Fault Tolerant on the Grid using Distributed Data Mining Services

T. HEMALATHA
M.Tech. Student,
Department of Computer Science & Engineering
Sri Venkateswara University College of Engineering
Tirupati, India

DR. CH. D. V. SUBBA RAO
Professor,
Department of Computer Science & Engineering
Sri Venkateswara University College of Engineering
Tirupati, India

Abstract:
Fault tolerance is an important issue in service-oriented architectures like Grid and Cloud systems, where many and heterogeneous machines are used. Fault Tolerance is a non-functional requirement that requires a system to continue to operate, even in the presence of faults.In this work we present a flexible fault tolerant which extends the service –oriented architecture for Distributed Data Mining services on Grid for all the machines (primaries and replica), in order to have each machine both serving client requests and acting as backup for the other machines. Where as in previously a mechanism for handling machine failures in a Grid environment is proposed. By this the users can achieve failure recovery whenever a crash can occur on a Grid node involved in the computation.

Keywords: *Distributed Data Mining, Fault Tolerance, Grid Computing, Service-oriented Architecture*

1. Introduction

Grid computing differs from conventional distributed computing because it focuses on large-scale resource shar-ing, offering innovative applications, and, in some cases, it is geared toward high-performance systems Grid is a basic infrastructure of national high performance computing and information service, it targets on the integration and interconnection of many kinds of high performance computers, data servers, large-scale storage systems and visualize systems which are physical distributed and heterogeneous. The driving Grid applications are traditional high-performance applications, such as high-energy particle physics, astronomy and environmental modeling, in which experimental devices create large quantities of data that require scientific analysis. For these reasons, Grids must offer effective support to the implementation and use of data mining and knowledge discovery systems. To achieve such a goal, several distributed data mining systems exploiting the Grid infrastructure has been designed a performed by the services on the worker nodes. A resource is associated to each service: the GlobalModel Resource to the GlobalMiner-WS and the LocalModel Resource to the LocalMiner-WS. Such resources are used to store the state of the services, in this case represented by the computed models (globally and locally, respectively). The services development can be done by using the Java WSRF library provided by the WS-Core, a component of the Globus Toolkit 4 .and implemented [14], [12], [8], [2]. In a previous work [4] we present a flexible failure handling

frame-work which extends that proposed in [3], addressing the requirements for fault tolerance in the Grid. The framework allows users to achieve failure recovery whenever a crash can occur on a Grid node involved in the computation.

In computer science applications, a general problem to take into account is machine failure due to faults in some component, such as a processor, memory, device, cable, or software. A fault is a malfunction, possibly caused by a design error, a manufacturing error, a programming error, physical damage, deterioration in the course of time, and many other causes. Not all faults lead (immediately) to system failures, but they can do. In particular, this aspect becomes relevant in a scenario, like the Grid, where many and heterogenous machines are involved. Developing, de-ploying, and running applications on a environment poses significant challenges due to the diverse failures and error conditions encountered during execution. As observed in [10], although the mean time to failure of any entity in a computational Grid is high, the large number of entities in a Grid (hardware, network, software, grid middleware, core services, etc.) means that a Grid can fail frequently. For example, in [13], the authors studied the failure data from several high performance computing systems operated by Los Alamos National Laboratory (LANL) over nine years. Although failure rates per processor varied from 0.1 to 3 failures per processor per year, systems with 4096 processors averaged as many as 3 failures per day. Thus, although the number of failures per processor is relatively low, the aggregate reliability of a system clearly deteriorates as the number of processors is increased. So, the reliability of a computational Grid is a real problem to deal with.

In this work we present a flexible fault tolerant which extends the service –oriented architecture for Distributed Data Mining services on Grid for all the machines (primaries and replica), in order to have each machine both serving client requests and acting as backup for the other machines In the following we describe all the steps composing the whole process. Let us suppose that a client wants to execute a distributed mining algorithm on a dataset D, which is partitioned in N partitions, $\{D_1, ..., D_N\}$, each one stored on one of the nodes $\{Node_1, ..., Node_N\}$. A request to the framework of performing a mining process can be labeled in three different main phases, each one composed of various steps, as described in the following.

