A Theory for Observational Fault Tolerance (Extended Abstract)

Adrian Francalanza[1] and Matthew Hennessy[2]

[1] University of Malta, Msida MSD 06, Malta, afra1@um.edu.mt
[2] University of Sussex, Brighton BN1 9RH, England, matthewh@sussex.ac.uk

Abstract. In general, faults cannot be prevented; instead, they need to be tolerated to guarantee certain degrees of software dependability. We develop a theory for fault tolerance for a distributed pi-calculus, whereby locations act as units of failure and redundancy is distributed across independently failing locations. We give formal definitions for fault tolerant programs in our calculus, based on the well studied notion of contextual equivalence. We then develop bisimulation proof techniques to verify fault tolerance properties of distributed programs and show they are sound with respect to our definitions for fault tolerance.

1 Introduction

One reason for the study of programs in the presence of *faults*, i.e. defects at the lowest level of abstractions [2], is to be able to construct more *dependable* systems, meaning systems exhibiting a high probability of *behaving* according to their *specification* [13]. System dependability is often expressed through attributes like maintainability, availability, safety and *reliability*, the latter of which is defined as a measure of the *continuous delivery* of *correct behaviour*, [13]. There are a number of approaches for achieving system dependability in the presence of faults, ranging from fault removal, fault prevention and *fault tolerance*.

The fault tolerant approach to system dependability consist of various techniques that employ *redundancy* to prevent faults from generating *failure*, i.e. abnormal behaviour caused by faults [2]. Two forms of redundancy are *space redundancy* (*replication*), i.e. using several copies of the same system components, and *time redundancy*, i.e. performing the same chunk of computation more than once. Certain fault tolerant techniques are based on *fault detection* which subsequently trigger *fault recovery*. If enough redundancy is used, fault recovery can lead to *fault masking*, where the specified behaviour is preserved without noticeable glitch.

Fault tolerance is of particular relevance in distributed computing; distribution yield a natural notion of *partial failure*, whereby faults affect a *subset* of the computation. Partial failure, in turn, gives scope for introducing redundancy as *replication*, distributed across independently failing entities such as locations. In general, the higher the replication, the greater the potential for fault tolerance. Nevertheless, fault tolerance also depends on how replicas are managed. One classification, due to [13], identifies three classes, namely *active replication* (all replicas are invoked for every operation), *passive replication* (operations are invoked on primary replicas and secondary replicas are updated in batches at checkpoints), and *lazy replication* (a hybrid of the previous two, exploiting the separation between write and read operations).

In this paper we address fault tolerance in a distributed setting, focussing on simple examples using *stateless* (read-only) replicas which are invoked only once. We code these examples in Dπ [8] with failing locations [5], a simple distributed version of the standard π-calculus [11], where the locations that host processes model closely physical network nodes.

Example 1. Consider the systems server_i, three server implementations accepting client requests on channel *req* with two arguments, x being the value to process and y being the reply channel on which the answer is returned.

$$\mathsf{server}_1 \Leftarrow (\nu\, data)\left(\begin{array}{l} l\llbracket req?(x,y).\mathsf{go}\ k_1.data!\langle x,y,l\rangle\rrbracket \\ \mid k_1\llbracket data?(x,y,z).\mathsf{go}\ z.y!\langle f(x)\rangle\rrbracket \end{array}\right)$$

$$\mathsf{server}_2 \Leftarrow (\nu\, data)\left(\begin{array}{l} l\left\llbracket req?(x,y).(\nu sync)\left(\begin{array}{l} \mathsf{go}\ k_1.data!\langle x, sync, l\rangle \\ \mid \mathsf{go}\ k_2.data!\langle x, sync, l\rangle \\ \mid sync?(x).y!\langle x\rangle \end{array}\right)\right\rrbracket \\ \mid k_1\llbracket data?(x,y,z).\mathsf{go}\ z.y!\langle f(x)\rangle\rrbracket \\ \mid k_2\llbracket data?(x,y,z).\mathsf{go}\ z.y!\langle f(x)\rangle\rrbracket \end{array}\right)$$

$$\mathsf{server}_3 \Leftarrow (\nu\, data)\left(\begin{array}{l} l\left\llbracket req?(x,y).(\nu sync)\left(\begin{array}{l} \mathsf{go}\ k_1.data!\langle x, sync, l\rangle \\ \mid \mathsf{go}\ k_2.data!\langle x, sync, l\rangle \\ \mid \mathsf{go}\ k_3.data!\langle x, sync, l\rangle \\ \mid sync?(x).y!\langle x\rangle \end{array}\right)\right\rrbracket \\ \mid k_1\llbracket data?(x,y,z).\mathsf{go}\ z.y!\langle f(x)\rangle\rrbracket \\ \mid k_2\llbracket data?(x,y,z).\mathsf{go}\ z.y!\langle f(x)\rangle\rrbracket \\ \mid k_3\llbracket data?(x,y,z).\mathsf{go}\ z.y!\langle f(x)\rangle\rrbracket \end{array}\right)$$

Requests are forwarded to internal databases, denoted by the scoped channel *data*, distributed and replicated across the auxiliary locations k_i. A database looks up the mapping of the value x using some unspecified function $f(-)$ and returns the answer, $f(x)$, back on port y. When multiple replicas are used, as in $\mathsf{server}_{2,3}$, requests are sent to all replicas in an arbitrary fashion, without the use of failure detection, and multiple answers are synchronised at l on the scoped channel *sync*, returning the first answer received on y.

