On the Evaluation of *heartbeat*-like Detectors

Luiz Angelo B. Estefanel — *angelo@inf.ufrgs.br*
Ingrid Jansch-Pôrto — *ingrid@inf.ufrgs.br*

Graduate Computer Science Program
Instituto de Informática – UFRGS – Caixa Postal 15064
91501-970 – Porto Alegre – RS - Brazil

***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. Some authors have commented on the most traditional detectors, like the *heartbeat* and *interrogation* models, with basis in their general characteristics. Although these results could change considering different application scenarios, their differences have not been well evaluated. 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 the evaluation results concerning a specific group of detectors, the *heartbeat*-like, under several condition, allowing us to identify the behavior of these detectors and their real cost to the system.**

***Index Terms* - Failure detectors, consensus, distributed agreement, heartbeat, practical evaluation**

I. 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 every situation (the situations where the detectors fail 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 *heartbeat* detectors have a distinct behavior from the *interrogative* detectors. 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]. In result, the combination of these factors may lead to false suspicions in a real system. 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.

The evaluation of *heartbeat*-like detectors is notably important, as they are probably the most popular detectors used with consensus operations. Moreover, these detectors approach (gather knowledge using *ping* messages) are very common in many other applications and systems, increasing the importance of such comparison.

In this paper, we present a fast revision on the basic principles of these detectors, in section II. In section III, we discuss some variations on the basic model, found in the literature. In section IV, we discuss the practical environment, the test situations where we have taken the metrics and some implementation issues. In section V, we present our results and a brief comparative analysis. Finally, some conclusions finish the paper.

II. Basic Structure for a *Heartbeat* Detector

Despite the importance of the failure detectors and their definition presented by Chandra and Toueg [1], the basic principles of heartbeat-like detectors are simple, and can be found in many other systems and applications. The main idea is that one process (the failure detector) needs to know the status of other processes. Thus, they send signals to that detector, meaning that they are alive. In the *heartbeat* model, monitored processes need to notify the detector using some kind of "*I am alive!*" message. If the message does not arrive within the maximum time allowed (the message ***time-out***), the detector should start its suspicion procedures (Fig. 1 shows an example of this detection).

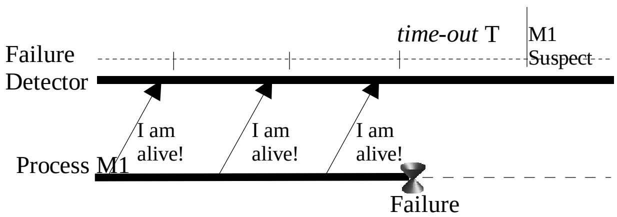

Fig. 1. The *heartbeat* communication model

This notification method is relatively efficient, because the processes can use one-way messages that have a low cost to the system. In addition, the implementation might use some multicast service provided by the network to optimize the communication process, if there are many detectors to notify [2]. However, this mechanism requires that the monitored processes are active, sending these messages periodically. Although it is possible to easily implement this state with some concurrent threads, it forces all the monitored processes to be aware of the presence of the detectors, as they need to specify the identity of their receiver detector to send the "*I am alive!*" signals.

Thus, an important point on the use of this class of detectors is that both processes and detectors should be tuned up with the same parameters. As the detectors control the time-out of the messages sent by the processes, there is a trade-off between messages interval and time-out. If message period is longer than the time-out allows, the processes will probably be suspect. In fact, the detectors should usually send their messages in slightly shorter intervals than the maximum allowed, dealing with network and processing delays.

While sending messages in shorter intervals allows the processes to deliver their messages in the proper time, the network traffic increases, as well as the system load. So, setting these detectors to balance the detection latency and the system overload have a direct impact on the occurrence of false suspicions, and relates directly to the *completeness* and *accuracy* properties.

