Improving Grid Systems Reliability with a Failure History Service

Catalin Leordeanu and Valentin Cristea
Faculty of Automatic Control and Computers,
University Politehnica of Bucharest
Email:{catalin.leordeanu,valentin.cristea}@cs.pub.ro

Thomas Ropars and Christine Morin
INRIA Rennes-Bretagne Atlantique, Rennes, France
Email: {thomas.ropars,christine.morin}@inria.fr

***Abstract*—The field of Grid computing has seen a lot of interest in recent years due to the demand for more and more computing power. There are already systems using thousands of nodes and this increase in size also leads to new challenges like the need for increased reliability. A very important part in ensuring reliability in distributed systems is failure detection and the need for accurate information about the reliability of the available nodes. This paper proposes a failure history service that allows services and applications executing in the grid to share their information about failures. This novel service ensures that the information about the current state of a node, as well as its failure history, is as accurate as possible even in a large scale dynamic system. By providing accurate failure data and allowing to analyze failures over time, this solution aims to increase the reliability of Grid systems.**

I. Introduction

With the growing scale of Grid systems, managing failures has become a main issue. Failure detection is the first step in failure management. However detecting failures in an asynchronous distributed system is complex because it is impossible to make the difference between a failed node and a delayed node, making failure detection unreliable.

In existing grid systems, like Vigne [1] or XtreemOS [2], failure detection is implemented at different layers. These systems are based on peer-to-peer overlays that implement algorithms to detect failed nodes. Furthermore services and applications executing in the grid can have their own failure detection mechanisms. The grid system can also provide a monitoring service [3]. These failure detectors may employ different mechanisms to detect node failures so they may provide different feedbacks about the same suspected node.

In this paper, we propose a failure history service as a basic block for reliable grid systems. The aim of this service is, first to allow all the applications and services executing in the grid to share their information about failures, and second to gather and analyze failure data coming from various failure detectors over time.

By comparing the output of various failure detectors, our failure history service can be used to improve failure detection and also to better understand the cause of the failures. Our service can also provide failures information to applications that don't have their own failure detection mechanisms. Failure history information could also be used to evaluate the reliability of grid nodes, for instance to allocate the more reliable nodes to priority applications.

The work on this failure history service is carried out in the context of Vigne grid middleware. Several issues have to be solved in the design of such a grid service. In this paper, we present our solution based a structured peer-to-peer overlay [4] to provide a scalable and accurate storage for failures data in the grid. We also describe how we manage those failures information to provide grid applications and services with data about the current status and the failure history of a node.

II. Failure Data Management

From the point of view of the grid applications and services, our service can be seen as a simple storage system, which can be queried to receive accurate information about the history of a node or its current state. We describe this storage system in the next section.

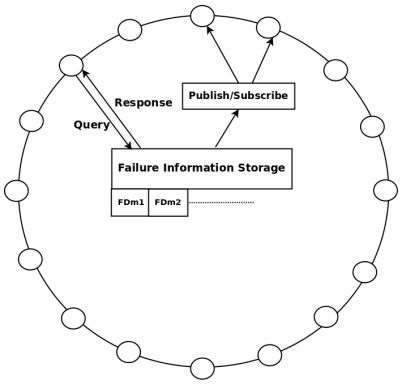

Figure 1. Capabilities of the Failure History Service

Our service provides two ways for a grid entity to get data about the failures of a grid node as shown in figure 1: query/response and publish/subscribe. An entity can query the service to receive the complete failure history of a node or its current status. The entire history is composed of the entire list of known failure events recorded from all of the available failure detectors. The current status of a node in the Grid represents the current feedback from the failure detectors suspecting the target node. The service can also notify node state changes to the interested entities.

Since we are relying on data from various failure detection mechanisms they could provide conflicting information on a

node state. We can define a degree of accuracy for failures information related to the similarity of the data provided by different failures detectors, e.g. if all the failure detectors see a node as failed, it must be failed. An application can subscribe to receive notifications about the state of another node when it is available with a certain degree of accuracy.

III. Storing Failure Information

Considering the scale of a Grid, the amount of failure data can be very large. Thus our failure history service must be based on a scalable storage solution. To store failure data provided by various failure detectors, we use a storage solution based on a structured peer-to-peer overlay, i.e. Pastry.

