KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2007
Cluj-Napoca (Romania), June 6–8, 2007, pp. 244–251

A VIEW ON FAULT TOLERANT TECHNIQUES APPLIED FOR MEDIOGRID

DACIAN TUDOR[(1)], VLADIMIR CRETU[(2)], AND HORIA CIOCARLIE[(3)]

Abstract. In this paper we analyze the characteristics of fault tolerance for grid systems, which serves as a basis to identify the features and techniques of the Globus framework and the way these can be applied for the MedioGrid project. Based on the general MedioGrid architecture and the fault tolerant analysis, we suggest an extension of dynamic replication strategies to address data life cycle using dynamic policies. Finally, we highlight some directions in order to enhance the fault tolerance at the MedioGrid application level, both simple and parallel.

1. Introduction

Classic systems approach fault tolerance by either avoiding error conditions, through structured programming and certified component reuse, or using error reduction techniques by enforcing strong testing techniques. In case of grid systems which exhibit dynamic and unforeseen interactions between their heterogeneous components, which reside in different administrative and privileged domains, such approaches are not feasible and insufficient. One of the main reasons is complexity as grid applications have a high degree of asynchronous protocol interactions. By its unpredictable nature, grid systems supply an execution environment where execution guarantees can be hardly satisfied. Another factor which makes complicates execution guarantee is the high execution time as some applications might be running even days or weeks until the results are obtained. In addition, applications might require multiple resources which are located in different administrative domains which can be error prone too. Defects might be exacerbated as well by the service composition, specifically, the fact that a grid service might invoke several other grid services. The failure of one of the invoked service might turn against the caller service, leading to return an error on its caller too. As the service composition protocol is non-deterministic, the potential grid errors are non-deterministic too.

2000 *Mathematics Subject Classification.* 68M14.
Key words and phrases. Grid computing, Fault Tolerance, MedioGrid.

All these new introduced condition in grid systems lead us to the idea that for grid systems, one must rely on a more complex fault model. In this paper we analyze a fault tolerance model for distributed systems through the grid perspective, then we are highlighting the Globus approach for the fault tolerance model and finally we evaluate the applicability of fault tolerance techniques on the MedioGrid project.

2. A VIEW ON FAULT TOLERANCE FOR THE GRID

When referring to a fault tolerant systems, we refer to a system which supplies a set of services to its clients, according to a well defined contract, in spite of error presence, through detecting, correcting and eliminating errors, while the systems continues to supply an acceptable set of services [1]. A fault tolerance model highlights possible causes and conditions where errors might appear, with the goal of improving system characteristics do detect and eliminate errors.

The main approach to attack fault tolerance is rollback technique [2], which implies application state logging at a certain time interval and restoring the last stable state in case the application is detected as entering a critical state. The used techniques are either check pointing types [3] where the application state is expected, or logging techniques [4] which implies application message logging and handling. For data grid systems, one of the most common and widespread fault tolerance techniques is provided by replication techniques, at both data provider and computing resources. In the later case, a certain application can be running in parallel on multiple resources and in case of error conditions, computation is continued on the healthy and active resources. Another approach is process migration when the executive state is becoming critical.

Based on [12], we present and discuss the main classes of errors that might appear in the grid systems.

2.1. **Network errors.**

Network errors are environmental errors caused by the communication channel and basically refer to package losses on the transmission path or corrupted incoming packages on the receiving path. These errors can be corrected by the network transmission protocol and in cases where no correction can be applied the communication path between the two endpoints is considered broken.

2.2. **Timing errors.**

Timing errors are errors that can occur either at the beginning of the communication as a result of the impossibility to establish a connection, or during the communication flow when for example the response time of the caller exceeds the response time expected by the caller. In case of grid systems which exhibit large and variable communication latencies, such timing conditions add a nondeterministic component to the expected approximate time.

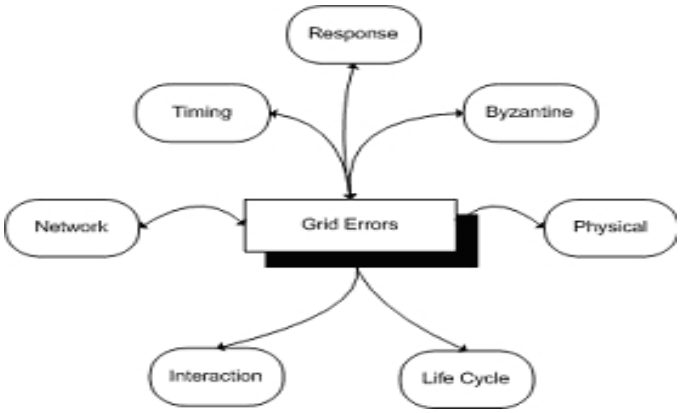

FIGURE 1. Grid Software Error Classes

2.3. **Response errors.** Response errors are caused by a service which returns values outside of the expected boundaries by the caller. In such situations, components have to be able to validate a certain response and to appropriately handle the exceptions. A system that is designed as a state machine, can execute uncontrolled transitions in the state space which can be further propagated to other services as a result of the grid service composition.

