HAL
open science

Analysis of Adaptive Fault Tolerance for Resilient Computing

William Excoffon, Jean-Charles Fabre, Michaël Lauer

► **To cite this version:**

William Excoffon, Jean-Charles Fabre, Michaël Lauer. Analysis of Adaptive Fault Tolerance for Resilient Computing. 13th European Dependable Computing Conference (EDCC 2017), Sep 2017, Geneva, Switzerland. 9p., 10.1109/EDCC.2017.22 . hal-01708205

HAL Id: hal-01708205

https://hal.science/hal-01708205v1

Submitted on 13 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Analysis of Adaptive Fault Tolerance for Resilient Computing

William Excoffon[1], Jean-Charles Fabre[1], Michaël Lauer[2]
CNRS-LAAS, Ave du Colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, [1]INP, [2]UPS, LAAS, F-31400 Toulouse, France
e-mail: {william.excoffon, jean-charles.fabre, michael.lauer}@laas.fr

***Abstract*—A system that remains dependable when facing changes is called resilient. The fast evolution of systems, including safety critical systems, requires that fault tolerance mechanisms – FTM – remain consistent with their assumptions and the non-functional requirements of the application. A change event may impose the adaptation of an FTM to the current assumptions that can be made. Consequently, system resilience should rely on adaptive fault tolerant computing.**

In this paper, we report on an analysis of the link between applications and their attached FTM. We show how a set of FTMs or their composition can be a solution according to a change event occurring in the system. We propose a measure to estimate the resilience of a system. According to application characteristics and fault tolerance requirements, we show the impact of assumptions on FTM selection. We finally draw some lessons learnt for the development of resilient systems.

Keywords – fault tolerance, resilience, evolution, analysis

I. INTRODUCTION

Systems have to evolve during their service life in order to cope with additional features requested by users. For dependable embedded systems, the challenge is greater, as evolution must not impair dependability attributes. The persistence of dependability when facing changes is called resilience [1]. Dependability relies at runtime on fault tolerance mechanisms (FTMs or Safety Mechanisms) attached to the application [2]. A challenge of resilient computing is to maintain the adequacy between any application and its attached FTMs during the operational life of the system despite change events.

Definition: a resilient computing system is a system able to adapt its fault tolerance mechanisms at runtime to comply with its fault-tolerance requirements.

Resilient computing[1] adds a new dimension to fault tolerant computing by including evolution events, in particular unexpected changes related to application updates, new fault tolerance requirements, system configuration changes, etc. In this respect, ideal fault tolerance strategies installed at one time can be invalidated by a change later on.

The word "Resilience" is often used in Cloud computing [3] and networking as a synonymous to fault tolerance that is not consistent with our definition. However, the metrics used in these fields (data losses, CPU efficiency…) are analog to Fault Tolerance measures. This work proposes an approach to measure the impact of unexpected change events on the persistence of dependability. The proposed measures are complementary to standard fault tolerance measures.

The need for *Adaptive Fault Tolerance* (AFT) was stated in [4] to address this challenge. AFT is gaining today more importance and on-line adaptation of FTMs has attracted research efforts for some time now. However, most of the solutions, as in [5], tackle adaptation in a preprogrammed manner: all FTMs necessary during the system lifetime are known and deployed from the beginning and adaptation consists in choosing the appropriate execution branch or tuning some parameters, e.g., number of replicas or interval between checkpoints. Clearly, predicting all events and threats a system may encounter throughout its service life and making provisions for them is impossible.

An innovative approach was proposed in [4], taking advantage of component-based software engineering techniques to develop adaptive fault tolerance mechanisms. FTM are developed as a collection of *lego-bricks* that can be combined and manipulated dynamically at runtime under certain assumptions [6]. Whatever adaptive fault tolerance is implemented, the important question is the following: ***how to determine a suitable FTM after a change event in order to maintain system dependability to improve resilience?***

Answering this question is the challenge of the work reported in this paper. Our analysis takes as inputs the application characteristics that have an impact on the selection of an FTM with respect to a given set of faults. For any application, the result of the proposed analysis can be: i) an FTM or a combination of available FTMs exist to comply with both application characteristics and their fault-tolerance requirements, or ii) there is no solution with the set of available FTMs, and thus new FTMs or variants of available FTMs must be developed.

We define a notion of *Consistency Ratio* (CR) to quantify the capacity of a set of FTM to comply with application characteristics and fault tolerance requirements. For a given set of FTM, a list of application characteristics and a set of faults to be tolerated, the CR corresponds to the proportion of the cases where a solution, i.e. a suitable FTM, is found. Each case can be seen as a cell of a table having application characteristics as rows and type of faults as columns. The content of the cell is one or several FTM or nothing. A change event for a given application corresponds to a jump from one cell in the table to another one. The system remains resilient if an FTM is found in the new cell for this

[1] ReSIST NoE, Resilience for Survivability in IST, http://www.resist-noe.org/.

application. The probability of finding a suitable FTM in the cell is directly related to the notion of CR.

