A Hierarchical Failure Detection Service with Perfect Semantics*

Francisco V. Brasileiro, Jorge C. A. de Figueiredo, and Lívia M. R. Sampaio

Universidade Federal da Paraíba
Departamento de Sistemas e Computação
Av. Aprígio Veloso, 882
58.109-970 Campina Grande, Paraíba, Brazil
Tel: (+55) 83 310 11 19 Fax: (+55) 83 310 11 24
{fubica,abrantes,livia}@dsc.ufpb.br

Abstract

A *failure detector* is an important abstraction to support the implementation of higher level fault-tolerant protocols on distributed asynchronous systems. In this paper we show, via a counter example, that using the best possible failure detector of a given class is not always the key to achieve the best performance for a specific higher level consensus protocol. We argument that this behaviour is due to structural limitations of the consensus protocol that are unlikely to be circumvented, unless stronger abstractions are provided. Thus, we advocate that the designer of a generic failure detection service should concentrate her efforts in implementing the strongest failure detector possible - even if it is not the best within its class, instead of trying to implement the best failure detector of a weaker class. Following this philosophy, we present the basis of the design of a hierarchical failure detection service with the strongest semantics known, namely that of a perfect failure detector.

1 Introduction

In the last few years, with the increasing popularity of distributed applications, the number of applications with dependability requirements has greatly increased. For these applications, failures in the components that form the system can cause undesirable consequences, such as loss of income, clients, information and confidentiality. To attain their dependability requirements, these applications must execute extra mechanisms that are able to tolerate faults.

*This work is supported by grants from CTPETRO and CNPq/Brazil (300.904/94-0 and 300.646/96).

Most off-the-shelf infrastructures for deploying distributed applications are characterised by the absence of upper bounds on both message transmission and scheduling delays, *i.e.* they are asynchronous systems. Unfortunately, a well known impossibility result presented in [FLP85] shows that it is impossible to reach consensus [Fis83] (a basic building block for many fault tolerance mechanisms) in a pure asynchronous distributed system subject to failures. The following observation is the core of the impossibility result presented in [FLP85]: due to the uncertainty on communication and scheduling delays, it is impossible to differentiate a processor that has failed from one that is simply slow.

In face of this result, a number of models that strengthen the pure asynchronous distributed system model, allowing solutions for the consensus problem have been defined. Among them, the asynchronous distributed system model augmented by an unreliable failure detector [CT96] has been widely studied. This system model assumes the existence of an "oracle", named a failure detector, that is able to give an idea of which processors have crashed. Although this oracle may make mistakes (*e.g.* by suspecting a correct process), the information it provides is precise enough to allow deterministic solutions for the consensus problem [CT96].

The semantics of a failure detector is characterized by two properties namely: *completeness* and *accuracy*. The first property defines how broad is the reach of the failure detector, while the other restricts the mistakes that the failure detector may make. The stronger the semantics of the failure detector, the less restrictive the assumptions required to guarantee the correctness of the higher level protocol [CT96] and, possibly, the more efficient the protocols. On the other hand, one can argue that the weaker the semantics of the failure detector, the simpler its implementation. In [CHT96] it is proved that the weakest class of failure detectors that allows a solution to the consensus problem is named $\diamond S$[1] and is defined by the following properties [CT96]:

- **strong completeness**: eventually every process that crashes is permanently suspected by every correct processes.
- **eventual weak accuracy**: there is a time after which some correct process is never suspected by any correct processes.

Following this result, several implementations of $\diamond S$ failure detectors have been reported in the literature. To allow the comparison of different implementations of failure detectors, three primary quality of service (QoS) metrics have been defined [CTA00] for the failure detector module of a process q (FD_q) that monitors a process p:

- **detection time (T_D)**: it is the time that elapses since p has failed until it is permanently suspected by FD_q.
- **mistake duration (T_M)**: it is the time that takes for FD_q to stop mistakenly suspecting a correct p.

[1] In fact, [CHT96] proves that $\diamond W$ is the weakest failure detector to solve consensus, however [CT96] shows that $\diamond W$ and $\diamond S$ are equivalent classes of failure detectors.

- **mistake recurrence time** (T_{MR}): it is the time elapsed between two consecutive mistakes of FD_q.

