A new unreliable failure detector for self-healing in ubiquitous environments

Anubis Graciela de Moraes Rossetto*, Carlos O. Rolim†, Valderi Leithardt†,
Guilherme A. Borges†, Cláudio F. R. Geyer†, Luciana Arantes‡ and Pierre Sens‡
*Federal Institute Sul-rio-grandense (IFSUL), Passo Fundo, Brazil,
Email: anubis.rossetto@passofundo.ifsul.edu.br
†Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
Email: {carlos.oberdan, vrqleithardt, gaborges, geyer}@inf.ufrgs.br
‡Sorbonne Universités, UPMC Univ. Paris 06, CNRS, Inria, LIP6
Email: {luciana.arantes, pierre.sens}@lip6.fr

***Abstract*—Due to the nature of ubiquitous systems, nodes (e.g., sensors) are frequently prone to failures. Such systems must, therefore, present self-healing capabilities in order to detect failures and make the necessary adjustments to prevent their impact on applications. In such a context, this work proposes a new and flexible unreliable failure detector, denoted the Impact failure detector (FD), for self-healing system in ubiquitous environments. The output of the Impact FD concerns the confidence in the system as a whole. By expressing the relevance of each node by an impact factor value as well as a margin of acceptable failures of the system, the Impact FD enables the user to tune the failure detection configuration in accordance with the requirements of the application: in some scenarios, the failure of low impact or redundant nodes does not jeopardize the confidence in the system, while the crash of a high impact factor one may seriously affect it. A softer or stricter monitoring is thus possible. Performance evaluation results using real PlanetLab [1] traces confirm the degree of flexible applicability of our failure detector and, due to the margin of failure, the number of false responses may be reduced when compared to traditional unreliable failure detectors.**

I. Introduction

Ubiquitous computing has its origins in the visionary work of Marc Weiser who, at the beginning of the 1990s, predicted the existence of environments saturated with computing devices and communication capabilities, highly integrated with human users [2]. However, faults in ubiquitous systems occur frequently due to the exposure to the physical environment [3]. These systems must, thus, present autonomic computing capabilities which render them more autonomous, to some extent, in the presence of failures so as not to depend on human intervention for preventing undesirable consequences [4] induced by the failures.

One of the properties of autonomic computing is self-healing. A system designed with this feature automatically discovers, diagnoses, and reacts to disruptions [5]. The system must be able to detect faults and make the necessary adjustments to prevent the failures from having an undesirable impact on the application which should keep active and available. Hence, failure detection service which delivers monitory information about the liveness of the system nodes is a crucial feature of self-healing systems. In order to meet such a requirement, this paper proposes a new unreliable failure detector (FD), denoted **Impact Failure Detector**.

In our approach, we consider that the Impact FD is one of the components of a Self-healing Module [6]. The latter aims to cater for the needs of ubiquitous systems which must manage applications that are distributed and adaptive to the changing requirements of the environment. Hence, a Self-healing Module should be available anywhere and at any time and, whenever a failure is detected, it has to make the necessary adjustments to avoid error propagation which can result in major damage to the application. Notice that the self-healing module can be incorporated in any ubiquitous system, regardless of its architecture. Basically, it has two components: the Failure Detector (FD) and Adaptation Manager (AM), as shown in Figure 1. The former consists of an adaptive unreliable failure detector that is responsible for detecting crash failures of different entities (nodes, sensor, etc.) that need to be monitored in the system. The AM provides suitable adaptation strategies, aiming at reducing the impact of the detected failure on the application. We emphasize that the focus of this paper is not the Adaptation Manager but to propose a suitable failure detector for self-healing service.

Contrarily to traditional unreliable failure detectors [7] [8] that output the set of nodes suspected of having failed, the Impact FD outputs a *trust level* and a *status* (trusted or not trusted) concerning a given system *S*. The output can be considered as the degree of confidence in the system. To this end, an *impact factor*, defined by the user, is assigned to each node and the *trust level* is equal to the sum of the impact factor of trusted nodes, i.e., those not suspected of failure. Furthermore, an input *threshold* parameter defines a trust level limit value, over which the confidence degree on *S* is not affected. Hence, by comparing the trust level with the *threshold*, the system is considered trusted or not. If it is not the case, the AM decides if some measures must be taken (urgently or not, with regard to the trust level output). In other words, when the FD informs, by *status* output, that *S* is not trusted, the Adaptation Manager analyzes the *trust level* output and decides about the recovery strategy and/or system reconfiguration.

In [9], the authors propose the Accrual ϕ failure detector which, similarly to our approach, outputs a suspicion level on a continuous scale. However, the ϕ FD does not analyze the output but just replays it to the application.

We should point out that both the *impact factor* and the $threshold$ render the estimation of the confidence of *S* more flexible. For instance, it might happen that some processes in *S* are faulty or suspected of being faulty but *S* is still considered to be trusted. Consequently, the Adaptation Manager will be less requested, since there is some flexibility for nodes failure. Furthermore, the Impact FD is easily configurable and adaptive according to the needs of the application or system requirements that can dynamically changes. For instance, the application may require a stricter monitoring of nodes during the night than during the day. For such an adaptation, it is only necessary to adjust the *threshold*.

The paper is structured as follows. In Section II, some application scenarios are presented. Section III outlines some basic concepts of unreliable failure detectors. In Section IV, the Self-healing Module is described and Section V presents the Impact Failure Detector. Section VI presents some preliminary evaluation results of experiments conducted with real traces on PlanetLab [1]. Section VII discusses some existing related studies. Finally, Section VIII concludes the paper and outlines some of our research research work.

