On the Evaluation of Failure Detectors Performance

Luiz Angelo Barchet Estefanel[1]
Ingrid Jansch-Pôrto
{angelo,ingrid}@inf.ufrgs.br

Programa de Pós-Graduação em Computação
Instituto de Informática - UFRGS
Porto Alegre - RS - Brazil
Caixa Postal: 15064 CEP 91501-970
Phone: +55 (51) 316-6159 FAX: +55 (51) 319-1576

Practical Experience Report

Abstract

Despite the great number of papers that broach the consensus agreement, just a few specifically discuss the structure or the implementation of failure detectors. Even these papers usually present only the structure of the detectors, and avoid presenting a comparison between the suggested detector and other existing ones. In fact, at the literature, we have found only one work that compares by simulations different failure detector models. We have chosen to use that paper as basis, and decided to make a comparison among some failure detectors models, using practical systems. In a practical environment, the implementation issues and the operational system can influence both the performance of the detectors and the consensus termination time. This paper presents our results, which allows us not only to judge the efficiency of these models, but also allow us to learn how these detectors work under different failure situations and under diverse system loads.
Key-words: Failure Detectors, Consensus Termination, Asynchronous Distributed Systems

1. Introduction

Since Chandra and Toueg [1] defined the properties of the failure detectors, many authors have suggested algorithms to solve consensus in a wide variety of environments and failure models. Despite this variety, there are few failure detectors described in the literature. Usually, consensus algorithms consider detectors as independent modules, which should work fine in almost all situations (the situations where the detectors makes false suspicions should be handled by the consensus).

However, this vision about the detectors is unreal, since their implementation model can interfere in the consensus performance through many factors. Felber [2] pointed out that the detectors with *Push* approach (also known as *heartbeat* or *I am alive* detectors) have distinct reactions than the *Pull* detectors (known as *interrogative* or *Are you alive?* detectors) if the application wants more accuracy or wider dissemination. Although his remarks are interesting to show that this difference exists, his suggestions of use for each failure detector model are based only on obvious tendencies. Actually several factors can influence the detection and the consensus: the communication model may overload the network if system parameters are not well tuned; and messages processing may consume system resources [3]. So, in a real system,

1 supported by CNPq

the combination of these factors may lead to false suspicions. As each detector presents a particular reaction to different failure scenarios, choosing the right detector and setting the best parameters for each situation is the only way to keep the consensus performance at a suitable level.

In fact, we know a single paper that had proposed a comparison among failure detectors, published by Sergent, Défago and Schiper [3]. In this work, they had used simulations to compare the behavior from three different failure detector models, evaluating their impact on the consensus termination. While this work is very useful to understand the influence of the communication model, many practical aspects were not included in the simulation specification, but some of them could represent potential points to reduce the performance of the consensus.

So, taking into account the limitations due to the use of simulation models, we tried to explore the question from another point of view, evaluating those detectors (and also some other model we found in the literature) through practical implementation, where the performance of the consensus and the detectors could be influenced by the system and network overhead.

In this paper, we present a fast revision on Sergent's report, in the section 2. After, in section 3, we discuss the practical environment, the used metrics and the test situations where the metrics were taken. In section 4, we present the considered failure detectors model, identifying their differences and similarities. In section 5, our results and a brief comparative analysis are shown. Finally, we conclude the paper in section 6.

2. A Brief Review on Sergent's Report

Usually, papers discussing about failure detectors, or more specifically, "asynchronous systems augmented with failure detectors", do not care about timing issues, and so, authors tend to consider consensus in asynchronous systems as inefficient or time spending [4]. In fact, this is a natural consequence of the failure detectors definition given by Chandra and Toueg, which allows a clear separation of "logic issues" (proof of safety and liveness of the algorithm) from the "engineering issue" (failure detectors implementation)[3].

Sergent's report has tried to show that optimal performance can be achieved, when tuning up the detectors. So, the choice of the better detector for a given situation must consider both implementation and algorithm characteristics.

To simulate the consensus and detectors operation, Sergent has represented the system as a private LAN, where identical workstations with only one process running on each machine communicate through datagram messages (like UDP/IP), in a point-to-point communication. The computation time was considered negligible compared to the time needed to transmit the messages, so the local computation time was set to 0. The transmission delays had been measured on a real network during a previous work.

