A Task Replication and Fair Resource Management Scheme for Fault Tolerant Grids

Antonios Litke, Konstantinos Tserpes, Konstantinos Dolkas, and Theodora Varvarigou

Department of Electrical and Computer Engineering,
National Technical University of Athens,
9, Heroon Polytechniou Str, 15773 Athens, Greece
{ali, tserpes, dolkas, dora}@telecom.ntua.gr
http://telecom.ece.ntua.gr/

Abstract. In this paper we study a fault tolerant model for Grid environments based on the task replication concept. The basic idea is to produce and submit to the Grid multiple replicas of a given task, given the fact that the failure probability for each one of them is known a priori. We introduce a scheme for the calculation of the number of replicas for the case of having diverse failure probabilities of each task replica and propose an efficient resource management scheme, based on fair share technique, which handles the task replicas so as to maintain in a fair way the fault tolerance in the Grid. Our study concludes with the presentation of the simulation results which validate the proposed scheme.

1 Introduction

Grid can be an appropriate solution for many computational intensive and grand scale applications ranging from scientific, industrial and engineering field. It is also an emerging solution for utility and pervasive computing. However, Grids as all the large scale distributed platforms are prone to failures, which restrain it to become a reliable execution platform for high performance and distributed applications. So the fault tolerance feature is of vital importance. By the term *fault tolerance* we denote the ability of the Grid system to perform correctly in the presence of faults.

Fault tolerance is of big importance in Grid computing, as the emerging grid-oriented applications have a significantly increased size and complexity from the traditional ones. Experience has shown that systems with interacting and complex activities are inclined to errors and failures. Thus, Grid computing is not expected to be fault free, despite the fact that individual techniques such as *fault avoidance* and *fault removal* [1] may additionally be applied to its resources. The fault tolerance feature is introduced in the Grid systems in order to enhance them with the appropriate reliability, which is mandatory in the context of diverse, dependable and cross-organizational environments. The reliability in Grid comprises the probability of all grid applications to be executed fully with no errors in the grid computing environment. As applications scale to take advantage of a Grid's vast available resources, the probability of failure is no longer negligible and must be taken into account.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1022 – 1031, 2005.

There are various approaches to make grid computing fault tolerant [1],[2],[3]. The basic however are the checkpoint recovery and the task replication. The former is a common method of ensuring the progress of a long-running application by taking a checkpoint, i.e., saving its state on stable storage periodically. A checkpoint recovery is an insurance policy against failures. In the event of a failure, the application can be rolled back and restarted from its last checkpoint—thereby bounding the amount of lost work to be recomputed. Task replication is another common method that aims to provide fault tolerance in distributed environments by scheduling redundant copies of the tasks, so that to increase the probability of having at least a simple task executed. A brief overview of the options in the fault tolerant computing on the Grid can be found in [2].

There has been a variety of implementations that have addressed the problem of fault tolerance in Grid and distributed systems. Globus [15] provides a heartbeat service to monitor running processes to detect faults. The application is notified of the failure and expected to take appropriate recovery action. Legion [16] provides mechanisms to support fault tolerance such as checkpointing. Other Grid systems like Netsolve [17], Mentat [18] and Condor-G [19] have their failure detection mechanisms and their failure recovery mechanisms. They provide a single user-transparent failure recovery mechanism (e.g. retrying in Netsolve and in Condor-G, replication in Mentat). The difference between these systems and our proposed scheme relies on the fact that the one presented here addresses the fault tolerance as a metric that is adjusted in a *fair* way for all the Grid users. Moreover it applies to all Grid environments and is especially beneficial for low *workload* jobs in unreliable environments, such as Mobile Grids [20], which consist of mobile resources (hosts and users) connected by wireless links and forming arbitrary and unpredictable topologies.