II. Motivation Scenarios

The Impact FD can be applied to different distributed scenarios and it is flexible enough to meet different needs. It is quite suitable for environments where there is node redundancy or nodes with different capabilities. The following examples illustrate scenarios where the module can be used.

A system in the area of healthcare requires the use of several sensors to measure different kinds of information about the health status of a person, such as, vital signs, location, falls, gait patterns, and acceleration. From this perspective, this scenario is critical since faults in the components can risk the patient's life. For instance, let's consider a scenario with four sensors: q_1 - *body temperature;* q_2 - *pulse;* q_3 - *electrocardiogram(ECG); and* q_4 - *galvanic skin* as well as node *p*, responsible for collecting information from these sensors and taking appropriate action based on the output of the Impact FD. In this example, some sensors are not considered to be critical, such as the sensor q_1 which measures the temperature. On the other hand, q_3, the ECG sensor, is crucial. Therefore, the impact factor assigned to q_3 should be higher than q_1*'s*. Furthermore, q_3 collects data about both the heartbeats and electrical activity of the heart while q_2 is a type of sensor that also collects data about the heartbeats. Hence, there is redundancy of information, i.e., the failure of q_2 sensor is not critical enough to make the system vulnerable and endanger the life of the monitored person. We could then define a threshold equals to the sum of the impact factor of all the sensors minus q_2*'s* impact factor since, the failure of q_2 does not jeopardize the trustness of the system.

Another important scenario that motivates our proposal is Ubiquitous Wireless Sensor Network (WSNs). These kinds of networks are deployed to monitor physical conditions in various places such as geographical regions, agriculture lands, battlefields, etc. In WSNs, there are a variety of sensor nodes with different battery resources and communication or computation capabilities [10]. However, these sensors are prone to failures (e.g., battery failure, process failure, transceiver failure, etc.) [11]. Hence, it is necessary to provide failure detection and adaptation strategies in order to ensure as much as possible that the failure of sensor nodes does not affect the overall task of the network. Redundant use of sensor nodes, reorganization of sensor network, and overlapped sensing regions are some of the techniques used to increase the fault tolerance and reliability of the network [12].

Let us take as an example a ubiquitous WSN which collects environmental data within a vineyard, which is grouped into management zones according to different characteristics (e.g., soil properties). Each zone is composed of sensors of different types (e.g., humidity control sensors, temperature control sensors, etc.) and the density of sensors in each zone depends on the characteristics of the latter. That is, the number of sensors can be different for each type of sensor within a given zone. Furthermore, sensors' redundancy ensures both area coverage and connectivity in case of failure. We can thus consider each management zone as a single set whose sensors of the same type are grouped into subsets. Such grouping approach enables the definition of a threshold which is equal to the minimum number of sensors that each subset must have in order to keep connectivity and the application functioning all time. Moreover, in some situation, there might be a need to dynamically reconfigure the density of the zones. In this case, the *threshold* value would change.

III. Unreliable Failure Detectors

In the following of this article, we consider that there is one process by node (site) or sensor. Therefore, the word process can mean a node, sensor or site.

A *correct* process is a process that never fails during the whole execution; otherwise it is *faulty*.

In synchronous distributed systems, message transmission delays and process speed are bounded and known, such that a simple timeout mechanism can be used to surely assert if a node has failed or not. On the other hand, in asynchronous distributed systems there are not bounds on process speed neither message delay. Therefore, no mechanism can ensure the failure of a remote process since it is impossible to know whether the latter has actually crashed or whether its message transmissions are delayed for some reason [13].

An important abstraction for the development of fault tolerant distributed systems is the unreliable failure detector [14]. It aims to encapsulate the uncertainty of the communication delay between two distributed entities and is usually implemented using wait time bound (timeouts).

An unreliable FD can be seen as an oracle that gives information, not always correct, about processes failures and it is based on the state notion of processes (trusted or suspected). It thus usually provides a list of processes suspected of having crash. According to [15], an unreliable FD module can make mistakes (1) by erroneously suspecting some correct process (false suspicion), or (2) by not suspecting a process that has actually crashed. If later the FD detects its mistake, it corrects the mistake. For instance, an FD can stop suspecting at time *t* + 1 some process that it suspected at time *t*. Unreliable failure detectors are characterized by two properties, *completeness* and *accuracy*, as defined in [14]. The completeness is related to the FD capability of suspecting every faulty process

permanently, while accuracy concerns the capability of not suspecting correct processes. FD are then classified according to two completeness proprieties and four accuracy properties [14]:

- Strong (resp. weak) completeness: Eventually every process that crashes is permanently suspected by every (resp. some) correct process.
- Strong (resp. weak) accuracy: No (resp. some) process is suspected before it crashes.
- Eventual strong (resp. weak) accuracy: There is a time after which correct processes (resp. some correct process) are (is) never suspected by any correct process.

Notice that the type of accuracy depends on the synchronism or stability of the network. For instance, a strong accuracy requires a synchronous system while a eventual strong one relies on a partially synchronous system which eventually ensures a bound for message transmission delays and processes speed.

The combination of the above properties yields eight classes of failure detectors as shown in Table I.

TABLE I. FAILURE DETECTORS CLASSIFICATION

Completeness	Accuracy			
	Strong	Weak	Eventual strong	Eventual weak
Strong	P	S	$\Diamond P$	$\Diamond S$
Weak	Q	W	$\Diamond Q$	$\Diamond W$

A. Implementation of Failure Detectors

The literature contains several proposals for implementation of unreliable failure detectors which usually exploit timers to detect faults. Basically, there are two main strategies: *heartbeat* and *ping*.