The theory developed in [5] enables us to *differentiate* between these systems, based on the different behaviour observed when composed with systems such as

$$\mathsf{client} \;\Leftarrow\; l\llbracket req!\langle v, ret\rangle\rrbracket$$

in a setting where locations may fail. Here we go one step further, allowing us to *quantify* in some sense the difference between these systems. Intuitively, if locations $k_i, i = 1, 2, 3$, can fail in fail-stop fashion[12] and observations are limited to location l only, then server_2 seems to be more *fault tolerant* than server_1; observers limited to l, such as client, cannot observe changes in behaviour in server_2 when *at most 1* location from k_i fails. Similarly, server_3 is more *fault tolerant* than server_1 and server_2 because $\mathsf{server}_3 \,|\, \mathsf{client}$ preserves its behaviour at *l up to 2* faults occurring at $k_{1..3}$.

In this paper we give a formal definition of when a system is deemed to be fault tolerant up to n-faults, which coincides with this intuition. As in [5] we need to consider systems M, running on some network, which we will represent as $\Gamma \triangleright M$. Then we will say that M is fault-tolerant up to n faults if

$$F^n[\Gamma \triangleright M] \cong \Gamma \triangleright M \qquad (1)$$

where $F^n[\ ]$ is some context which induces at most n faults, and $\cong$ is some behavioural equivalence between systems descriptions. A key aspect of this behavioural equivalence is the implicit separation between *reliable* locations, which are assumed not to fail, and *unreliable* locations, which may fail. In the above example l is reliable, at which observations can be made, while the k_i are assumed unreliable, subject to failure. Furthermore it is essential that observers not have access to to these unreliable locations, at any time during a computation. Otherwise (1) would no longer represent M being fault tolerant; for example we would no longer have

$$F^1[\Gamma \triangleright \mathsf{server}_2] \cong \Gamma \triangleright \mathbf{server}_2$$

as an observer with access to k_i would be able to detect possible failures in $F^1[\Gamma \triangleright \mathsf{server}_2]$, not present in $\Gamma \triangleright \mathbf{server}_2$.

We enforce this separation between reliable, observable, locations, and unreliable, unobservable, locations, using a simple type system in which the former are called *public*, and the latter *confined*. This is outlined in Section 2, where we also formally define the language we use, DπLoc, give its reduction semantics, and also outline the behavioural equivalence $\cong$; this last is simply an instance of *reduction barbed congruence*, [6], modified so that observations can only be made at public locations. In Section 3 we give our formal definition of fault-tolerance; actually we give two versions of (1) above, called *static* and *dynamic* fault tolerance; we also motivate the difference with examples. Proof techniques for establishing fault tolerance are given in Section 4; in particular we give a complete co-induction characterisation of $\cong$, using labelled actions, and some useful up-to techniques for presenting witness bisimulations. In Section 5 we refine these proof techniques for the more demanding fault tolerant definition, *dynamic fault tolerance*, using *simulations*. Finally Section 6 outlines the main contributions of the paper and discusses future and related work.

2 The Language

We assume a set of *variables* Vars, ranged over by $x, y, z, \ldots$ and a separate set of *names*, Names, ranged over by $n, m, \ldots$, which is divided into locations, Locs, ranged over by $l, k, \ldots$ and channels, Chans, ranged over by $a, b, c, \ldots$. Finally we use $u, v, \ldots$ to range over the set of *identifiers*, consisting of either variables and names.

The syntax of our language, DπLoc, is a variation of Dπ [8] and is given in Table 1. The main syntactic category is that of *systems*, ranged over by M, N: these are essentially a collection of *located processes*, or *agents*, composed in parallel where location

Table 1. *Syntax of typed DπF*

Types			
$\mathtt{T} ::= \mathtt{ch}_{\mathtt{v}}\langle\tilde{\mathtt{P}}\rangle \mid \mathtt{loc}^{\mathtt{s}}_{\mathtt{v}}$	*(stateful types)*	$\mathtt{s} ::= \mathtt{a} \mid \mathtt{d}$	*(status)*
$\mathtt{U} ::= \mathtt{ch}_{\mathtt{v}}\langle\tilde{\mathtt{P}}\rangle \mid \mathtt{loc}_{\mathtt{v}}$	*(stateless types)*	$\mathtt{v} ::= \mathtt{p} \mid \mathtt{c}$	*(visibility)*
$\mathtt{P} ::= \mathtt{ch}_{\mathtt{p}}\langle\tilde{\mathtt{P}}\rangle \mid \mathtt{loc}_{\mathtt{p}}$	*(public stateless types)*		
Processes			
$P, Q ::= u!\langle V\rangle.P$	*(output)*	$\mid u?(X).P$	*(input)*
$\mid\ \mathsf{if}\ v{=}u\ \mathsf{then}\ P\ \mathsf{else}\ Q$	*(matching)*	$\mid\ *\,u?(X).P$	*(replicated input)*
$\mid\ (\nu\, n{:}\mathtt{T})P$	*(channel/location definition)*	$\mid \mathsf{go}\ u.P$	*(migration)*
$\mid\ \mathbf{0}$	*(inertion)*	$\mid P \vert Q$	*(fork)*
$\mid\ \mathsf{ping}\ u.P\ \mathsf{else}\ Q$	*(status testing)*		
Systems			
$M, N, O ::= l[\![P]\!]$	*(located process)*	$\mid N \vert M$	*(parallel)*
$\mid\ (\nu\, n{:}\mathtt{T})N$	*(hiding)*		

