

Fault Tolerance within a Grid Environment

Paul Townend and Jie Xu

Department of Computer Science
University of Durham, DH1 3LE, United Kingdom
p.m.townend@dur.ac.uk jie.xu@dur.ac.uk

Abstract

Fault tolerance is an important property in Grid computing as the dependability of individual Grid resources may not be able to be guaranteed; also as resources are used outside of organizational boundaries, it becomes increasingly difficult to guarantee that a resource being used is not malicious in some way. As part of the e-Demand project at the University of Durham we are seeking to develop both an improved fault model for Grid computing and a method for providing fault tolerance for Grid applications that will provide protection against both malicious and erroneous services.

We have firstly begun to investigate whether the traditional distributed systems fault model can be readily applied to Grid computing, or whether improvements and alterations need to be made. From our initial investigation, we have concluded that *timing*, *omission* and *interaction* faults may become more prevalent in Grid applications than is the case in traditional distributed systems. From this initial fault model, we have begun to develop an approach for fault tolerance based on the idea of job replication, as anomalous results (either maliciously altered or simply wrong) should be caught at the voting stage. This approach combines a replication-based fault tolerance approach with both dynamic prioritization and dynamic scheduling.

1 INTRODUCTION

Traditionally, software dependability has been achieved by *fault avoidance* techniques (such as structured programming and software reuse) in order to prevent faults, or *fault removal* techniques (such as testing) in order to detect and delete faults [LYU95]. However, in the case of Grid computing, these approaches – although still very much useful – may not be enough.

As applications scale to take advantage of Grid resources, their size and complexity will increase dramatically. However, experience has shown that systems with complex asynchronous and interacting activities are very prone to errors and failures due to their extreme complexity, and we simply cannot expect such applications to be fault-free no matter how much effort is invested in fault avoidance and fault removal. In fact, the likelihood of errors occurring may be exacerbated by the fact that many Grid applications will perform long tasks that may require several days of computation, if not more. In addition to this, it may also be the case that the cost and difficulty of containing and recovering from faults in Grid applications is higher than that of normal applications. Furthermore, the heterogeneous

nature of Grid nodes means that many Grid applications will be functioning in environments where interaction faults are more likely to occur between disparate Grid nodes, whilst the dynamic nature of the Grid – resources may enter and leave at any time, in many cases outside of the applications control – means that a Grid application must be able to tolerate (and indeed, expect) resource availability to be fluid.

It is therefore necessary to investigate the application of *fault tolerant* techniques for Grid computing. Fault tolerance is the survival attribute of computer systems; [AVI85] describes the function of fault tolerance “...to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service.”

As an example, consider a Grid application, decomposed into 100 services. Assume each service has a MTTF of 120 days, and requires a week of computation. Assuming an exponentially distributed failure mode, the composed application would have a MTTF of 1.2 days; therefore, without any form of fault tolerance, the application would rarely finish.

A further cause for concern when considering Grid applications is that in many cases, an organisation may send out jobs for remote execution on machines upon which trust cannot be placed; for example, the machines may be outside of its organisational boundaries, or may be desktop machines that many people have access to. A fault tolerant approach may therefore be useful in order to potentially prevent a malicious node affecting the overall performance of the application.

As part of the e-Demand project at the University of Durham, we are seeking to develop both an improved fault model for Grid computing and a fault tolerant architecture for Grid applications that will provide protection against both malicious and erroneous services.

2 AN IMPROVED FAULT MODEL

A fault model attempts to identify and categorise the faults that may occur in a system, in order to provide clues as to how to fine-tune the software development environment and how to improve error detection and recovery. A question that needs to be asked is: is the traditional distributed systems fault model appropriate for Grid computing, or are refinements necessary?