In the heartbeat strategy, the most current one, every process q periodically sends an *"I am alive"* message to the processes p in charge of monitoring q's liveness. If p does not receive such a message from q after the expiration of a timer, it adds q to its list of suspected processes. If p later receives an *"I am alive"* message from q, p then removes q from its list of suspected processes [13]. On the other hand, in the ping strategy, p monitors a process q by sending *"Are you alive?"* messages to q periodically. Upon reception of such messages, q replies with an *"I am alive"* message. If the timeout of p related to q expires, p adds q to its list of suspected processes. If later, p receives an *"I am alive"* message from q, p removes q from its list of suspected processes [13].

B. Estimation of heartbeat arrivals

Aiming at reducing both the number of false suspicions and the time to detect a failure, Chen et al. [7] propose an approach to estimate the arrival of the next heartbeat which is based on the history of heartbeats arrival time and a safety margin (α). The timer is then set according to this estimation.

The estimation algorithm is the following: process p takes into account the n most recent heartbeat messages received from q, denoted by $m_1, m_2, \ldots, m_n$; $A_1, A_2, \ldots, A_n$ are

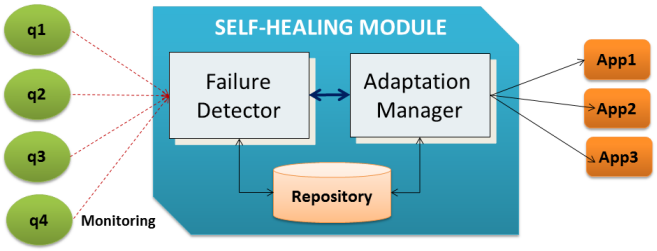

Fig. 1. Self-healing Module

their actual receipt times according to p's local clock. When at least n messages have been received, the theoretical arrival time $EA_{(k+1)}$ for a heartbeat from q is estimated by:

$$EA_{(k+1)} = \frac{1}{n} \sum_{i=k-n}^{k} (A_i - \Delta_i * i) + (k+1)\Delta_i$$

where Δ_i is the interval between the sending of two q's heartbeats. The next timeout value which will be set in p's timer and will expire at the next freshness point $\tau_{(k+1)}$, is then composed by $EA_{(k+1)}$ and the constant safety margin α:

$$\tau_{(k+1)} = \alpha + EA_{(k+1)}$$

Bertier et al. [8] have extended Chens approach by proposing an estimation function which combines Chens with Jacobsons [16] estimation. However, their approach is more suitable for LAN environments.

IV. SELF-HEALING MODULE

Figure 1 presents the Self-Healing Module which can be incorporated in a ubiquitous system, regardless of the architecture of the latter. It has two components: the *Failure Detector* and the *Adaptation Manager*. The former consists of an adaptive failure detector which is responsible for detecting crash failures of the different entities (nodes, sensors, etc.) that need to be monitored in the system. The Adaptation Manager makes decisions in order to adopt suitable adaptation strategies, aiming at reducing the impact of the detected failure.

A. Failure Detector

In order to fulfill the requirements of ubiquitous environments, a failure detector should present the following features:

- Strong Completeness: the FD oracle of a correct process should eventually detect all failures.
- Grouping: The user of the FD, in our case the Adaptation Manager, is interested in knowing if the system is trusted or not. Therefore, the FD output should express the confidence about the system as a whole (set of nodes) and not about each node individually. Moreover, it must provide the possibility of organizing nodes with some common characteristics in groups, i.e., subsets (see section II).

- Flexibility: nodes can be of different relevance or have different roles in the system. Consequently, their respective failures may have different impact on the proper functioning of the system. The FD must take into account such an heterogeneity. Furthermore, some systems tolerate a margin of failures (e.g. systems with redundant nodes) which the FD should also consider.
- Adaptability: The FD should be configurable in order to cope with different system confidence requirements, i.e., the fault margin may vary depending on the environment, situation, or context, that can dynamically change.

B. Adaptation Manager

According to the FD output, the Adaptation Manager can decide about the most suitable adaptation. Nevertheless, it is noteworthy that the application should have implemented the action of adaptation as well as the interaction with the Self-healing Module which triggers its execution. In other words, there exists an interaction protocol between the self-healing module and the application. Whenever requested, the Failure Detector informs if the system is trusted or not. If it is not the case, the Adaptation Manager takes a decision about the need of an adaptation. If the latter is necessary, it chooses a reconfiguration strategy and communicates it to the application which will activate it. We highlight once again that the Adaptation Manager is not the focus of this paper.

V. Impact Failure Detector

We consider a distributed system that consists of a finite set of processes $\Pi = \{q_1, \ldots, q_n\}$ with $|\Pi| = n$. Failures are only by crash. Other types of failures (e.g. misbehavior, transient, etc) is the object of a near future work. We assume the existence of some global time denoted T. A failure pattern is a function F: $T \rightarrow 2^{\Pi}$, where F(t) is the set of processes that have failed before or at time *t*. The function correct(F) denotes the set of correct processes, i.e., those that never belong to failure pattern (F), while faulty(F) denotes the set of faulty processes, i.e., the complement of correct(F) with respect to Π.