The CR is a static notion at a given point in time whereas the resilience is a dynamic notion during the lifetime of a system. During the operational life, the list of application characteristics and fault types may change and thus have an impact on the CR. The resilience of the system can be estimated by the combination of the CR for all applications in the system after a change event during the system lifetime.

This work provides a basic tool to estimate the resilience of a system. The sensitivity analysis with respect to the set of FTM, the application characteristics and the set of faults, shows the impact, positive or negative, of a change event on the CR, and so on the resilience of the system.

After a brief summary of the context in Section II, we perform in Section III a manual analysis of the system resilience using a basic set of FTMs and application characteristics. In Section IV, we describe our analysis more formally and show how a solution can be identified by computation. A sensitivity analysis to both application and fault model changes is proposed in Section V. Lessons learnt are given in Section VI.

II. CONTEXT AND PROBLEM STATEMENT

A. Adaptation and Change Model

An appropriate *Fault Tolerance Mechanism* (FTM) for a given application depends on several parameters grouped in three classes: 1) application characteristics (AC); 2) fault model to consider (FM); 3) available resources (AR).

At any point in time, the FTM(s) attached to an application must be consistent with the current values of (AC, FM, AR). These parameters enable to discriminate FTMs. We denote (AC, FM, AR) the change model.

Several application characteristics (AC) have an impact on the selection of an FTM. In this paper we first consider the following: i) behavioral determinism, ii) application statefulness, iii) state accessibility and iv) fail-silence.

Regarding the fault model (FM), we consider well-known fault types, e.g., crash, omission and transient faults.

The available resources (AR) play also an important role. Firstly, in the FTM selection, since FTMs require resources to be implemented such as bandwidth, CPU, battery life/energy. This resource criterion may invalidate a solution.

However, resources can be a trigger for FTM change. A lack of resources at a given point in time in the operational life of the system may invalidate an FTM. This implies that a new FTM must be installed according the available resources. This aspect has not been considered in this paper.

Any parameter variation during the service life of the system may invalidate the initial FTMs selected, thus requiring a transition towards a new one. Transitions may be triggered by a new application version with different characteristics or new threats (i.e. fault model change).

Definition: A configuration (A ◊ FTM) must remain consistent despite changes in the characteristics of the application and its related fault tolerance requirements.

We assume that any application A is attached (denoted ◊) to an FTM that complies with its fault tolerance requirements F and that resources to run this FTM are available. This is called a configuration.

B. FTM and Assumptions

To illustrate our modeling approach, we consider some conventional fault tolerance mechanisms that will be used as a guiding thread through the remainder of the paper.

Duplex protocols tolerate crash faults using passive (e.g. *Primary-Backup Replication* denoted PBR[2]), or semi-active replication strategies (e.g. *Leader-Follower Replication* denoted LFR[3]). Each replica is considered as a *self-checking* component in both cases. At least 2 independent processors (error confinement areas) are necessary to run these FTMs.

Time Redundancy (TR) tolerates transient faults leading to omission or value errors using repetition of the computation and comparison. This can also be perceived as a way to improve the self-checking nature of a replica.

In this paper, our fault model includes permanent and transient hardware faults or random operating system faults. We do not consider common mode faults.

The above *Duplex strategies* can be combined with *TR* to tolerate both transient and permanent physical faults.

TABLE I. ASSUMPTIONS AND FAULT TOLERANCE MECHANISM

Assumptions / FTM		PBR	LFR	TR
Fault Model (FT)	Crash	✓	✓	
	Omission			✓
	Transient			✓
Application Characteristics (AC)	Deterministic		✓	✓
	State access	✓		✓
	Fail silent	✓	✓	

The considered FTMs, in terms of fault model and application characteristics, are shown in TABLE I. For instance, PBR and LFR tolerate the same fault model (namely crash), but have different application behavioral assumptions. PBR allows non-determinism of applications because only the Primary computes client requests while LFR only works for deterministic applications as both replicas compute all requests. LFR could tackle non-determinism if all non-deterministic actions can be captured. This is not what we consider in a first step. As mentioned previously, PBR requires state access (if any) for checkpointing application state, while LFR does not require state access when cloning replicas is not considered. TR also requires state access (if any) to restore the previous state of the computation before repetition of the processing.

[2] With PBR, only one replica is active, the Primary. The state of the computation is forwarded to the Backup in a checkpoint. The Backup replica handles checkpoints in normal operation and takes over when the Primary crashes.

[3] With LFR, both replicas are active, the Leader and the Follower. The Leader replies to client's request, while the Follower just executes the request to update its computational state. The Follower takes over when the Leader crashes.

C. Evolution scenarii example and possible transitions

During the service life of the system, the characteristics of the application or its fault tolerance requirements of parameters can change.

An application can become non-deterministic when a new version is developed. The fault model can also become more complex, e.g., from crash-only it can become crash-and-value. For instance, the PBR→LFR transition (denoted →) is triggered by a change in application characteristics (e.g. inability to access application state). A transition can occur in both directions, w.r.t parameters variation. A transition obviously implies an off-line validation of the new configuration (A ◊ FTM).

