Failure Detectors for Large-Scale Distributed Systems

Naohiro Hayashibara
nao-haya@jaist.ac.jp

Adel Cherif
adel@jaist.ac.jp

Takuya Katayama
katayama@jaist.ac.jp

Japan Advanced Institute of Science and Technology
Graduate School of Information Science
1-1 Asahidai, Tatsunokuchi, Ishikawa 923-1292, Japan

Abstract

This paper discusses the problem of implementing a scalable failure detection service for Grid systems. More specifically, traditional implementations of failure detectors are often tuned for running over local networks and fail to address some important problems found in wide-area distributed systems, such as Grid systems. We identify some of the most important problems raised in the context of Grids. We then survey recent propositions that can help in solving some of these problems.

Keywords failure detection, fault tolerance, distributed systems, Grid system, Grid computing.

1 Introduction

Failure detection services have received much attention in the literature and many protocols for failure detection have been proposed and implemented. Most of the proposed protocols were developed for local area networks and are not efficient in the context of a wide area distributed system consisting of many components and characterized by a high level of asynchrony, long message delay, high probability of message loss and which topology might change as a result of a reconfiguration.

A *Grid system* (may be called only *grid* in below) enables applications and individuals to efficiently use and share a large number of computing resources consisting of computers, networks, data stores, and software components that are distributed over several institutions and in different domains. The Grid underlying infrastructure can be reconfigured and can change dynamically. The large number of components and their distribution in the Grid as well as the dynamic structure of the Grid increases the risk of failures. Thus, providing a scalable and generic failure detection service as a basic service for Grids is of primary importance in such systems.

In this paper we discuss the implementation of a failure detection service for Grids. We identify some of the problems that need to be addressed in the context of the Grid. We briefly describe some of the proposed protocols for failure detection and discuss their effectiveness and how they address the identified problems.

The remainder of this paper is organized as follows. Section 2 presents general concepts of failure detectors. Section 3 explores problems related to the implementation of failure detectors in a wide area distributed system (particularly a Grid). Section 4 describes some of the proposed failure detection protocols and their implementations. Section 5 discusses the proposed protocols in the context of a Grid and their effectiveness in addressing the problems identified in Section 4. Section 6 presents some concluding remarks and future work.

2 Failure Detectors

Failure detectors monitor systems components in order to detect failures and notify registered components when a failure of one or more of the monitored components occurs. Monitored components may be any of the system components such as computers or processes.

2.1 Theoretical concepts

The importance of failure detectors in asynchronous distributed systems was highlighted by the work of Chandra and Toueg [2]. They showed that by augmenting the asynchronous system model with failure detectors that verify certain properties it becomes then possible to circumvent the FLP impossibility result [6] and to solve fundamental problems in distributed systems such as the consensus problem.

They considered distributed failure detectors where each process has access to a local failure detector module and each module monitors a subset of the processes in the system and maintains a list of those that it currently suspects to have crashed.

They classified failure detectors in a number of classes depending on two properties termed the completeness and accuracy properties. Completeness requires that a failure detector eventually suspects every process that eventually crashes. Accuracy restricts the mistakes that a failure detector can make. A class of particular interest is the class $\Diamond W$ of failure detectors. This is because $\Diamond W$ is the weakest failure detector to solve the Consensus problem in asynchronous distributed systems [1].

2.2 Implementation issues

Implementation of failure detection protocols are based on timeouts. There are two *interaction policies* between the failure detectors and the monitored components which abstract behaviors of monitoring protocols used by failure detectors to monitor system components [4]. One is the *push* model and the other is the *pull* model. We introduce these models in what follows.

2.2.1 The Push Model

In this model, monitored components are active and the monitor (failure detector) is passive. Each monitored component periodically sends messages (heartbeat messages) to the failure detector which is monitoring the component. A failure detector suspects a component failure, which means crashed component, when it fails to receive a heartbeat message from the component within a certain time interval (timeout) T (see Fig. 1).

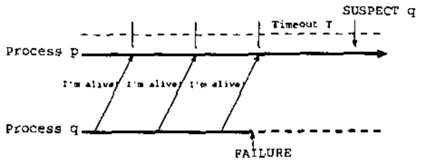

Figure 1. Push model

In the push model, the monitor suspects the failure of a component in the system after a certain time interval T. However, there is a large number of messages sent on the network. If there is a large number of monitored components such messages can flood the network.

2.2.2 The Pull Model

In this model, monitored components are passive while the monitor or failure detector is active. The monitor sends liveness requests (*"Are you alive?"* messages) periodically to monitored components. Upon the reception of a liveness request, the monitored component sends a reply to the monitor (see Fig. 2). When the monitor does not receive a reply from a monitored component within a certain time interval (timeout), it suspects the failure of the monitored component.