and channel names may be scoped to a subset of agents. The syntax for processes, P, Q, is an extension of that in Dπ: there is input and output on channels - here V is a tuple of identifiers, and X a tuple of variables, to be interpreted as a pattern - and standard forms of parallel composition, inertion, replicated input, local declarations, a test for equality between identifiers and migration. The only addition on the original Dπ is $\mathsf{ping}\, k.P\ \mathsf{else}\ Q$, which tests for the *status* of k in the style of [1, 10] and branches to P if k is alive and Q otherwise. For these terms we assume the standard notions of *free* and *bound* occurrences of both names and variables, together with the associated concepts of α-conversion and *substitution*. We also assume that systems are *closed*, that is they have no free variable occurrences.

As explained in the Introduction we use a variation (and simplification) of the type system of Dπ [8] in which the the two main categories, channels and locations, are now annotated by visibility constraints, giving $\mathtt{ch}_{\mathtt{v}}\langle\tilde{\mathtt{P}}\rangle$ and $\mathtt{loc}_{\mathtt{v}}$, where $\mathtt{v}$ may either be $\mathtt{p}$ (i.e. public) or $\mathtt{c}$ (i.e. confined); in Table 1 these are called *stateless* types, and are ranged over by $\mathtt{U}$. As explained in [5] a simple reduction semantics can be defined if we also allow types which record the status of a location, whether it is alive, $\mathtt{a}$, or dead, $\mathtt{d}$; these are refereed to as *stateful* types, ranged over by $\mathtt{T}$. Finally $\mathtt{P}$ ranges over *public* types, the types assigned to all names which are visible to observers.

Type System: Γ denotes a type environment, an unordered list of tuples assigning a single *stateful* type to names, and we write $\Gamma \vdash n : \mathtt{T}$ to mean that Γ assigns the type $\mathtt{T}$ to n; when it is not relevant to the discussion we will sometimes drop the various annotations on these types; for example $\Gamma \vdash n : \mathtt{ch}\langle\mathtt{U}\rangle$ signifies that $\mathtt{ch}_{\mathtt{v}}\langle\mathtt{U}\rangle$ for some visibility status $\mathtt{v}$. Typing judgements take the form $\Gamma \vdash N$ and defined by the rules in Table 2. In these rules, we use an extended form of type environment, Σ, which, in addition to names, also maps *variables* to *stateless* types. Note that none of the rules depend on the status (dead or alive) of names in the environment. Also the visibility constraints are enforced indirectly, by virtue of the formation rules for valid types, given in Figure 1.

In this extended abstract we omit even the statement of the Subject Reduction and appropriate Type Safety result for our language.

Table 2. *Typing rules for typed DπLoc*

Processes

(t-out)
$$\frac{\Sigma \vdash u : \mathrm{ch}\langle \tilde{\mathtt{U}} \rangle \quad \Sigma \vdash V : \tilde{\mathtt{U}} \quad \Sigma \vdash P}{\Sigma \vdash u!\langle V \rangle.P}$$

(t-in-rep)
$$\frac{\Sigma \vdash u : \mathrm{ch}\langle \tilde{\mathtt{U}} \rangle \quad \Sigma, X : \tilde{\mathtt{U}} \vdash P}{\begin{array}{c}\Sigma \vdash u?(X).P \\ \Sigma \vdash *u?(X).P\end{array}}$$

(t-nw)
$$\frac{\Sigma, \tilde{n} : \tilde{\mathtt{T}} \vdash P}{\Sigma \vdash (\nu\, \tilde{n} : \tilde{\mathtt{T}})P}$$

(t-cond)
$$\frac{\Sigma \vdash u : \mathtt{U},\ v : \mathtt{U} \quad \Sigma \vdash P,\ Q}{\Sigma \vdash \text{if } u = v \text{ then } P \text{ else } Q}$$

(t-fork)
$$\frac{\Sigma \vdash P,\ Q}{\Sigma \vdash P|Q}$$

(t-axiom)
$$\frac{}{\Sigma \vdash \mathbf{0}}$$

(t-go)
$$\frac{\Sigma \vdash u : \mathtt{loc} \quad \Sigma \vdash P}{\Sigma \vdash \text{go } u.P}$$