The PBR→PBR+TR (FTMs composition denoted +) transition is triggered by a change in the considered fault model (e.g. crash-and-value). It worth noting that the composition of FTMs is not straightforward and requires a deep analysis of the impact of the first FTM on the second one. This analysis is out of the scope of this work. The composition of FTM must validated off-line before using it, taking care of possible interferences between FTMs [7].

III. BASIC ANALYSIS OF AFT FOR RESILIENT COMPUTING

In this Section we perform a manual analysis of the *Consistency Ratio* a given set of FTM can offer to an application. This analysis will be automated is section IV. For the example we assume that only PBR, LFR and TR strategies are available. Also any duplex strategy (PBR or LFR) can be combined with TR. The aim of this analysis is to show that even slight changes in the basic assumptions we use can drastically alter the *Consistency Ratio*.

In this analysis, the application characteristics that have an impact on the selection of an FTM are the determinism, statefullness, state accessibility, and fail-silence. These characteristics are the main differentiators between the FTM considered in the paper, as in [6]. To characterize the application we use the following notation:

- DT for DeTerministic, !DT if not.
- ST for STateful, !ST if stateless.
- SA for State Access, !SA if not.
- FS for Fail Silent, !FS if not.

The fault-tolerance requirements notation is as follows:

- C for Crash faults, !C if not considered.
- O for Omission, !O if not.
- V for Value errors, !V if not.

All possible combinations of application characteristics and fault tolerance requirements must be considered. With the set of FTM defined in Section II.B, namely {PBR, LFR, TR} (see also TABLE I.), our objective here is to check whether a solution can be found for each combination (AC, FM) that can occur during the lifetime of the system due to change events.

In the following analysis, application and their fault-tolerance requirements have the same probability of occurrence. In practice, a real system may exhibit different probabilities that must be included in the final evaluation of the system resilience. This is part of our ongoing work.

A. Analysis with FTM strict definitions

In this basic analysis, we do consider strict definitions of the FTM as follows:

a) Our duplex protocols PBR and LFR tolerate crash faults leading to clear stop errors:

- both assume a fail-silent behavior of the application.
- PBR needs access to the state of the computation to perform checkpointing.
- LFR is only valid for deterministic applications and does not need access to the state of the computation.

b) TR tolerates transient faults leading to absence of result (omission) or to erroneous results (value error):

- the behavior must be deterministic.
- the state of the computation must be accessible to be restored to repeat the computation.
- The comparison among several results decides on a valid value.

In TABLE II. we list in columns all possible combinations of fault tolerance requirements an application can specify. In rows, we have all possible combinations of application characteristics. Green Boxes mean that at least one FTM was found to comply with both application characteristics and fault-tolerance requirements. We ignore the case where no fault tolerance requirements are requested, i.e. !C,!O,!V.

TABLE II. ANALYSIS WITH STRICT DEFINITIONS OF PBR, LFR AND TR.

FM → / AC ↓	C !O !V	!C O !V	!C !O V	C O !V	C !O V	!C O V	C O V
!DT,!ST,!SA,!FS							
!DT,!ST,!SA,**FS**	PBR						
!DT,!ST,**SA**,!FS							
!DT,!ST,**SA,FS**	PBR						
!DT,**ST**,!SA,!FS							
!DT,**ST**,!SA,**FS**							
!DT,**ST,SA**,!FS							
!DT,**ST,SA,FS**	PBR						
DT,!ST,!SA,!FS		TR	TR			TR	
DT,!ST,!SA,**FS**	LFR, PBR	TR	TR	LFR +TR	LFR +TR	TR	LFR +TR
DT,!ST,**SA**,!FS		TR	TR			TR	
DT,!ST,**SA,FS**	LFR, PBR	TR	TR	LFR +TR	LFR +TR	TR	LFR +TR
DT,ST,!SA,!FS							
DT,ST,!SA,**FS**	LFR						
DT,ST,SA,!FS		TR	TR			TR	
DT,ST,SA,FS	LFR, PBR	TR	TR	LFR +TR	LFR +TR	TR	LFR +TR

For non-deterministic applications, only PBR works when the state is accessible to tolerate crash faults and the application is fail-silent. Regarding transient faults, the repetition of the computation may lead to correct but different results and thus the comparison may lead to false alarms (good different results may lead to error messages!).

For deterministic applications, LFR works whatever the state is accessible or not, and when the application is fail-silent.

The result is the following: 30% of the cases are solved. It is worth noting, that for deterministic applications 55% of the cases are solved with the limited set of mechanisms we consider. The bad news is that for non-deterministic applications only about 5% of the cases are solved.

B. Analysis with a first extension of the FTM set

In the previous step of our analysis, our definition of TR does not help solving omission errors in all cases. TR can solve omission errors when considered as value error (absence of result) in the previous table.

A reduced version of TR with no comparison, just a repetition of the computation, enables omission error to be tolerated more largely. We can thus distinguish two variants of TR: i) repetition of the computation with comparison / voting for tolerating value errors (denoted TR), and ii) simple repetition to tolerate omissions (denoted TR0). In both cases, the access to the state of the computation (if any) is mandatory.

