Proceedings of the 6th WSEAS International Conference on Applied Informatics and Communications, Elounda, Greece, August 18-20, 2006 (pp28-33)

Data Management Policies and Scheduling in Grid Computing

MASSIMILIANO CARAMIA, STEFANO GIORDANI
Dipartimento di Ingegneria dell'Impresa
Università di Roma "Tor Vergata"
Via del Politecnico, 1 – 00133 Rome
ITALY

Abstract: - Grid computing is emerging as a new paradigm for solving large-scale problems and is becoming an established technology for providing transparent access to large-scale distributed computational resources. Resource allocation and application scheduling are two of the most important aspects of Grid computing. In general, a grid application also requires datasets that may not be available at the local computing site where the application has to be executed, and hence in this case the required data has to be fetched before running the application. In this paper, we tackle with the local scheduling problem by means of a rectangle packing model combined with different policies for dataset scheduling, with the aim of maximizing the system efficiency.

Key-Words: - Grid scheduling, dataset policies, rectangle packing, on-line algorithms, experimental analysis

1 Introduction

Recently, there has been an increasing interest in availing distributed computer systems for very large-scale computing purposes. Grid computing is emerging as a new paradigm for solving large-scale problems and is becoming an established technology for providing transparent access to large scale distributed computational resources. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing. Resources site contains, in general, cluster commodity computers consisting of PCs or workstations interconnected by vendor-independent networks that are connected to the processing elements through PCI ports. In such a framework two of the most important issues are the efficient allocation of computational resources to user application and the scheduling of such applications on the allocated resources.

Grid computing has been extensively studied in the past. In particular, many successful strategies have been developed for scheduling applications on the Grid: examples include the AppLeS project [1, 2, 3], and the GrADS project [4]. Moreover, application tools for Grid program development were also developed (e.g., see [5]).The typical mechanism of Grid is as follows [6]: a user submits an application (task) request. Grid plays the role of finding and allocating feasible resources (computers, storages) to satisfy the request of the user. Then, it monitors the correct task processing, and notifies the user when the results are available.

One of the most known Grid model is the one introduced by Ranganathan and Foster in [7]. In this architecture, users submit requests for application execution from any one of a number of sites. At each site, besides the local computing system, the system model is composed by three components: an External Scheduler (ES) responsible for determining a particular site where a submitted task can be executed; a Local Scheduler (LS), responsible for determining the order in which tasks are executed at that particular site; a Dataset Scheduler (DS), responsible for determining if and when to replicate data and/or delete local files.

In general, on receipt of a task request, the ES interrogates the LSs to ascertain whether the task can be executed on the available resources and meet the user-specified due date. If this is the case, a specific site in which executing that task is chosen. Otherwise, the ES attempts to locate a LS of a site, controlled by another ES, that can meet the task processing requirements, through search mechanisms. If a LS cannot be located within a preset number of search steps, the task request is either rejected or passed to a scheduler that can minimize the due date failure depending on a task request parameter. When a suitable site is located, the task request is passed from the ES to this site and is managed by the associated LS.

In this paper, we focus the attention on the local scheduling problem and we extend the analysis of the scheduling model and algorithm presented in [8] to be addressed by each LS by considering also the effect of some dataset policies adopted by a DS for managing the dataset locally available. We suppose

that a group of applications (independent tasks) have to be executed on a given local machine cluster (site). The cluster is assumed to be formed by a number of processing nodes and a limited storage. Tasks are not known in advance by the LS, and are assumed to be presented one by one to the LS according to the over time paradigm. Therefore, the task characteristics (e.g.: duration, number of allocated (required) processing nodes, required dataset) become known to the LS when the task is presented to it.

We model the scheduling problem as (on-line) rectangle packing (e.g. see [9]) consisting in orthogonally packing a subset of a set of rectangular-shaped boxes, without overlapping, into a single bounding rectangular area, maximizing the ratio between the area occupied by the packed boxes and the area of the bounding rectangle. We provide an on-line algorithm for the rectangle packing that whenever a new box (task) arrives as to decide to pack it in the rectangular bounding area or reject it The algorithm is coupled with dataset policies. Indeed, the effective duration of a task comprises also the time needed to load the required dataset if it is not present in the local site, therefore the dataset policy adopted may affect the performance of the system. We experiment different dataset policies and evaluate the performance of the algorithm on different Grid scheduling scenarios.