(t-ping)
$$\frac{\Sigma \vdash u : \mathtt{loc} \quad \Sigma \vdash P,\ Q}{\Sigma \vdash \text{ping } u.P \text{ else } Q}$$

Systems

(t-rest)
$$\frac{\Gamma, \tilde{n} : \tilde{\mathtt{T}} \vdash N}{\Gamma \vdash (\nu\, \tilde{n} : \tilde{\mathtt{T}})N}$$

(t-par)
$$\frac{\Gamma \vdash N,\ M}{\Gamma \vdash N|M}$$

(t-proc)
$$\frac{\Gamma \vdash l : \mathtt{loc} \quad \Gamma \vdash P}{\Gamma \vdash l\llbracket P \rrbracket}$$

Observers

(t-obs)
$$\frac{\mathbf{pub}(\Gamma) \vdash O}{\Gamma \vdash_{\text{obs}} O}$$

Table 3. *Reduction Rules for DπLoc*

Assuming $\Gamma \vdash l : \mathbf{alive}$

(r-comm)
$$\frac{}{\Gamma \rhd l\llbracket a!\langle V \rangle.P \rrbracket \mid l\llbracket a?(X).Q \rrbracket \ \longrightarrow\ \Gamma \rhd l\llbracket P \rrbracket \mid l\llbracket Q\{^{V}\!/X\} \rrbracket}$$

(r-rep)
$$\frac{}{\Gamma \rhd l\llbracket *a?(X).P \rrbracket \ \longrightarrow\ \Gamma \rhd l\llbracket a?(X).(P| * a?(X).P) \rrbracket}$$

(r-fork)
$$\frac{}{\Gamma \rhd l\llbracket P|Q \rrbracket \ \longrightarrow\ \Gamma \rhd l\llbracket P \rrbracket \mid l\llbracket Q \rrbracket}$$

(r-eq)
$$\frac{}{\Gamma \rhd l\llbracket \text{if } u = u \text{ then } P \text{ else } Q \rrbracket \longrightarrow \Gamma \rhd l\llbracket P \rrbracket}$$

(r-neq)
$$\frac{}{\Gamma \rhd l\llbracket \text{if } u = v \text{ then } P \text{ else } Q \rrbracket \longrightarrow \Gamma \rhd l\llbracket Q \rrbracket}\ u \neq v$$

(r-go)
$$\frac{}{\Gamma \rhd l\llbracket \text{go } k.P \rrbracket \longrightarrow \Gamma \rhd k\llbracket P \rrbracket}\ \Gamma \vdash k : \mathbf{alive}$$

(r-ngo)
$$\frac{}{\Gamma \rhd l\llbracket \text{go } k.P \rrbracket \longrightarrow \Gamma \rhd k\llbracket \mathbf{0} \rrbracket}\ \Gamma \nvdash k : \mathbf{alive}$$

(r-ping)
$$\frac{}{\Gamma \rhd l\llbracket \text{ping } k.P \text{ else } Q \rrbracket \longrightarrow \Gamma \rhd l\llbracket P \rrbracket}\ \Gamma \vdash k : \mathbf{alive}$$

(r-nping)
$$\frac{}{\Gamma \rhd l\llbracket \text{ping } k.P \text{ else } Q \rrbracket \longrightarrow \Gamma \rhd l\llbracket Q \rrbracket}\ \Gamma \nvdash k : \mathbf{alive}$$

(r-new)
$$\frac{}{\Gamma \rhd l\llbracket (\nu\, n : \mathtt{T})P \rrbracket \longrightarrow \Gamma \rhd (\nu\, n : \mathtt{T})\, l\llbracket P \rrbracket}$$

(r-str)
$$\frac{\Gamma \rhd N' \equiv \Gamma \rhd N \quad \Gamma \rhd N \longrightarrow \Gamma' \rhd M \quad \Gamma' \rhd M \equiv \Gamma' \rhd M'}{\Gamma \rhd N' \longrightarrow \Gamma' \rhd M'}$$

(r-ctxt-rest)
$$\frac{\Gamma + n : \mathtt{T} \rhd N \ \longrightarrow\ \Gamma' + n : \mathtt{U} \rhd M}{\Gamma \rhd (\nu\, n : \mathtt{T})N \ \longrightarrow\ \Gamma' \rhd (\nu\, n : \mathtt{U})M}$$

(r-ctxt-par)
$$\frac{\Gamma \rhd N \ \longrightarrow\ \Gamma' \rhd N'}{\begin{array}{c}\Gamma \rhd N|M \ \longrightarrow\ \Gamma' \rhd N'|M \\ \Gamma \rhd M|N \ \longrightarrow\ \Gamma' \rhd M|N'\end{array}}$$