We observe that simple repetition (TR0) can be used to tolerate omission errors when the state of the computation (if any) is accessible or when there is no state, whatever the application is deterministic or not. Combining a duplex strategy, e.g. PBR with TR0, enables both crash fault and transient faults leading to omission errors to be tolerated simultaneously in TABLE III. (cf. Dark Green Boxes).

TABLE III. ANALYSIS WITH STRICT DEFINITIONS OF PBR, LFR AND TWO TR STRATEGIES.

FM → AC ↓	C !O !V	!C O !V	!C !O V	C O !V	C !O V	!C O V	C O V
!DT,!ST,!SA,!FS		TR0					
!DT,!ST,!SA,**FS**	PBR	TR0		PBR+ TR0			
!DT,!ST,**SA**,!FS		TR0					
!DT,!ST,**SA,FS**	PBR	TR0		PBR+ TR0			
!DT,**ST**,!SA,!FS							
!DT,**ST**,!SA,**FS**							
!DT,**ST,SA**,!FS		TR0					
!DT,**ST,SA,FS**	PBR	TR0		PBR+ TR0			
DT,!ST,!SA,!FS		TR, TR0	TR			TR	
DT,!ST,!SA,**FS**	LFR, PBR	TR, TR0	TR	LFR +TR	LFR +TR	TR	LFR +TR
DT,!ST,**SA**,!FS		TR, TR0	TR			TR	
DT,!ST,**SA,FS**	LFR, PBR	TR, TR0	TR	LFR +TR	LFR +TR	TR	LFR +TR
DT,ST,!SA,!FS							
DT,ST,!SA,**FS**	LFR						
DT,ST,SA,!FS		TR, TR0	TR			TR	
DT,ST,SA,FS	LFR, PBR	TR, TR0	TR	LFR +TR	LFR +TR	TR	LFR +TR

The result is the following now: 38% of the cases are now solved. For deterministic applications, the percentage of cases solved remains 55%. What is more interesting is that for non-deterministic applications we have now about 21% of the cases are solved, compared to 5% previously.

C. Analysis with a revision of the FTM definitions

Examining cases that remain unsolved, we need to either extend again the set of mechanisms or just revise our definitions. We have shown in the previous section that adding variants of the FTMs improves the results. We show here that the revision of the strict definitions help solving more cases.

We considered previously that duplex strategies cannot be applied when the application is not fail-silent. The notion of fail silence is probabilistic and depends on the coverage of the error detection mechanisms integrated within the application. The coverage cannot be 100%. The validation process and the measurements carried out in particular using fault injection enables the error detection coverage to be estimated. It is thus the responsibility of the developer to declare if its application is fail-silent or not according to the measurements obtained. However, in both cases, the application being fail-silent or not (FS and !FS), as determined by the developer, crash faults can be tolerated with duplex strategies. Even for non fail-silent applications, the developer may consider that crash fault must be tolerated despite some value errors can be observed.

Interestingly, the combination of TR and a duplex strategy (PBR or LFR) improves the situation described above. The application of TR improves the error detection coverage for non fail-silent applications, i.e. tolerance to value errors, and thus improves the fail-silent assumption required by any duplex strategy. This is true for deterministic applications and when the state of the computation (if any) is accessible.

TABLE IV. ANALYSIS WITH REVISED DEFINITIONS.

FM → AC ↓	C !O !V	!C O !V	!C !O V	C O !V	C !O V	!C O V	C O V
!DT,!ST,!SA,!FS	PBR	TR0		PBR +TR0			
!DT,!ST,!SA,**FS**	PBR	TR0		PBR +TR0			
!DT,!ST,**SA**,!FS	PBR	TR0		PBR +TR0			
!DT,!ST,**SA,FS**	PBR	TR0		PBR +TR0			
!DT,**ST**,!SA,!FS							
!DT,**ST**,!SA,**FS**							
!DT,**ST,SA**,!FS	PBR	TR0		PBR +TR0			
!DT,**ST,SA,FS**	PBR	TR0		PBR +TR0			
DT,!ST,!SA,!FS	LFR	TR, TR0	TR	LFR +TR0	LFR +TR	TR	LFR +TR
DT,!ST,!SA,**FS**	LFR, PBR	TR, TR0	TR	LFR +TR	LFR +TR	TR	LFR +TR
DT,!ST,**SA**,!FS	LFR	TR, TR0	TR	LFR +TR0	LFR +TR	TR	LFR +TR
DT,!ST,**SA,FS**	LFR, PBR	TR, TR0	TR	LFR +TR	LFR +TR	TR	LFR +TR
DT,ST,!SA,!FS	LFR						
DT,ST,!SA,**FS**	LFR						
DT,ST,SA,!FS	LFR	TR, TR0	TR	LFR +TR0	LFR +TR	TR	LFR +TR
DT,ST,SA,FS	LFR, PBR	TR, TR0	TR	LFR +TR	LFR +TR	TR	LFR +TR

The Blue Boxes represent the additional solution with revised definitions (TABLE IV.). The result is that 55% of the cases are now solved. For deterministic applications, the percentage of cases solved is now 89%. For non-deterministic applications, the percentage remains low, only 32% of the cases are solved.