The Impact FD can be defined as an unreliable failure detector that provides an output related to the trust level with regard to a set of processes. If the trust level, provided by detector, is equal or greater than a given threshold value, defined by the user, the confidence in the set of processes is ensured. We thus say that the system is trusted. We denote ${I_p}^S$ the Impact failure detector module of process *p*.

Let $S \subset \Pi$ be a set of processes. Each process $q \in S$ has an *impact factor* ($I_q | I_q > 0 : I_q \in \mathbb{N}$). Moreover, the set *S* can be partitioned into *m* disjoint subsets $\{S_1, \ldots, S_m\}$.

Notice that the grouping feature of the Impact FD allows *S* to be partitioned into disjoint subsets, according to some criterion. For instance, in a scenario where there are different types of sensors, those of the same type can be gathered in the same subset, as in the example of the ubiquitous WSN of Section II.

An acceptable margin of failures, denoted $threshold^S$, which characterizes the acceptable degree of failure flexibility in relation to set *S*, is also defined. The $threshold^S$ is related to the minimum trust level required for each subset, i.e., it is defined as a set which contains the respective threshold of each subset of *S*: $threshold^S = \{threshold_1, \ldots, threshold_m\}$.

The Impact FD needs as input the set of subsets of S and the impact factor of processes that compose S as well as the respective *threshold* value of each subset of S. Moreover, each process knows its own impact factor as well as the subset to which it belongs and include such information in the messages it sends to the other processes of S.

Figure 2 shows several examples of sets and its respective threshold. In the first example (a) there is just one subset with three processes. Each process has impact factor equal to 1 and the *threshold* establishes that the sum of impact factor of non faulty processes must be at least equals to 2, i.e., the system is considered trusted whenever there are two or more correct processes. The example (b) shows a configuration where the processes must be monitored individually. Each process is in a subset and the threshold defines that if any of the processes fails, the system is not trusted anymore. In the next example (c), *S* has two sets with three processes each. The *threshold* requires at least two correct processes in each subset. The last example (d) has a single subset with five processes with different impact factor and the *threshold* delimits that the set is trusted when the sum of impact factor of correct processes is at least equals to seven.

When invoked in *p*, the Impact FD (${I_p}^S$) returns the ${trust\_level_p}^S$ and ${status_p}^S$ (trusted/not trusted) values. The ${trust\_level_p}^S$ is a set that contains the trust level of each subset, i.e., it expresses the confidence that *p* has in the set *S*. The ${status_p}^S$ informs whether the system is *trusted* or *not trusted* according to the analysis of the *threshold*.

Let FD (${I_p}^S$) be the failure detector module of process *p*. ${trusted_p}^S(t) = \{trusted_1, \ldots, trusted_m\}$, where each $trusted_i$ $(1 \leq i \leq m)$ contains the processes of S_i not considered faulty by p at time $t \in T$.

The *trust level* at $t \in T$ of processes $p \notin F(t)$ of *S* is the function ${trust\_level_p}^S$ such that ${trust\_level_p}^S(t) = \{trust\_level_i | trust\_level_i = sum(trusted_i); 1 \leq i \leq m\}$. The function $sum(set)$ returns the sum of impact factor of all elements of set.

The ${status_p}^S$ is generated at t based on the comparison of the $threshold^S$ with ${trust\_level_p}^S(t)$. If, for each subset of *S*, the $trust\_level_i(t) \geq threshold_i$, *S* is considered to be *trusted* at *t* by *p*, i.e., the confidence of *p* in *S* has not been compromised; otherwise *S* is considered *not trusted* by *p* at *t*.

Figure 3 shows an example with three subsets. The values of the set $threshold^S$ define that the subsets S_1 and S_2 must have one correct process at least and the subset S_3 must have 2. Several situations are shown and the set *S* is considered trusted when, for each subset S_i, $trust\_level_i(t) \geq threshold_i$.

We should point out that both the *impact factor* and the $threshold^S$ render the estimation of the confidence of *S* flexible. For instance, it might happen that some processes in *S* are faulty or suspected of being faulty but *S* is still considered to be trusted. Furthermore, the $threshold^S$ increases the tolerance

a)	$S = \{\{q_1, q_2, q_3\}\}$ $I_{q1}=1; I_{q2}=1; I_{q3}=1;$	$threshold^S = \{2\}$
b)	$S = \{\{q_1\}\{q_2\}\{q_3\}\}$ $I_{q1}=1; I_{q2}=1; I_{q3}=1;$	$threshold^S = \{1,1,1\}$
c)	$S = \{\{q_1, q_2, q_3\}\{q_4, q_5, q_6\}\}$ $I_{q1}=1; I_{q2}=1; I_{q3}=1; I_{q4}=2; I_{q5}=2; I_{q6}=2;$	$threshold^S = \{2, 4\}$
d)	$S = \{\{q_1, q_2, q_3, q_4, q_5\}\}$ $I_{q1}=1; I_{q2}=1; I_{q3}=1; I_{q4}=5; I_{q5}=5;$	$threshold^S = \{7\}$