2.4. **Byzantine errors.** Byzantine errors are arbitrary errors that could appear during the execution of an application. They refer to catastrophic conditions such as crashes and omission errors. A system entering in a Byzantine state has an undefined behavior which might be caused either by the execution impossibility or erroneous execution, or by arbitrary execution outside of the one specified by design.

2.5. **Physical errors.** Physical errors refer to critical conditions of the physical resources such as processor, memory, storage or communication medium. Such errors have to be detected and corresponding resources be declared as non-functional.

2.6. **Life cycle errors.** Life cycle errors are particular to components which expose services which can expire at a certain moment. They can apply to component versioning as well. An example of this condition is updating a service while its clients expect that the service is working properly according to its previous specification. Service changes could be both syntactical and structural with different implications on the service callers.

2.7. **Interaction errors.** Interaction errors are caused by incompatibilities at the communication protocol stack level, security, workflows or timing. These are the most common errors in large scale grid systems because all these conditions appear

while running the applications and the environmental and interaction states cannot be reproduced during the application testing phases. We expect that for complex grid applications to observe a high probability of interaction error occurrence. Some of these, as for example the ones due to different security levels, could be isolated and eliminated during the testing phases in a high percentage as there is a limited number of calls between virtual organizations.

3. FAULT TOLERANCE IN GLOBUS

Globus Toolkit [4] offers a reference implementation of grid standardized protocols, together with a set of tools and helper services in order to facilitate grid application development and deployment. In terms of the fault tolerance conditions presented in the previous section, one of the most important services are data transfer services, replication and grid execution management services.

The most basic data transfer service which serves as a basis for all other data manipulation services is GridFTP service, which represents an implementation of the grid extended well known FTP protocol. Its extended features for the grid include security level integration, parallel data transfer flows, partial transfers, automatic setting of TCP transfer buffers, transfer flow monitoring and restarting. The most interesting part of the GridFTP component in the context of building fault tolerant data services, is that Globus does not supply a server side library, thus peer-to-peer transfer scenarios cannot be constructed using Globus. In addition, it requires a preinstalled GridFTP server which reduces the degree of resource discovery and automatic replacement in case of physical damage.

Reliable File Transfer service (RFT) provides a Globus service which guarantees a successful file delivery between grid nodes in the presence of failures during a given transfer operation. Transfer status is kept into a database which supplies the necessary information to eventually resume the transfer. Globus uses a SOAP description for the transfer request and the GridFTP protocol to initiate the transfer. RFT offers support for concurrent transfer flows which gives good performance for modest size parallel file transfers.

The Data Replication Service (DRS) is a higher level Globus service which gives support for automatic file replication between multiple sites and it is working as following:

- Grid clients specify a set of files to be replicated to other nodes.
- Each node uses RLS to determine which are the files that are missing locally and where these files are available.
- The missing files are replicated locally by submitting a RFT request to the RFT service.
- After the files are locally replicated they are registered to LRC.

Analyzing data management services, we can notice that the Globus solution is mostly a static solution based on fixed configuration sets of grid components.

Hypothetical situations where a damaged grid node is dynamically replaced by another one reduce to the a priori determination of replacement nodes. Globus does not offer support for peer-to-peer data sharing with automatic discovery of candidate nodes for such a service.

At the execution level, Globus offers an interface to launch and monitor jobs on grid nodes through the GRAM service. The GRAM service takes care for input and output data transfers as well as the security level integration. For execution planning, GRAM supplies a "plug-in" architecture which permits an extension to adapters for local resource schedulers. One can notice that the Globus execution level offers only job state checking during execution. The fault tolerance level has to be assured by the concrete implementation of the resource scheduler. One of the most popular resource schedulers is Condor [6]. Condor offers technical support for fault tolerance by providing job migration and check pointing mechanisms. The fault tolerance level of Condor takes care of restarting monitoring and job submission services in case a local error occurs. In case of network type errors, Condor can restore the communication flow as soon as the nodes become available and can restore the previous state through persistent state logging mechanisms. In cases where the grid nodes which execute the job are detected as faulty nodes, Condor implements a job migration policy to other available nodes which are continuously monitored through a dedicated service.