Initially, four models of failure detectors have been considered: two traditional detectors, with *Push* and *Pull* characteristics, and two specialized ones, named *ad-hoc "no message"* and *ad-hoc "heart-beat"*[2], whose goal was to reduce the number of failure detection messages using the messages of the consensus to get the necessary detection information. For his simulations, Sergent considered that the *ad-hoc "heart-beat"* outperforms the *Push* detector, since it sends messages only when required, so the evaluation has been applied only to the remaining three detectors.

2 As Felber's nomenclature (*Push* and *Pull*) we also kept the names used by Sergent in our paper.

As simulation environment, Sergent considered two situations. In the first one, no process crashes, so the consensus must reach the decision in the minimal time (this situation was called ***failure free*** case). In the other situation, the coordinator of the round crashes just before sending its `propose` message, so this can be considered as the worst case that the algorithm can face in one round. The first case shows how the detectors may affect the consensus performance, while the second situation remarks the algorithm agility to detect the coordinator failure and start a new round. As these situations are antagonistic, and the best solution to the first case (longer time-outs and message intervals) becomes harmful to the worst case (longer time-outs lead to slower detection), tuning up the detector and the parameters demands a wide evaluation. Unfortunately, the analysis has been focused in the best performance points, so the behavior of each detector through all the tested parameters has not been completely shown.

Sergent analysis has identified *ad-hoc "no message"* as the detector with the minimal impact, presenting better detections than the other detectors. Closer to its results was the *ad-hoc "heart-beat"* detector, but the network overload caused by its messages reduced the final performance. The *Pull* detector was considered inefficient when compared to the *ad-hoc* ones, because its detection mechanism disputes resources (network and memory) with the consensus.

3. Assumptions for the Practical Environment

To evaluate implementations of the detectors in a real environment, we have judged that the tools used by Sergent were poor. We considered that the best environment for the experiments should face the same interactions as a normal distributed application. We used five machines running Linux (Pentium II 233Mhz, 64MB, kernel 2.2.16), connected through a 10-BaseTX Ethernet for the experiments. As there were not only these machines in the network, we have executed the experiments only at unsocial moments, like at night and weekends, to reduce possible interactions. The consensus and the failure detector algorithms were implemented in Java, using only UDP messages. The choice for the Java environment is due to our main interest in the behavior evaluation of the detectors, not in their particular performance. The results here presented comprise the average from at least 1000 consensus operations.

Also, as our measures probably could not be directly compared with those presented in Sergent's report without an expensive evaluation, we had chosen to compare the behavior of each detector through the tests. This decision makes the analysis more independent from the environment or the implementation techniques, and allowed us to include some other models of failure detectors that we found in the literature.

To evaluate the detectors we considered only crash failures, and we have used two hypothetical test situations and a "normal" one. The two hypothetical situations represent both the ***best case*** and the ***worst case*** that a consensus algorithm can face in one round. In the best-case situation, there is no suspicion, so it solves the consensus in only one round. This leads to the analysis of the overhead caused by the failure detectors, because only the system overhead influences the termination time in this situation. As the detectors can make suspicions even if no real failure has occurred (false suspicions), this case forces the consensus to ignore suspicions from the detector. Thus, we became able to measure the impact of the detectors processing without inserting additional rounds in the consensus operation.

The second situation, named worst case, allows us to measure the detection latency from the detectors. This case represents those situations where the coordinator crashes just after it had gathered messages from the majority of the processes (more specifically, just before sending its propose message). When the detector suspects that the coordinator has crashed, the processes start a new round, and solve the consensus. As this is not a restricted situation and the consensus could delay for many rounds due to false suspicions, it would be hard to compare the results. Thus, we restricted the detection to the second round, forcing the consensus to finish in exactly two rounds, as this situation is enough to show the detection latency of each model.

In the ***normal case*** situation, we have evaluated the detectors in a normal operation, where they are free to cast suspicion on anyone. With this situation, we can evaluate the moment when the overhead from the detectors begins to prejudice the detection, leading to false suspicions. For example, in Figure 1 we show the three corresponding situations for one of the *Push* detectors presented ahead.