III. *Heartbeat* Variations

While traditional *heartbeat* detectors are widely known and used, they present some disadvantages. As the detectors communicate all the time, they overload the network and the system. Moreover, these detectors are strictly tied to shared parameters (because one process needs to deliver messages within the time-out controlled by the other processes), and are unable to adapt themselves to the relative speed from different processes and links. Possible variations on the *heartbeat* model were found in the literature [4,6], and intend to reduce some of those problems. We will study them in the following.

A. Adaptive Detectors

Actually, adaptive detectors do not represent a different model of detector. These detectors have a special suspicion mechanism that tries to increase the accuracy of the detection. This is desirable because with the traditional *heartbeat* detector there is only one fixed time-out, and often the network delay or the processing speed of other detectors leads to erroneous suspicions. Since slow processes are more likely to send their messages too late, the other processes frequently suspect them. A simple solution would be to increase this global time-out until all correct processes are able to send their messages in time, but this solution only increases the detection latency. Even if we give a different time-out for each process, setting fixed time-outs is a hard task, because it depends on many environmental aspects. In fact, the adaptive detectors try to set these time-outs dynamically, adapting to the delay of each network or process (Fig. 2 shows how this mechanism works).

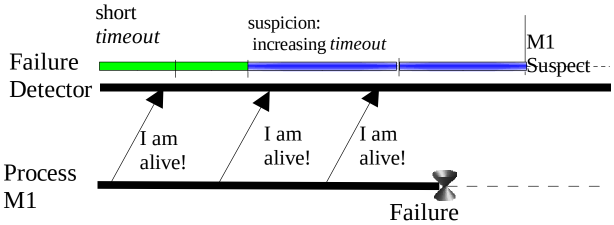

Fig. 2. Failure detection: the adaptive implementation

Chandra and Toueg [1] have introduced the main idea of adaptive detectors while discussing detection in partially synchronous environments. Adaptive detectors try to incrementally achieve the proper time-out for each correct detector. Obviously, when the time-out value is not enough, processes unable to deliver their messages in time become suspect. Moreover, when correct processes start to deliver their messages in time, it increases the probability that the silent detectors really crashed.

Currently, there is also another proposal or 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.

B. Specialized Detector

Sergent *et al.* [4] have presented the *ad-hoc "heart-beat"* detector in the paper where they have compared different models of failure detectors. This is a specialized implementation, which runs inside the consensus algorithm, using its basic structure and messages. Moreover, this kind of detector is only activated in specific moments, when the consensus demands the detection.

This detector intends to work between the transmission and reception of the `estimate` and `propose` messages from the consensus algorithm. The *ad-hoc "heart-beat"* implementation sends "*I am alive"* messages while the `propose` message has not been sent (Fig. 3).

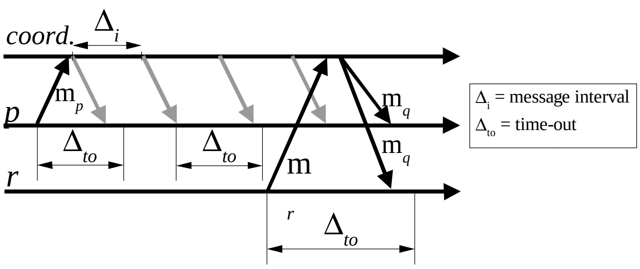

Fig. 3. Detection with the *ad-hoc "heart-beat"* [4]

While this detector generates fewer messages (only the coordinator of each round sends "*I am alive*" messages to the others), the "fixed time-out" problem appears again. As each process defines its time-out, it shall be long enough to deal with the delays from all possible coordinators. Moreover, this time-out has to consider both send and receive time, because the first *heartbeat* message only arrives after each process had sent its `propose` message.

IV. Evaluation on the Practical Environment

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.

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.

The main metric used to compare the detectors was the consensus termination time, measured between the consensus invocation and the return to the application. However, just one metric may not be able to show all aspects from the operation [5], so we also considered the CPU time used by the processes and the memory usage. The CPU time allows us to filter "stand by" situations, which can happen, for example when one process has to wait for a specific message to progress. The memory use helps us to identify when the system and network buffers become overloaded by the exchanged messages.