Table 4. *Structural Rules for DπLoc*

(s-comm)	$N \vert M \equiv M \vert N$	
(s-assoc)	$(N \vert M) \vert M' \equiv N \vert (M \vert M')$	
(s-unit)	$N \vert l[\![\mathbf{0}]\!] \equiv N$	
(s-extr)	$(\nu\, n{:}\mathtt{T})(N \vert M) \equiv N \vert (\nu\, n{:}\mathtt{T})M$	$n \notin \mathbf{fn}(N)$
(s-flip)	$(\nu\, n{:}\mathtt{T})(\nu\, m{:}\mathtt{U})N \equiv (\nu\, m{:}\mathtt{U})(\nu\, n{:}\mathtt{T})N$	
(s-inact)	$(\nu\, n{:}\mathtt{T})N \equiv N$	$n \notin \mathbf{fn}(N)$

Reduction Semantics: We call pairs $\Gamma \triangleright N$ *configurations*, whenever $\Gamma \vdash N$. Reductions then take the form of a binary relation over configurations

$$\Gamma \triangleright N \quad\longrightarrow\quad \Gamma \triangleright N'$$

defined in terms of the reduction rules in Table 3, whereby systems reduce with respect to the status of the locations in Γ; we write $\Gamma \vdash l{:}\,\mathbf{alive}$ as a shorthand for $\Gamma \vdash l : \mathtt{loc}^{\mathrm{a}}$. So all reduction rules assume the location where the code is executing is alive. Moreover, (r-go), (r-ngo), (r-ping) and (r-nping) reduce according to the status of the remote location concerned. The reader is refered to [5] for more details; but note that here the status of locations is unchanged by reductions.

Behavioural equivalence: First note that the type system does indeed enforce the intuitive separation of concerns discussion in the Introduction. For example let Γ_e denote the environment

$$\Gamma_e \;=\; l{:}\mathtt{loc}^{\mathrm{a}}_{\mathrm{p}},\ k_1{:}\mathtt{loc}^{\mathrm{a}}_{\mathrm{c}},\ k_2{:}\mathtt{loc}^{\mathrm{a}}_{\mathrm{c}},\ k_3{:}\mathtt{loc}^{\mathrm{a}}_{\mathrm{c}},\ \mathit{req}{:}\mathtt{ch}_{\mathrm{p}}\langle \mathtt{T}, \mathtt{ch}_{\mathrm{p}}\langle \mathtt{T}\rangle\rangle,\ a{:}\mathtt{ch}_{\mathrm{p}}\langle \mathtt{A}\rangle,\ \mathit{ret}{:}\mathtt{ch}_{\mathrm{p}}\langle \mathtt{T}\rangle$$

where $\mathtt{T}$ is an arbitrary public type; Then one can check

$$\Gamma_e \vdash \mathsf{server}_i$$

where server_i is defined in the Introduction, provided the locally declared channels *data* and sync are declared at the types $\mathtt{ch}\langle \mathtt{T}, \mathtt{ch}_{\mathrm{p}}\langle \mathtt{T}\rangle, \mathtt{loc}_{\mathrm{p}}\rangle$ and $\mathtt{ch}\langle \mathtt{T}\rangle$ respectively. Now consider

$$\mathsf{serverBad} \Leftarrow \mathsf{server}_1 \mid l[\![a!\langle k_1\rangle]\!]$$

which attempts to export a confined location k_1, which could subsequently could be tested for failure by a public observer. Once more one can check that $\Gamma_e \nvdash \mathsf{serverBad}$.

Intuitively an observer is any system which only uses public names. Formally let $\mathbf{pub}(\Gamma)$ be the environment obtained by omitting from Γ any name not assigned a public type. Then $\mathbf{pub}(\Gamma) \vdash O$ ensures that O can only use public names. For example consider

$$\begin{aligned} \mathsf{observer} &\;\Leftarrow\; l[\![\mathit{req}!\langle v, \mathit{ret}\rangle]\!] \\ \mathsf{observerBad} &\;\Leftarrow\; l[\![\mathsf{go}\ k_1.\mathsf{go}\ l.\mathit{ok}!\langle\rangle]\!] \end{aligned}$$

Here one can check that $\mathbf{pub}(\Gamma_e) \vdash \mathsf{observer}$ and $\mathbf{pub}(\Gamma_e) \nvdash \mathsf{observerBad}$.

Our behavioural equivalence will in general relate arbitrary configurations; but we would expect equivalent configurations to have the same *public interface,* and be preserved by public observers.

Definition 1 (p-Contextual). *A relation over configurations is called p-Contextual if, whenever* $\Gamma \triangleright M \;\mathcal{R}\; \Gamma' \triangleright N$

- *(p-Interfaces:)* $\mathbf{pub}(\Gamma) = \mathbf{pub}(\Gamma')$
- *(Parallel:)* $\Gamma \triangleright M \mid O \;\mathcal{R}\; \Gamma' \triangleright N \mid O$ *and* $\Gamma \triangleright M \mid O \;\mathcal{R}\; \Gamma' \triangleright N \mid O$ *whenever* $\mathbf{pub}(\Gamma) \vdash O$
- *(Fresh extensions:)* $\Gamma, n : \mathtt{P} \triangleright M \;\mathcal{R}\; \Gamma', n : \mathtt{P} \triangleright N$ *whenever n is fresh*