The rest of the paper is organized as follows. Section II describes the original framework that supports no fault tolerant functionalities. Section III describes a fault tolerant framework for distributed data mining, as extension of that proposed in [4]. Section IV gives some concluding remarks.

2. Background: A Service Oriented Architecture for Distributed Data Mining On The Grid

Typically in Distributed Data Mining (DDM) data sets are stored in local databases or file systems, hosted by local computers/repositories, which are connected through a computer network. One of the most common DDM approaches includes the analysis of the local data sets at each site, inferring local models or statistics. Then, the locally discovered knowledge is usually transmitted to a merger (or central) site, where the integration/refinement of the distributed local models is performed. Such two steps can be re-executed, until a convergence condition has been reached. Examples of solutions adhering to this design pattern fall in various categories, i.e. clustering ([3], [8]), classification ([7]), association rule and frequent item sets mining ([7]), ensemble learning ([11]), collective data mining ([8]) and meta-learning ([9]). In order to provide a service oriented architecture for the execution of distributed data mining algorithms on the Grid, in [2] we have designed the Grid service architectural model shown in Figure 1. It is composed of two Grid Services: the GlobalMiner-WS and the LocalMiner-WS.

The overall architecture resembles the aforementioned DDM schema through an entity acting as coordinator (the GlobalMiner-WS)) and a certain number of entities acting as miners (LocalMiner-WS) on local sites. Thus, it is a master/worker architecture, in which the service on the master node arranges the operations performed by the services on the worker nodes. A resource is associated to each service: the GlobalModel Resource to the GlobalMiner-WS and the LocalModel Resource to the LocalMiner-WS. Such resources are used to store the state of the services, in this case represented by the computed models (globally and locally, respectively). The services development has been done by using the Java WSRF library provided by the WS-Core, a component of the Globus Toolkit 4 .

In the following we describe all the steps composing the whole process. Let us suppose that a client wants to execute a distributed mining algorithm on a dataset D, which is partitioned in N partitions, $\{D_1, ..., D_N\}$, each one stored on one of the nodes $\{Node_1, ..., Node_N\}$. A request to the framework of performing a mining process can be labeled in three different main phases, each one composed of various steps, as described in the following.

From the following Fig.: 1 we can see that how a job can be submitted and executed. First the client submits the job to the Global Miner-WS and immediately the Global Miner-WS searches for an appropriate resource (Global Model Resource) for storage of the job. Then the Global Model takes the help of Local Miner-WS for the execution of the job and for finding out the faults in the machine. Finally the Global Model Resource submits the job to the Local Miner-WS. Once if the

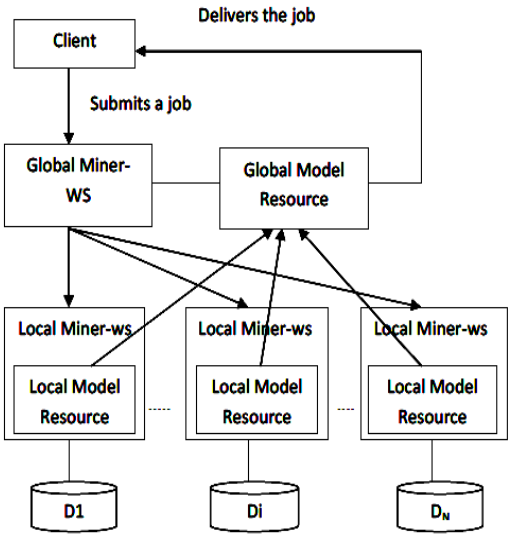

Fig. 1. General Grid Service Architecture for a machine

Job is submitted to the Local Miner-WS then the Global Model Resource it has no control over the job processes. Now the Local Miner-WS starts the required process of the job. After execution of the job. The results of the executed job are stored in the Local Model Resource, when once the results are collected then the Local Model Resource submits the job results to the Global Model Resource. At last finally the Global Model Resource will subm it the job results to the client.