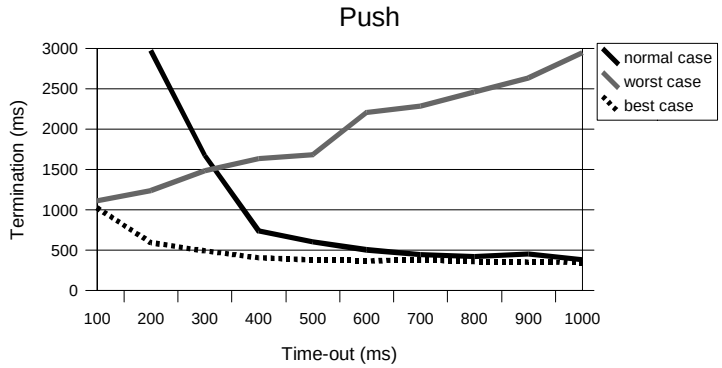

Figure 1 - A *Push* detector analysis

Here, we can see that the termination in the best case situation is influenced when the time-out is lesser than 400 ms, because the overhead increases. Meanwhile, the worst case situation shows that the longer the time-out is, the slower will be the detection of the coordinator's crash, increasing the consensus termination time. The analysis of the normal case shows that for time-outs longer than 400 ms the termination time will be close to that seen in the best case. However, when we use smaller time-outs, the termination time increases so far, that it also overtakes the worst case time. This means that the detector makes too many false suspicions, so the consensus takes many rounds before its termination.

4. Failure Detectors Models

We decided to add in our comparison more detectors that were found in the literature. Our main interest was to compare detectors with different mechanisms, communication models and suspicion procedures. In fact, we gathered samples from four kinds of detectors. In the group of the traditional detectors, we evaluated the *Push* and *Pull* detectors. The group of the specific detectors had the *ad-hoc "no message"* and *ad-hoc "heart-beat"*. We also considered the adaptive detectors (based in [1]), variations from *Push* and *Pull* detectors that allow their time-out to be dynamically set. Finally, we tried a gossip-styled detector, called *Heartbeat* [5], as its diffusion mechanism can be useful when the systems faces with excessive message traffic, because it exchanges messages only with a limited number of processes each time [6]. Currently, there are also other proposals on adaptive detectors as, for

example, the mechanisms proposed by Macedo [7] or Chen *at al*. [8]. Those models, however, consider the provision of additional information about the network and system status to make their computations. Although these detectors can achieve better results, we considered them excessively complex to the objectives from this work.

The detection in the traditional detectors is shown in Figures 2 and 3. They mainly differ on the time-out control and the tuning among the processes. For example, a *Pull* detector can ask for information with longer intervals, because each detector handles the time that the messages must take to be returned. A *Push* detector however must send its messages periodically, because the other processes control the time-out of the messages of a given process. Even if each one of these models has its advantages and disadvantages, they are the most popular detectors referred in the literature.

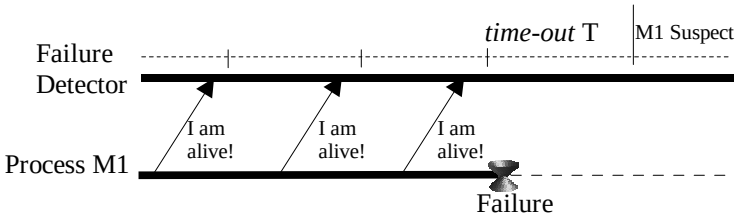

Figure 2 - Failure detection: the *Push* implementation

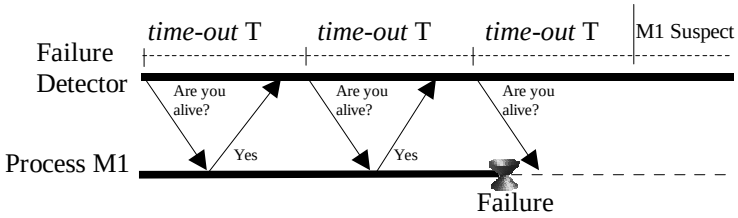

Figure 3 - Failure detection: the *Pull* implementation

The specialized detectors run over the consensus algorithm, using its basic structure and messages. Also, these detectors are activated only in specific moments, when the consensus needs the detection. The first detector, which is also the simplest one, is called ***ad-hoc "no message"***. It does not send detection messages; it just controls the time-out of a specific message from the application. In this case, the *ad-hoc "no message"* monitors the transmission and reception of the `estimate` and `propose` consensus messages. The set of specific messages that are monitored by the detector are called critical messages; this characterization makes them different from the other messages produced by the algorithm. The *ad-hoc "no message"*, however, can induce several mistakes if some process does not receive the critical anwer in the proper time. This happens because, in the consensus, a slow process may induce the coordinator to not send the `propose` message within the proper time, as the consensus has not yet received messages from a majority of processes (Figure 4). Due to this problem, Sergent also considered another variation on this detector, called ***ad-hoc "heart-beat"***. The *ad-hoc "heart-beat"* avoids these false suspicion situations by making the coordinator to send *I am alive* messages while no critical response is dispatched (Figure 5).