D. General comments about the analysis

We have shown that with a very limited extension of the mechanisms and a revision of the assumptions required to apply a given FTM, we can impact drastically the number of cases solved, i.e. the FTM *Consistency Ratio*. However, some cases remain unsolved because of two problems that can be summarized as follows: i) non-deterministic application for which our current set of FTM cannot tolerate value error, or ii) non-deterministic or deterministic stateful application whose state is not accessible.

Solving the first problem implies solving the non-determinism issue. This means that all non-deterministic decisions must be identified in the implementation of the application and notified. A variant of LFR can be developed, a variant in which the *Follower* synchronizes its behavior thanks to the notifications sent by the *Leader* as soon as a non-deterministic decision is made.

The second problem relates to the capture of the state of the computation. This is far more difficult to solve and thus required much more complex mechanisms. For instance, a transparent solution to this problem implies capturing all actions that modify the state of the computation and save them in a log (notion of journalization). The capture may involve a specific language or compiler facilities, virtual machine, hypervisor or even a specific hardware. Logs are used to restore the state of the computation by replaying the actions on a spare copy for instance.

We observed that non-determinism has a strong impact on the capabilities to provide solution to value errors and all combinations including them. In other words, the number of cases not solved for deterministic applications remain limited even with a simple set of FTMs, which means that solving non-deterministic issues is clearly of high interest.

Concerning the fail silence assumption required for any duplex strategy, non fail-silent applications can take advantage of TR to improve the coverage of built-in error detection mechanisms (e.g. autotest, defensive programming, exception handling, etc.). Then, the combination of TR with PBR or LFR in this order, i.e. TR first, makes sense to tolerate both crash and remaining value errors.

Finally, the developer is responsible for the definition of the application characteristics and the fault tolerance requirements in its application domain and semantics, the dependability objectives, the system environment. In a certain sense, he has to fill a questionnaire any time a change occurs, proving an answer to two set of questions:

- application characteristics: `DT? ST? SA? FS? (yes/no).`
- fault tolerance requirements: `C? O? V? (yes/no).`

The work reported in this section shows that change events occurring during the lifetime of the system may move applications from one "cell" to another. The fault tolerance requirements of the application are guaranteed as soon as a solution exists in the corresponding cell.

The analysis was done "manually" in this section, but such manual analysis is not scalable with many more characteristics, fault models, and FTM.

IV. Automated Analysis and Measurements

A. Formal notation

Our objective is now to automate this analysis. Based on the previous analysis, we can now define a formal notation to represent an application A and its fault tolerance requirements, from which a suitable FTM can be determined. As a result, a configuration A ◊ FTM is consistent according to the definition given below.

> *Definition: The design of a critical application is* ***consistent if and only if the*** *FTM assumptions and capabilities match Application characteristics and fault tolerance requirements.*

An application `A` has a set of non-functional characteristics, denoted `(aci), k ∈ [1..N]`, *N* being the total number of characteristics considered in the model. The `boolean` application characteristics are those used previously: `determinism, statefulness, state access, fail silence.` This list can obviously be extended.

The fault tolerance requirements of an application `Ai` correspond to the types of faults `fmj` the application `Ai` must tolerate. We use again a `boolean` notation to represent this set of faults, `(fmj), j ∈ [1..P]`, *P* being the total number of possible fault types affecting an application component in the system. Examples of such fault types are those used previously: `value fault, omission fault, crash fault.` This list can also be extended with other fault types.

An application is thus represented by two vectors, one for application characteristics, one for the fault types that must be tolerated. The FTM attached to the application in a configuration must tolerate such types of faults and be valid for the application characteristics.

$$A = \left(\begin{pmatrix} ac_1 \\ ac_2 \\ \ldots \\ ac_N \end{pmatrix}, \begin{pmatrix} fm_1 \\ fm_2 \\ \ldots \\ fm_P \end{pmatrix} \right)$$

The vector (ac_i) represents application characteristics. The vector (fm_j) represents the fault tolerance requirements.

The objective now is to determine FTMs making the configuration *A ◊ FTM* consistent. As shown in the previous section, an FTM provides a solution to tolerate some types of faults, but its validity depends on application characteristics. Our final aim is to compute the tables automatically, from the application model given above and a formal definition of the validity of fault tolerance mechanisms.

Let's examine the simple set of mechanisms we have considered previously {PBR, LFR, TR} with their strict definition. Any of these FTM relies on assumptions to tolerate a given type of fault. As an example, the assumptions for the use of PBR are `state access` if the application is `stateful` and `fail silent` behavior. When such assumptions are strictly true, then PBR tolerates the `crash` of the application. The same reasoning applies to other FTM and their combination.

