The theoretical complexity of the mapping algorithm will be studied. The main difficulty will be to explore the complexity of the estimation of the end of task. More comparisons with other mapping algorithms will be done. Real execution of a set of jobs during many weeks will be done on a small grid over 3 different sites.

References

1. Warren Smith, Ian Foster and Valerie Taylor. *Scheduling with advanced reservations.* Proceeding of the IPDPS Conference, May 2000.
2. Dror G. Feitelson and Morris A. Jette. *Improved utilization and responsiveness with gang scheduling.* Proceeding JSSPP 1997 : 238-261. Job scheduling strategies for parallel processing, IPPS'97 workshop, Geneva, Switerlang.
3. Dmitry Zotkin and Peter J. Keleher. *Job-length estimation and performance in backfilling schedulers.* 8th Intl Symp. High Performance Distributed Comput., august 1999.
4. P.Bacquet, O.Brun, J.M.Garcia, T.Monteil, P.Pascal, S.Richard. *Telecommunication network modeling and planning tool on ASP clusters.* Proceedings of the International Conference on Computational Science (ICCS'2003) Melbourne, Australia, June 2-4, 2003.
5. Atsuko Takefusa, Satoshi Matsuoka, Henri Casanova, Francine Berman. *A study of deadline scheduling for client-server systems on the computational grid.* Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco, California, August 7-9, 2001.
6. Rajkumar Buyya. *Economic-based distributed resource management and scheduling for grid computing.* Thesis, April 2002.
7. T.L. Casavant and J.G. Kuhl. *Effects of Response and Stability on scheduling in distributed computing systems.* IEEE Transactions on software engineering, vol. 14, No 11, pp. 1578-1588, november 1988.
8. Y.C. Chow, W.H. Kohler. *Models for dynamic load balancing in a heterogeneous multiple processor system.* IEEE Transactions on computers, vol. c-28, No 5, pp. 354-361, 1979
9. C.Y. Lee. *Parallel machines scheduling with non simultaneous machine available time.* Discrete Applied Mathematic North-Holland 30, pp 53-61, 1991.
10. F. Bonomi and A. Kumar. *Adaptative optimal load balancing in a non homogeneous multiserver system with a central job scheduler.* IEEE Transactions on computers, vol. 39, No 10, pp. 1232-1250, october 1990.
11. Henri Casanova, Arnaud Legrand and Loris Marchal *Scheduling Distributed Applications: the SimGrid Simulation Framework.* Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03).
12. Patricia Pascal and Thierry Monteil. *Influence of Deterministic Customers in Time Sharing Scheduler.* ACM Operating Systems Review, 37(1):34-45, January 2003.
13. http://www.cs.huji.ac.il/labs/parallel/workload/logs.htmlsdscsp2
14. http://joblog.npaci.edu/
15. http://www.cs.huji.ac.il/labs/parallel/workload/
16. B. Kingsbury. *The network queuing system.* 16 May 1998. http://pom.ucsf.edu/srp/batch/sterling/READMEFIRST.txt.