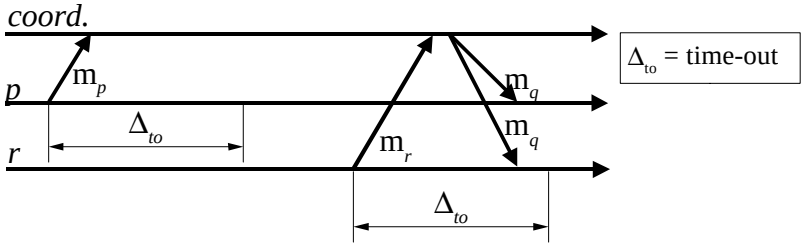

Figure 4 - Incorrect suspicions with the *ad-hoc "no message"* detector [3]

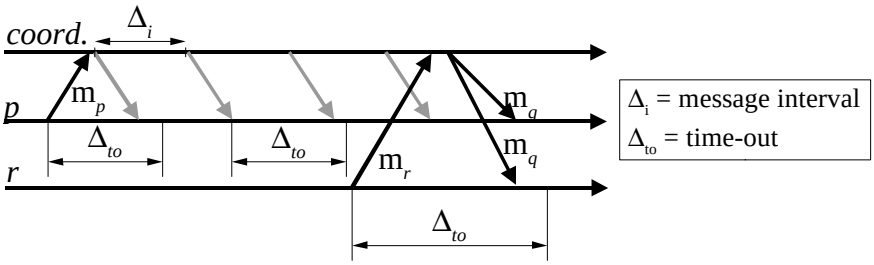

Figure 5 - Detection with the *ad-hoc "heart-beat"* [3]

In Table 1 we list the parameters and their characteristics - fixed or variable - as used in our experiments. The relations between message interval (Δ_i) and timeout (Δ_{to}) stand for the proportionality required by the detector to manage the network delay. As *Push* detectors need that its messages to arrive before the timeout, they sends the messages before the timeout limit (for example, $\Delta i = 98\% \Delta to$). The closer the relation is, the less will be the allowed delay, but the network overload will be small as there will be less messages being sent in a time period. On the other hand, this report is not necessary to a *Pull* detector, so it can send *are you alive?* messages in larger intervals ($\Delta i = 150\% \Delta to$, for example). The adaptive detector, however, does not have a fixed timeout, so we just gave it an initial value and an increment rate, instead of a parameters relation. The *Heartbeat* detector, as the *Push* model, needs to set its relation, but also needs to manage the number of neighbors to which it sends his messages in each communication step; thus, we have chosen the minimal set of neighbors to generate the minimum number of additional messages.

Table 1 - Failure detectors evaluated

Detector	Fixed parameter	Variable parameter
Push	$\Delta i = 98\% \Delta to$ and $\Delta i = 75\% \Delta to$	Δto
Pull	$\Delta i = 150\% \Delta to$ and $\Delta i = 100\% \Delta to$	Δto
ad-hoc "no message"	-	Δto
ad-hoc "heart-beat"	$\Delta i = 75\% \Delta to$	Δto
AdaptivePush	Δto starts as 100 ms, increment = 50 ms	Δi
AdaptivePull	Δto starts as 100 ms, increment = 50 ms	Δi
Heartbeat	$\Delta i = 75\% \Delta to$ and neighbors = 2	Δto

5. Results Assessment

For our analysis, we decided to separate the detectors in three groups. The time-out based group consists of those detectors that use the time-out as their fixed parameter. This group has most of our examples. In the second group we have evaluated the adaptive detectors, because they use Δ_i as their variable parameter. The last group has only the Heartbeat detector, because its results were much higher than the results from the other detectors, so we reasoned that a special analysis on its behavior would be interesting.

