Failure Detection in Large-Scale Systems: a Survey

Marcia Pasin
UFSM – Brazil
marcia@inf.ufsm.br

Stéphane Fontaine
INPG – LI, France
Stephane.Fontaine@inria.fr

Sara Bouchenak
Univ. of Grenoble I – France
Sara.Bouchenak@inria.fr

Abstract

Failure detection is a basic service for building dependable systems. The large-scale distribution of computing systems naturally makes failure detectors much harder to build. Moreover, providing QoS (quality of service) guarantees in this context is a challenging task. The objective of this paper is twofold: (i) proposing a complete set of classification criteria to compare different failure detection mechanisms, and based on these criteria (ii) surveying the main failure detection solutions for large-scale distributed systems.

1. Introduction

Automatic failure detection is a basic service for building dependable systems. In large scale systems, maintaining QoS (quality of service) guarantees for failure detection [4] is not straightforward due size and geographical scalability. Automatic failure detection is usually proposed as ad-hoc solution, and commercial tools are often relatively static and slow. Implementations typically address only a small number of QoS guarantees and have reduced usability.

Large scale failure detection is subject to active research but still has some lacks. [9, and 14] do not provide flexibility (i.e. support different types of applications) and propose a generic way to detect faulty nodes. [4, 9 and 12] use limited environments. [5 and 6] do not treat size scalability, and [9, 14 and 16] are not adaptable. The state of the art of failure detection is discussed briefly in [5 and 10]. [16] compares mathematically and experimentally algorithms across detection time, mistake rate, network band-width, and message loss rate. [6] presents an experimental evaluation of different estimations and safety margins of a distributed push failure detector. [10] presents a survey about failure detection for grids but it uses a limited set of properties. Moreover, properties as message explosion and scalability are put together at the same level. [6] focus on implementation choices.

In this context, this paper proposes a complete set of classification criteria to compare different failure detection mechanisms and, based on these criteria, surveys the main failure detection solutions for large scale distributed systems.

2. Background on failure detection

Failure detectors [3 and 4] are process which collect information about faulty nodes. They maintain a list of suspects and a list of monitored nodes, which can be static or dynamic. A dynamic list maintains a variable number of nodes and is a more realistic model due to the ability to deal with network dynamism.

Failure detector finds faulty nodes using two types of keep-alive messages: heartbeat and ping. Heartbeat is a message periodically sent from a monitored process to the failure detector to inform that it is still alive [7]. If the heartbeat does not arrive before a timeout expires, the failure detector suspects the node is faulty. Ping is a message continuously sent from a failure detector to a monitored process. The failure detector expects to receive as answer an *ack*. If a keep-alive message fails, then a probe (i.e. a series of messages separated by a time interval) can be used to verify if a process is really faulty. In practice, it is difficult to distinguish faulty from healthy processes in asynchronous environments because message delays are unpredicted. So, processes which do not answer the failure detector question are addressed as suspect.

3. Classification for large scale failure detectors

In this section, we propose a set of classification criteria to compare different large scale failure detectors. These criteria are summarized in table 1.

Centralized versus distributed. A *centralized failure detector* is a single and monolithic module able to monitor different processes. They are easy to maintain, but represent single points-of-failure and are potential bottlenecks. Distributed failure detectors [2, 9 and 14] avoid these drawbacks and can be viewed as a set of failure detection modules, each one attached to a

different process in the system. Upon request, each module provides its own list of suspect nodes.

Push versus pull. There are two types of failure detectors with regard to keep-alive messages [7]: push and pull approach. The push approach uses heartbeats and the direction of both control and information flow is the same. In pull failure detectors, the direction of the flows is opposite. Ping messages are used in pull failure detectors. Authors [1 and 6] claim that heartbeat approaches have advantages over ping approaches. Heartbeats require half as many messages as ping failure detectors and estimation of timeout delay considers one-way trip messages. One advantage of ping failure detectors is that time control is executed only in the failure detector process.

Table 1 – Classification criteria for large scale failure detectors

Criteria	**Classification**
Architectural organization	*Centralized or distributed*
Keep-alive message	*Push or pull*
Application awareness	*Active or passive*
Liveness information	*Baseline or sharing*
Time values frequencies	*Adaptive or constant*
Application of time values	*Global or local*
Monitoring patterns	*All-to-all, randomized or neighborhood-based*
Propagation patterns	*One-to-all, randomized, circular or hierarchical*

Active versus passive. [7] classifies failure detectors with regard to application awareness. *Active protocols* continuously send or receive keep-alive messages. *Lazy or passive protocols* [8] take advantage of application's messages and, if data traffic is frequent, it can be sufficient for failure detection. However, [16] describes situations in which passive protocols are inadequate. In these situations, active protocols are required.

Baseline versus sharing. [16] classifies failure detectors with regard to share or not *liveness* information. In the *sharing approach*, a failure detector shares *liveness* information of monitored nodes with other modules. Adjacent nodes are typically used to take advantage of the network topology. Sharing algorithms differ in the type of information exchanged, and in the amount of keep-alive state they maintain. In the *baseline approach*, each module independently makes a decision about a suspect node.

