A New Failure Detector to Detect Failures in a Distributed System

Sheikh Tania, Jannatul Maowa, Afsana Ahmed Munia

Abstract— Process groups in distributed applications and services rely on failure detectors to detect process failures *completely*, and as *quickly*, *accurately*, and *scalably* as possible, even in the face of unreliable message deliveries. Failure detector is a simulation application that is responsible for detection of node failures or crashes in a distributed system. It is impossible to distinguish with certainty a crashed process from a very slow process in a purely asynchronous distributed system. Some parameters are used to evaluate a Failure Detector such as complete, quick, accurate, and scalable even in the face of unreliable message deliveries. In contrast to previous failure detectors that have been used to circumvent impossibility results, the heartbeat failure detector is implementable, and its implementation does *not* use timeouts. Here we introduce a failure detector which is based on heartbeat message.

Index Terms— Distributed system, failure detection, asynchronus system, simulation, crash.

1 Introduction

Failure detector is an application that is responsible for detection of node failures or crashes in a distributed system. A failure detector is a distributed oracle that provides hints about the operational status of other processes. The design and verification of *fault- tolerant* distributed system is a difficult problem. The *detection of process failures* is a crucial problem, system designers have to cope with in order to build fault tolerant distributed platforms. Unfortunately, it is impossible to distinguish with certainty a *crashed process from a very slow process* in a purely asynchronous distributed system.

The ability of the failure detector to detect process failures *completely* and *efficiently*, in the presence of unreliable messaging as well as arbitrary process crashes and recoveries, can have a major impact on the performance of these systems. "Completeness" is the guarantee that the failure of a group member is eventually detected by every non-faulty group member. "Efficiency" means that failures are detected *quickly*, as well as *accurately* (i.e., without too many mistakes).

The recent emergence of applications for large scale distributed systems has created a need for failure detector algorithms that minimize the network load (in bytes per second, or equivalently, messages per second with a limit on maximum message size) used, as well as the load imposed on participating processes. Failure detectors for such settings thus seek to achieve good *scalability* in addition to efficiency, while still (deterministically) guaranteeing completeness. Recently, Chen et al. [6] proposed a comprehensive set of metrics to measure the Quality of Service (QoS) of complete and efficient failure detectors. This paper presented three primary metrics to quantify the performance of a failure detector at *one* process detecting crash-recovery failures of a *single* other process over an unreliable network. The authors proposed failure detection time, and recurrence time and duration times of mistaken detection as the primary metrics for complete and efficient failure detectors. However, the paper neither deal with the optimal relation among these metrics, nor focussed on distributed or scalable failure detectors

2 Related Work

Chandra and Toueg [5] were the first to formally address the completeness and accuracy properties of failure detectors. Subsequent work has focused on different properties and classifications of failure detectors. This area of literature has treated failure detectors as *oracles* used to solve the Distributed Consensus/Agreement problem [12], which is unsolvable in the general asynchronous network model. These classifications of failure detectors are primarily based on the weakness of the model required to implement them, in order to solve the Distributed Consensus/Agreement problem [10].

Proposals for implementable failure detectors have sometimes assumed network models with weak unreliability semantics eg., timed-asynchronousmodel [6], quasi-synchronous model [2], partial synchrony model [11], etc. These proposals have treated failure detectors only as a tool to efficiently reach agreement, ignoring their efficiency from an application designer's viewpoint. For example, most failure detectors such as [11] provide *eventual* guarantees, while applications are typically concerned about *real* timing constraints.

In most real-life distributed systems, the failure detection service is implemented via variants of the "Heartbeat mecha-

- *Sheikh Tania is currently pursuing masters degree program in computer scienceand engineering in Bangladesh University of Engineering and Technology, Bangladesh, PH-8801717042036. E-mail: sheikhtania327@gmail.com.*
- *Jannatul Maowa is currently pursuing pursuing masters degree program in computer scienceand engineering in Bangladesh University of Engineering and Technology, Bangladesh, PH-8801670685446. E-mail: jms.shopno@gmail.com.*
- *Afsana Ahmed Munia is currently pursuing masters degree program in computer scienceand engineering in Bangladesh University of Engineering and Technology, Bangladesh, PH-8801713492457. E-mail: afsana.106@gmail.com*

nism" [1, 2, 3, 4, 6, 7, 8], which have been popular as they guarantee the completeness property. However, all existing heartbeat approaches have shortcomings. Centralized heartbeat schemes create hot-spots that prevent them from scaling. Distributed heartbeat schemes offer different levels of accuracy and scalability depending on the exact heartbeat dissemination mechanism used, but we show that they are inherently not as efficient and scalable as claimed.