Both traditional and adaptive detectors have the same structure design, as we considered the adaptive detector only as a slightly modified version from the traditional detector. Nevertheless, the design of the specialized detector did not follow the same pattern. As this detector should work within the consensus algorithm, we implemented it using all available mechanisms from the consensus. In this way, the detector sends *"I am alive"* messages through the consensus communication facilities; the message reception uses the same communication channels than the consensus does; and shared variables speed the interaction between the consensus and the detector. Actually, the only addition to the consensus was the suspicion thread, which handles the timeout and the processing from the *"I am alive"* message.

In Table I, 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 *heartbeat*-like detectors need 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. 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.

Table I -Test Parameters

Detector	**Fixed parameter**	**Variable parameter**
heartbeat	$\Delta_i = 98\% \ \Delta_{to}$ and $\Delta_i = 75\% \ \Delta_{to}$	Δ_{to}
adaptive	Δ_{to} starts as 100 ms, increment = 50 ms	Δ_i
ad-hoc "heart-beat"	$\Delta_i = 75\% \ \Delta_{to}$	Δ_{to}

V. Results Assessment

For our analysis, we had to separate the detectors in two groups, because the adaptive detector uses messages interval (Δ_i) as a variable parameter, while the other detectors use the timeout (Δ_{to}). However, they can be compared in a tough way, which is enough for our purposes.

A. Best case situation

The performance analysis of the termination time in the best case (Fig. 4) has shown that the influence from the failure detector mechanisms was generally similar in all implementations, except for the *ad-hoc* "*heart-beat"*. The performance of the detectors presented the most significant reductions only under shorter timeouts (or message intervals, depending on the detector model). Although already expected, this behavior does not explain the results from the *ad-hoc "heart-beat"*. Therefore, we still have to look at the CPUTime and memory usage graphics (Fig. 5 and Fig. 6).

The CPUTime metric shows that the *ad-hoc "heart-beat"* detector does not use so much CPU as suggested the former graphic. Moreover, Fig. 6 shows that the memory usage in this detector is even lower than in the others, so the problem is not related to excessive message traffic. In fact, we believe that the performance problem presented by the *ad-hoc "heart-beat"* detector is related with three factors: its detection mechanism, the communication channels and the occurrence of false suspicions. As the network *roundtrip* limits the *ad-hoc* detector, the first *"I am alive"* message will not arrive before this time. In addition, this detector appeared to be very liable to message delays, since it was not possible to test the relation $\Delta_i = 98\% \Delta_{to}$ due to the absence of enough messages arriving in time. The main reason for these delays probably is the communication scheme, which shares the same channels with the messages from consensus. Because of these factors (*roundtrip* and delays), it increases the probability to make false suspicions, and the consensus is delayed through many rounds before its termination.