Definition 2 (p-Barb). $\Gamma \triangleright N \Downarrow^{p}_{a@l}$ *denotes a* p-observable barb *by configuration* $\Gamma \triangleright N$*, on channel a at location l, defined as:*

$\exists N'.\ \Gamma \triangleright N \longrightarrow^{*} \Gamma \triangleright N'$ *such that* $N' \equiv (\nu\, \tilde{n} : \tilde{\mathtt{T}}) M | l[\![a!\langle V\rangle.Q]\!]$ *where* $\Gamma \vdash l : \mathtt{loc}^{\mathtt{a}}_{\mathtt{p}}, a : \mathtt{ch}_{\mathtt{p}}\langle\tilde{\mathtt{P}}\rangle$

Using this concept, we can now modify the standard definition of *reduction barbed equivalence*, [6]:

Definition 3 (Reduction barbed congruence). *Let* $\cong$ *be the largest relation between configurations which is p-contextual, reduction-closed (see [6]) and preserves p-barbs.*

3 Defining Fault Tolerance

Our first notion of *n*-fault-tolerance, formalising the intuitive (1), is when the faulting context induces at most *n* location failures, prior to the execution of the system; of course these failures must only be induced on locations which are not public. Formally for any set of location names $\tilde{l}$ let $F^{\tilde{l}}_{S}$ be the function which maps any configuration $\Gamma \triangleright N$ to $\Gamma - \tilde{l} \triangleright N$, where $\Gamma - \tilde{l}$ is the environment obtained from Γ by changing the status of every l_i to *dead*. We say $F^{\tilde{l}}_{S}$ is a *valid static n-fault context* with respect to Γ, if the size of $\tilde{l}$ is *at most n*, and for every $l_i \in \tilde{l}$, l_i is confined and alive ($\Gamma \vdash l_i : \mathtt{loc}^{\mathtt{a}}_{\mathtt{c}}$).

Definition 4 (Static Fault Tolerance). *A configuration* $\Gamma \triangleright N$ *is n-static fault tolerant if*

$$\Gamma \triangleright N \cong F^{\tilde{l}}_{S}(\Gamma) \triangleright N$$

for every static n-fault context $F^{\tilde{l}}_{S}$ *which is valid with respect to* Γ*.*

With this formal definition we can now examine the systems server_i, using the Γ_e defined above. We can easily check that $\Gamma \triangleright \mathsf{server}_1$ is not 1-fault tolerant, by considering the fault context $F^{k_1}_{S}$. Similarly we can show that $\Gamma_e \triangleright \mathsf{server}_2$ is not 2-fault tolerant, by considering F^{k_1,k_2}_{S}. But establishing positive results, for example that $\Gamma_e \triangleright \mathsf{server}_2$ is 1-fault tolerant, is difficult because the definition of $\cong$ quantifies over all valid observers. This point will be addressed in the next section, when we give a co-inductive characterisation of $\cong$.

Instead let us consider another manner of inducing faults. Let $l[\![\mathsf{kill}]\!]$ be a system which asynchronously kills a confined location *l*. Its operation is defined by the rule

$$\text{(r-kill)}\quad \frac{}{\Gamma \triangleright l[\![\mathsf{kill}]\!] \longrightarrow (\Gamma - l) \triangleright l[\![\mathbf{0}]\!]}$$

For any set of locations $\tilde{l}$ let $F^{\tilde{l}}_{D}$ denote the function which maps the system *M* to $M \mid l_1[\![\mathsf{kill}]\!] \mid \ldots \mid l_n[\![\mathsf{kill}]\!]$. It is said to be a valid dynamic *n*-fault context with respect to Γ if again the size of $\tilde{l}$ is at most *n* and $\Gamma \vdash l_i : \mathtt{loc}^{\mathtt{a}}_{\mathtt{c}}$, for every l_i in $\tilde{l}$.

Definition 5 (Dynamic Fault Tolerance). *A configuration $\Gamma \triangleright N$ is n-dynamic fault tolerant if*

$$\Gamma \triangleright F_D^{\tilde{l}}(M) \cong \Gamma \triangleright M$$

for every dynamic n-fault context which is valid with respect to Γ.

Example 2. The system sPassive defined below uses two identical replicas of the distributed database at k_1 and k_2, but treats the replica at k_1 as *primary* replica and the one at k_2 as a *secondary* (backup) replica - once again $\mathtt{W} = \mathsf{ch}\langle \mathtt{T}, \mathsf{ch_p}\langle \mathtt{T}\rangle, \mathtt{loc_p}\rangle$.

$$\mathsf{sPassive} \Leftarrow (\nu\, data : \mathtt{W}) \left(\begin{array}{l} l \left\llbracket \begin{array}{l} serv?(x,y).\mathsf{ping}\ k_1.\mathsf{go}\ k_1.data!\langle x,y,l\rangle \\ \qquad\qquad\qquad\qquad \mathsf{else\ go}\ k_2.data!\langle x,y,l\rangle \end{array} \right\rrbracket \\ \mid k_1\llbracket data?(x,y,z).\mathsf{go}\ z\ .y!\langle f(x)\rangle\rrbracket \\ \mid k_2\llbracket data?(x,y,z).\mathsf{go}\ z\ .y!\langle f(x)\rangle\rrbracket \end{array} \right)$$