5.1. Time-out based detectors

The performance analysis in the best case (Figure 6) has shown that the influence of the failure detector mechanisms was, in general, similar among the detectors. Except for the *ad-*

hoc "heart-beat", the other detectors behave in a close manner. Just in the higher overhead situation (about 100 ms) their performance was reduced and had presented the most significant differences. We have also seen specially that the *ad-hoc "no message"* was better than the others, but these results were not really significant. Consequently, we have concluded that the presence (or absence) of the detection mechanism (sending detection messages) has not an expressive impact in the consensus performance as thought before.

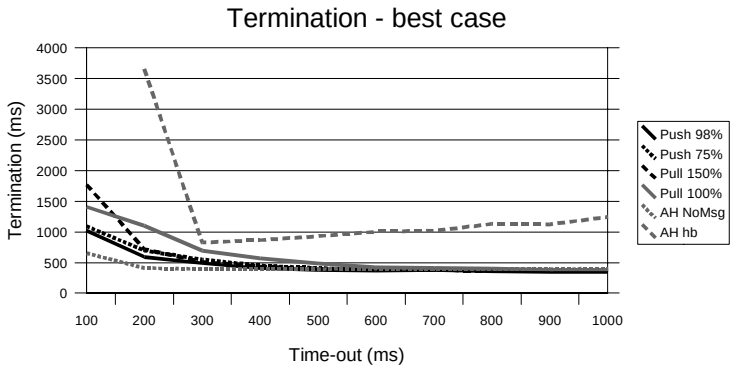

Figure 6 - Time-out based detectors: best case comparison

In the worst case (Figure 7), we have seen that these detectors, which have a permanent message passing ,can detect failures faster than the *ad-hoc* detectors. As the *ad-hoc* detectors have to start the detection procedure every time, they do not keep the knowledge that the other detectors gather all the time. So, with *ad-hoc* detectors, the consensus algorithm has a longer waiting time, which lasts until the detector suspects the coordinator or the coordinator answers the messages.

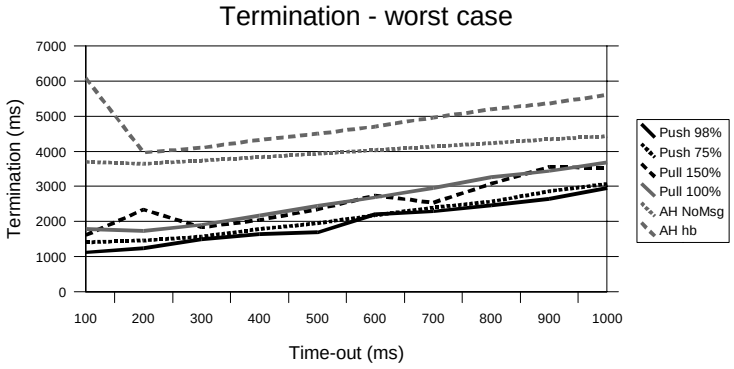

Figure 7 - Time-out based detectors: worst case comparison

Our great surprise has occurred in the normal case (Figure 8). First, some detectors have a special tendency to make false suspicions when the system overload increases. Besides this, the network *roundtrip* has a special impact on the detectors which depend on two-steps communication. This is the case of the *Pull* detectors, which cannot work with short time-outs, like 100 ms. Also the *ad-hoc* detectors are influenced, because despite of their communication scheme, they still depend on the *roundtrip* represented by the `estimate/propose` or `estimate/heartbeat` message pairs. The average *roundtrip* in our tests was about 133 ms; thus, when the time-outs became smaller than 200 ms, these detectors became unable to receive messages within the time-out, so they started to make false suspicions.

The bad performance shown by the *ad-hoc "heart-beat"* was important to curb the use of specialized detectors. Its performance was harmed by the activation cost from the threads

and objects involved in the detection process, as well as the dispute for communication channels with the consensus algorithm. Because of this behavior, even if the *ad-hoc "no message"* has worked well, the use of detectors specifically designed to achieve the lower cost is very risky, and without a well-conducted evaluation, they do not show significant differences from the traditional detectors.