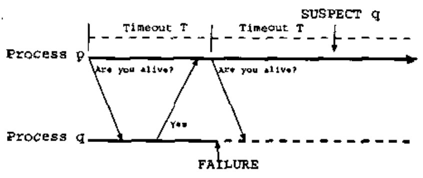

Figure 2. Pull model

In the pull model the load on the network is reduced and depends on the number of liveness requests sent by the monitor. However, the monitor may not suspect or detect the failure of a component until after it sends it a liveness request.

3 Implementing Failure Detectors in Grids

Many implementations of failure detection services have been proposed for local area networks. Such services may be efficient in the context of a LAN. However, they do not perform well in the context of a large scale distributed system. In the following we identify some of the problems that should be considered when implementing failure detectors in a Grid System.

Problem 1 (message explosion) Despite the large number of components that need to be monitored and their distribution in the system, the failure detector must prevent flooding or overloading the network with failure detection related messages.

Problem 2 (scalability) Applications running on a Grid system often require a large number of resources distributed over a wide area network. A failure detection service must be able to efficiently monitor such large number of resources. It must be able to quickly detect failures while minimizing the number of wrong suspicions.

Problem 3 (message loss) The failure detector must be sensitive to network conditions. In global networks, message losses occur with high probability as result of transitional faults or signal attenuation by the noise in the cable.

Also messages may be arbitrarily delayed. These might be identified as component failures by a failure detector. Thus, this problem may result in incorrect suspicions.

Problem 4 (flexibility) Different types of applications use the Grid. The failure detection service should be able to adapt to different types of applications and application requirements for monitoring system components. This will reduce the number of messages for failure detection and monitoring messages sent over the network.

Problem 5 *(dynamism)* The grid system is highly dynamic, with component leaving/joining all the time, usage patterns changing, network topology changes, etc. Failure detectors have to be aware of the Grid reconfiguration and adapt to it.

Problem 6 (security) Failure detectors could be used for DoS attack because the failure detector could behave like an optimal adversary and generate suspicions always at the worst time. The security problem will not be further addressed in this paper.

4 Existing approaches

There are some proposals in the literature to address the problems identified in the previous section (see Sect. 3). In this section, we briefly summarize some of the proposed protocols for failure detection, describe how they address the identified problems, and discuss their effectiveness and limitations.

4.1 Globus Failure Detection Service

Stelling *et al* [10] proposed a failure detection service for the *Globus toolkit*[1].

The architecture of the proposed failure detector service has two layers: the lower layer includes *local monitors* and the upper layer includes *data collectors*(see Fig. 3). The local monitor is responsible for monitoring the host on which it runs as well as selected processes on that host. It periodically sends heartbeat messages to data collectors including information on the monitored components. The data collectors receive heartbeats from local monitors, identifies failed components, and notifies applications about relevant events concerning monitored components.

This approach improves the failure detection time in a grid. Thus, it solves Problem 2.

However, it has drawbacks. In this approach, each local monitor broadcasts heartbeats to all data collectors. Therefore, this approach probably does not solve Problem 1. A grid may change its topology by component leaving/joining at runtime but the proposed architecture is static and does not adapt well to such changes in system topology (Problem 5).

[1]Globus toolkit has been designed to use existing fabric components, including vendor-supplied protocols and interfaces [7]

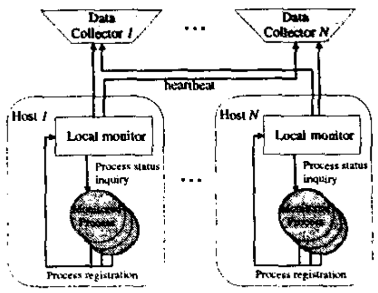

Figure 3. The architecture of the globus failure detection service

4.2 The Hierarchical approach

Felber *et al* [4] proposed a failure detection service with a hierarchical structure as shown in Fig. 4. In the figure, there are three subnets (LAN1,LAN2,LAN3), and each subnet has some failure detectors. Monitoring messages between a failure detector and monitored components are sent only within the subnet. Messages between failure detectors are sent over the subnets, messages between a failure detector and a client follows the same flow.

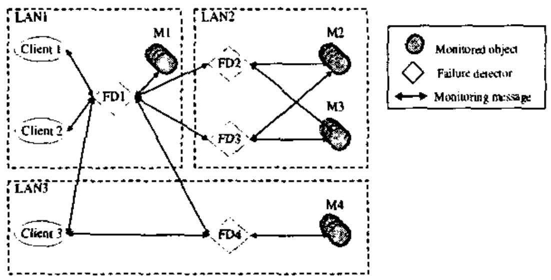

Figure 4. The hierarchical configuration

In this figure, there are two level hierarchies for the failure detection. *FD2* and *FD3* located in a subnet(*LAN2*) and *FD4* located in a subnet(*LAN3*) are in a low level hierarchy. They monitor monitorable objects located in the same subnet. *FD1* located in a subnet(*LAN1*) is in a high level hierarchy and it monitors monitorable object located in the same subnet and also monitors failure detectors located in lower hierarchies. Failure detectors in lower level hierarchies send information about suspected objects to failure detectors in upper level hierarchies.