Probabilistic network models have been used to analyze heartbeat failure detectors in [4, 6], but only with a *single* process detecting failures of a *single* other process. [6] was the first paper to propose metrics for non-distributed heartbeat failure detectors in the crash-recovery model. These metrics were not inclusive of scalability concerns.

In this paper we proposed a new approach of Failure Detector that has strong accuracy, strong/weak completeness and the approach is scalable in terms of Network load (messages per unit time).

3 Scalable And Efficient Failure Dtetectors

The first formal characterization of the properties of failure detectors was offered in [4], which laid down the following properties for distributed failure detectors in process groups:

- {**Strong/Weak**} **Completeness**: crash-failure of any group member is detected by *{all/some}* non-faulty members.
- **Strong Accuracy**: no non-faulty group member is declared as failed by any other non-faulty group member.

[4] also showed that a perfect failure detector i.e., one which satisfies both Strong Completeness and Strong Accuracy, is sufficient to solve distributed Consensus, but is impossible to implement in a fault-prone network.

Subsequent work on designing efficient failure detectors has attempted to trade off the Completeness and Accuracy properties in several ways. However, the completeness properties required by most distributed applications have lead to the popular use of failure detectors that guarantee Strong Completeness always, even if eventually. This of course means that such failure detectorscannot guarantee Strong Accuracy always, but only with a probability less than 1. For example, all-to-all (distributed) heartbeating schemes have been popular because they guarantee Strong Completeness (since a faulty member will stop sending heartbeats), while providing varying degrees of accuracy.

The requirements imposed by an application (or its designer) on a failure detector protocol can thus be formally specified and parameterized as follows:

1. Completeness: satisfy eventual Strong Completeness for member failures.
2. Efficiency:
(a) Speed: every member failure is detected by *some* non-faulty group member within T time units after its occurrence (T >>worst-case message round trip time).
(b) Accuracy: at any time instant, for every nonfaulty member Mi not yet detected as failed, the probability that no other non-faulty group member will (mistakenly) detect Mi as faulty within the next T time units is at least $(1-PM(T))$.

T and $PM(T)$ are thus parameters specified by the application (or its designer). For example, an application designer might specify $T = 3$ seconds, and $PM(3 \text{ seconds}) = 10^{-8}$.

4 Theoritical Concept of The Proposed Model

Our proposed mechanism for detecting failure is a diffusion work of Heartbeat algorithm proposed in [1]. The output of the failure detector module of HB at a process p is a vector of counters, one for each neighbor q of p. If neighbor q does is live, its counter increases with no bound. If q crashes, its counter eventually stops increasing. The basic idea behind an implementation of HB is that each process periodically sends a heartbeat message to every other process and every process receiving a heartbeat increases the corresponding counter. Though quiescent reliable communication can be achieved with HB failure detector that can be implemented without timeouts in systems with process crashes and lossy links, the major drawback of the procedure is message explosion that is the network will be overloaded with failure detection related messages. For a network with n number of nodes it needs to transmit n^2 messages periodically. HB is not like existing implementations of failure detectors (in Ensemble and Phoenix, have modules that are also called heartbeat [9, 4]). Although existing failure detectors are also based on the repeated sending of a heartbeat, they use timeouts on heartbeats in order to derive lists of processes considered to be up or down; applications can only see these lists. In contrast, HB simply counts heartbeats and shows these counts to applications.

We propose a new approach in which only a single node becomes a failure detector and every other node periodically sends a heartbeat message to it. The FD maintains a vector of counters one for each neighbor and increases the counter when receives a HB message from corresponding neighbor. When the FD detects a node as suspected it announces the node as Suspected to all other nodes. Periodically FD node shows the vector counts for all other nodes and others show the suspected list of nodes. For a network with n number of nodes it needs to transmit n messages periodically. The significant reduction of network load makes the proposed approach very much efficient.