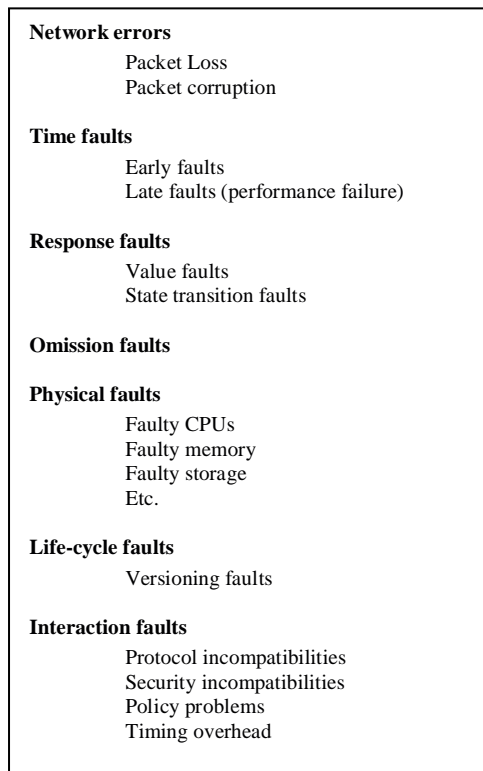


Figure 1 – A partial fault model for a traditional distributed system

Figure 1 shows a list of some of the potential faults that can occur in a traditional distributed system. Some potential faults that may occur in a Grid system however, are not present. For example, Grid service instances have the potential to expire at set times; should a service expire whilst still being used by a Grid application, then an error may occur. Therefore, service expiry may be considered a potential life-cycle fault. A potentially new type of fault may also arise; future Grid services may provide metered access [FOS01] - should the meter run out half-way through a job, then errors may occur.

As Grid applications may feature dozens – if not hundreds – of services, a fault in one service may be amplified as the effects of the fault move further down the chain. For example, figure 2 shows how a time-out in a service may cause another service that is blocking for it to time-out as well.

The increased number of interactions between large numbers of services, many of which may be dynamically bound at run-time and hence not known to the original application developers, will undoubtedly lead to increased incidence of interaction faults. This may be due to different services supporting different protocols, or may also be due to the faults listed previously.

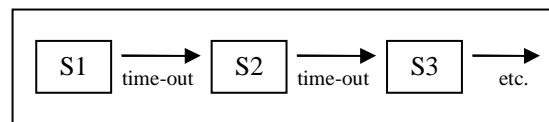


Figure 2 – An example of cascading time-outs

Omission faults will also arise when resources become unavailable, due to the dynamic nature of many Grids. Again, this may have a “knock on” effect on the workflow of an application.

We are currently working on this fault model, in order to see if further changes need to be made; however, as can be seen, we have currently found that in addition to the two extra types of fault discovered, *interaction*, *timing*, and *omission* faults become more prevalent in a Grid/e-Science context.

3 A FAULT TOLERANT SOLUTION

One of the most attractive features of the Grid is the ability for a client application to be able to send out (potentially computationally expensive) jobs to resources on the Grid, dynamically located at run-time. What is striking about the Grid is how few assumptions can be made about the resources

available; these resources may be widely heterogeneous; each may run a different combination of hardware and operating system environment. Some resources may be privately available within corporate intranets, whilst others may be publically available over the internet. Some clusters of resources may be stored in one location, whilst others may be geographically distributed. Some resources may be metered, whilst others may be freely available. Different resources may have widely differing quality of service (QoS) guarantees – or none whatsoever.

However, with most compute resources available on the Grid, it is fair to say that at least some of the following properties will hold:

- We may not be able to guarantee that a resource is non-malicious (i.e. it may record or alter results of compute tasks)
- We may not be able to guarantee reliability (either of the resource’s software or hardware)
- We may not be able to guarantee processing power (i.e. we may have no control over the resource’s scheduling or load)

In order to minimise the problems caused by the above properties, we propose a new fault tolerance scheme based upon the notion of replication (i.e. an n-copy system). A replication-based system is proposed as it will allow us to take advantage of the large number of resources available on the Grid, whilst not requiring any additional software to be supplied on the part of the client application. Our initial scheme is shown in figure 3 and is detailed below.

Grid resources with a fault tolerance service compatible with our scheme register themselves with a UDDI repository; their fault tolerance service is capable of receiving jobs, executing them, performing checksum operations on them, and sending the jobs back.

A client application then instantiates one or more “co-ordination” services; these services are designed to be able to function under the Distributed Recovery Block scheme first proposed by [KIM84a]; there can be any number of these services, but it is recommended that at least two be used, to guard against a single point of failure. The co-ordination services contact one or more UDDI registries in order to determine the location and number of compatible resources available. They may also have the capability to contact any appropriate metering services, in order to determine the relevant costs of using the different processing nodes (provided this information is not made available in meta-data provided by the nodes themselves).

The co-ordination service(s) then wait for the

client to send them a job via a multicast; once this has been received, the primary co-ordination service (presuming no failure occurs, else the secondary will take over and so on) determines which nodes to send replicas of the job to, and then sends replicas of the job to these nodes.