Fig. 2. Examples of sets and threshold

$S=\{\{q_1,q_2\}, \{q_3\}, \{q_4, q_5, q_6\}\}$

$I_{q1}= 1;\ I_{q2}= 1;\ I_{q3}= 3;\ I_{q4}= 4;\ I_{q5}= 4;\ I_{q6}= 4;$

t	F(t)	$trusted_p^S(t)$	$trust\_level_p^S(t)$	Status
1	$\{\{q_2\}, \{\}, \{\}\}$	$\{\{q_1\}, \{q_3\}, \{q_4, q_5, q_6\}\}$	{1, 3, 12}	TRUSTED
2	$\{\{q_2\}, \{\}, \{q_6\}\}$	$\{\{q_1\}, \{q_3\}, \{q_4, q_5\}\}$	{1, 3, 8}	TRUSTED
3	$\{\{q_2\},\{\},\{q_5, q_6\}\}$	$\{\{q_1\}, \{q_3\}, \{q_4\}\}$	{1, 3, 4}	NOT TRUSTED
4	$\{\{q_2\}, \{q_3\}, \{q_5, q_6\}\}$	$\{\{q_1\}, \{\}, \{q_4\}\}$	{1, 0, 4}	NOT TRUSTED

$threshold^S = \{1, 3, 8\}$

Fig. 3. Example of FD output related to S with three subsets

S to false suspicions, reducing thus, wrong decisions of the Adaptation Manager.

It is also worth noting that the Impact FD is easily configurable according to the needs of the environment. The $threshold^S$ can be tuned in order to provide a more restrict or softer monitoring. Such an adaptability, as mentioned in the previous section, is essential in dynamic environments such as ubiquitous ones. Notice that the Impact FD can also be applied when the application needs information about each process of S individually. In this case, each process should be defined as a subset (see example (b) of Figure 2).

VI. Performance Evaluation

In this section, we firstly describe the environment in which the experiments were conducted and the QoS metrics which were used. Then, we present some evaluation results with different configurations of node sets with regard to both the impact factor and the threshold as well as comparison with the Chen FD [7], whose output is a list of suspect processes.

A. Environment

We used realistic traces files collected from ten nodes of PlanetLab [1], as summarized in Table II. The experiment started on July 16, 2014 at 15:06 UTC, and finished one full week later. Each site sent heartbeat messages to other sites at a rate of one heartbeat every 100 ms (the sending interval). Table III shows information about the heartbeat messages received by site number *1* (the monitor node). We observe that the mean inter-arrival times of received heartbeats is very close to 100 ms. However, in some sites, the standard deviation is very high, like in site *5* which the standard deviation is 310.958 ms

TABLE II. Sites of Experiments

ID	Site	Local
0	ple4.ipv6.lip6.fr	France
1	planetlab1.jhu.edu	USA East Coast
2	planetlab2.csuohio.edu	USA, Ohio
3	75-130-96-12.static.oxfr.ma.charter.com	USA, Massachusetts
4	planetlab1.cnis.nyit.edu	USA, New York
5	saturn.planetlab.carleton.ca	Canada, Ontario
6	PlanetLab-03.cs.princeton.edu	USA, New Jersey
7	prata.mimuw.edu.pl	Poland
8	planetlab3.upc.es	Spain
9	pl1.eng.monash.edu.au	Australia

TABLE III. Sites and Heartbeat Sampling

Site	Messages	Mean (ms)	Min (ms)	Max (ms)	Stand. Dev.(ms)
0	5424326	100.058	0.025	26494.168	19.525
2	1759989	100.415	0.031	509.093	9.275
3	5426843	100.012	0.027	1227.349	1.709
4	5414122	100.247	0.003	1193.276	18.595
5	5413542	100.258	0.006	657900.226	310.958
6	5426700	100.015	0.003	3787.643	2.557
7	5424117	100.062	0.006	59603.188	31.229
8	5424560	100.054	0.027	11443.359	100.714
9	5422043	100.100	0.004	30600.076	18.798

with a minimum of 0.006 ms, and a maximum of 657900.226 ms. Such values inform that, for a certain time interval during execution, the site stopped sending heartbeats and started again afterwards. Note also that site *2* stopped sending messages after approximately 48 hours and, therefore, there are just 1759990 received messages. Finally, we observe that sites *3* and *6* (resp. *5* and *8*) are the most stable (resp. unstable) sites.

We should point out that, despite the "large-scale" system and the high latency among the nodes, these traces of PlanetLab contain a large amount of data concerning the sending and reception of heartbeats, including unstable periods of links and message loss that induce false suspicions. Therefore, such traces can characterize any distributed system that uses FDs based on heartbeats, including self-healing ones. Furthermore, since the sending and arrival times of each heartbeat are recorded in the trace files using the experimental, all experiments were conducted with exactly the same scenarios and history of heartbeats.

B. QoS Metrics

In order to evaluate the Impact FD, we use three of the QoS metrics proposed by [7]: detection time, average mistake rate, and query accuracy probability. Considering two processes q and p where p monitors q, the QoS of the FD at p can be determined from the transitions between the "trusted" and "not trusted" states with respect to q.

- Detection Time (T_D): it is the time that elapses from the moment that process q crashes until the FD at p starts suspecting q permanently.
- Average Mistake Rate (λ_R): represents the number of mistakes that FD makes in a unit time, i.e., the rate at each a FD makes mistakes.
- Query Accuracy Probability (P_A): it is the probability that the FD output is correct at a random time.