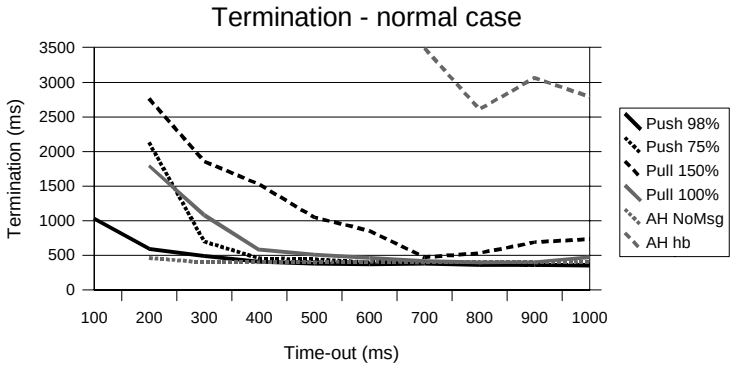

Figure 8 - Time-out based detectors: normal case comparison

Our conclusion is that, within this group, both *Push* detector with the relation $\Delta_i = 98\% \Delta_{to}$ and *ad-hoc "no message"* are the most efficient detectors, while the *ad-hoc "heart-beat"* is the worst detector concerning performance as the criterion. Both *Pull* detectors evaluated, together with the *Push* $\Delta_i = 75\% \Delta_{to}$, stand in a second level group, which have a good performance only when the parameter settings do not lead to generate excessive overload.

5.2. Adaptive detectors

Adaptive detectors have been analyzed as a different group because they use the message interval (Δ_i) as variable parameter, instead of time-outs (Δ_{to}), as the other detectors do. When we analyze the best case situation (Figure 9), our first perception is that these detectors are easily affected by the system overload. But, although their rates are pretty close to the termination time of the other detectors, this extra overhead can be explained by the adaptive processing itself, as the awareness and the reaction to an expired time-out leads to more processing time.

When dealing with the worst case (Figure 10), these detectors show their purpose. As the time-out is dynamically tuned up to allow distinguishing between the moments when a message is just late and when a process has crashed, the detectors keep the termination time constant, without regard to the defined message interval.

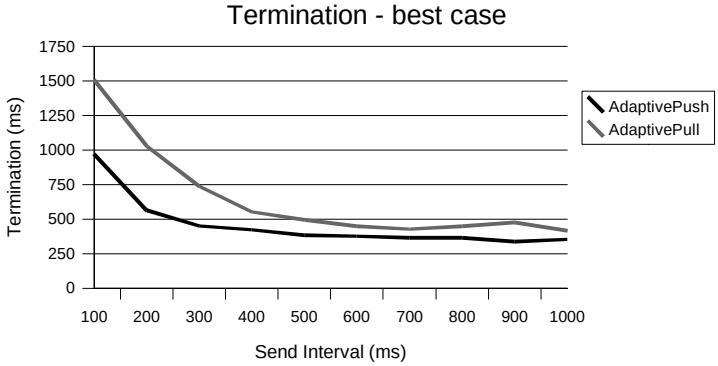

Figure 9 - Adaptive detectors: best case comparison

This regular behavior also can be seen with the Adaptive Push detector in the normal case (Figure 11). However, the behavior presented by the Adaptive Pull was quite unexpected, as it was very variant through the test. We made some observations and reasoned about the possible factors that could explain the behavior we found. First, there are stable periods, between 200 ms and 400 ms and between 500 ms and 900 ms. The relative difference between them come from the increasing number of false suspicions; the detector tries to adapt, but the more suspicions we have, the bigger will be the termination time. Our second observation is related to the growth of the termination time, when the message interval is greater than 900 ms. These results correspond to the average from many consensus. Consequently we presume that this growth reflects the excessive number of aborted rounds that the first consensus has had before the time-out was adapted to its parameters.