5 Features of Proposed FD

Our proposed heartbeat failure detector has the following features. The output of the algorithm at failure detector, FD is a vector $\{s_1, s_2, s_3, \dots, s_n\}$ where s_n is the status of the node n and s_n is a non negative integer. Clearly s_n increases when node n is live and stops increasing when n crashes. Again FD prints the suspected node lists periodically.

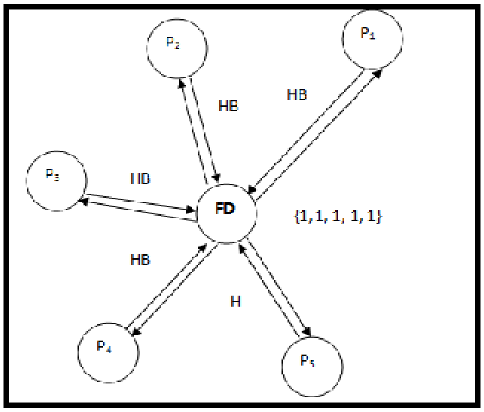

Fig. 1. Proposed FD

6 Proposed Algorithm

Task 1: Repeat periodically
 for all p ε π do
  Send HB to FD
 end for

Task 2: When receive HB from some p
 if p ε SuspectedFD then
  HostStatusFD[p] + +
 else
  Update HostStatusFD[p]
  Send RECOVERED to all p ε π
 end if

Task 3: Repeat periodically
 for all p ε π do
  print HostStatusFD[p]
  Update SuspectedFD
  Send SuspectedFD to all p ε π
 end for

for all p ε π do
 print SuspectedListp

end for

7 Experimental Result

Our proposed Failure Detector maintains the basic properties correctness and accuracy strongly. The main contribution of this work is the signi_cant reduction of network load.
In HB algorithm the network needs to transmit n2 messages per unit of time.
In our proposed algorithm the network needs to transmit n messages per unit time. If any node is suspected then it needs to transmit n messages again. So total cost for each period of time is maximum 2n.

8 Conclusion

In our simulation we initially introduce two processes as failure detectors. So the probability of failure of FD's reduces. If this probability can be reduced more the quality of the approach will be increased. The problem can be solved if number of FDs can be increased as the size of the system increases. Our future mission is to significantly reduce the probability by adding more backup FD's.

References

[1] M. K. M. K. Aguilera, W. Chen, and S. Toueg. Heartbeat: a timeout-free failure detector for quiescent reliable communication. In *Proceedings of 11th International Workshop on Distributed Algorithms (WDAG'97)*,

[2] C. Almeida and P. Verissimo. Timing failure detection and real-time group communication in real-time systems. In *Proceedings of 8th Euromicro Workshop on Real-Time Systems*, June 1996.

[3] R. Bollo, J.-P. L. Narzul, M. Raynal, and F. Tronel. Probabilistic analysis of a group failure detection protocol. In *Proceedings of 4th International Workshop on Object-Oriented Real-Time Dependable Systems*, 1998.

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

[5] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. In *Proceedings of 30th International Conference on Dependable Systems and Networks (ICDSN/FTCS-30)*, June 2000.

[6] S. A. Fakhouri, G. S. Goldszmidt, I. Gupta, M. Kalantar, and J. A. Pershing. Gulfstream - a system for dynamic topology management in multi-domain server farms. Technical Report RC 21954, IBM T.J. Watson Research Center, February 2001.

[7] C. Fetzer and F. Cristian. Fail-awareness in timed asynchronous systems. In *Proceedings of 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, pages 314–321a, May 1996.

[8] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proceedings of International Conference and Distributed Systems Platforms and Open Distributed Processing (IFIP)*, 1998.

[9] K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, December 1993.

[10] J. M. Helary and M. Hurfin. Solving Agreement problems with failure detectors; a survey. *Annals of Telecommunications*, 52(9-10):447–464, September-October 1997.

[11] M. Larrea, A. Fernandez, and S. Arevalo. Optimal implementation of the weakest failure detector for solving Consensus. In *Proceedings of 19th Annual ACM-SIGOPS Symposium on Principles of Distributed Computing (PODC 2000)*, July 2000.

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