The coordinating interface at l uses the ping construct to *detect failures* in the primary replica: if k_1 is alive, the request is sent to the primary replica and the secondary replica at k_2 is *not invoked*; if, on the other hand, the primary replica is dead, then the passive replica at k_2 is promoted to a primary replica and the request is sent to it. This implementation saves on *time redundancy* since, for any request, only one replica is invoked. Another passive replication server is sMonitor, defined as

$$\mathsf{sMonitor} \Leftarrow (\nu\, data : \mathtt{W}) \left(\begin{array}{l} l \left\llbracket serv?(x,y).(\nu\, sync : \mathsf{ch}\langle\rangle) \left(\begin{array}{l} \mathsf{go}\ k_1.data!\langle x, sync, l\rangle \\ \mid \mathsf{mntr}\ k_1.\mathsf{go}\ k_2.data!\langle x, sync, l\rangle \\ \mid sync?(z).y!\langle z\rangle \end{array} \right) \right\rrbracket \\ \mid k_1\llbracket data?(x,y,z).\mathsf{go}\ z\ .y!\langle f(x)\rangle\rrbracket \\ \mid k_2\llbracket data?(x,y,z).\mathsf{go}\ z\ .y!\langle f(x)\rangle\rrbracket \end{array} \right)$$

where again, $\mathtt{W} = \mathsf{ch}\langle \mathtt{T}, \mathsf{ch_p}\langle \mathtt{T}\rangle, \mathtt{loc_p}\rangle$. It uses a *monitor* process for failure detection

$$\mathsf{mntr}\ k.P \Leftarrow (\nu\, test : \mathsf{ch}\langle\rangle)(\, test!\langle\rangle \mid\, *\, test?().\mathsf{ping}\ k.\ test!\langle\rangle\ \mathsf{else}\ P\,)$$

instead of a *single* ping test on the primary replica at k_1; mntr $k.P$ *repeatedly* tests the status of the monitored location (k) and continues as P when k becomes dead. Similar to $\mathsf{server}_{2..3}$, sMonitor synchronises multiple answers from replicas with channel *sync*.

Using the techniques of the next section, one can show that both $\Gamma_e \triangleright \mathsf{sPassive}$ and $\Gamma \triangleright \mathsf{sMonitor}$, are 1-static fault tolerant, similar to server_2. However there is a difference between these two systems;if k_1 fails *after* sPassive tests for its status, then an answer will never reach l. Thus sPassive is *not* 1-dynamic fault tolerant; formally one can show $\Gamma_e \triangleright F_D^{k_1}(\mathsf{sPassive}) \ncong \Gamma_e \triangleright \mathsf{sPassive}$. But, as we will see in the next section, sMonitor can be shown to be 1-dynamic fault tolerant, just like $\mathsf{server}_{2..3}$.

4 Proof Techniques for Fault Tolerance

We define a labelled transition system (lts) for DπLoc$_e$, which consists of a collection of actions over (closed) configurations, $\Gamma \triangleright N \xrightarrow{\mu} \Gamma' \triangleright N'$, where μ can be an internal action, τ, a bound input, $(\tilde{n} : \tilde{\mathtt{T}})l : a?(V)$ or bound output, $(\tilde{n} : \tilde{\mathtt{T}})l : a!\langle V\rangle$. These actions

Table 5. *Operational Rules for Typed DπLoc*

Assuming $\Gamma \vdash l : \mathbf{alive}$

(l-in)

$$\frac{}{\Gamma \triangleright l\llbracket a?(X).P\rrbracket \xrightarrow{l:a?(V)} \Gamma \triangleright l\llbracket P\{{}^{V}\!/X\}\rrbracket}\ \Gamma \vdash_{\mathsf{obs}} l,\ \Gamma \vdash a : \mathtt{ch_p}\langle\tilde{\mathtt{W}}\rangle,\ V : \tilde{\mathtt{W}}$$

(l-fork)

$$\frac{}{\Gamma \triangleright l\llbracket P|Q\rrbracket \xrightarrow{\tau} \Gamma \triangleright l\llbracket P\rrbracket \mid l\llbracket Q\rrbracket}$$

(l-out)

$$\frac{}{\Gamma \triangleright l\llbracket a!\langle V\rangle.P\rrbracket \xrightarrow{l:a!\langle V\rangle} \Gamma \triangleright l\llbracket P\rrbracket}\ \Gamma \vdash_{\mathsf{obs}} l, a$$

(l-in-rep)

$$\frac{}{\Gamma \triangleright l\llbracket *a?(X).P\rrbracket \xrightarrow{\tau} \Gamma \triangleright l\llbracket a?(X).(P| * a?(Y).P\{{}^{Y}\!/X\})\rrbracket}$$

(l-eq)

$$\frac{}{\Gamma \triangleright l\llbracket \mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q\rrbracket \xrightarrow{\tau} \Gamma \triangleright l\llbracket P\rrbracket}$$