4. FAULT TOLERANCE IN MEDIOGRID

In case of the MedioGrid system [6] , we consider the main error sources as being caused by communication path corruption, failure of the processing or data units and application specific errors.

Communication path failure can lead to the impossibility to communicate with the parallel processes spawn across the grid to solve a certain environmental problem. It is desired that the entire operation is not restarted as a whole, but only specific computations that are assigned to the processing units who cannot communicate anymore with the rest of the systems. Using a fault tolerant scheduler like Condor [6] removes the impossibility to continue computations on an alternative resource. The GRAM architecture and the interaction with the scheduler are presented in Figure 2. We consider communication path failure as an extreme failure condition that can be compensated by providing alternative communication paths which requires more research into the problem of network design which is not in the scope of the MedioGrid project.

Referring to computing resources and excluding the ones that are managing data, any error at this level is detected and managed by the Condor job scheduler. The support offered by Condor is sufficient to get the job execution to its end, even if nodes are signalling errors.

Data unit failures are mainly addressed by the data model and the level of data replication [8]. The proposed replication solution, which is supported by Globus

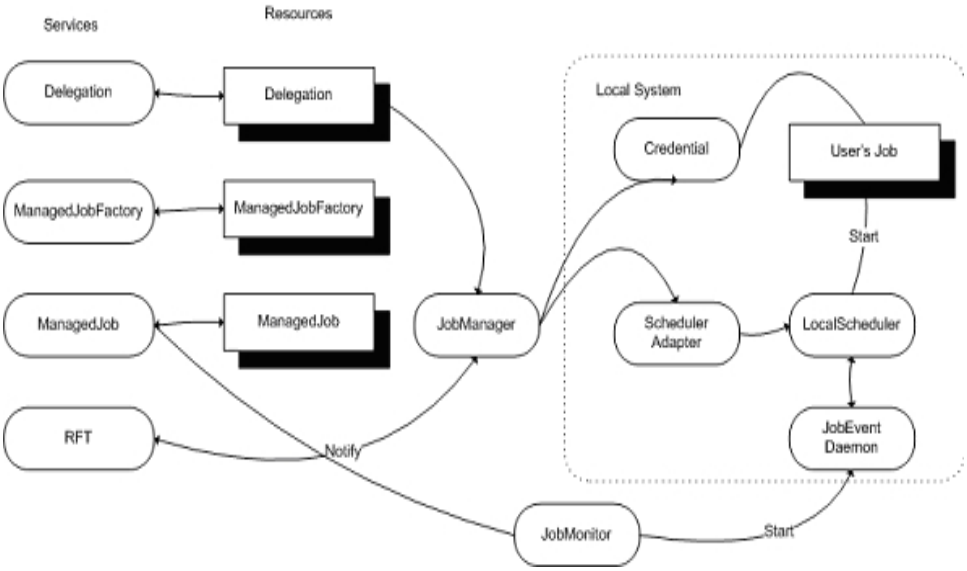

FIGURE 2. GRAM Architecture, as presented in Globus [4]

toolkit, represents a static solution which can handle the failure of up to two data storage units, considering the primary replication group composed of three data resources. A solution which offers a configurable degree of fault tolerance and also addresses the data life cycle is to supply a data management service which guarantees the existence of a file f in the MedioGrid network as being replicated in n locations. At the present form of the MedioGrid system, n is equal to 3 and replication is fixed.

The proposed service achieves a rule based variable replication. The rules determine the replication degree in respect to the data use rate and data age. It is expected that users are accessing more recent data than older ones. These policies come to enforce the data life cycle management in contrast to the current system that keeps data at three fixed locations. Such a service can be implemented based on existing Globus services and existing higher level dynamic replication strategies for grids as the ones presented in [13].

In addition to the availability aspect of data replication in MedioGrid, which has been a goal from the beginning, we would like to extend the dynamic replication scope to address the data life cycle as well. One of the drawbacks of the fixed replication solution is that data is always replicated with the same ratio even if its usage rate is low. Such solution is simply wasting storage that could be used for newer data items.

At the application level, fault tolerance has to components: execution context and the application code itself. At the execution context, the analysis boils down to the resource scheduler and its capabilities to support fault tolerance. As far as the application code is concerned, we distinguish two cases: when the applications runs stand alone and when the application is a parallel application comprising several grid concurrent jobs.