2.1 Proposed Architecture

In the proposed architecture[Gridbus [6], it is composed of two Grid Services : the Grid Service Broker and the Grid Service Provider.This architecture is similar to the Grid Service Architecture. To the proposed architecture we use the **Peer-To-Peer Computing** mechanism for an efficient

exploitation of all the machines (primaries and replica), in order to have each machine both serving as client requests and acting as backup for the other machines.

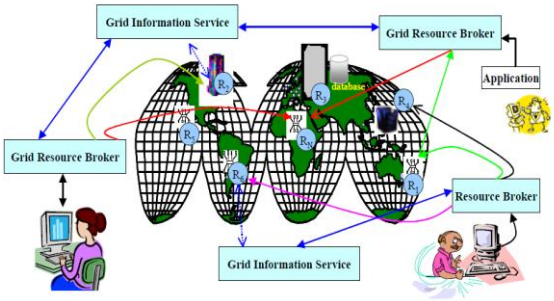

Fig. 2. Proposed architecture of P2P computing

From the Fig.:2 we can see the proposed architecture, how it process. The client submits the job to the Grid Service Broker and then the Grid resource broker performs resource discovery based on user-defined characteristics, using the Grid information service(step 1). The broker identifies the list of data sources or replicas and selects the optimal ones(step 2) . The broker ensures that the user has the necessary credit or authorized share to utilise resources(step 3) . The broker scheduler maps and deploys data analysis jobs on resources that meet user quality-of-service requirements(step 4) . The broker agent on a resource executes the job and returns results(step 5) . The broker collects the results and passes them to the user(step 6).

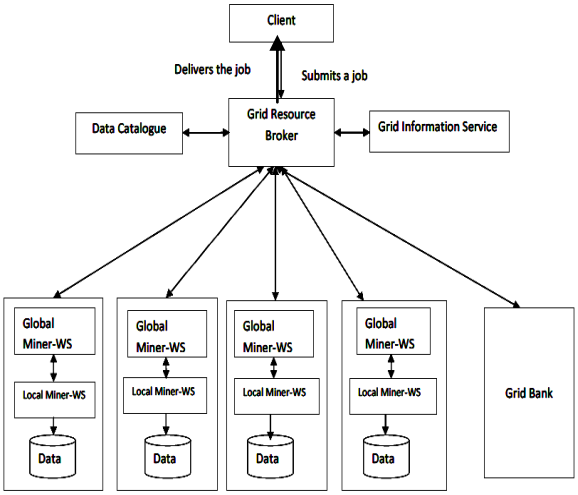

Fig. 3. Proposed Architecture

2.2 Unreliability Aspects

The system described in the Section II-A presents no strategy to tolerate faults. In particular, crashes can occur both on the Grid Service Broker and on the Grid Service Provider, so there are two points where the fault tolerance has to be handled. The first service acts both as the coordinator of the system and as collector of the results delivered by the local services; for such a reason, any failure occurring on it stops the whole computation. The second services, acting as workers in the architecture, execute local computations. Since the global model can be built only if all the local models are delivered, any crash on a local site (and consequently no deliver of its local model to the central site) prevents the Grid Service Broker to build a global model. In the rest of the paper we address such issues and propose a solution to deal with them.

2.2.1 To overcome these two reasons or faults

Many fault-tolerance systems have been proposed in literature. Most of them exploit the two well-known classical Primary-Backup and Active-Replication approaches. In the following, we will give a brief description of such two techniques. Then, we will cite some systems devoted at handling fault tolerance in Grids

The Primary-backup method is based on a primary server and a given number of backup servers. The essential idea is that, at any instant, only the primary is running and does all the work. If the primary fails, a cut over from the primary to the backup is handled by a suitable protocol (that in most cases does not involve the client). At this point, the backup acts as new primary (and new backup should be activated).