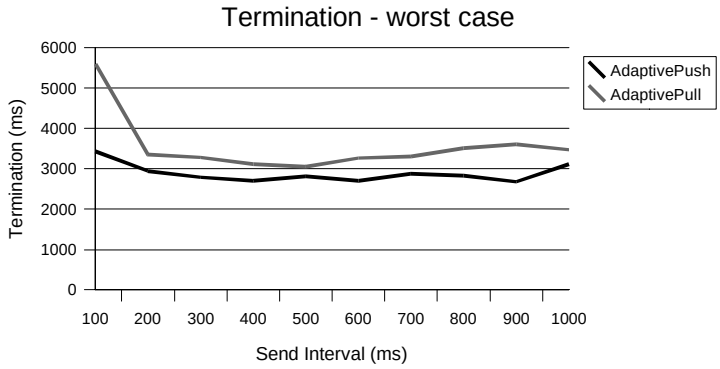

Figure 10 - Adaptive detectors: worst case comparison

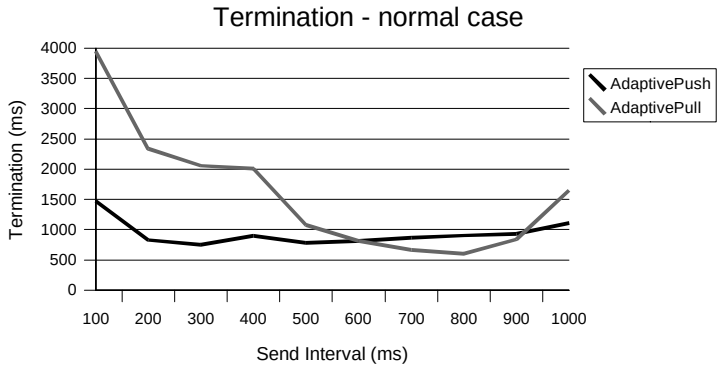

Figure 11 - Adaptive detectors: normal case comparison

Because of this huge variation in the behavior, we selected the Adaptive Push detector as the best (and stable) adaptive detector. In fact, this is a good detector if the application needs a stable service through different situations.

5.3. Heartbeat detector

We have included the Heartbeat detector in this work to explore a different kind of mechanism, generally referred as *gossip*. We had previously studied this detector [9,10], and we were anxious to see if its structural innovations would reduce the impact of the detection in the consensus operation.

However, this detector has proved to be inefficient, in comparison with the others. The first problem is related to the total number of exchanged messages. Even if the gossip

mechanism reduces the amount of the originally sent messages, the detector also uses a "retransmission" procedure that makes the total number of messages be greater than in the other detectors. Figure 12 shows that even with only one neighbor (that cannot be used in normal operation as a single failure may lead to a logic partition) the total number of exchanged messages is equal to those sent by the *Push* detector to its four neighbors. This happens because the number of redirected messages increases too much (each possible path will generate another message) [10].

As there are many messages carrying information from one process, it takes too much time to detect a crash, because every message with the old information must stop running. Adding this fact to the non-deterministic behavior from the gossip diffusion scheme, we got a terrible performance in the worst case, as shown in Figure 13.

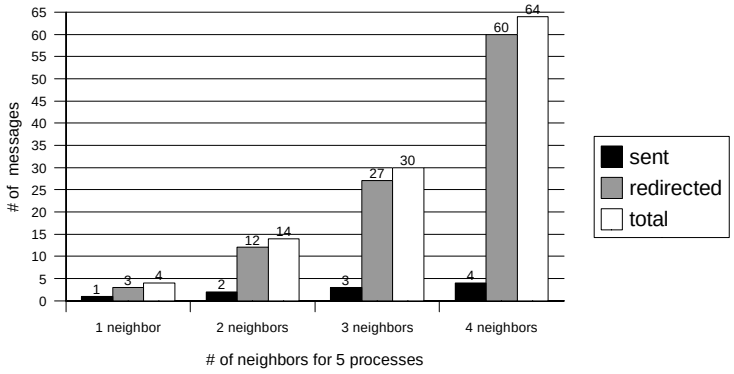

Figure 12 - Number of messages exchanged by a single process in one communication step (based on [10])

On the other hand, the "redundant information" that makes so higher the cost of a crash detection also helps the detector to avoid false suspicions. As we may see in Figure 14, the performance in the normal case is relatively close to the performance from the best case. We believe that the termination time is not lower than the presented because this detector must deal with the high cost of processing all the information carried by the messages. In fact, we believe that the significant increase to the system overhead has been caused by this excessive processing.