B. FTM assumptions and properties

The assumptions for each mechanism in the set of FTM with respect to application characteristics can be defined by

logical assertions. We first use assertions to test the ***compatibility*** of an FTM with the application characteristics. Assertions for the FTM considered are defined as follows:

Assumption for PBR:	`FS and !(ST and !SA)`
Assumption for LFR:	`DT and FS`
Assumption for TR:	`DT and !(ST and !SA)`

The assertion for the combination of several FTM can be deduced from the above logical expressions, such as:

Assumptions for LFR+TR:	`FS and DT and !(ST and !SA)`

The above assertion is stronger than needed; if it is true we guaranty that the FTMs composition is valid. However, a composition may require reduced assumptions. For instance, applying TR implies that fail silence (`FS`) is no longer necessary since TR is a way to improve the fail silence. This is not considered in our algorithms, but reducing the assumption set could improve the CR.

The Boolean expressions allow us to take in account the dependencies between some characteristics. For instance, the state characteristics has an impact on the CR only when the application is stateful. We use assertions to test the ***adequacy*** of an FTM with respect to fault tolerance requirements. Assertions for the FTM considered are defined as follows:

Fault model for PBR:	`!O and !V`
Fault model for LFR:	`!O and !V`
Fault model for TR:	`!C`

Now, we can define properties to check the consistency of any application configuration *(A ◊ FTM)*.

> *Definition of* ***compatibility****: FTM is compatible with A* ***if and only if*** *FTM assumptions comply with the application characteristics of A.*
>
> *Definition of* ***adequacy****: FTM is adequate with A* ***if and only if*** *FTM tolerates the fault model required by A.*
>
> *Definition of* ***consistency****: A configuration (A ◊ FTM) is consistent* ***if and only if*** *it complies with both the compatibility and adequacy properties.*

C. Notion of Consistency Ratio

During its lifetime, the several versions of a given application developed and loaded into the system may have an impact on the application characteristics and thus invalidate the FTM that has been attached to the application A in a first place. To help quantify the resilience offered by a set of FTM, we define a notion of *Consistency Ratio*.

> *Definition of* ***Consistency Ratio****: For all combinations of application characteristics and fault tolerance requirements, the CR is the ratio of consistent configurations (A ◊ FTM) we can obtain for a given set of FTM.*

This is exactly what we have done manually in Section III. The two questions we want to address now are:

- Can we compute the configuration tables from the application model and the FTM logical expressions?
- How to compute the CR for such configuration tables and FTM definitions?

D. Computing configuration tables and Consistency Ratio

The proposed algorithm creates the tables from the application characteristics and fault tolerance requirements using the logical assertions established for each FTM in the set of FTMs. From the tables obtained, it computes the CR. Using the same application characteristics, fault tolerance requirements and FTMs with strict definitions we should be able to reproduce the TABLE II. and the corresponding CR.

All application characteristics and fault tolerance requirements are encoded with `boolean` values. The application is represented by two `boolean` vectors. For instance, a deterministic (`DT`), stateful (`ST`), with accessible state (`SA`), fail silent (`FS`) application requiring tolerance to crash faults is represented like this:

$$A = \left(\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right)$$

Each FTM assertion is implemented as a function returning a `boolean` value. The output is `1` if the application *A* matches the FTM assumptions and fault model, `0` otherwise. The assertion for the PBR mechanism is:

```
FS and !(ST and !SA) ) and (!O and !V)
```

Here is in pseudo-code our simple algorithm to automate the computation of the table and the CR. The implementation of the algorithm was validated using TABLE II. as an oracle using the same characteristics, fault tolerance requirements and FTMs with strict definitions.

```
INPUT: Application model, FTMs set
FOR each application characteristics SA
   FOR each fault model FM
      FOR each FTM
         IF (AC,FM) ◊ FTM is consistent THEN
            Store FTM in the cell (AC,FM)
         END IF
      END FOR
      IF the cell (AC,FM) is not empty
         Increment NbrOfConsCells
      END IF
   END FOR
END FOR
RETURN: CR=NbrOfConsCells/TotNbrOfCells
```

Fig. 1. Algorithm for CR computation

Clearly, CR values for the other scenario used in the manual analysis (see TABLE III. and TABLE IV.) can also be obtained with the algorithm. The approach can be extended to any characteristic, fault model, and set of FTM.

V. SENSITIVITY ANALYSIS

This section proposes a sensitivity analysis of the CR offered by a given set of FTM. The aim is to identify application characteristics and types of faults that have the most impact on the CR. We use the algorithm presented in the previous Section to measure the CR when fixing some of the parameters. First, we will address the sensitivity regarding application characteristics (Section V.A) and then, we will discuss the sensitivity to fault types (Section. V.B).

This analysis will help us to find the most efficient way to improve the *Consistency Ratio*.

A. Sensitivity to application characteristics

In this Section, we analyse the impact of application characteristics on the CR. For each application characteristic, we measure the CR when the characteristic is set to a particular value, and the other one can vary. In Fig. 2 each bar corresponds to a CR. The X-axis represents each specific characteristic and its assigned value (1 or 0). When a given characteristic is set to 1, it means that it is true for the application. For example, the bar "DT=1" represents the CR offered to an application which will always be deterministic regardless of future updates.