Active replication, sometimes referred as state machine approach, is a well-known technique for providing fault tolerance using physical redundancy. The strategy is based on independent replicated server (replicas). The client sends the invocation to all the replicas, which execute the computation and send the responses. In other words, the computation is made by more servers are executing exactly the same work and are able to finalize the computation and reply to the client.

3. A Fault-Tolerant Framework

This section presents the extension of the framework described in the Section II with the goal of making it fault-tolerant. As cited in the Section II-B, we deal with two-level fault tolerant strategies: the first one applied to the Grid Service Broker level, the second one to the Grid Service Provider level.

3.1 Fault tolerance on the Grid Service Broker

The fault-tolerance on the Grid Service Broker has been designed by adopting and implementing the general primary-backup strategy. The proposed fault-tolerant framework supposes the presence of a set of Grid Service Broker replica, whose just one is the primary at any time. The others are named backups. The primary-backup strategy [7] contemplates the following general steps:

1. The client sends the invocation to the Grid Service Broker.
2. The primary receives the invocation and asks for local computations; as soon as such computation results are returned and the Global Model has been re-computed, the primary sends a model-update message to the backups. The primary sends a model-update message to the backups.
3. If the primary crashes during step 2, a new primary is elected among the replicas, and it becomes the new primary of the system, taking care of the computation.
4. Once the primary has received a reply of the state update (step 2) from all backups, the response is sent to the client.

Three delicate phases, as in any implementation of a primary-backup mechanism [3], should be analyzed in detail:

- **Checkpointing** or transfer of application state. The primary periodically needs to send the change in the Global Model (its state) to the backups; it basically consists of storing a snapshot of the current application state. In particular, the consistency has to be guaranteed among the backup states, i.e., the primary can continue its work (or reply to the client) only when it is known that the backups have applied the state change.
- **Failure detection**. Crashes of the primary node can be detected by a periodic message, i.e. heartbeat that is sent to the backup; if no messages are sent for a given time, then this can be an indication of a failure on the primary node.

- **Recovery phase,** or switching to a new primary. Originally, one of the service instances is designated as a primary and others as backups. After a failure of the primary , the backups agree on a new primary that restarts the execution from the last Checkpoint state. Hereafter, all future requests are directed to and processed by it.

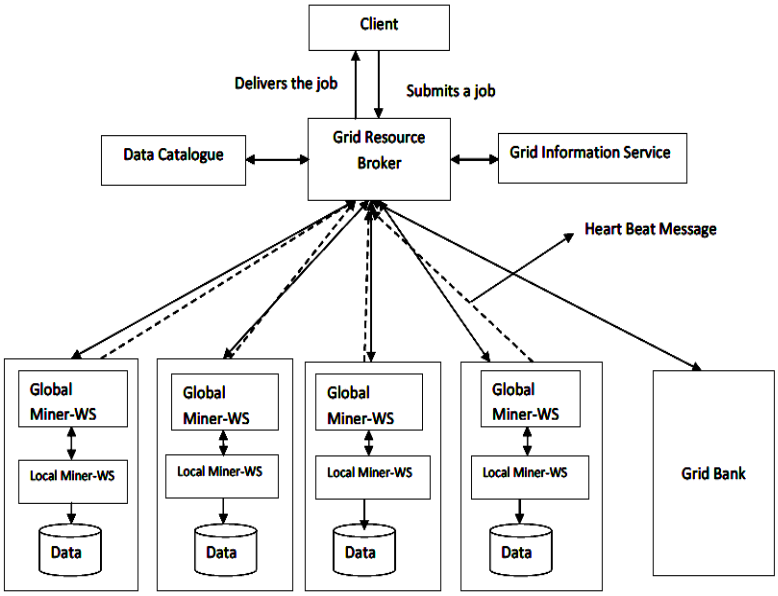

Fig. 5. Fault Tolerant Architecture

Figure 3 shows the system with no faults and process is going on properly with the backups consisting internally in it by sending the heart beat message to the Grid Service Broker. . The client submits the job to the Grid Service Broker and then the Grid resource broker performs resource discovery based on user-defined characteristics, using the Grid information service(step 1). The broker identifies the list of data sources or replicas and selects the optimal ones(step 2) . The broker ensures that the user has the necessary credit or authorized share to utilise resources(step 3) . The broker scheduler maps and deploys data analysis jobs on resources that meet user quality-of-service requirements(step 4) . The broker agent on a resource executes the job and returns results(step 5) . The broker collects the results and passes them to the user(step 6).