TABLE IV. SET CONFIGURATIONS

Config	Impact Factor of each site
Set 0	I_0=2; I_2=1; I_3=6; I_4=6; I_5=1; I_6=6; I_7=1; I_8=2; I_9=2;
Set 1	I_0=1; I_2=2; I_3=6; I_4=6; I_5=2; I_6=6; I_7=2; I_8=1; I_9=1;
Set 2	I_0=6; I_2=2; I_3=1; I_4=1; I_5=2; I_6=1; I_7=2; I_8=6; I_9=6;
Set 3	I_0=2; I_2=6; I_3=1; I_4=1; I_5=2; I_6=1; I_7=2; I_8=6; I_9=6;
Set 4	I_0=2; I_2=1; I_3=6; I_4=6; I_5=1; I_6=6; I_7=2; I_8=2; I_9=1;
Set 5	I_0=3; I_2=3; I_3=3; I_4=3; I_5=3; I_6=3; I_7=3; I_8=3; I_9=3;

For the estimation of the arrival time of the next heartbeat we have applied Chen's approach [7], described in section III. The authors suggest that the safety margin α should range from 0 to 2500 ms. We set the window size for all experiments to 100 samples, which means that the FD relies only on the last 100 heartbeat message samples in order to compute the estimation of the next heartbeat arrival time.

C. Set Configuration

We defined a set composed of sites 0, 2, 3, 4, 5, 6, 7, 8 and 9 (S= {{*0, 2, 3, 4, 5, 6, 7, 8, 9*}}). Site *1* was the monitor node (p). Table IV shows the five configurations with regard to impact factor values that we have considered for S in the experiments. For all configurations, the sum of impact factor of processes is 27.

D. Experiments

1) Experiment 1 - Query Accuracy Probability: The aim of this experiment is to evaluate the Query Accuracy Probability (P_A) with different threshold values (18, 20, 21, 22, 23, 24, and 25) and different impact factor configurations (Table IV). We considered the fault margin α = 400 ms.

Figure 4 shows that the P_A decreases when the threshold increases. It is important to remember that the *threshold* is a limit value defined by the user and if the FD trust level output value is equal or greater than the threshold, the confidence in the set of processes is ensured. Hence, the results confirm that when the threshold is more flexible, the Query Accuracy Probability is greater.

On the one hand, *"Set 0"* configuration has the highest P_A for all *thresholds* due to the assignment of high (resp. low) impact factors for the most stable (resp. unstable) sites. On the other hand, *"Set 2"* and *"Set 3"* have the lowest P_A since unstable sites have high impact factor values in these sets. For instance, for site *8*, the mean of inter-arrival times of received heartbeats is 100.05 ms, but the standard deviation is 100,71 ms. The impact factor assignment has thus an impact on the P_A performance.

"Set 5" presents an abrupt decrease when the *threshold* = 25. Such a behavior can be explained since, in this set configuration, all sites have the same impact factor (3). Therefore, every false suspicion leads the *trust_level* to be smaller than then *threshold* (25) which increases the mistake duration and, consequently, the P_A decreases.

Notice that site *2* failed after approximately 48 hours. Thus, after its crash, the FD output, which indicates *status* = *not trusted* is not a mistake, i.e., it is not a false suspicion. Hence, in *"Set 3"*, whose impact factor of site *2* is 6 (high), the *PA* is

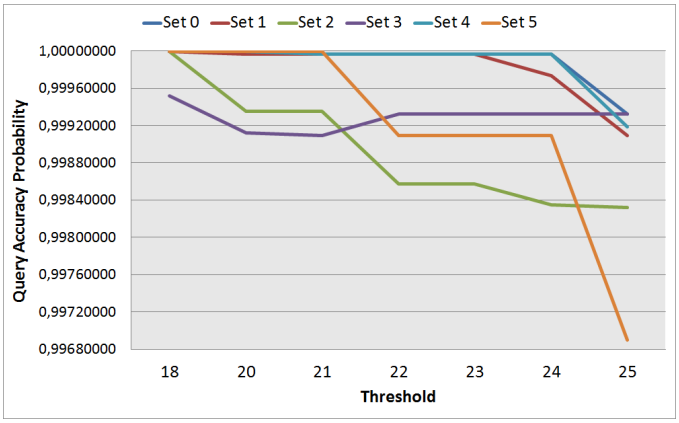

Fig. 4. PA vs. threshold with different set configurations

constant for *threshold* greater or equal to 22: after the crash of site *2*, the FD output $state$ is always "not trusted" and false suspicions related to other sites do not modify it. The average mistake duration of the experiment is thus smaller after the crash, which improves P_A.

In order to further compare the P_A of Impact FD with an approach that monitors the processes individually, we monitored each site using the same Chen algorithm and parameters (*WS=100*; α*=400 ms*). The mean P_A obtained was 0,979788. Such a result shows that, independently of the set configuration, Impact FD presents higher P_A than Chen FD since the former has flexibility to tolerate failures, i.e., the mistake duration only starts to be computed when the Impact FD output *state* informs that the system is "not trusted", contrarily to individual monitoring, as Chen FD, where every false suspicion increases mistake duration.

The results of this experiment highlight that the assignment of heterogeneous impact factors to nodes can degrade the performance of the failure detector, specially when unstable sites have high impact factor.

2) Experiment 2 - Detection time: In the second experiment, we have evaluated the average Query Accuracy Probability (P_A) average regarding the average detection time (T_D). To this end, we varied the safety margin (Chen's estimation) to get different values of detection time. It was varied with intervals of 100 ms, starting at 100 ms. In this experiment we used the set configuration *"Set 0"* and defined different thresholds. We have chosen such a set because it presented the best P_A for all thresholds in Experiment 1. We have also evaluated the P_A and T_D for the Chen's algorithm, which outputs the set of suspected nodes.