The following sections are organized as follows. Section 2 describes the Grid scheduling framework; Section 3 the scheduling model and the algorithm. Finally, Section 4 reports on the data management policies and the experimental results.

2 The Grid scheduling framework

We refer to the Grid scheduling framework introduced in [7], that is depicted in Figure 1, in which the scheduling logic is encapsulated in three modules:

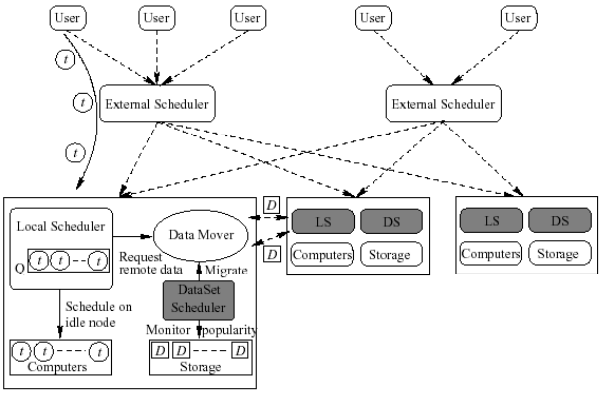


Fig. 1. The Grid framework.

- *External Scheduler* (ES): Users submit tasks to the ES they are associated with. The ES decides to which site tasks must be sent. To this aim a *JobLocal* discipline [7] is considered, with which each task is dispatched to the local site (domain) where it was submitted by the user. Moreover, the ES should manage the rejected tasks by a LS, sending them to another one or deciding to reject them definitely from the system.
- *Local Scheduler* (LS): Once a task is assigned to a particular site (and sent to a queue), it is then managed by the LS. Each LS manages the set of processors (computing units) of the relative site, within a given time window that represents their availability. The LS attempts to schedule each task dispatched to the site, within the time window, otherwise it rejects it and sends it back to the ES.
- *Dataset scheduler* (DS): The DS manages the datasets locally available. Datasets are preassigned to different sites, and dynamic replication policy is in place. Data are fetched from remote sites for a particular task, if the required data is not available, in which case it is cached. A cached dataset is then available to the local site as a replica for the duration of the task.

3 Scheduling model and algorithm

On the basis of the aforementioned framework we consider the Grid composed by m clusters (sites). Each site is modeled as a set of H identical processors available for a time window of length W, and the task j as the request of h_j processors, and of a dataset f_j, for a certain processing time p_j.

The local scheduling problem we consider is therefore a multiprocessor task scheduling problem where a set of H processors, and a set J of non preemptive tasks are given; each task $j \in J$ requires h_j processors for a certain time $w_j = p_j + c(f_j)$, where $c(f_j)$ is the communication time (depending on the dataset policy) for loading dataset f_j, and the objective is to schedule as many tasks as possible without interruption within a deadline W, maximizing the processor total busy time.

Assuming the processors being indexed and organized as an array and restricting the subset of processors assignable to a task j formed by h_j consecutive indexed processors, we model the scheduling of task j as packing a box of width w_j and height h_j on a bounding rectangular area of width W and height H. Therefore the local

scheduling problem can be modeled as a rectangle packing, that is the problem of orthogonally packing a set of boxes into a bounded rectangular area without overlapping such that the (efficiency) ratio ρ between the size of the area occupied by the packed boxes and the size $W \cdot H$ of the rectangular bounding area is maximized.

We adopt the on-line algorithm PACK_R for rectangle packing proposed by the authors in [8]. The algorithm packs incoming boxes one by one into the smallest free rectangular subarea of the bounding area, and rejects an incoming box if there is no free rectangular subarea where that box can be packed. For the algorithm's details the reader is referred to [8].

4 The data management policies and the system performance

Experimental results presented have been conducted fixing the following parameter values for all the runs:

- Number of tasks $n = 100$;
- Maximum processing time for a task $w_{max} = 300$ seconds.