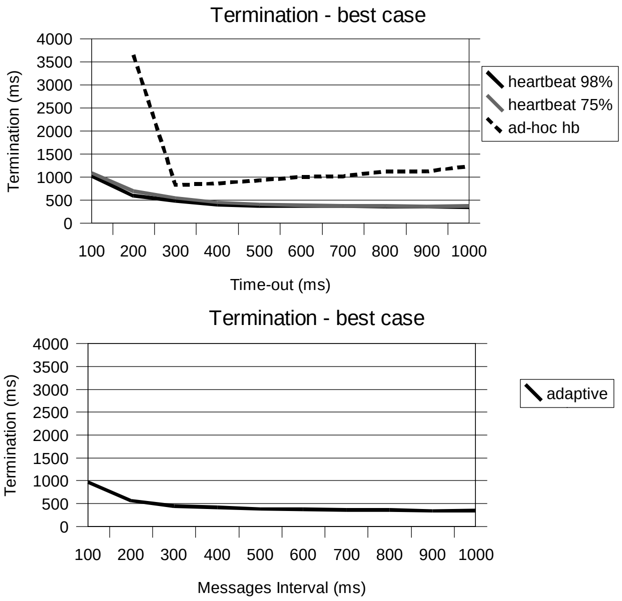

Fig. 4. Consensus Termination: best case comparison

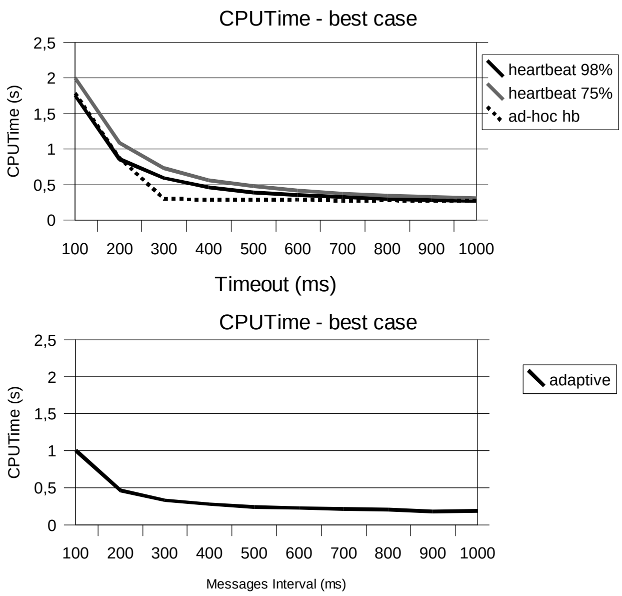

Fig. 5. CPUTime: best case comparison

Another interesting fact refers to the adaptive detector, when analyzed in this test situation. Our first perception is that although its performance rates are pretty close to the termination time of the other detectors, it uses more memory. The adaptive processing itself may explain this extra memory usage, because the detector demands more control structures to be aware and react when time-outs expire.

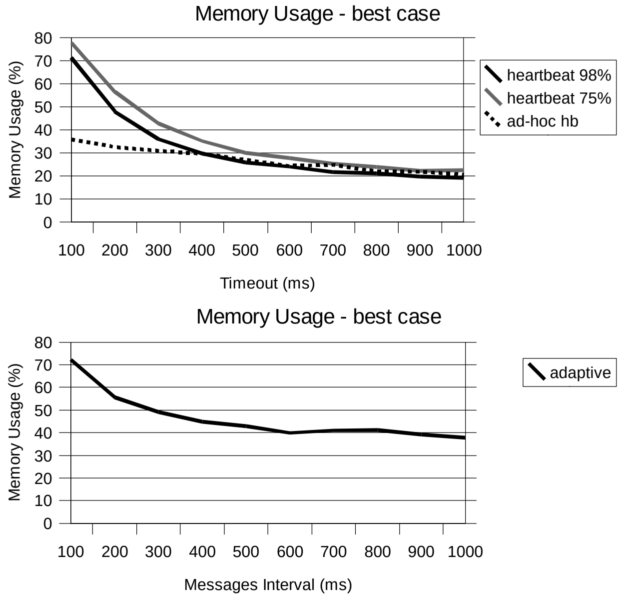

Fig. 6. Memory Usage: best case comparison

B. Worst case situation

The analysis of the termination time in the worst case situation (Fig. 7), has shown that detectors that have permanent message exchange are able to detect failures faster than the *ad-hoc* detector. As the *ad-hoc* detector have to start its detection procedure every time, it does not keep the knowledge that other detectors gather through their operation, and in addition, it is limited by the network *roundtrip*. Consequently, the use of an *ad-hoc* detector brings longer waits to the consensus algorithm. The wait lasts until the detector suspects the coordinator or the coordinator answers the messages. That is why the bad rates from the *ad-hoc "heart-beat"* do not relate to excessive processing or message traffic, as Fig. 8 and Fig. 9 show.