Figure 5 shows that for high threshold and detection time close to 200 ms, the P_A of the Impact FD is smaller, independently of the threshold, because the safety margin (used to compute the expected arrival times) is, in this case, equals to 100 ms, which increases both the number of failure suspicion and mistake duration. However, when T_D is greater than 230 ms, the P_A of Impact FD is considerably higher than Chen. After the detection time of approximately 400 ms, the P_A of Impact FD becomes constant regardless of the detection time and threshold, getting close to 1. Such a behavior can

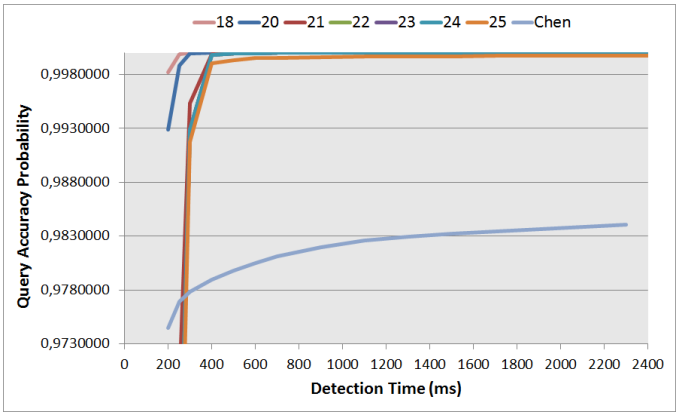

Fig. 5. P_A vs. T_D with different thresholds

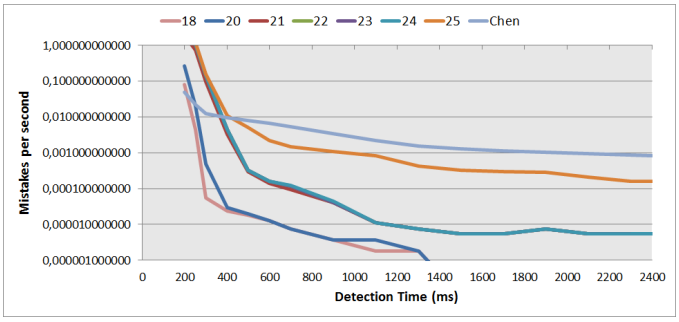

Fig. 6. λ_R vs. T_D with different thresholds

be explained since the higher the safety margin, the smaller the number of false suspicions, and the shorter the mistake duration which confirms that when the timeout is short, failures are detected faster but the probability of having false detections increases [17].

3) Experiment 3 - Average mistake rate: In this experiment, we have evaluated the average detection time vs. the mistake rate (mistakes per second). We consider *"Set 0"* configuration and the mistake rate is expressed on a logarithmic scale. We can observe in Figure 6 that the mistake rate of the Impact FD is higher when the detection time is low (smaller than 400 ms) and the threshold is high (from 23 to 25). Such a result is in accordance with Experiment 2: whenever the safety margin is small and *threshold* tolerates fewer failures, the Impact FD makes mistakes more frequently. In other words, the mistake rate decreases when threshold is more flexible or the time detection increases.

VII. Related Work

Related studies can be divided into two groups: (1) ubiquitous systems with failure handling and (2) failure detectors respectively.

In the first group, we find middlewares like Gaia, SAFTM, or MARKS. Gaia provides fault tolerance based on fail-stop model. Only devices (e.g., laptops, portable devices, etc.) can host applications [18]. Whenever it detects a lack of heartbeat messages, it infers a contextually appropriate surrogate device where the application can be restarted (rollback). The fault tolerant self-adaptive SAFTM middleware [19] detects failures by continuously monitoring of the state of the components (e.g. CPU, memory, OS, I/O, network operations, etc.) and dynamically building the self-adaptive mechanism in accordance with the various types of failures. The MARKS (ad-hoc) middleware has an unit called ETS (efficient, transparent, and secure) which is self-healing [20]. By predicting failures, it conducts an analysis of the changing rate of the status of each device (e.g. memory, energy, communication signal, etc.). With regard to fault containment, it isolates the faulty device and assigns the service to a provider of alternative resources. Bourdenas et al. [3] proposed the Starfish, a self-healing framework for pervasive systems, that follows the Self-Managed Cell (SMC) architectural paradigm. Starfish was an instantiation of an SMC for wireless sensor networks. It supports adaptation on nodes thereby allowing deployment of new strategies at run-time. However, it only provides recovering from sensor failures and does not consider other type of failures of pervasive computing.

We observe in the above works that there exist some limitations with regard to adaptation: in presence of failures, most of them use a fixed criterion for adaptation. Moreover, they do not provide a failure detector tailored for the features of ubiquitous environment, such as grouping, flexibility, or adaptability, like proposed in the current work.

In the second group, there are some important studies addressed to failure detectors. The ϕ Accrual failure detector [9] proposes an approach where the output is a suspicion level on a continuous scale, instead of providing information of a binary nature (trust or suspect). It is based on an estimation of inter-arrival times assuming that inter-arrivals follow a normal distribution. The suspicion level captures the degree of confidence with which a given process is believed to have crashed. If the process actually crashes, the value is guaranteed to accrue over time and tends toward infinity. In [17], the authors extended the Accrual FD by exploiting histogram density estimation. Taking into account a sampled inter-arrival times and the time of the last received heartbeat, the algorithm estimates the probability that no further heartbeat messages arrive from a given process, i.e., the latter has failed. Accrual failure detectors aim to decouple monitoring and interpretation. FD of class Σ (Sigma or quorum) output, for any failure pattern, any time τ, and any process p_i, a set of processes that are said to be *trusted* by p_i at time τ, such that the two following properties are satisfied: (1) the two sets of trusted processes intersect; (2) Eventually every trusted process is correct [21]. The FD of class Ω outputs the id of a process at each process and there is a time after which it outputs the id of the same correct process at all correct processes [22].