These parameters are required in the execution of the GenPro algorithm which generates the processing times for the tasks; moreover, in the PACK_R scheduling algorithm further parameters are needed to determine time windows, within which tasks are scheduled, and the number of processors (computing units) of a local computing site (i.e., the width W and the height H of the rectangular bounding area in the rectangle packing problem). In particular,

- $W = 1200$ seconds, represents the width of the schedule, i.e., the overall time windows within which tasks must be scheduled;
- $H = 80$, is the number of identical parallel processors that are available at a local site to execute tasks;
- $h_{max} = 30$, is the maximum number of parallel processors that a task requires for its execution.

It is important to notice that the processor number that each operation requires uniformly varies in the interval [1, 30], and is determined by the scheduling algorithm.

In general, this parameter tends to affect the solution even though the penalty introduced by GenPro tries to minimize such effect, moving the attention on the data replication phase at the intermediate storage servers.

The three policies of data replication considered are:

1. PLAIN CACHING. There is only the primary server with a storage capacity equal to 3% of the overall files where files may be replicated;
2. CASCADING REPLICATION. There is a primary and secondary storage server with a capacity equal to 3% and 20%, respectively;
3. NO REPLICA SERVER. There are no storage servers in the path followed by files from the server containing the latter to the computing site.

The analysis is done on two different scenarios:

- The number N of repetitions of algorithm GenPro, that is analyzing the n tasks that arrive in the system when $(N - 1)$ n tasks have been scheduled. The interesting aspect of this study is in the analysis of the servers past records and in the files that are currently replicated in the servers;
- The probability that files requested by tasks for their execution are contained or not in the cluster formed by the first five files (out of 100), i.e., in the cluster of files mostly requested.

The performance indicators used in the comparative analysis are:

- The average number of tasks rejected by the systems, since the local scheduler could not be able to allocate all the tasks;
- The efficiency ρ of the system.

4.1 Analyisis on the number of the GenPro algorithm repetitions

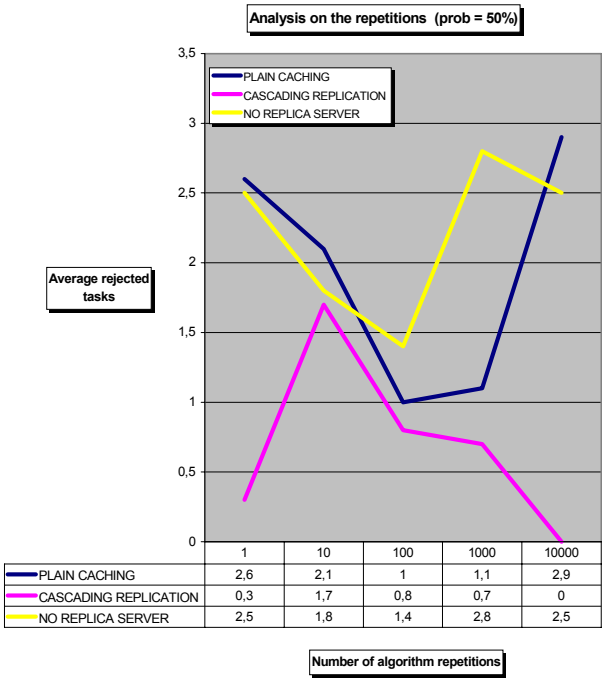
The system at the N-th repetition has received $(N - 1)$ n tasks to schedule. Based on the file that each task has requested for its execution, the servers, using the index NOA (number of access), can define an ordered list of the files that have been most frequently requested until the N-th algorithm repetition. After that, the GenPro algorithm, exploiting this ordered list, executes a scan in the server and updates over time the latter copying and deleting files (in case the storage limit is reached) leaving an immediate availability to the user of only those files with higher probability of being requested by forthcoming tasks. Hence, based on the data replication policy used, the present files in the servers will be more or less exploited for the user request execution.

We start our analysis by studying the impact of the number of repetitions of the GenPro algorithm. The performance measurement index used are the

average number of tasks rejected by the system and the ratio between the occupied area for task scheduling and the total available area. Note that in this first analysis the probability of a specific file request is 50%.