The fault tolerance level for a simple application can be achieved by using grid check-pointing strategies [9]. The idea of the project is to provide transparent check-pointing support for Java applications, by capturing and restoring local and global states through the use of an execution coordinator. There are more particular approaches which aim to offer a certain degree of fault tolerance for Java, which consist basically of a fault tolerant RMI layer. Such layer replaces remote references automatically in case a reference is in the impossibility to execute an operation over a connection. The applicability of such strategies is dependent on the algorithms and decided at this point in time.

In case of a parallel application containing more collaborative or independent tasks which are running in parallel, the Condor job scheduler mentioned in the previous section is being able to launch only one job. The are a series of efforts towards a MPI-like fault tolerant solution [10, 11], but they are still considered immature to be used in a real grid application. Techniques based on GridRPC to obtain a fault tolerant service have main target applications that are running uninterruptible computations, where stopping the calculus for a limited duration is not critical. One the other pole, MedioGrid applications aim to provide the result immediately, which reduces the applicability of these techniques for our project. Of course, in case of some grid operations which have to be completed without any timing constraints, such techniques could be applied for MedioGrid too.

5. CONCLUSIONS

In this paper we have presented a view on fault tolerance for grid systems and we have evaluated the main solutions and concepts provided by the Globus toolkit to construct fault tolerant grid applications. As a result of the analysis, we concluded that Globus provides limited support for fault tolerant services, both at data management and task execution, but provides the necessary basic concepts to build higher level services. One of the major drawbacks of the Globus solution is the built-in static configurations which limits dynamic service construction using Globus components. Based on our analysis of the main grid error classes, we have assessed the MedioGrid system in terms of fault tolerance and we suggested a dynamic data replication extension based on configurable life-cycle policies. Depending on the MedioGrid applications, we have given a few directions towards adopting supplemental fault tolerance levels in terms of parallel applications.

References

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

[2] Manivannan, D., Singhal, M.,"Quasi-synchronous checkpointing: Models, characterization, and classification". In: IEEE Transactions on Parallel and Distributed Systems. Volume 10. (1999) 703-713

[3] Alvisi, L., Marzullo, K., "Message logging: Pessimistic, optimistic, causal, and optimal". Software Engineering 24 (1998) 149-159

[4] Globus toolkit homepage, http://www.globus.org, Globus Alliance 2006

[5] Condor homepage, http://www.cs.wisc.edu/condor/

[6] Ordean, M., Melenti, C., and Gorgan, D., "Mediogrid system in meteorological and environment applications". International Conference on Advances in the Internet, Processing, Systems and Interdisciplinary Research, IPSI - 2005 Amalfi, Italy, pp: 203-207, ISBN: 86-7466-117-3, 2005

[7] Muresan, O. and Gorgan, D., "Arhitectura retelei MedioGrid". Atelier de Lucru MEDIOGRID vol 1, ISBN: 973-713-090-1, Ed MEDIAMIRA Cluj-Napoca, 2006.

[8] Colesa, A., Ignat, I., Opris, R., "Providing High Data Availability in MEDIOGRID", 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, September 26-29, 2006

[9] Stone, N., Simmel, D., and Kielmann, T., "GWD-I: An architecture for grid checkpoint recovery services and a GridCPR API". Grid Checkpoint Recovery Working Group Draft 3.0, Global Grid Forum, http://gridcpr.psc.edu/GGF/docs/draft-ggf-gridcpr-Architecture-2.0.pdf, May 2004.

[10] Graham, E. F., and et al., "HARNESS and fault tolerant MPI", Parallel Computing, vol. 27, pp. 1479-1496, 2001.

[11] Bosilca, G., and et al., "Mpich-v: Toward a scalable fault tolerant mpi for volatile nodes", in Proceedings of Supercomputing, 2002.

[12]]Townend, P., Xu, J., "Fault Tolerance within Grid environment", Proceedings of AHM2003, page 272, 2003

[13] Ranganathan, K., Foster, I.T., "Identifying Dynamic Replication Strategies for a High-Performance Data Grid", Proceedings of the Second International Workshop on Grid Computing Vol. 2242, pages: 75-86, 2001

(1) Computer Science and Engineering Department, "Politehnica" University of Timisoara, V. Parvan street, no. 2, 30023, Timisoara, Romania
E-mail address: dacian@cs.utt.ro

(2) Computer Science and Engineering Department, "Politehnica" University of Timisoara, V. Parvan street, no. 2, 30023, Timisoara, Romania
E-mail address: vcretu@cs.utt.ro

(3) Computer Science and Engineering Department, "Politehnica" University of Timisoara, V. Parvan street, no. 2, 30023, Timisoara, Romania
E-mail address: horia@cs.utt.ro