Intuitively, the best $\diamond S$ failure detector module would have: i) the smallest T_D; ii) the smallest T_M; and iii) the largest T_{MR}. Unfortunately, the use of better failure detectors does not necessarily guarantee better performance to the higher level protocols. In this paper we show, via a counter example, that the performance of a consensus protocol based on a $\diamond S$ failure detector may improve when the QoS of the failure detector it uses worsens. This is because the performance of the protocol is dependent on the performance of a process that plays the role of the round coordinator that first reaches a decision.

A careful design of a failure detection service can increase the performance of higher level protocols, however this can only be achieved by using *ad hoc* implementations tailored for a particular setting and protocol [SDS99]. Therefore, we advocate that the designer of a generic failure detection service should not strive to implement the best failure detector possible of a given class. Rather, she should concentrate her efforts in building *any* failure detector of the *strongest* class possible. Following this idea we present in this paper the design of a hierarchical failure detection service of the class P (perfect) [CT96]. The class of perfect failure detectors is the strongest class among those proposed in [CT96]. We believe that it is possible to build consensus protocols on top of a perfect failure detector that not only are able to tolerate more faults, but are also simpler and more efficient than those built on top of a $\diamond S$ one.

The rest of the paper is structured as follows. Section 2 discusses the basis of our argument on the inadequacy of trying to implement better failure detectors to provide generic failure detection services. Then, in Section 3 we present the basis of the design of a hierarchical failure detection service that provides the semantics of a perfect failure detector. Finally, Section 4 concludes the paper with our final remarks.

2 The Counter Example

The counter example uses one of the consensus protocols based on the rotating coordinator paradigm presented in [CT96]. We will use the protocol that is supported by a $\diamond S$ failure detector and whose functioning can be summarised as follows. n processes, from which at most $\lfloor (n-1)/2 \rfloor$ may fail, participate in the consensus. Each process has access to its local $\diamond S$ failure detector module that gives it hints about which processes might have failed.

The protocol is executed in asynchronous rounds and it is assumed that all processes have an a priori knowledge of the identity of the process that plays the role of the coordinator of each round. Within each round the protocol proceeds in the following four phases. In the first phase every process sends its estimate of the decision value to the current round coordinator. In the second phase the round coordinator gathers $\lceil (n+1)/2 \rceil$ estimates, chooses one of them[2] and send it to all processes as their new estimate value. In phase three processes wait for the new estimate from the round coordinator. To avoid the possibility

[2]This choice must respect a locking mechanism that is not important for the purpose of this paper; the interested reader should refer to [CT96].

of blocking due to a faulty coordinator, processes constantly query their failure detector module to assess the round coordinator status. If the round coordinator is suspected the process sends a *nack* message to the round coordinator (notice that a suspicion does not mean that the round coordinator has indeed failed). On the other hand, if it receives the new estimate value from the round coordinator it adopts the new estimate value and sends an *ack* message to the round coordinator. In phase 4 the round coordinator collects $\lceil (n+1)/2 \rceil$ replies (*acks* and *nacks*) and if all replies are *acks* it decides for the estimate value it has proposed. The processes are informed of the decision via the execution of a reliable broadcast protocol [CT96]. A process finishes the execution of the protocol when it reliably delivers the decision value.

As discussed in Section 1, based on the QoS metrics proposed in [CTA00], improvements on the performance of a $\diamond S$ failure detector (or any failure detector, for that matter) can be achieved by reducing T_D, reducing T_M, or increasing T_{MR}. Since in our example we will only consider the most frequent runs where processes are not faulty, we will disregard T_D.

We will first consider all runs of the consensus protocol described above where processes are non-faulty, and their $\diamond S$ failure detector modules behave in such a way that $T_M = 0$ and $T_{MR} = \infty$. This is to say that in the runs selected the failure detector modules do not make mistakes and behave as failure detector modules of class P. Considering only these runs, and the QoS metrics used, it is not possible to implement better $\diamond S$ failure detector modules. Let us assume that process p is the coordinator of the first round of all executions of the consensus protocol and that for some reason p (or its communication with the other processes) is much slower that the other processes (say k times slower). Since the failure detector modules do not make mistakes, p will never be suspected by any process and the performance of the protocol will be hugely influenced by the performance of p.