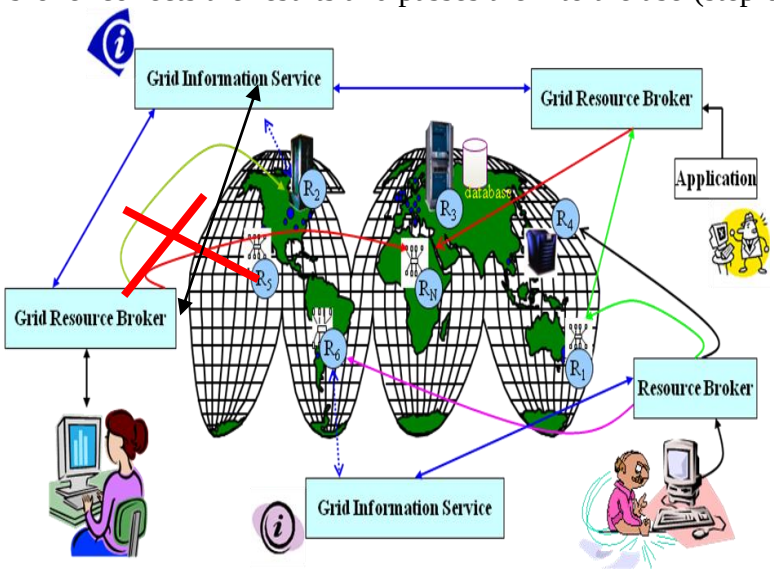

Fig. 6. Architecture with backup

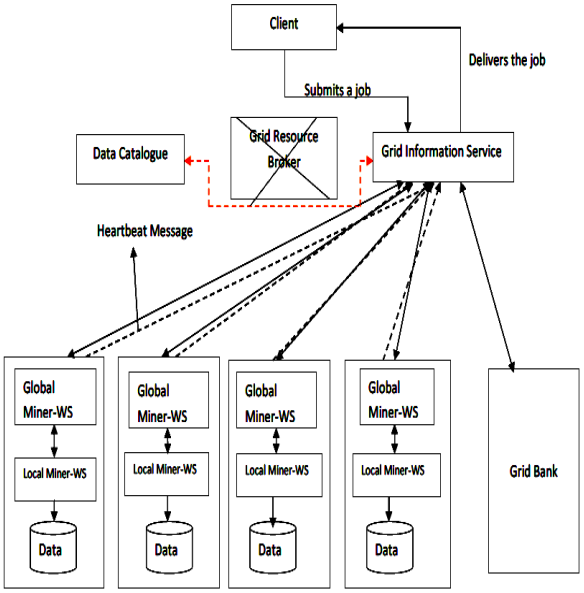

Fig. 4. Fault Tolerant Architecture with backup

Figure 4 shows an efficient exploitation of all the machines (primaries and replica), in order to have each machine both serving client requests and acting as backup for other machines. Now, let us make some considerations on the protocol above described, giving some more details on it:

- Even though state updates are asynchronous events, they are sent from the primary to the backup by the synchronous updateModel operation; this guarantees that the primary has a control on the reception of the updates by the backups. In fact, for consistency reasons, it is important that the computation can go ahead only if all the backups have stored the last checkpointed global model.
- Whenever a role of primary is assigned to a backup, it re-starts the computation from the last committed GlobalModel. Since it has no control on the operations performed by the old primary after the last checkpoint and before the failure, the new primary performs the following choices: if the last checkpointed global model is the final model, it is just delivered to the client; otherwise, if it needs more processing, the new primary submits to the local services the task to be performed (probably, the same submitted by the old primary before its failure).