4.1.1 The impact of the number of repetitions on the number of rejected tasks

Let us first analyze the impact on the average number of rejected tasks. Experimental results are shown in Figure 2, where, besides the table with the values obtained by the different data replication policies, we report the chart of the trends related to such values.



	1	10	100	1000	10000
PLAIN CACHING	2,6	2,1	1	1,1	2,9
CASCADING REPLICATION	0,3	1,7	0,8	0,7	0
NO REPLICA SERVER	2,5	1,8	1,4	2,8	2,5

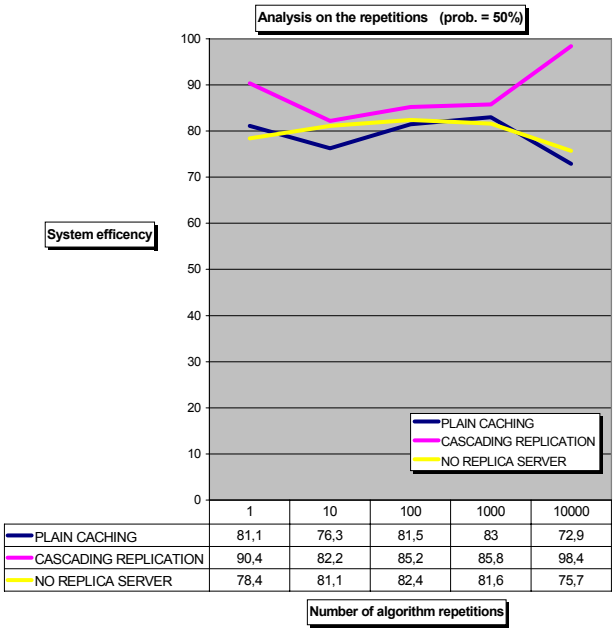
Fig. 2. The impact of the number of repetitions on the average number of rejected tasks.

Tests have been conducted with number of repetitions N = 1, 10, 100, 1000, and 10000. As it can be inferred by the chart, the Cascading Replication policy is the one that gives the best performance. In fact, yet when N = 1, the average number of rejected tasks is less than 1, and in the worst case such a value equals 1.7. This behavior can be explained considering that the probability used in the experimentation is quite high (50%) and the introduction of data replication servers renders processing times limited with respect to the other policies. When N is sufficiently high (in our tests equal to 10000), the number of rejected tasks with the Cascading Replication policy is zero, meaning that after a short transition period, the servers have stored the most frequently requested files.

It is interesting to note that there is not a large difference in terms of rejected tasks between the Plain Caching policy and the absence of Data Replication. In fact, for high values of N (i.e., $N >$ 1000) it is not convenient the use of a replication server.

4.1.2 The impact of the number of repetitions on the efficiency of the system

As showed in Figure 3, the system efficiency related to the Cascading Replication policy is clearly superior than the other two policies. The system efficiency is certainly related to the average number of rejected tasks: indeed, if there are a huge number of rejections, it is clear that the available area to scheduling tasks will be not properly used and the ratio between the used area and the total area will be low.



	1	10	100	1000	10000
PLAIN CACHING	81,1	76,3	81,5	83	72,9
CASCADING REPLICATION	90,4	82,2	85,2	85,8	98,4
NO REPLICA SERVER	78,4	81,1	82,4	81,6	75,7

Fig. 3. The impact of the number of repetitions on the system efficiency.

Analyzing the trends of the curves in the chart, we have the same situation occurred in Section 4.1.1. Also the trend of the Cascading Replication policy, that in the transitory phase (for $N < 10$) is not increasing, when $N > 100$ tends to increase, and for N = 10000 the efficiency reaches 98.4%. Note that the efficiency in the other two policies is not greater than 83%.