With respect to the above works, none of them deal with subsets nor processes' relevance and are not easily configurable to the needs of the system. Furthermore, among them, only Accrual and Adaptive Accrual have the output as a suspicion level. However, they do not analyze the output but just replays it to the application, contrarily to the Impact FD that compares the trust level output with the threshold defined by the user.

VIII. Conclusion and Future Work

Tailored for a self-healing module for ubiquitous computing, we have presented a new unreliable failure detector,

the Impact FD, which provides an output related to a set of processes and not just to each one individually. Both its *impact factor* and the *threshold* offer a degree of flexibility since they enable the user to tune the Impact FD in accordance with the specific needs and acceptable margin of failures of the application. As a result, the communication between the Impact FD and Adaptation Manager is reduced. In some scenarios and configurations, they also might weaken the rate of false responses when compared to traditional unreliable failure detectors. Performance evaluation results show that the assignment of high (resp. low) impact factor to more stable (resp. unstable) nodes increases the Query Accuracy Probability of the failure detector.

As a future work, we intend to extend the Impact FD in order to address misbehavior failures. Another direction of our research is the dynamic adaption of nodes' impact factor values according to their respective stability. A third aim is to conduct other experiments on different networks such as WiFi or LAN in order to compare the Impact FD with other well-known failure detectors. Finally, we look forward to evaluating the self-healing module in ubiquitous scenarios.

ACKNOWLEDGMENT

This research was supported by the UbiArch project - Ubiquitous Architecture for Context Management and Application Development at UFRGS.

REFERENCES

[1] http://www.planet-lab.org, access date: September 16, 2014.

[2] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[3] T. Bourdenas, M. Sloman, and E. C. Lupu, "Self-healing for pervasive computing systems," in *Architecting dependable systems VII*. Springer, 2010, pp. 1–25.

[4] R. Sterritt, "Autonomic computing," *Innovations in systems and software engineering*, vol. 1, no. 1, pp. 79–88, 2005.

[5] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM systems Journal*, vol. 42, no. 1, pp. 5–18, 2003.

[6] A. Rossetto, C. Rolim, V. Leithardt, C. F. Geyer, and L. Arantes, "An architecture for resilient ubiquitous systems," in *International Conference on Health Informatics*, 2014.

[7] W. Chen, S. Toueg, and M. K. Aguilera, "On the quality of service of failure detectors," *Computers, IEEE Transactions on*, vol. 51, no. 5, pp. 561–580, 2002.

[8] M. Bertier, O. Marin, P. Sens *et al.*, "Performance analysis of a hierarchical failure detector." in *DSN*, vol. 3, 2003, pp. 635–644.

[9] N. Hayashibara, X. Defago, R. Yared, and T. Katayama, "The φ accrual failure detector," in *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on*. IEEE, 2004, pp. 66–78.

[10] K. Ishibashi and M. Yano, "A proposal of forwarding method for urgent messages on an ubiquitous wireless sensor network," in *Information and Telecommunication Technologies, 2005. APSITT 2005 Proceedings. 6th Asia-Pacific Symposium on*. IEEE, 2005, pp. 293–298.

[11] D. Geeta, N. Nalini, and R. C. Biradar, "Fault tolerance in wireless sensor network using hand-off and dynamic power adjustment approach," *Journal of Network and Computer Applications*, vol. 36, no. 4, pp. 1174–1185, 2013.

[12] A. Z. Abbasi, N. Islam, Z. A. Shaikh *et al.*, "A review of wireless sensors and networks' applications in agriculture," *Computer Standards & Interfaces*, vol. 36, no. 2, pp. 263–270, 2014.

[13] L. Arantes, F. Greve, and P. Sens, *Handbook of Research on Mobility and Computing: Evolving Technologies and Ubiquitous Impacts*. IGI Global, 2010, ch. Unreliable Failure Detectors for Mobile Ad-hoc Networks, p. 20.

[14] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 225–267, 1996.

[15] N. Hayashibara, X. Défago, and T. Katayama, "Two-ways adaptive failure detection with the ϕ-failure detector," in *Workshop on Adaptive Distributed Systems (WADiS03)*. Citeseer, 2003, pp. 22–27.

[16] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4. ACM, 1988, pp. 314–329.

[17] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer, "A new adaptive accrual failure detector for dependable distributed systems," in *Proceedings of the 2007 ACM symposium on Applied computing*. ACM, 2007, pp. 551–555.

[18] S. Chetan, A. Ranganathan, and R. Campbell, "Towards fault tolerance pervasive computing," *Technology and Society Magazine, IEEE*, vol. 24, no. 1, pp. 38–44, 2005.

[19] H. Cai, C. Peng, L. Jiang, and Y. Zhang, "A novel self-adaptive fault-tolerant mechanism and its application for a dynamic pervasive computing environment," in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012 15th IEEE International Symposium on*. IEEE, 2012, pp. 48–52.

[20] M. Sharmin, S. Ahmed, and S. I. Ahamed, "Marks (middleware adaptability for resource discovery, knowledge usability and self-healing) for mobile devices of pervasive computing environments," in *Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on*. IEEE, 2006, pp. 306–313.

[21] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui, "Shared memory vs message passing," *Technical Report*, vol. 77, 2003.

[22] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg, "The weakest failure detectors to solve certain fundamental problems in distributed computing," in *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. ACM, 2004, pp. 338–346.