Now consider the same runs as above but supported by worse $\diamond S$ failure detector modules. Let us assume that these failure detector modules have a non-zero T_M and a much smaller T_{MR} and, because of that, causes all other processes to suspect p and advance to round two. Since all processes are non-faulty the faster processes that advanced to round two can reach a decision in that round without the aid of p. For that to happen T_{MR} must be such that all failure detector modules mistakenly suspect the slow p but do not suspect any of the other faster processes. Given the time required to execute the consensus protocol and the speed of the faster processes it is not difficult to find values for T_{MR} and k such that the second scenario will always outperform the first one.

We have modeled the consensus protocol based on the rotating coordinator paradigm presented in [CT96] by means of a Coloured Petri Net (CPN). Without loss of generality, we have modeled a 3-process instance of the referred consensus protocol that is able to tolerate one process failure. In order to make performance analysis of the protocol, we simulated the CPN model using the Design/CPN tool [CPN93]. Two scenarios were analysed, by tunning the failure detector QoS metrics previously discussed and using different communication delays between the processes. First, we have set the T_M and T_{MR} to 0 and ∞, respectively. Such a configuration corresponds to a failure detector that does not make mistakes (a failure detector that behaves as a perfect one). By increasing the T_M value and reducing the T_{MR} value we have worsen the QoS of the $\diamond S$ failure detector. We have simulated the two configurations within three situations, namely: i) communication delays between processes

is the same; ii) communication delays between the coordinator and processes is 5 times slower than between the other processes; and iii) communication delays between the coordinator and processes is 10 times slower than between the other processes. Table 1 shows the consensus time[3] obtained in the simulations (time is expressed in ms).

T_M	T_{MR}	$Delay_{coordinator->processes}$	$Delay_{processes->processes}$	Consensus Time
0	∞	1	1	4
0	∞	5	1	20
0	∞	10	1	40
1	50	1	1	4
1	50	5	1	8
1	50	10	1	13

Table 1: Results of the Consensus Simulation Using Different $\diamond S$ Failure Detectors

The results obtained through these simulations show that the performance of the protocol is hugely influenced by the performance of the round coordinator. They also demonstrate that in some situations, considering a worse $\diamond S$ failure detector yields better performance for the consensus protocol. Further, they suggests that it is not easy to assess the impact of a failure detector on the performance of an application based solely on the QoS metrics proposed by [CTA00].

It may well be possible that there exists a consensus protocol based on a $\diamond S$ failure detector that does not possess the unwanted property discussed above. However, it is more likely that one such protocol would require a stronger failure detector. In the next section we propose the basis of the design of a failure detection service with a perfect semantics, which is the strongest semantics known for failure detectors.

3 A Hierarchical Failure Detection Service

In this section we propose a hierarchical failure detection service (FD service) for wide area networks (WANs). Initially, we will consider that the service runs over a non-partitionable WAN.

The proposed FD service has a perfect semantics (*i.e.* it implements a failure detector of class P) and is structured in two levels: the local level (within a local area network - LAN, or a segment of a LAN) and the global level (within the WAN that connects all local level segments). Each LAN segment is referred to as a local domain of detection, which is supervisioned by a local FD service. A set of local domains of detection connected via a WAN is named a federation of detection, which is supervisioned by a global FD service. Figure 1 illustrates this structuring.

In the local domain of detection, a perfect failure detector can be implemented by adding an extra communication channel for the network nodes in the domain. Such a channel will be used exclusively to convey traffic of the local FD service. This implies that the communi-

[3]Consensus time was measured as the time required for two processes to decide.