4.2 Analysis of the performance when the probability of dataset request varies

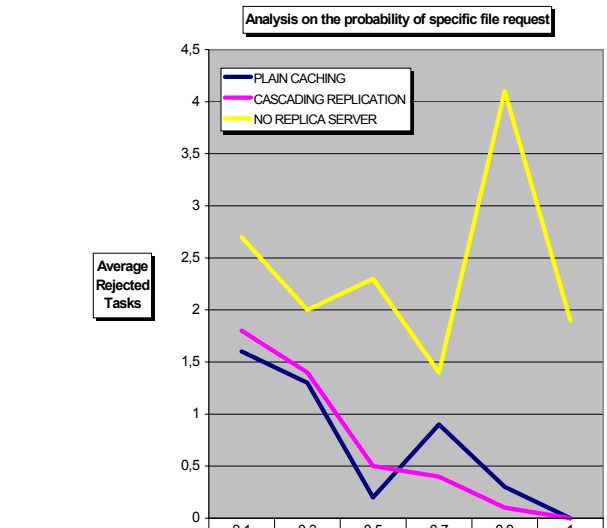
Let us now discuss the case in which the probability that each task requires for its execution a dataset

contained in the cluster of the first five out of one hundred sets, i.e., in the set of most requested datasets.

In the problem of replicating data on intermediate servers for distributed scheduling, we note that the scenario analyzed in this sub-section occurs very frequently in practice; in fact, the hypothesis of introducing these servers along the path between the computational site and the main storage site is thought for those situations in which tasks require for their execution a restricted number of datasets, and that this latter are requested with a certain frequency of the system that has to schedule tasks with a defined probability.

4.2.1 The impact of the probability of specific dataset requirement on the number of rejected tasks

As it was quite foreseeable from what we said before, since these policies were analyzed specifically in the case of repeated requests of specific datasets, the performance of the system in the case when there are intermediate servers (primary, secondary, or both) are much better than for the case in which there are no storage sites.



	0,1	0,3	0,5	0,7	0,9	1
PLAIN CACHING	1,6	1,3	0,2	0,9	0,3	0
CASCADING REPLICATION	1,8	1,4	0,5	0,4	0,1	0
NO REPLICA SERVER	2,7	2	2,3	1,4	4,1	1,9

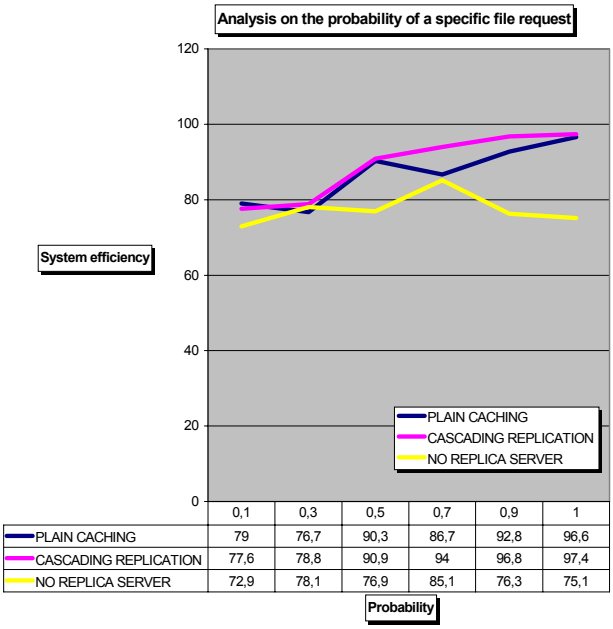
Fig. 4. The impact of the probability of specific dataset request on the average number of rejected tasks.

As one can observe from the chart of Figure 4, the trends of the curves related to Plain Cashing and Cascading Replication are quite similar, even if the the trend of the curve related to Cascading Replication (i.e., double servers) is more regular. When the probability of specific dataset requirement tends to one, that is, all the user applications require datasets contained in the group of the most popular datasets, the number of rejected tasks tends to zero, reaching this value in the extreme case.

When dataset replication policies are not adopted, the average number of rejected task is about 2.5, with peak values of about 4.1, which appear to be too high over a number of one hundred tasks (nearly 3% of rejected tasks).

4.2.2 The impact of the probability of specific dataset requirement on the system efficiency

Previously from the analysis of the experimental results, we have seen that varying the number of repetitions the curves related to both two performance indices follows a quite similar trend; this is mainly due to the fact that there exists a tight relation, as showed before, between the average number of rejected tasks and the system efficiency ρ.