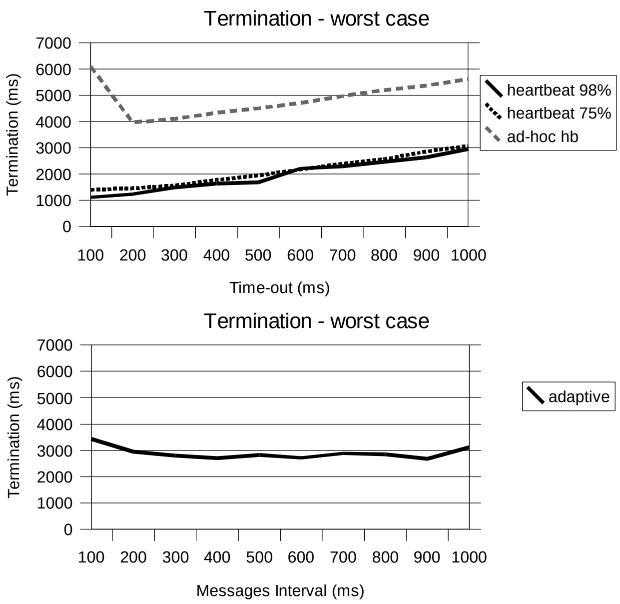

Fig. 7. Consensus Termination: worst case comparison

The adaptive detector has presented a very stable behavior, coherent with its main purpose. As timeouts are dynamically tuned up to allow the detector to distinguish crashed or slow processes, the adaptive detector was able to keep constant the termination time, without regard to the messages interval. However, this stability demands more memory and processing time (as shown in Fig. 8 and Fig. 9), so this detector probably will be more liable to system overload.