In Pastry, all nodes are associated with a unique identifier (nodeID) and are ordered clockwise in a logical ring. Failure data are distributed over all the grid nodes. The failure information and history of a node are stored on the k nodes with closest nodeID to that node. Thus data failure availability is ensured despite node failures. In our current prototype, the value of k is global and fixed for the entire system.

To get information about a node, a request can be sent to any node in the system. The request is routed to the nodes in the neighborhood of the target node using the routing mechanisms of Pastry. The storage system is able to provide failure data about the target even if it has failed and is unavailable at the time of the query because the response will be given by one of its neighbors which also contains all the failure information of the target node.

To ensure the consistency of the nodes history, we have to manage nodes having a different nodeID after recovering from a failure. In this case we can complete the identity of a node with its hardware address. If a node appears in the distributed system with the nodeID of a failed node and has the same hardware address then it is considered that it has recovered from the failure and the storage system will append its failure history accordingly.

To be able to evaluate the downtime of the nodes, we need to save data related to node failures and also node arrival. The actual stored data are organized as a collection of events which contain the following information:

- *Timestamp.* This is the local timestamp of the node that receives the event from the failure detector. It is used to order the events.
- *Failure detector ID.* The identity of the failure detector which sent the notification about the event.
- *Type.* The type of event. It depends on the failure detector. For instance, it can be a process failure or a node failure. It can also be an event signaling a node that joins the system.
- *NodeID.* The identity of the node which is the cause of the event. The NodeID is the same as the one used in the Pastry DHT.
- *Hardware address.* The hardware address of the node. Since a node may fail and then return with a different NodeID we use the hardware address as a second method to identify the node.

When a node joins the system after a failure, it has to update its history. To do so, it contacts its neighbors to get the events which took place while it was down and also to recover any lost data due to the failure.

We performed a preliminary evaluation of our storage system through simulation using SPLAY [5]. Our preliminary experiments show that under moderate churn, the use of data replication and update mechanisms ensures the availability and accuracy of the failure information. During our experiments, the total size of the stored data was not very large, but it does present a steady increase. This would require an increasing amount of space as the storage system would run for a longer amount of time. A policy for the archiving of old information could solve this problem.

IV. Conclusions and Future Work

This paper presents a grid failure history service that could be used as basic block in the design of reliable grid systems. This service allows applications and services executing in the grid to share their information about failure. By gathering information provided by various failure detectors available in the grid, the service aims at improving failure information provided to application and services to better handle the consequences of failures.

To implement such a services, several technical issues have to be solved. In the paper, we have described how the service could interact with grid entities. We have also detailed our solution to store failure data. This scalable solution, based on a structured peer-to-peer overlay, ensures the availability and the accuracy of the data even in a dynamic system. We are now working on the interface between the failure detectors and our service. This interface should allow grid users and developers to describe their failure detectors so that the service could make a better use of the data they provide.

Future work also includes the evaluation of the accuracy of a failure detector by comparing the information it provides with the one provided by other failure detectors. Another possible research direction is the evaluation of the reliability of a node based on its failure history. We could also use statistical methods to evaluate the probability of node failure.

References

[1] L. Rilling, "Vigne: Towards a Self-Healing Grid Operating System," in *Proceedings of Euro-Par 2006*, vol. 4128 of *Lecture Notes in Computer Science*, (Dresden, Germany), pp. 437–447, Springer, August 2006.

[2] XtreemOS, "http://www.xtreemos.eu."

[3] T. Ropars, E. Jeanvoine, and C. Morin, "GAMoSe: An Accurate Monitoring Service for Grid Applications," in *6th International Symposium on Parallel and Distributed Computing (ISPDC 2007)*, (Hagenberg Autriche), 07 2007.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of International Middleware Conference (Middleware 2001)* (R. Guerraoui, ed.), vol. 2218 of *Lecture Notes in Computer Science*, pp. 329–350, Springer, 2001.

[5] L. Leonini, E. Rivière, and P. Felber, "SPLAY: Distributed Systems Evaluation Made Simple (or how to turn ideas into live systems in a breeze)," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI'09)*, (Berkeley, CA, USA), pp. 185–198, USENIX Association, 2009.