In this paper we study a fault tolerant model for Grid environments based on the task replication concept. The basic idea is to produce and schedule in the Grid infrastructure multiple versions (replicas) of a given task, based on the fact that the failure probability for each one of them is known a priori. The replication model that is adopted is based in static replication [4] meaning that when a replica fails it is not substituted by a new one. The failure of a task replica is based on aspects that concern the task itself and not the resource on which it is going to be executed. This approach implies that the Grid infrastructure remains unchanged concerning its topology and total computational capacity, and independent from the faults that occur in the Grid environment. The introduction of task replicas causes an overhead in the workload that is allocated for execution to the Grid environment. Moreover, scheduling and resource management are important in optimizing Grid resource allocation, and determining its ability to deliver the negotiated QoS and provide fair access to all users [5][3]. The basic idea that is applied in this study is to address the additional overhead caused by the task replicas in the Grid system with a fair scheme of resource management that will provide a fair share of computational resources to the Grid users [8][9].

The remainder of this paper is structured as follows: Section 2 provides the problem formulation for the fault tolerance and task replication in the Grid and the notation that will be used. Section 3 provides the task replication model for tasks' whose failure probability is a random function. In section 4 we describe the need for adopting a mechanism for the efficient handling of the additional load that has been caused

by the replicas and which is based on the *max min fair share* scheme, aiming to satisfy as many as possible users with the available resources. Finally, in section 5 we present the simulation results of the developed scheme and conclude, in section 6, with a discussion on future work as well as on potential improvements and enhancements on our proposed scheme.

2 Notation and Problem Formulation

We consider that a set of M processors forms a Grid infrastructure. Each processor has a fixed computational capacity denoted as $c_j, j \in \{1,2,...,M\}$, thus the total computational capacity of the Grid is $C = \sum_{j=1}^{M} c_j$. We also consider a set of N different tasks $T_i, i \in \{1,2,...,N\}$ to be assigned to the Grid for execution. We assume that the tasks are non-preemptable and non-interruptible [10]. This means that a task cannot be broken into smaller sub-tasks or modules and it has to be executed as a whole on a single processor. Additionally as soon as a task starts its execution on a processor, it cannot be interrupted and it consumes the whole processor computational capacity as long as it is executed.

Each task T_i has an *execution time* ET_i and a *deadline* D_i. The execution time corresponds to the time interval that the execution of T_i lasts if it is executed in a processor of unitary *capacity* $c = 1$. The deadline of the task represents the latest time at which the Grid has to deliver the results to the user. It is a quantity specified by the end-user who is willing to pay for the Grid resources used. During a task execution on the Grid, various errors might occur causing task failure. In this study we will deal with those cases that are based upon the distributed systems fault model, which includes omission, timing and arbitrary faults [12]. These kinds of errors are commonly met in distributed systems as well as in Grid environments. We will omit other types of failures such as hardware failures [2][4][13], etc.

We define the *failure probability* Pf_i of a task T_i, which is the probability that the task fails to be executed on the Grid. Respectively, *success probability* Ps_i is the probability that the task T_i concludes its execution within the Grid system, providing the presumable results. The correlation between failure probability and success probability is:

$$Pf_i = 1 - Ps_i . \tag{1}$$

At this point we introduce the concept of *workload*. Workload $w_i, i \in \{1,2,\text{K},N\}$ is the computational capacity that is required by a task T_i in order to be executed in unitary time on a given resource. For simplicity reasons we have focused only on computational capacity and we have omitted other parameters such as communication delays, disk input/output delays etc. Moreover, in our study we have reduced the computational capacity into unitary, when referring to the workload, in order to treat the resources as homogeneous simplifying thus the presented model. However, the

extended scheme incorporating various heterogeneous resources can be derived in the same manner. The workload is equal to the inversed execution time ET_i, and can be written as:

$$w_i = (ET_i)^{-1} . \tag{2}$$

Task replicas are generated and assigned to the Grid for execution. The term *replica* or *task replica* is used to denote an identical copy of the original task. By producing task replicas, a low probability of task failure can be achieved. We assume that m_i replicas –denoted by $T_{ik}, k = 1,\text{K}, m_i$- of a task T_i are produced and are placed among other tasks and replicas that are to be submitted for execution. Given the failure probability Pf_{ik} of each one of the m_i replicas T_{ik} of task T_i, the failure probability is defined as:

$$Pf_i = \prod_{k=1}^{m_i} Pf_{ik} . \tag{3a}$$

The above corresponds to the probability of the event "all task replicas fail". Respectively, the success probability Ps_i is equal with the probability of the event "at least one task replica succeeds" and is given by the equation:

$$Ps_i = 1 - Pf_i = 1 - \prod_{k=1}^{m_i} Pf_{ik} . \tag{3b}$$

It can be assumed that the similarity of the replicas implies that the variations of the failure probabilities of each one of them cannot be large. Given that assumption, we can define two bounds for the failure probability Pf_{ik} of each replica T_{ik}, namely u_i and l_i are the maximum and the minimum value that Pf_{ik} can take for each of the replicas of T_i. These two bounds can be either estimated by the Grid user or can be statistically determined by the previous history of the system according to the relative tasks that have been already submitted. Alternatively, prediction models can be applied for the estimation of the failure probabilities based on the individual task features in a similar way as described in [14].

In order to guarantee a low failure probability our scheme produces as many task replicas as needed so as to satisfy the constraint of success probability. We now define a *probability threshold* δ, which denotes the probability that each task (including its replicas) will not finish its execution. We can write:

$$Pf_i \le \delta . \tag{4}$$

where δ is a constant between 0 and 1.

3 Task Replication Model

We will present a way to calculate the number of replicas that is required in order to secure a fault tolerant operation of the Grid for all cases. We will distinguish between

two cases that will be examined in this section. In the first case it is assumed that two different positive numbers u_i, l_i bound the failure probability Pf_{ik} of a replica T_{ik}. In the second case Pf_{ik} is unbounded and it can take random values, so a simple algorithm is used to produce replicas in order to follow inequality (4).

First case: $l_i < Pf_i < u_i$

The failure probability Pf_{ik} for each replica of a task can be bounded by two positive real numbers u_i, l_i. From (3a) and (4) we have:

$$l_i^{m_i} \le u_i^{m_i} \le \delta, \tag{5}$$

and in the sequel:

$$m_i \le \frac{\log(\delta)}{\log(u_i)} \Rightarrow m_i \le \left\lfloor \frac{\log(\delta)}{\log(u_i)} \right\rfloor. \tag{6}$$

In the simple case of a constant failure probability $Pf_i = F_i = const$, it becomes:

$$m_i \le \left\lfloor \frac{\log(\delta)}{\log(F_i)} \right\rfloor. \tag{7}$$

Second Case: Pf_{ik} is a function of *k*.

We use a simple algorithm to specify the number of the needed replicas for each task. The idea is to produce a replica of a task each time the failure probability is bigger than δ. The replication procedure stops when inequality (4) holds true for a given number of replicas m_i. The proposed algorithm computes the number of replicas to be produced for a task T_i in order to reduce the failure probability Pf_i at least bellow the value δ.

The replication procedure for both cases takes place in the *Grid middleware*, which is responsible for keeping the level of fault tolerance for the whole Grid environment. This approach does not make necessary any communication between the replicated tasks themselves. The presented scheme, however, does not exclude any parallel programming applications from being executed on the specific platforms, since it does not comprise a constraint to these tasks. Moreover, although the failure probability is attached to tasks, this assumption does not affect the generality of our approach, since in the second case, where the failure probability is a random function of *k,* we can assign to *k* values that are dependent to the resource itself.

4 Efficient Fault Tolerant Mechanism with Fair Share

We assume that the workload of each task is the sum of the workloads of its replicas, which can be written as $w_i = \sum_{k=1}^{m_i} w_{ik}$ or $w_i = m_i \cdot w_{i1}$, since every replica is identical

to the primary task which was assigned by the user. In this way every task can be considered as a *virtual task*, comprising as the set of itself and all of its replicas and which has a workload equal to an integral multiple of the original's workload.

We propose a *max min fair share* mechanism for the management of the fault tolerant feature of each task, by reducing in a fair way the replicas of each task. It is important to clarify the difference between the demanded capacity d_i and the capacity provided by the Grid which will be symbolized with a_i. In our case we will consider as required capacity the workload w_i. The number of replicas actually allowed by the Grid is other than the one required (m_i) to guarantee the fault tolerance of the Grid. We denote the number finally assigned replicas as n_i. This fact raises the need to determine the new number of replicas that actually can be assigned in order to have the maximum possible probability threshold δ which utilizes the total of the Grid capacity so as to satisfy the tasks' deadlines. We will apply a *max min fair share* technique [8][9] to determine the fair share that can be allocated at each task.