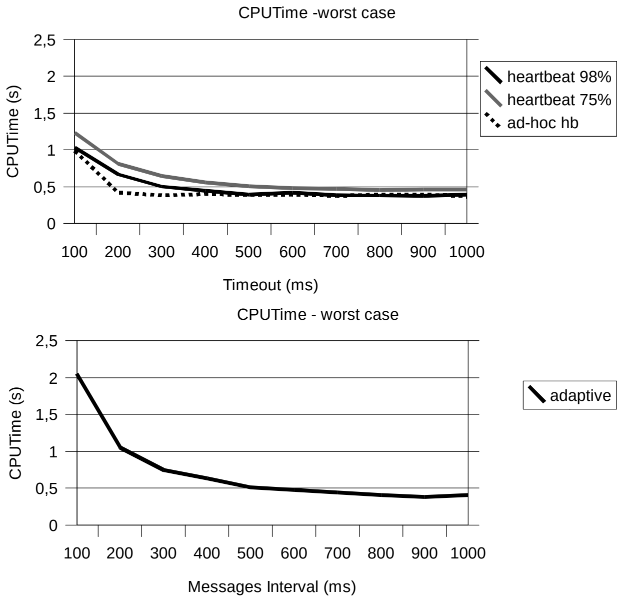

Fig. 8. CPUTime: worst case comparison

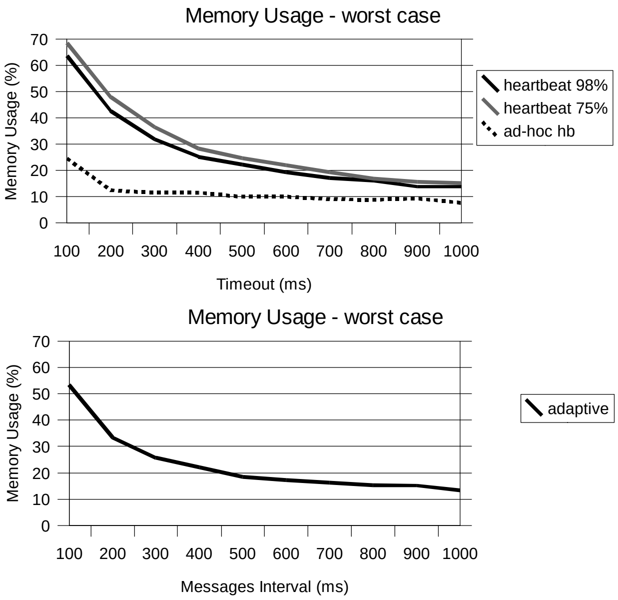

Fig. 9. Memory Usage: worst case comparison

C. Normal case situation

While the hypothetical situations (best case and worst case) are useful to show the impact of the detectors in the consensus operation, the normal case situation is the most important. That is because this situation points which detector has more tendency to make false suspicions, when the system overload increases. As told before, the network *roundtrip* had a special impact on the detectors that depend on two-step communication. Because the average *roundtrip* in our tests was about 133 ms, when timeouts became lesser than 200 ms, the *ad-hoc* detector became unable to receive messages within the timeout, making more false suspicions than the other detectors.

Fig. 10 presents the termination time of the detectors. It shows that the *ad-hoc "heart-beat"* starts to make false suspicions much earlier than the other detectors. The impact of these false suspicions in the *ad-hoc* detector appears well in Fig. 11, which presents the three testing situations for this specific detector.

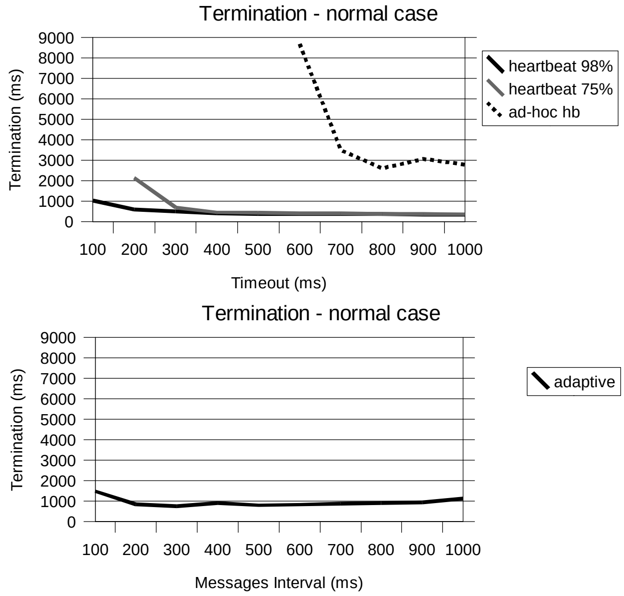

Fig. 10. Consensus Termination: normal case comparison

Especially in the termination graphics, the curve from the normal case situation quickly surpasses the curve from the worst-case situation. As the worst-case situation solves the consensus in exactly two rounds (the application forces this limitation to avoid multiple rounds), the *ad-hoc "heart-beat"* detector demands several rounds before its termination, in the normal case. We believe, however, that using its own communication channels would improve the performance from the *ad-hoc "heart-beat"* detector, despite the *roundtrip* limitation.

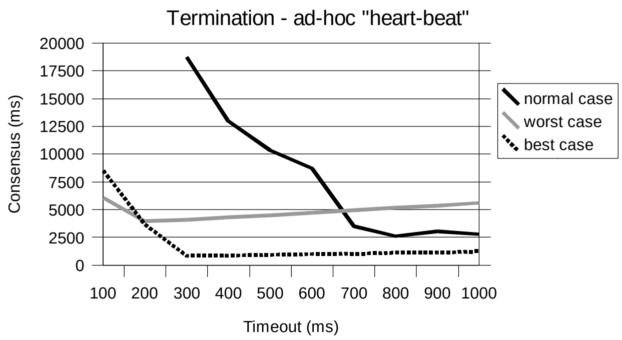

Fig. 11. The behavior from the *ad-hoc "heart-beat"*

Fig. 12 also presents this behavior. In addition, Fig. 13 shows the same memory usage tendencies already observed in the worst case situation.

Despite the *ad-hoc* detector, there are still two important remarks about these results. The first one refers to the adaptive detector. This detector presented termination results a little worse than the heartbeat detectors, and the other metrics also confirmed this evaluation. However, this detector worked in a constant pace through all test situations, presenting a desirable behavior for reliable systems.