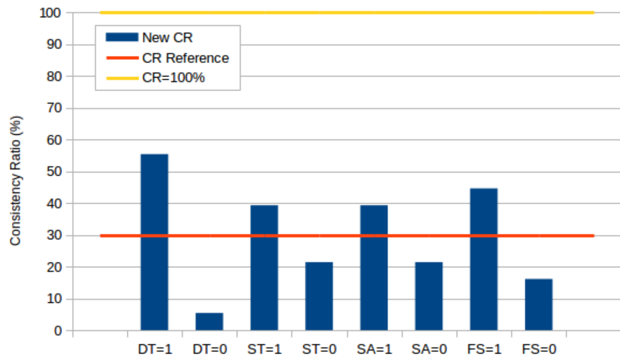

Fig. 2. Sensitivity to application characteristics

For this analysis, the set of FTMs is the one used in Section III.A, i.e. {PBR, LFR, TR}, with their strict definitions. The CR for this set of FTM was about 30% when no characteristic is imposed (see. TABLE II.) This CR value is our *Reference* value in our sensitivity analysis. The question now is: *what is the impact on this reference value when fixing one application characteristic for this FTMs set?*

The CR obtained when fixing a characteristic to 1 is higher that the reference CR value. For instance, when the application is guaranteed to be always deterministic (DT=1) the CR value is improved from 30% to 55%. As a consequence, we can anticipate that when the component is not deterministic, the CR value is lower. All this is confirmed by the results obtained in Fig. 2 where we show the impact of each characteristic. In our analysis, with the set of FTM we consider, we observed that determinism has the most important impact on the CR value. Other characteristics have also an impact, but it is less significant.

The conclusion is that improving the CR implies focussing on this characteristic first. Two solutions are possible to improve it: Either we force the component to be deterministic or we need to include new FTM compliant with non-deterministic applications.

B. Sensitivity to fault types

In this Section, we analyse the impact of the fault types to the CR. The following analysis is done with the extended set of FTMs considered in Section III.B – TABLE IV. The reference is 55% in this case. We use the same method as previously: when a fault type is set to 1, it is part of the fault tolerance requirements of the application. When it is set to 0, the fault type is not part of the faults considered.

The results are given in Fig. 3. This figure shows the CR variation when a type of fault is removed (i.e. set to 0). As shown, the CR can increase or decrease. This enlightens the strong impact of the fault model, positive or negative, on the CR for a given set of FTM.

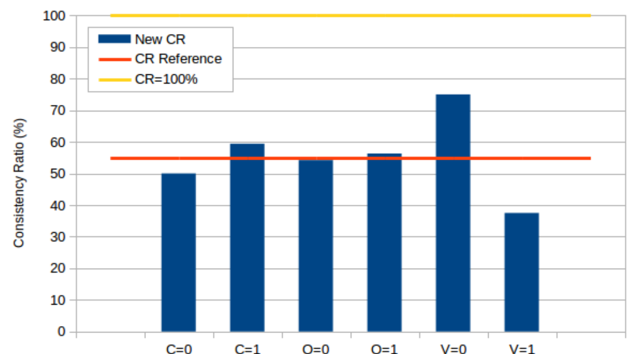

Fig. 3. Sensitivity to fault types

We observe that the CR is increased when we remove a fault type (e.g. value faults removed) from the set of fault types; the reason for this is that this type of fault is not efficiently tolerated with the current set of FTMs, whereas other faults are better tolerated.

Conversely, when the CR is decreased (e.g. crash faults removed) we can infer that these types of faults (namely crash faults here) are better tolerated than other types of faults (value and omission). It is indeed the case as shown previously with the simple set of FTM we consider in this analysis. A first solution to improve the CR could be to extend our set of FTM to include mechanisms dedicated to value faults. Another solution is to work on the application development itself to prevent by construction this kind of faults, e.g. implementing a self-checking version of the application.

In conclusion, this analysis is a mean to identify the fault types which are badly covered and finding a solution to improve the resilience. Two options are possible: i) extending the FTMs set or ii) removing these faults by design. The conclusion regarding the impact of application characteristics is similar: i) extending the FTMs set with mechanisms handling more efficiently some application characteristics or ii) removing by design those characteristics that have a bad impact on the *Consistency Ratio*.

VI. LESSONS LEARNT AND DESIGN PROCESS

The resilient computing interpretation of the CR value refers to the probability of having a compliant FTM after an update of the application. The higher this ratio is, the higher the probability to comply with new application characteristics or fault tolerance requirements. Developing a resilient system requires first the selection of an FTM compliant with fault tolerance requirements and the characteristics of the application. However, as we have shown, a single FTM may not offer much in terms of resilience. Thus, following the principles of Adaptive Fault Tolerance, some additional FTMs need to be included to future-proof the system. A question then arises: How to select these FTMs?