Adaptive versus constant. Keep-alive frequency, timeouts and other time values can be adaptive or constant. Constant rate is easy to be applied because it is computed one single time for each node, for example, when the node joins the system but has limited efficiency in presence of network dynamism. Adaptive failure detectors were proposed by [2, 4, 5 and 11]. In [11], the idea is to use the time of later timeouts to forecast the time of arrive of the new heartbeats. Computing rates and timeouts is not a trivial task, and can include some network information. Adaptive and constant rates can be computed by mathematical formulas [4, 6 and 15] or, alternatively using heuristics.

Global versus local. Other issue with regard to time values is about the application of time values individually or globally. A simple approach is use a global keep-alive message rate for all nodes. This approach makes sense if nodes are homogeneous and when all nodes exhibit the same session duration. Alternatively, if nodes are heterogeneous, individual nodes can compute their own rates.

Before introducing other criteria, we will describe the operation phases of a large scale failure detector: *normal, propagation* and *reconfiguration.* Failure detectors modules send keep-alive messages to monitored nodes in normal phase. When a failure is detected, propagation phase starts and failure information is sent to other modules. Reconfiguration starts after propagation, and comprises local and global ones. Local reconfiguration occurs when the current module repairs the failure. For example, the failure detector can remove a node from a group. Global reconfiguration happens when information about faulty components is propagated to the other failure detection modules, which will be able to repair them system view. Two performance issues in large scale failure detection are *monitoring* and *propagation patterns*. Efficient monitoring allows fast detection time whilst fast propagation helps to maintain system consistency.

Monitoring patterns. Monitoring patterns are related with communication between failure detection modules and monitored nodes during normal phase. Monitoring patterns can be all-to-all, randomized, and neighborhood-based. In *all-to-all failure detection,* each module sends keep-alive messages to all monitored processes. Indeed, for small groups this approach could run efficiently but the scalability is limited if the number of processes grows up due massive network traffic.

In a *randomized failure detector* [9 and 13], each node maintains a list for each member its address and a time value used for failure detection. Each node in a group randomly selects other k nodes to send a keep-alive message. Gossiping protocol [13] is a special kind of randomized failure detector based on heartbeats. The

gossiping is very efficient for small groups and has a multi-level version for large groups. Randomized failure detectors can improve scalability and reduce detection time via random and periodic communication among monitored nodes. Detection time depends on the probability of being randomly selected.

In the *neighborhood-based approach*, each process sends keep-alive messages to adjacent nodes. Performance is improved by restricting communication and taking advantage of the locality. Neighbors are selected statically and do not change over the time unless a failure had been detected. In this case, a reconfiguration is required to remove the faulty node and select a new neighbor, and network information must be taken in account.

Failure propagation. When a node is addressed as faulty, this information must be propagated to other modules. Failure propagation is time consuming in large scale systems, and different approaches were proposed to improve propagation time: one-to-all, randomized, (ring or) circular space, and hierarchical. With *one-to-all propagation pattern*, failures are immediately propagated to all failure detection modules. Network traffic can be intense if failures and churn are frequent. The implementation could take advantage of IP-multicast for performance reasons.

In the *randomized propagation pattern*, a module or a group of modules is chosen to receive information about a failure from the current failure detection module. Communication is cheaper than one-to-all propagation pattern but propagation time depends on the possibility of being selected by another node.

Circular failure detectors [12] arrange nodes in a virtual ring. Communication is achieved only between adjacent nodes. Therefore, when a new node joins or leaves the ring, adjacent nodes must be rearranged. Another drawback is mapping the virtual ring to the network topology which cannot be trivial Failure propagation can be time consuming for large rings.

Hierarchical failure detectors [2, 12 and 13] arrange nodes into a multi-level hierarchy and partition monitoring in small groups. Failures are reported along a tree to improve scalability. Hierarchical failure detectors take into account the network infrastructure to be efficient and are commonly used in large-scale systems to connect small groups running the other approaches.

4. Discussion

Based on classification formerly presented, in this section we survey the main failure detection solutions for large scale distributed systems.

Sharing liveness information to improve detection time. Detection time could be beneficiated by sharing *liveness* information with neighborhood: the node which first detects the failure announce to the others. In [16], experiments were conducted in a network with 2,000 nodes and detection time was almost constant, even if the number of neighbors grows up. In addition, the same authors claim that sharing information helps to tolerate high churn rates.

Adaptive solutions. In large scale networks, adaptive algorithms [1, 8 and 15] are proved to be more efficient than algorithms with constant timeout. These approaches support different application requirements and network changes. Adaptive approaches have some open drawbacks: how to find an appropriate mechanism to tuning the timeout in the failure detector? How to choose a frequency in which keep-alive messages must be issued? [15] proposes using a router and, based on the current network traffic, dynamically adapting heartbeats in a large scale system. The authors claim that this solution offers short response time, good stability and fine robustness. However, the router can be a bottleneck. A more interesting solution is the accrual failure detector [11] which provides suspicious information about faulty nodes. Currently, there is no experimentation in large-scale systems but it should be a very promissory solution because it is application targeted to decide about the QoS level of the provided information.