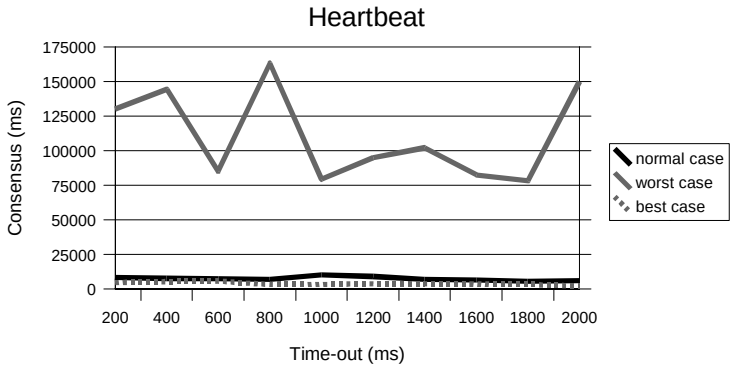

Figure 13 - Heartbeat performance

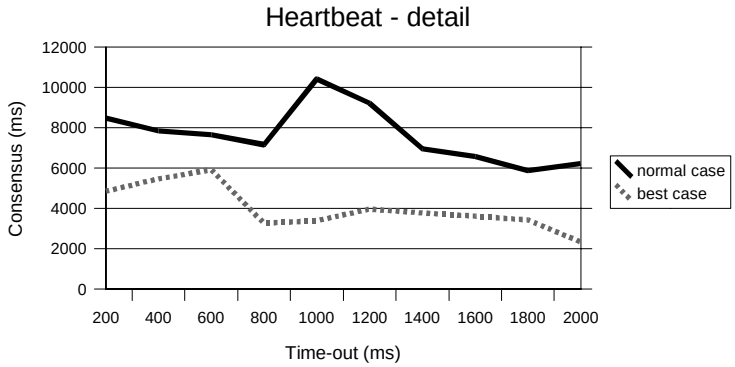

Figure 14 - Detail in the Heartbeat performance

Despite this experience, we think that correcting some of these problems, specially the information exchange model, this detector could be able to achieve good results.

6. Conclusion

This paper has presented a comparison among the most representative models of failure detectors. The main goal was to learn more about the behavior of the detectors through different scenarios, and specially to identify adequate efficiency of the detectors to the consensus agreement.

Since we do not know any other comparison except Sergent's simulations, we think that this paper may contribute to the implementation and the suggestion of new failure detectors. Also, the data acquired in this work was very helpful to compare practical issues with simulation ones. As we saw, some results we got are different from simulation ones, because even when communication model is efficient and economic, the processing cost can overlap those benefits. Other detectors, however, had proved their efficiency, in both cases. This trade-off between communication efficiency and processing cost should always be in mind when designing a failure detector, together with the logical issues like the proof of safety and liveness of the algorithm.

More than simply define the most efficient detector, this paper allows us to analyze the behavior and tendencies from the detectors, contributing to the choice of the better detector that a specific application would need. This would be very important to our future works, because these specifications will conduct the construction of other components for group communication.

To conclude, we think that this work strengthened the qualities from those detectors which are modular and permanently-working. We saw that they behave as efficiently (or in other words, they make so lower impact on consensus termination) as the specific detectors, but are less intrusive in the application algorithm and can be reused with few or without modifications.

References

[1] T.D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, March 1996.

[2] P. Felber. The CORBA Object Group Service. *PhD. Thesis.* EPFL, Lausanne, Switzerland, 1998.

[3] N. Sergent, X. Défago and A. Schiper. Failure Detectors: implementation issues and impact on consensus performance. *Technical Report*, EPFL, Lausanne, Switzerland, 1999.

[4] R. Guerraoui and A. Schiper. Consensus: The Big Misunderstanding. *Proceedings of the IEEE International Workshop on Future Trends in Distributed Computing Systems (FTDCS'97)*, October 1997.

[5] M. K. Aguilera, W. Chen and S. Toueg. Heartbeat: a timeout-free failure detector for quiescent reliable communication. *Proceedings of the 11th International Workshop on Distributed Algorithms*. May 1997.

[6] K. H. Guo. Scalable Message Stability Detection Protocols, *PhD. Thesis*, Cornell University, May 1998.

[7] R. Macêdo. Failure Detection in Asynchronous Systems. *Anais do II Workshop de Testes e Tolerância a Falhas*. Curitiba, Brasil. Julho 2000.

[8] W. Chen, S. Toueg and M. K. Aguilera. On the Quality of Service of Failure Detectors. *Proceedings of the IEEE International Conference on Dependable Systems and Networks*. New York. June 2000.

[9] L. A. B. Estefanel. Detectores de Defeitos Não Confiáveis. *Publicação Interna - UFRGS*, 90 pg., UFRGS, Porto Alegre, Brasil. Janeiro 2000. Available at <*http://www.inf.ufrgs.br/~angelo/ publications/ TIDetectores.zip*> (in Portuguese)

[10] L.A.B. Estefanel and I. Jansch-Pôrto. Avaliação Prática de um Detector de Defeitos: teoria *versus* implementação. *Anais do II Workshop de Testes e Tolerância a Falhas*. Curitiba, Brasil. Julho 2000.