	0,1	0,3	0,5	0,7	0,9	1
PLAIN CACHING	79	76,7	90,3	86,7	92,8	96,6
CASCADING REPLICATION	77,6	78,8	90,9	94	96,8	97,4
NO REPLICA SERVER	72,9	78,1	76,9	85,1	76,3	75,1

Fig. 5. The impact of the probability of specific dataset request on the system efficiency.

Analogously to the previous considerations, we may note that (see Figure 5) also if we only consider the efficiency of the system as performance index, the policies of Data Replication adopted (Plain Caching and Cascading Replication) allow a task scheduling with a very high system efficiency. In fact, for high probability values (prob. > 0.7), we obtain a system efficiency greater than 94% for the Cascading Replication and greater than 87% for the Plain Caching policy.

In the case in which there is no intermediate server for dataset replication, the average system efficiency reaches 78%, with a minimum of less than 73%.

4.3 Some technical-economic considerations

We conclude the analysis making some considerations on the trade-off between the cost of installing and maintaining the servers for data replications and the Grid system performance.

An aspect that comes in evidence from the analysis of the results is that analog considerations may be done both for the case in which we consider the number of rejected tasks and for the case in which we consider the system efficiency as performance measures.

We restrict the analysis varying the probability of specific dataset request for task execution. We note from the charts of Figures 4 and 5 that the trends of the curves related to Plain Caching and Cascading Replication are quite similar; in fact, for a value sufficiently high of this parameter the performance of PACK_R, with respect to both the number of rejected tasks and system efficiency, are quite good. In this situation, the average number of rejected tasks tends to zero and the system efficiency reach 97%. Therefore, in the case we are interested in guaranteeing an high system performance, it seems better to adopt a single dataset replication server instead of two servers. Also in the case in which we concern with cost minimization, the Plain Caching policy seems to be the best one because with no dataset replication the system efficiency goes down noticeably to 75% and the average number of rejected tasks goes up to 4%.

5 Conclusion

In this paper, we experimented an on-line scheduling algorithm based on rectangle packing for scheduling task in computing sites of a Grid system. The algorithm is experimented taking into account different dataset replication policies and storage server configurations: *Plain Caching*, i.e., there is only the primary server with a storage capacity equal to 3% of the overall files; *Cascading Replication*, i.e., there is a primary and secondary storage server with a capacity equal to 3% and 20%, respectively; *No Replica Server*, i.e., there are no storage servers in the path followed by files from the database containing the latter to the computing site.

References:

[1] F. Berman, R. Wolski, S. Figueira, J. Shopf and G. Shao, Application-level scheduling on distributed heterogeneous networks, *Proc. of Supercomputing '96*, 1996.

[2] H. Casanova, G. Orbetelli, F. Berman and R. Wolski, The AppLeS parameter sweep template: user-level middleware for the Grid, *Proc. of Supercomputing '00*, 2000.

[3] A. Su, F. Berman, R. Wolski and M.M. Strout, Using AppLeS to schedule simple SARA on the computational Grid, *International Journal of High Performance Computing Application*, Vol. 13, 1999, pp. 253-262.

[4] H. Dail, H. Casanova and F. Berman, A decoupled scheduling approach for the GrADS program development environment, *Proc. of the 2002 ACM/IEEE Conf. on Supercomputing*, Baltimore, Maryland, USA, November 16-22, 2002, CD-ROM. ACM. 2002.

[5] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon and R. Wolski, The GrADS project: software support for high-level Grid application development, *Internat. J. of Supercomputer Application*, Vol. 15, 2001, pp. 327-344.

[6] I. Foster and C. Kesselman, *The Grid: blueprint for a new computing infrastructure*, Morgan Kaufmann, 1999.

[7] K. Ranganathan and I. Foster, Decoupling computation and data scheduling in distributed data-intensive applications, *Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 23-26, IEEE Computer Society, 2002, pp. 352-358.

[8] M. Caramia, S. Giordani, A. Iovanella, Grid Scheduling by On-Line Rectangle Packing, *Networks*, Vol. 44, 2004, pp. 106-119.

[9] Y.L. Wu, W. Huang, S.C. Lau, C.K. Wong and G.H. Young, An effective quasi-human based heuristic for solving the rectangle packing problem, *European J. of Operations Research*, Vol. 141, 2002, pp. 341-358.