Global or local timeout values. Maintaining, calculating, and tuning a timeout for each process in a large scale system could be unacceptable. On the other side, applying the same timeout for local and remote process could be unacceptable too. If this situation is unacceptable in large scale systems, where failure detectors could monitor several nodes simultaneously, a hybrid approach classifies monitored processes in small groups and addresses for each group a keep-alive rate.

Faulty versus busy nodes. Typically failure detectors are implemented at the process-level, with one process per machine and following the crash failure model. Faulty processes are simulated by the crash of an entire node. The effect is that a busy node cannot be differentiated from a faulty node because both are not able to answer the failure detector before the timeout expires. [5] applies failure detection at the application-level. In this approach, a node supports a set of different applications monitored by one local failure

detector, which satisfies a given level of QoS previously informed for each application.

Gossiping protocols. Gossiping protocols are less network bandwidth consuming than all-to-all protocols. Second [16], multi-level gossiping avoids message explosion and treats message loss. Gossip protocol also is efficient because it does not take into account the network structure. However, authors [5 and 16] report some drawbacks of this protocol. For example, when a lot of nodes crashed, the detection time could be very long in the basic (flat) protocol. In addition, developers must pay attention when they choose the randomized function. A non-equitable function could make some nodes more randomly selected than others and it allows that some non-popular nodes could undesirably maintain out-of-date information about other nodes.

Hierarchical protocols. Hierarchical protocols are proved to be efficient in large scale groups. They could be applied to circular space failure detectors, all-to-all failure detectors and randomized protocols. For example, failure propagation for all nodes in a circular space is time consuming in large rings and because messages follow the ring. To allow fast failure propagation, the ring could be divided in small groups with a leader per group. If a failure is detected by a node in a group, its group leader informs all other group leaders about the failure. Each group leader, in a high hierarchic, will inform its group members about the faulty node. [14] connects local groups in a hierarchical structure avoiding new leader election. The IP address is used to divide nodes in small groups. Most monitoring messages are sent by the basic monitoring protocol (i.e., a randomized protocol) within small groups while few monitoring messages are sent between groups. Using the IP address allows the hierarchical structure remains unaware of the network topology because the division of the network in small groups is done dynamically.

5. Conclusions

In the last years, experimental research in failure detection has been conducted taking into account large-scale systems. However, (practical) monitoring or failure detection tools are still required. It remains a challenge providing QoS guarantees. The solution must be as little intrusive as possible and must consider issues as network dynamism, scalability, performance, consistency, heterogeneity, and application flexibility.

Determinate a suitable timeout value for monitored nodes in large scale systems still remains a difficult task. Current solutions usually divide the system in small groups and use neighborhood information to improve the consistency in the process of failure detection. Future works are investigating failure detection in multi-application environments.

Bibliography

[1] M. Bertier, O. Marin, and P. Sens. Implementation and Performance Evaluation of an Adaptable Failure Detector. Proc. DSN 2002.

[2] M. Bertier, O. Marin, and P. Sens. Performance Analysis of Hierarchical Failure Detector. Proc. DSN 2003.

[3] T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. J. ACM, 43(2), 1996.

[4] W. Chen, S. Toueg, and M. K. Aguilera. On the Quality of Service of Failure Detectors. IEEE Transactions on Computers. V.51, N.5. May 2002.

[5] X. Defago et al. On the Design of a Failure Detection Service for Large Scale Distr. Systems. Proc. PBit 2003.

[6] L. Falai and A. Bondavalli. Experimental Evaluation of the QoS of Failure Detectors on WAN. Proc. DSN 2005.

[7] P. Felber, X. Defago, R. Guerraoui, and P. Oser. Failure Detectors as First Class Objects. Proc. DOA 1999.

[8] C. Fetzer, M. Raynal, and F. Tronel. An Adaptive Failure Detection Protocol. Proc. 8th IEEE PRDC 2001.

[9] I. Gupta, T. D. Chandra, and G. S. Goldszmidt. On Scalable and Efficient Distributed Failure Detectors. Proc. ACM Symp. PODC. Aug. 2001.

[10] N. Hayashibara, A. Cherif, and T. Katayama. Failure Detectors for Large Scale Systems. Proc. SRDS 2002.

[11] N. Hayashibara, X. Defago, R. Yared, and T. Katayama, The Accrual Failure Detector. Proc. SRDS 2004. pp.66-78.

[12] A. Mislove and P. Druschel. Providing Administrative Control and Autonomy in Structured Peer-to-Peer Overlays. Proc. IPTPS 2004.

[13] R. van Renesse, Y. Minsky, and M. Hayden. A Gossip-Style Failure Detection Service. Proc. Middleware 1998.

[14] K. C. W. So, and E. G. Sirer. Latency and Bandwidth-Minimizing Failure Detector. Proc.EuroSys 2007.

[15] N. Xiong, Y. Yang, J. Chen, and Y. He. On the Quality of Service of Failure Detectors Based on Control Theory. Proc. AINA 2006. pp.75-80.

[16] S. Zhuang et al. On Failure Detection Algorithms in Overlay Networks. Proc. IEEE INFOCOM Conference 2005.