In this approach, failure detectors obtain information of suspicions by either messages from other failure detectors or by monitoring target components directly. Therefore, the amount of messages is less than in the traditional approaches and it successfully addresses the scalability problem. However the hierarchical approach is based on a static tree structure and thus fails to address the dynamism problem.

4.3 Gossip-style protocols

Gossip-style failure detection(e.g., [11, 8]) is based on the idea that

Van Renesse *et al* [11] distinguish two variations of gossip-style protocols. One is named *basic gossiping* and the other is named *multi-level gossiping*.

In the basic gossiping protocol, a failure detector module is resident at each host in the network. It maintains a list with an entry for each failure detector module known to it. This entry includes a counter called heartbeat counter that will be used for failure detection. Each failure detector module picks another failure detection module randomly (without concern to the network topology) and sends it its list after incrementing its heartbeat counter. The receiving failure detector module will merge its local list with the received list and adopts the maximum heartbeat counter for each member. Occasionally each member broadcasts its list to recover from eventual network partitions (see Fig. 5). If a heartbeat counter for a host member A maintained at a failure detector at another host B has not increased after some timeout, host B suspects host A to have crashed.

To adapt it to a large scale network, a variant of the basic gossiping protocol called multi-level gossiping protocol is proposed. The multi-level gossiping protocol uses the structure of internet domains and subnets and their mapping into the IP address to identify domains and subnet and map them into different levels. Most gossip messages are sent by the basic protocol within a subnet, and few gossip messages are sent between subnets, and even fewer between domains (see Fig. 5).

Gossip-style protocols can address the problem of message explosion (Problem 1). The number of message is reduced even if this protocol is used in distributed systems with a large scale network. The number of message of the given domain only depends on the number of subnets in that domain.

In the gossip-style protocol, we can change some topology of the system dynamically. Therefore, it also solves Problem 5.

According to the result of [11], this protocol tolerates

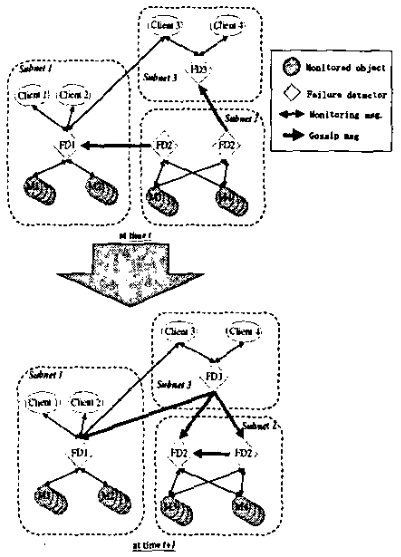

Figure 5. Gossip-style protocols

message losses (Problem 3). The detection time of failures increases gently by increasing the probability of message losses.

In contrast, there are drawbacks. Especially, this protocol does not work well when a large percentage of components crash or become partitioned away. Then, a failure detector may spend long time to detect crashed components by gossip messages. This protocol is quite simple but is less efficient than approaches based on hierarchical, tree-based protocols.

4.4 Failure Detector Adaptive

Network have one behavior during high-traffic hours and a completely different behavior during low-traffic hours. During peak hours, heartbeat messages may have a high probability of message losses, a higher expected delay and a higher variance of delay, than during off-peak hours. The network conditions also are changed very frequently due to bursty traffic. The failure detector adaptive [3] can adapt the changing conditions. Therefore, it can reconfigure itself dynamically and can meet given QoS requirements.

4.5 Lazy Failure Detection protocol

In the lazy failure detection protocol [5] processes monitor each other by using application messages whenever possible to get information on processor failures. This protocol requires that each message be acknowledged. In the absence of application messages between two processes control messages are used instead.

An application process can use three primitives to get information about monitored processes. The first one is the SEND primitive which is used by some process p_i to send an application message M to another process p_j. The SEND primitive includes some control information with the application message. The second one is the RECEIVE primitive used by p_i to receive an application message. The third one is the QUERY method which is used to know whether p_j is suspected to have crashed[2] (see Fig. 6).

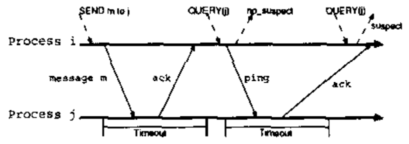

Figure 6. Lazy Failure Detection Protocol

The lazy failure detection protocol tries to reduce the number of monitoring messages sent by the failure detection protocol and thus addresses the message explosion problem. Although originally proposed in the context of a LAN the approach is also relevant to Grid systems. This approach depends highly on the communication patterns between application processes and may perform poorly for certain types of applications.