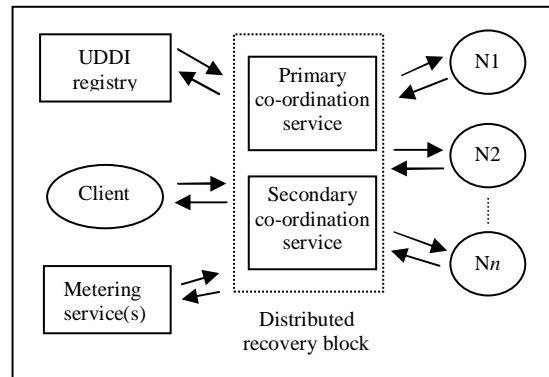


Figure 3 – Our initial scheme

The nodes then process the job until they complete (or until they fail and/or leave the system). Upon completion, the fault tolerance service on each node generates a checksum based upon the results it has generated, and broadcasts this checksum to the co-ordination service(s). The co-ordination services wait until a given number of nodes have completed, and then vote on the returned checksums. A “correct” checksum is assumed to be the one returned by the majority (i.e. $n/2 + 1$) of the nodes who broadcast a checksum. Should no consensus be reached, then the co-ordination service can either wait for more checksums to be received, send out more replicas (preferably to nodes belonging to organisations different from the nodes whose checksums did not reach consensus) or return an error message to the client application. Should consensus be reached, a node with a correct checksum is randomly selected and requested to send the full result back to the client application. The co-ordination service returns the appropriate checksum to the client, and the client then generates a checksum on the returned result, compares it with the consensus checksum returned by the co-ordination service and accepts the returned results if there is a match.

An obvious drawback to this approach is that of increased overhead; this is both due to the voting method (based on generating and comparing checksums) and also because voting cannot commence until a suitable number of results have been generated. This can however be lessened by increasing the number of replicas used; for example, should 3 results be needed in order to perform a satisfactory vote, then 8

replicated jobs can be sent out and voting will begin once the first 3 results have been received. Although traditionally replication schemes have assumed that the cost of additional replicas is high, this may not be the case on the Grid, although the co-ordination service must base the number of replicated jobs it chooses to send out on any cost considerations that may arise.

Another important issue that needs to be addressed is how to schedule multiple jobs. Some jobs may require more compute time than others and/or may be deemed more urgent; such jobs need to be scheduled in such a way that results can be returned within a given time frame. This is not a trivial task, as the client program cannot be assumed to have control over the resources it wishes to use – and so, for example, a fast node processing the client's job may suddenly begin to execute a job submitted by a different program too, and so slow down dramatically. Therefore, the co-ordination service may implement a dynamic scheduling approach, moving jobs from one Grid resource to another in order to minimise processing time.

The urgency of jobs may also change over time depending on the state of the client application, and so the co-ordination service may also have functionality to be able to dynamically re-prioritise jobs.

4 CONCLUSIONS

In this document, we have outlined some of the work currently taking place in the Fault Tolerance strand of the University of Durham e-Demand project. A fault model for Grid computing is being constructed, refining the traditional distributed systems fault model; this is being used as a basis for fine-tuning a replication-based approach to fault tolerance also being developed as part of the project. This approach uses one or more co-ordination services, constructed under a distributed recovery block scheme, to locate compute resources on the Grid, and to schedule, broadcast, receive and vote upon jobs submitted by a client program. This is in order to not only reduce the likelihood of faulty results being received by the client, but also to protect against malicious Grid resources deliberately altering the results they produce.

5 FUTURE WORK

Additional refinements are continuing to be made to both the new fault model for Grid computing and also the main replication-based

fault tolerance scheme. A priority in the near future is to produce a working implementation of the fault tolerance scheme.

6 REFERENCES

[AVI85] A. Avizienis, "The N-version Approach to Fault-Tolerant Software" - IEEE Transactions on Software Engineering - vol. 11 1985

[FOS01] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of Supercomputer Applications*, 15(3), 2001

[KIM84a] K.H. Kim, "Distributed execution of recovery blocks: an approach to uniform treatment of hardware and software faults", *Proc. 1st International Conference on Data Engineering*, pages 620-628, Los Angeles, April 1984

[LYU95] M.R. Lyu, "Software Fault Tolerance" - John Wiley & Sons - Chichester - UK - 1995

7 ACKNOWLEDGEMENTS

This work is part of the University of Durham e-Demand project (<http://www.dur.ac.uk/e-demand>).