The main idea of this scheme is that all users submitting their jobs in the Grid are allocated an equal share of the computational resources for the execution. Although many schemes can be adopted for the efficient handling of Grid resources, we deal in this study with the fair share model which is a decent way to share resources without having prioritized and weighted clients. In case where a specific job does not require all its assigned computational power, the remaining part is being distributed evenly to the remaining users in a recursive way. In order to classify the tasks that are submitted in the Grid according to the required computational power, we need to define the quantity of demanded capacity d_i which, in our case, corresponds to the workload w_i. We will refer to that amount as *fair share x* .

The demanded capacity for every task is compared with the fair share x. If the task's computational capacity allocated is bigger than or equal to the fair share, then it is considered as a *satisfied* task. In the other case, the task is classified as *unsatisfied* (the capacity given is smaller than the fair share). In the sequel the fair share is being recalculated. The new fair share accrues from the capacity U , which is remaining to be shared among the unsatisfied tasks. This algorithm runs iteratively until either all tasks are satisfied with the demanded capacity, or there is no remaining capacity U to be distributed. The proposed scheme assigns in a fair way the total capacity of the Grid to the virtual tasks that are submitted for execution. The virtual tasks that have received no sufficient resources for their execution can either be scheduled in a later phase having as disadvantage the deviation from the given deadline D_i.

5 Simulation Results

The proposed efficient fault tolerant mechanism has been implemented in C++ and evaluated against a set of tasks. In the following table we present the tasks that have been used for the evaluation of the proposed scheme. The tasks have been selected so as to provide a main separation between "reliable tasks" with a low failure probability Pf_i between 0 and 0.15 and those that have a higher failure probability between 0.2

and 0.35. The failure probabilities of the individual tasks have been randomly generated between each of the two values respectively. The workload of each original task ranges between 1 and 100 computational units. By this way, a mean value of 50 computational units can be assigned to each one of the original tasks, which in turn leads to a "hard" scenario, given the fact that the produced replicas will augment the demanded capacity overall. As fault tolerance threshold δ we have selected the value of 0.05. This threshold has been selected so as to provide a high degree of reliability, higher than the mean value of the "reliable" tasks.

Table 1. Input provided for the simulation results of the proposed fault tolerant scheme

***Fault tolerance threshold* δ =0.05**			
No of tasks	**No of "reliable" tasks**	**Grid's Capacity *C***	**% of satisfied tasks**
30	20	1500	80%
50	30	2500	52%
100	50	5000	63%
200	120	10000	85%
500	300	25000	65%
1000	400	50000	88.2%

Figure 1 shows the relation of the workload generated for each T_i *before* and *after* the use of the max min algorithm. Figure 2 depicts the relation between the failure probability Pf_i *before* and *after* the use of the max min fair share algorithm for each virtual task. Examining the presented results, we can see that in some cases there is not enough workload assigned to a virtual task, leading to the incapability of the procedure to provide service even to a single replica. This fact is clearly depicted in Fig. 2 where the failure probability after the max min fair share algorithm reaches to 1, which is interpreted as a complete failure to serve the certain task which actually means its rejection from the fault tolerant Grid and, consequently, its deviation from the user specified deadline. This rejection, although undesired, does not imply inefficiency of the proposed scheme, since the basic motivation of the work is to construct a mechanism of providing a fault tolerant Grid for both "reliable" and "less reliable" tasks and not to guarantee the achievement within the given deadline.

The result analysis in Fig. 1 shows that even in the case where the tasks are equally distributed among "reliable" and "less reliable", the tasks that are finally blocked are 3, while less than 10% of the tasks have a failure probability between 0.2 and 0.3. Again, the majority of the tasks, comprising the 80% of the total tasks assigned, is successfully scheduled in the Grid with a failure probability less or equal to 0.05, although their initial failure probability was significantly higher before applying the proposed scheme.