- The protocol guarantees hiding of any failure to the client, in fact no interaction is requested during all the process (even if a failure occurs).
- The protocol does not consider the failures of a backup service before its activation as primary; the only new issue are the detection of backup failures and the integration of new backup into the system. These steps would not interfere with the operation of the surviving system components and its implementation is not complex.

4. Conclusions and Future Work

Owing to the heterogeneity and complexity of the Grids, executing long distributed data mining tasks in a reliable way is a challenge. We introduced an efficient exploitation of all the machines (primaries and replica), in order to have each machine both serving client requests and acting as backup for the other machines.

As future we plan to introduce some mechanisms for load balancing which can be used in order to have a load-aware assignment of the mining tasks.

References

1. Alsairafi, S. F. S. Emmanouil, M. Ghanem, N. Giannadakis, Y. Guo, D. Kalaitzopoulos, M. Osmond, A. Rowe, J. Syed and P. Wendel .(2003).The Design of Discovery Net: Towards Open Grid Services for Knowledge Discovery. International Journal of High Performance Computing Applications, vol. 17, no. 3, pp. 297-315.
2. Beguelin, A. E. Seligman and P. Stephan (1997).”Application level fault tolerance in heterogeneous networks of workstations”. Journal of Parallel and Distributed Computing, vol. 43, no. 2, pp. 147-155.
3. Cannataro, M. and D. Talia (2003). The Knowledge Grid. Communi-tations of the ACM, vol. 46, no. 1, pp. 89-93.
4. Cesario, E. and D. Talia (2008).”Distributed Data Mining Models as Services on the Grid”. In Proc. of 10th International Workshop on High Performance Data Mining (HPDM 2008), in conjunction with ICDM’08, IEEE, 2008, pp. 409-495.
5. Cesario, E. and D. Talia. “ A failure Handling Framework for Distributed Data Mining Services on Grid”
6. Foster, I. C. Kesselman, J. Nick and S. Tuecke (2003). The Phys-iology of the Grid. In Grid Computing: Making the Global Infrastructure a Reality, Berman F., Fox G. and Hey A., Eds., Wiley. pp. 217-249.
7. Gridbus Toolkit.
8. Guerraoui, R. and A. Schiper.(1996).”Fault-Tolerance by Replication in Distributed Systems”. In Proc. of Conference on Reliable Software Technologies. pp. 38-57.
9. J. Hofer and P. Brezany. “DIGIT: Distributed Classifier Construction in the Grid Data Mining Framework GridMiner-Core”. In Proc.Workshop on Data Mining and the Grid,2004.
10. Johnson, E. and H. Kargupta. Collective(2005). Hierarchical Cluster-ing from Distributed, Heterogeneous Data. In LargeScale Par-allel KDD Systems, Zaki M. and Ho C., Eds., SpringerVerlag. pp. 217-249.
11. Kandaswamy, G. A. Mandal and D. A. Reed (2008).”Fault Toler-ance and Recovery of Scientific Workflows on Computational Grids”. In Proc. of the 2008 Eighth IEEE International Sym-posium on Cluster Computing and the Grid (CCGrid) pp. 777-782.
12. Prodromidis, A. L. P. K. Chan and S. J. Stolfo (200)0. Meta-learning in Distributed Data Mining Systems: Issues and Approaches. In Advances in Distributed and Parallel Knowledge Discovery, Kargupta H. and Chan P., Eds., AAAI/MIT Press: Menlo Park, pp. 81-87.
13. Schroeder, B. and G. A. Gibson (2006).”A large-scale study of failures in high-performance computing systems”. In Proc. of the International Conference on Dependable Systems and Networks.
14. Talia, D. P. Trunfio and O. Verta (2005). ”Weka4WS: A WSRF-Enabled Weka Toolkit for Distributed Data Mining on Grids”. In Proc. 9th European Conference on Principles and Practice of Knowledge Discovery in Databases.