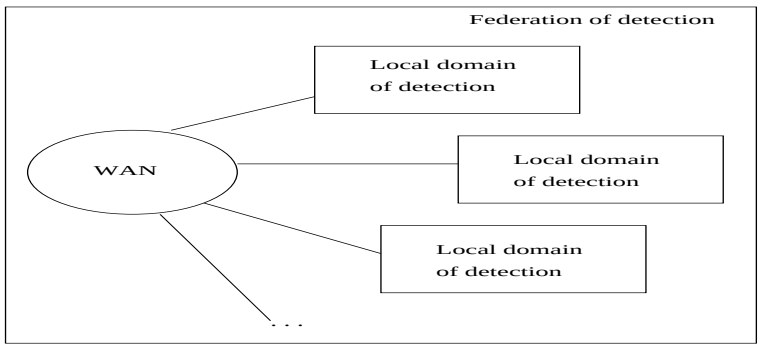

Figure 1: The Structuring of a Hierarchical Failure Detection Service

cation traffic generated through this channel is known and controlled [AV96], and therefore the maximum delay to transmit messages using this channel can be easily accounted.

To guarantee that the end-to-end maximal communication delay between any two functioning modules of the local FD service is bound, one needs to ensure that these modules are scheduled at the required time. One way to achieve that is using a real-time operating system [VCF00]. Another approach is to use a conventional operating system and implement the failure detection service within the system kernel (*e.g.* as a kernel thread), or as a user process that runs with maximal priority. In both approaches, by appropriately choosing the time interval between the transmission of two consecutive heartbeat messages, it is possible to guarantee that the maximal transmission delays in the redundant channel will never be violated [AV96, CB97]. This characterises a synchronous system within which implementing a perfect failure detector is a trivial task.

Our target system is composed of a number of Ethernet-based local area networks of PCs running Linux and connected by some routers. We have envisaged two strategies to implement the extra channel required at each local domain of detection. The first requires each machine in a local domain of detection to have an extra Ethernet card. These cards are connected to form a redundant communication channel (when recabling is not suitable, a wireless LAN can be used). Notice that the FD service does not impose any extra load in the asynchronous communication channel used by applications. The only cost it imposes is that related to the processing time required to execute each module of the FD service at the corresponding machine. The second approach further reduces this cost. It uses a failure detector device attached to each machine. This device has very simple processing and wireless communication capabilities. Whenever a machine is initialized the device starts detecting the other machines that are up (by listening to their heartbeats). It then chooses a slot of time for starting emitting heartbeats at some a priori agreed rate following a TDMA-based protocol. Finally, it starts monitoring the other machines. A watchdog mechanism implemented at kernel level (the only processing cost added) allows the failure detector device to detect the failure of its own host. Whenever the device detects the failure of its host it stops sending heartbeats, therefore allowing the other devices to also detect this failure.

The global FD service implemented within the federation is composed by a number of federation detection modules. Each local domain of detection in the federation executes one

of these modules. They exchange information about failures on the nodes belonging to their respective local domains in order to implement the global FD service. This information is provided by the underlying local FD service (see Figure 2).