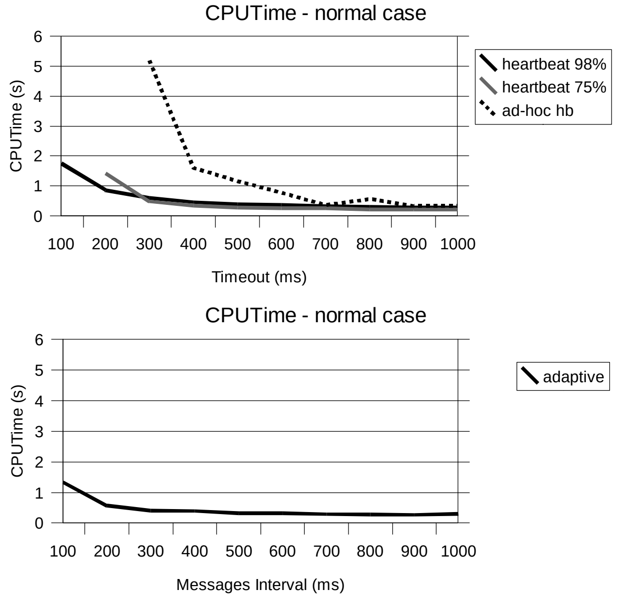

Fig. 12. CPUTime: normal case comparison

The second remark refers to the *heartbeat* detector with the relation $\Delta_i = 75\%\ \Delta_{to}$. In the hypothetical situations, there was very little difference between this detector and the *heartbeat* with the relation $\Delta_i = 98\%\ \Delta_{to}$. However, in the normal case situation, this detector presented higher levels of memory consumption. Even if this difference had not influenced so much the termination time, it can represent a possible weakness (perhaps, the better word would be liability) when the system overhead increases.

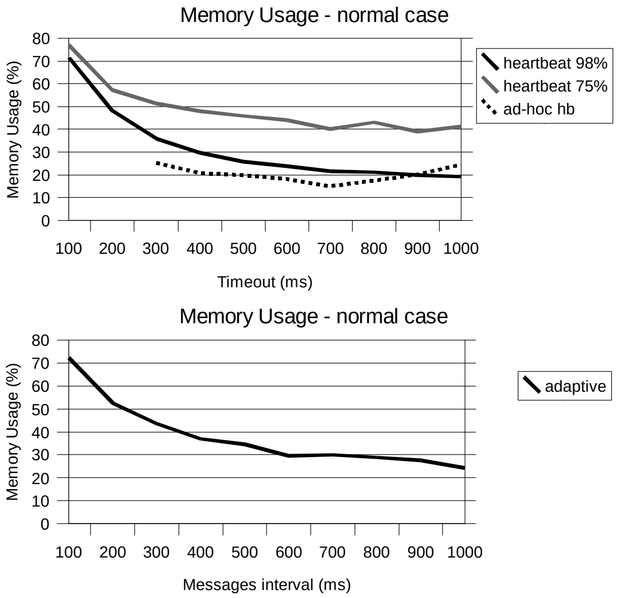

Fig. 13. Memory Usage: normal case comparison

VI. Conclusion

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

Since we do not know of any other comparison except Sergent's simulations (even though Chen *et al.* had done a good evaluation on the detection parameters), we think that this paper may contribute to the implementation issues and the suggestion of new failure detectors. While the "traditional" *heartbeat* detector presented the best performance, the adaptive detector behaved extremely stable in all situations (a desirable quality for reliable systems). Concerning the *ad-hoc heartbeat* detector, whose results were not so good, has presented the lower cost to the system (memory and processing); we still believe that the use of some usual techniques can improve it.

More than simply defining the most efficient detector, this paper allowed us to analyze the behavior and tendencies of 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 can conduct the construction of other components for group communication.

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] 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.

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

[5] R. Jain. The Art of Computer Systems Performance Analysis, New York: John Wiley, 1991

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

[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.