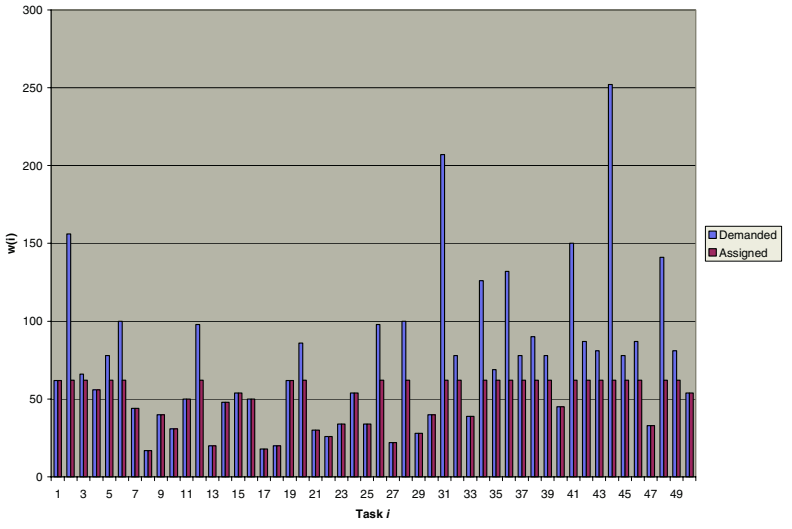

Fig. 1. The case of 50 different tasks with 30 reliable ones and fault tolerance threshold δ = 0.05. The respective workloads of the virtual tasks as resulted for the *demanded* and *assigned* case for δ = 0.05

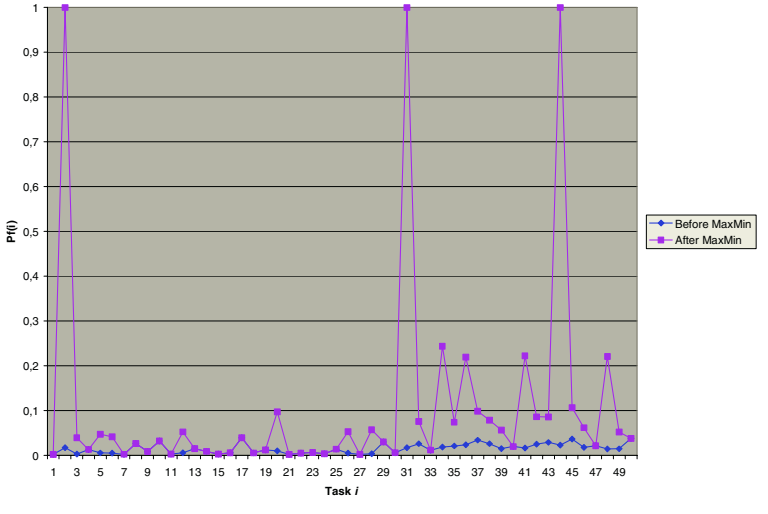

Fig. 2. The case of 50 different tasks with 30 reliable ones and fault tolerance threshold δ = 0.05. The failure probability for each task *before* and *after* the max-min fair share technique

From the workload perspective the maximum deviation between the primary demanded and the eventually assigned workload to each set of tasks ranges between 43.43% of the demanded workload in the set of the 1000 tasks and 77,54% in the set

comprising of 500 tasks. The largest number of not satisfied tasks in terms of workload is presented in the set of 50 tasks. In particular, 24 out of 50 tasks are not provided with the required amount of workload by the proposed mechanism. The average percentage of the deviations in the demanded workload of the 50 tasks set is 17.09%, a rather small number, if we consider the overall required satisfaction level of fault tolerance threshold. Some results of special interest are those of the biggest set of 1000 tasks. In that case, 88.2% of the tasks are satisfied in terms of assigned workload while at the same time the average deviation of the assigned workload is 2.89% of the demanded workload.