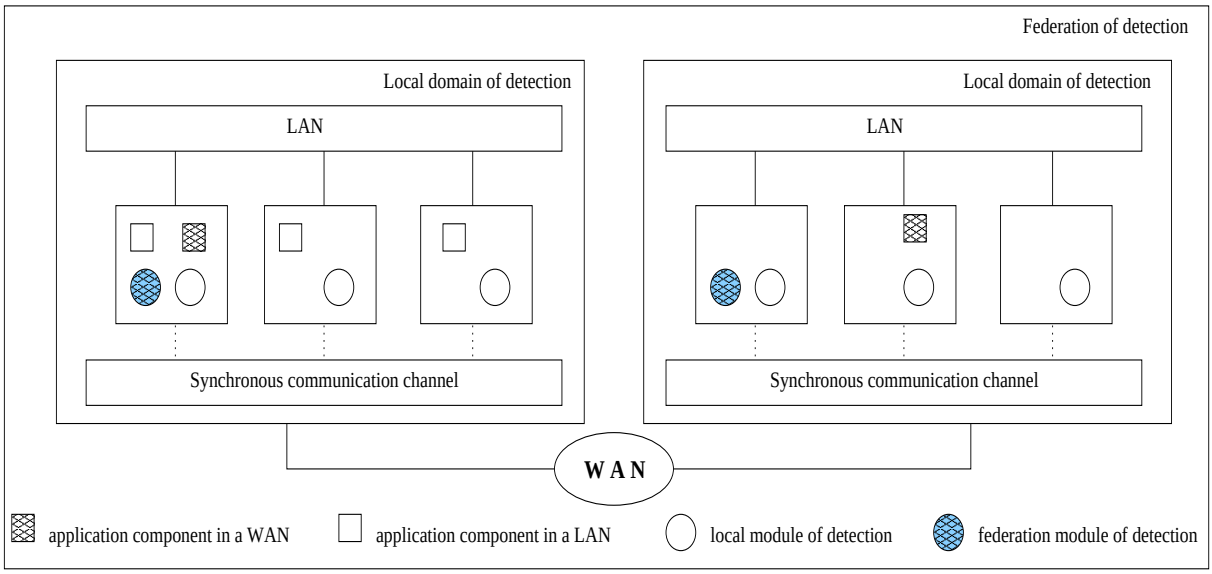

Figure 2: A Detailed View of the FD Service

It is important to notice that, although the transmission delay on the WAN is finite, (the WAN is non-partitionable) it is not bounded. Therefore, the federation detection modules executing on each local domain of detection must be fault-tolerant, otherwise it would be impossible to differentiate between a faulty module from one that is simply slow [FLP85]. The federation detection modules can be made fault-tolerant by having their implementation replicated within each local domain of detection. The required level of replication should be such that the probability of all replicated modules within a local domain fail is negligibly small.

Notice that when compared to traditional implementations of heartbeat based FD services, this two level hierarchy substantially reduces the number of messages required to be exchanged by failure detector modules, allowing for greater scalability. In fact, one can easily extend the two-level hierarchy with more levels to further increase scalability.

If it is not possible to assume a non-partitionable WAN, the global FD service cannot provide a perfect semantics. However, it can easily provide a weaker semantics, *e.g.* a $\diamond S$ failure detector [CT96]. In this case, applications spanning the federation would have to use protocols based on this weaker FD service. Nevertheless, those applications confined within a particular local domain of detection could still take advantage of a perfect FD service.

4 Concluding Remarks

We think that the result presented in [CHT96] has placed too much bias toward the implementation of $\diamond S$ failure detectors. Since these are the weaker failure detectors that allow

consensus to be solved, it is very much possible that they are the simplest and cheapest ones to implement, hence the interest they have raised recently. However, we are very confident that stronger failure detectors can indeed be built and at a very reasonable cost. Further, we believe that stronger failure detectors not only can reduce the assumptions that have to be made to guarantee the correctness of higher level protocols, but also yield substantially bigger performance gains if designers would be able to develop more efficient higher level protocols based on these stronger abstractions.

We are now modeling several consensus protocols based on a variety of failure detectors to gain insights on how to increase the performance of fault-tolerant applications. Our preliminary simulation analysis based on a Petri-net model supports our claims in favour of using stronger failure detectors. We are also implementing the FD service discussed in Section 3 to confront experimental results with the ones yield by our simulations.

References

[AV96] C. Almeida and P. Veríssimo. Timing failure detection and real-time group communication in quasi-synchronous systems. In *Proceedings of the 8th Euromicro Workshop on Real-Time System*, L'Aquila, Italy, Jun 1996.

[CB97] V. S. Catão and F. V. Brasileiro. Serviço de comunicação síncrona para nodos replicados. In *Anais do VII Simpósio de Computadores Tolerantes a Falhas*, Paraíba, Brazil, Jul 1997.

[CHT96] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, Jul 1996.

[CPN93] Design/cpn: User's manual. CPN group, 1993.

[CT96] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, Mar 1996.

[CTA00] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. In *DSN'2000*, Jun 2000.

[Fis83] M. J. Fischer. The consensus problem in unreliable distributed systems. Research Report 273, Yale University, Jun 1983.

[FLP85] M. J. Fischer, N. A. Lynch, and M. D. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2):374–382, Apr 1985.

[SDS99] N. Sergent, X. Défago, and A. Schiper. Failure detectors: Implementation issues and impact on consensus performance. Technical Report SSC/1999/019, École Polytechnique Fédérale de Lausanne, Switzerland, May 1999.

[VCF00] P. Veríssimo, A. Casimiro, and C. Fetzer. The timely computing base: Timely actions in the presence of uncertain timeliness. In *DSN'2000*, Jun 2000.