4.6 Application-based Failure Detection

Sergent *et al* [9] analyze several implementations of failure detectors in the context of a LAN. In this context, they propose to specialize the implementation of the failure detector to the communication pattern of the algorithm that uses them (Consensus in their case). By doing so, they are able to significantly reduce the number of failure detection messages that are being generated, and thus dramatically improve the overall performance of the algorithm.

This approach can be seen as a way to ensure that the failure detector becomes eventually silent (i.e., eventually stops sending control messages) and thus address Problem 1. However, unlike lazy failure detectors, application-based failure detectors are difficult to adapt to the context of Grid systems. Furthermore, application-based failure detectors must be embedded into algorithms, thus potentially leading to an explosion of messages in generic contexts such as a Grid system.

[2]The QUERY method returns *suspect* if p_i does not receive a reply from p_j until the timeout, otherwise it returns *no_suspect*.

5 Discussion

These protocols and approaches introduced in the previous section can address problems partially. However, each of these is not sufficient to implement an appropriate failure detector for Grids. We have to remember these problems to discuss the relationships between approaches/protocols and problems.

The globus failure detection service (see Sect. 4.1) can address Problem 2. The data collector periodically receives information of suspicions as heartbeats sent by the local monitor. Therefore, an application detects failed component with low latency by which it makes inquiries to the data collector. The hierarchical approach (see Sect. 4.2) also addresses this problem. Both approaches have a common drawback which is that they can not adapt to other configurations dynamically.

The problem of message explosion (Problem 1) is a critical problem to implement the failure detection service because it greatly lowers the performance of the service. The gossip-style protocol (see Sect. 4.3) and the hierarchical approach (see Sect. 4.2) address this problem. To reduce the number of messages, the former uses the random manner to send messages and the latter uses the hierarchical configuration which can reduce the number of messages when compared with traditional approaches. The lazy failure detection protocol (see Sect. 4.5) also addresses this problem.

Problem 3 decreases the accuracy of the failure detection as a result of wrong suspicions. We have to consider how to tolerate this problem. Experimental results published in [11] confirm that the gossip-style protocol is able to address this problem.

The problem of flexibility (Problem 4) is an interesting problem. The failure detection service should perform well for different types of applications. The application-based failure detection (see Sect. 4.6) and the lazy failure detection protocol depend on the application requirements and communication patterns. Thus, they fail to address Problem 4. However, they can solve Problem 1 for certain types of applications for which they were designed.

Problem 5 should be considered when designing the failure detection service in a grid. A Grid system may change its configuration during its execution. The failure detection service should be aware of the configuration changes, and should be able to adapt to the modified system. Gossip-style protocols can address this problem because they do not depend on the system configuration in its failure detection.

6 Conclusion

This paper identified problems for designing and implementing a scalable and generic failure detector service in a *Grid* system. Several approaches proposed in the literature were studied. Their effectiveness and limitations in addressing the identified problems were discussed. Each approach successfully addresses one or more of these problems but no approach provides a complete and satisfactory solution. Thus we feel that a combination of some of these approaches and concepts is required to provide an efficient failure detector service in a Grid.

As our future work, we plan to combine proposed approaches and to especially investigate a combination of the hierarchical approach with a gossiping protocol to implement an efficient failure detection service in a Grid system.

Acknowledgments

We would like to thank Xavier Défago for several discussions on the implementation of failure detectors, and for his valuable comments and feedback.

References

[1] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM (JACM)*, 43(4):685–722, Jul. 1996.

[2] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[3] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(2):13–32, 2002.

[4] P. Felber, X. Défago, R. Guerraoui, and P. Oser. Failure detectors as first class objects. In *Proc. of the 9th IEEE Int'l Symp. on Distributed Objects and Applications(DOA'99)*, pages 132–141, Sep. 1999.

[5] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In *Proc. of the 8th IEEE Pacific Rim Symp. on Dependable Computing(PRDC-8)*, 2001.

[6] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[7] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid. *Int'l Journal of Supercomputer Applications*, 2001.

[8] I. Gupta, T. D. Chandra, and G. S. Goldszmidt. On scalable and efficient distributed failure detectors. In *Proc. of the 20th Annual ACM Symp. on Principles of distributed computing*, pages 170–179. ACM Press, 2001.

[9] N. Sergent, X. Défago, and A. Schiper. Impact of a failure detection mechanism on the performance of consensus. In *Proc. of the 8th IEEE Pacific Rim Symp. on Dependable Computing(PRDC-8)*, pages 137–145, 2001.

[10] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. In *Proc. of the 7th IEEE Symp. on High Performance Distributed Computing*, pages 268–278, Jul. 1998.

[11] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Middleware '98*, 1998.