6 Conclusions

In this paper we have studied a task replication scheme that applies max-min fair share resource management for providing fault tolerance in Grids. The main contribution of this work relies in performing task replication by using the proposed algorithm which is designed for the case of having diverse failure probabilities between a task and its replicas, and in the handling of the fault tolerance fair share for the efficient assignment of the tasks in the Grid. The scheme has been implemented and validated for a variety of tasks with a diverse set of failure probabilities for the given tasks and their replicas. It showed that in cases where we have tasks equally distributed among "reliable" ones and "less reliable" ones, a fault tolerant Grid can be achieved by having rejected only a small number of tasks resulting in their deviation from their deadlines. The other tasks and their replicas are successfully scheduled in the Grid providing thus a high degree of fault tolerance. The presented results although in a preliminary form, are indicative for the evaluation of the proposed scheme. The approach that is presented can be further improved by taking into consideration the deviation from the deadline for each task and assuming this deviation as a criterion for prioritized scheduling.

References

1. M.R. Lyu,, Software Fault Tolerance, John Wiley & Sons – Chichester, 1995
2. J. B. Weissman. Fault Tolerant Computing on the Grid: What are My Options? HPDC 1999
3. F. Wang, K. Ramamritham, J.A. Stankovic. Determining redundancy levels for fault tolerant real-time systems, IEEE Trans. Computers, vol 44, issue 2, 1995, pp. 292-303
4. A. Nguyen-Tuong. Integrating Fault-Tolerance Techniques in Grid Applications, PhD Dissertation, University of Virginia, August 2000
5. Scheduling Working Group of the Grid Forum, Document: 10.5, September 2001
6. K. Ramamritham, J.A.Stankovic, and P.-F. Shiah. Efficient Scheduling Algorithms for Real-time Multiprocessor Systems, IEEE Trans. on Parallel and Distributed Systems, vol.1, no.2, 1990, pp.184-194
7. L. E. Jackson and G. N. Rouskas. Deterministic Preemptive Scheduling of Real Time Tasks, IEEE Computer, vol. 35, no. 5, 2002, pp. 72-79
8. A. Demers, S. Keshav and S. Shenker, Design and Analysis of a Fair Queuing Algorithm, Proc. of the ACM SIGCOMM, 1989

9. D. Bertsekas, R. Gallager, Data Networks, Prentice Hall, 1992. The section on max-min fairness starts on p.524
10. J.Y-T. Leung and M.L. Merrill, A Note on Preemptive, Scheduling of Periodic, Real-Time Tasks, Information Processing Letters, 11, no. 3, 1980, pp. 115-118
11. M. L. Dertouzos and A.K.-L. Mok, Multiprocessor On-line scheduling for Hard Real Time Tasks, IEEE Trans. on Software Eng., vol. 15, no. 12, 1989, pp. 1497-1506
12. A. S. Tanenbaum, M. van Steen, Distributed Systems: Principles and Paradigms, Prenctice Hall, Computer Science, 2002
13. T. Varvarigou, J. Trotter, Module replication for fault-tolerant real-time distributed systems, IEEE Transactions on Reliability, vol. 47, no. 1, 1998, pp. 8-18
14. N. Doulamis, A. Doulamis, A. Panagakis, K. Dolkas, T. Varvarigou and E. Varvarigos, A Combined Fuzzy -Neural Network Model for Non-Linear Prediction of 3D Rendering Workload in Grid Computing, IEEE Trans. on Systems Man and Cybernetics, Part-B (accepted for publication)
15. The Globus project. http://www-fp.globus.org/hbm/
16. A. Nguyen-Tuong, and A.S. Grimshaw, "Using Reflection to Incorporate Fault-Tolerance Techniques in Distributed Applications," Computer Science Technical Report, University of Virginia, CS 98-34, 1998.
17. H. Casanova, J. Dongarra, C. Johnson and M. Miller, "Application-Specific Tools", in I. Foster and C. Kesselman (eds.), The GRID: Blueprint for a New Computing Infrastructure, Chapter 7, pp. 159–180, 1998
18. A.S. Grimshaw, A. Ferrari and E.A. West, "Mentat", in G.V. Wilson and P. Lu (eds.), Parallel Programming Using C++, Chapter 10, pp. 382–427, 1996
19. F.C. Gartner, "Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments", ACM Computing Surveys, Vol. 31, No. 1, 1999
20. "Access to Knowledge through the Grid in a Mobile World" (AKOGRIMO) Integrated Project FP6-2003-IST-004293. http://www.akogrimo.org/