The design process could be the following: first the system designer needs to list all the FTM that could be implemented and for each one of them, construct all consistency assertions (see Section IV.B), respecting the critical assumptions for FTMs [8]. With these elements and thanks to the proposed automated analysis, it is possible to compute the CR value for each subset of these FTM. This exhaustive analysis[4] is illustrated in Fig. 4. We compute the CR value for each of the 2^6 subsets we can construct from the 6 FTM used in TABLE III. Each bar represents the CR value for a given subset and for readability reasons we have sorted the results in increasing order (i.e. the X-axis correspond to the list of 2^6 subsets ranked according to their corresponding CR value).

Then the designer can find all the subsets of FTM that satisfy some CR requirement. Finally, among all these subsets of FTM, the designer must use other criteria, such as required resources or development cost, to select the right set of FTM to implement. It is noteworthy that deciding the right CR requirement is obviously a complex problem in itself and it should at least take into consideration the criticality of the application, its envisioned frequency of updates, and the confidence in the initial fault model.

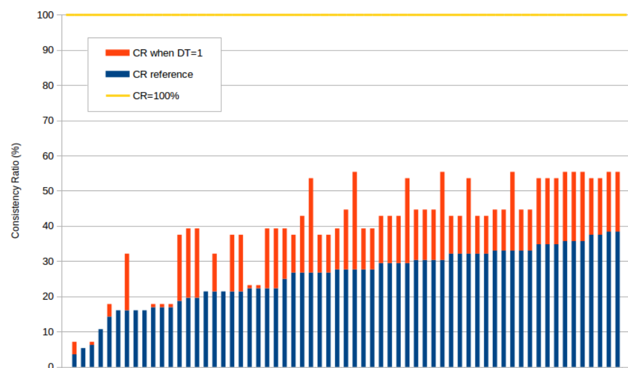

Fig. 4. Exhaustive CR analysis

If no subset of FTM is acceptable with respect to the desired CR, for example because implementation is too expensive, the sensitivity analysis can guide the designer towards other solutions as shown in section V. By way of example, Fig. 4 shows the CR values for all subsets of FTMs when the application is deterministic (in red). We clearly see that making this assumption offers the designer more choices. Such insights have an impact on the development of an application in a resilient system. Forcing determinism implies a development discipline, the identification of non-deterministic decisions, and also a non-concurrent implementation of the application. Lastly, FTMs must be adapted to synchronize with remaining non-deterministic decisions, if any, or new FTMs must be developed.

[4] Although this approach is exhaustive, we avoid scalability issues because our approach is solely focused on the critical assumptions and requirements for AFT.

VII. CONCLUSION

The work reported in this paper aimed at proposing an approach for the analysis of *Adaptive Fault Tolerance* for resilient computing. We have shown that the analysis requires a deep understanding of the application charateristics, the fault model and the core assumptions of fault tolerance mechanims. We introduced the notion of *Consistency Ratio* as an estimator of the system resilience, i.e. a measure of the system capability to remain dependable when facing changes. The CR can be computed with a simple algorithm. This approach is generic in the sense that it can be extended to any application characteristic, fault model, or FTM. The proposed sensitivity analysis is a mean to manage the development of resilient computing systems and is envisoned as a tool to help system designers to take appropriate decisions regarding resilience.

REFERENCES

[1] J.-C. Laprie, “From Dependability to Resilience", in 38th IEEE/IFIP International Conf. on Dependable Systems and Networks (DSN), suplemental volume, 2008.

[2] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso "Understanding replication in databases and distributed systems", *Distributed Computing Systems, 2000. Proceedings. 20th International Conference on*, 2000; pp. 464–474.

[3] M. Albanese, S. Jajodia, R. Jhawa and V. Piuri, "Dependable and Resilient Cloud Computing", 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), Oxford, 2016, pp. 3-3.

[4] K. H. K. Kim and T. F. Lawrence, “Adaptive Fault Tolerance: Issues and Approaches”, in Procs of the Second IEEE Workshop on Future Trends of Distributed Computing Systems. IEEE, 1990, pp. 38–46.

[5] C. Krishna and I. Koren, “Adaptive Fault-Tolerance for Cyber-Physical Systems”, in IEEE International Conference on Computing, Networking and Communications (ICNC), 2013, pp. 310–314.

[6] M. Stoicescu, J.-C. Fabre, M. Roy, "Architecting Resilient Computing Systems: A Component-Based Approach For Adaptive Fault Tolerance", Journal Of Systems Architecture, Elsevier Eds, Ref. Jsa-D-16-00131R1, Nov. 2016

[7] J. Lauret, J.-C.Fabre, H.Waeselynck,, “Fine-Grained Implementation of Fault-Tolerance Mechanisms with AOP: To what Extent”, SAFECOMP 2013, Toulouse (F), Sept.2013.

[8] W. Excoffon, J.-C. Fabre, M. Lauer, "Towards Modelling Adaptive Fault Tolerance for Resilient Computing Analysis", in SAFECOMP 2016: 15

[9] D. Powell, "Failure mode assumptions and assumption CR". *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, 1992; pp. 386–395

.