(l-neq)

$$\frac{}{\Gamma \triangleright l\llbracket \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\rrbracket \xrightarrow{\tau} \Gamma \triangleright l\llbracket Q\rrbracket}\ u \neq v$$

(l-new)

$$\frac{}{\Gamma \triangleright l\llbracket (\nu n : \mathtt{T})P\rrbracket \longrightarrow \Gamma \triangleright (\nu\, n : \mathtt{T})\ l\llbracket P\rrbracket}$$

(l-kill)

$$\frac{}{\Gamma \triangleright l\llbracket \mathsf{kill}\rrbracket \xrightarrow{\tau} (\Gamma - l) \triangleright l\llbracket \mathbf{0}\rrbracket}$$

(l-go)

$$\frac{}{\Gamma \triangleright l\llbracket \mathsf{go}\ k.P\rrbracket \xrightarrow{\tau} \Gamma \triangleright k\llbracket P\rrbracket}\ \Gamma \vdash k : \mathbf{alive}$$

(l-ngo)

$$\frac{}{\Gamma \triangleright l\llbracket \mathsf{go}\ k.P\rrbracket \xrightarrow{\tau} \Gamma \triangleright k\llbracket \mathbf{0}\rrbracket}\ \Gamma \nvdash k : \mathbf{alive}$$

(l-ping)

$$\frac{}{\Gamma \triangleright l\llbracket \mathsf{ping}\ k.P\ \mathsf{else}\ Q\rrbracket \xrightarrow{\tau} \Gamma \triangleright l\llbracket P\rrbracket}\ \Gamma \vdash k : \mathbf{alive}$$

(l-nping)

$$\frac{}{\Gamma \triangleright l\llbracket \mathsf{ping}\ k.P\ \mathsf{else}\ Q\rrbracket \xrightarrow{\tau} \Gamma \triangleright l\llbracket Q\rrbracket}\ \Gamma \nvdash k : \mathbf{alive}$$

(l-open)

$$\frac{\Gamma + n : \mathtt{T} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Gamma' \triangleright N'}{\Gamma \triangleright (\nu\, n : \mathtt{T})N \xrightarrow{(n:\mathtt{T},\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Gamma' \triangleright N'}\ l, a \neq n \in V$$

(l-weak)

$$\frac{\Gamma + n : \mathtt{T} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Gamma' \triangleright N'}{\Gamma \triangleright N \xrightarrow{(n:\mathtt{T},\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Gamma' \triangleright N'}\ l, a \neq n \in V$$

(l-rest)

$$\frac{\Gamma + n : \mathtt{T} \triangleright N \xrightarrow{\mu} \Gamma' + n : \mathtt{U} \triangleright N'}{\Gamma \triangleright (\nu\, n : \mathtt{T})N \xrightarrow{\mu} \Gamma' \triangleright (\nu\, n : \mathtt{U})N'}\ n \notin \mathbf{fn}(\mu)$$

(l-par-ctxt)

$$\frac{\Gamma \triangleright N \xrightarrow{\mu} \Gamma' \triangleright N'}{\begin{array}{c}\Gamma \triangleright N \mid M \xrightarrow{\mu} \Gamma' \triangleright N' \mid M \\ \Gamma \triangleright M \mid N \xrightarrow{\mu} \Gamma' \triangleright M \mid N'\end{array}}$$

(l-par-comm)

$$\frac{\uparrow(\Gamma) \triangleright N \xrightarrow{(\tilde{n}:\Uparrow(\tilde{\mathtt{T}}))l:a!\langle V\rangle} \Gamma' \triangleright N' \qquad \uparrow(\Gamma) \triangleright M \xrightarrow{(\tilde{n}:\Uparrow(\tilde{\mathtt{T}}))l:a?(V)} \Gamma'' \triangleright M'}{\begin{array}{c}\Gamma \triangleright N \mid M \xrightarrow{\tau} \Gamma \triangleright (\nu\, \tilde{n} : \tilde{\mathtt{T}})(N' \mid M') \\ \Gamma \triangleright M \mid N \xrightarrow{\tau} \Gamma \triangleright (\nu\, \tilde{n} : \tilde{\mathtt{T}})(M' \mid N')\end{array}}$$

are defined by transition rules given in Table 5, inspired by [7, 6, 5]. In accordance with Definition 2 (observable barbs) and Definitions 1 (valid observers), (l-in) and (l-out) restrict external communication to *public* channels at *public* locations ($\Gamma \vdash_{\mathsf{obs}} l, a$). Furthermore, in (l-in) we require that the type of the values inputted, V, match the object type of channel a; since a is public and configurations are well-typed, this also implies that V are public values defined in Γ. The restriction on output action, together with the assumption of well-typed configurations also means that, in (l-open), we only scope

Table 6. *β-Transition Rules for Typed DπLoc*

Assuming $\Gamma \vdash l : \textbf{alive}$

(b-in-rep)
$$\frac{}{\Gamma \triangleright l\llbracket * a?(X).P\rrbracket \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l\llbracket a?(X).(P| * a?(Y).P\{^Y\!/X\})\rrbracket}$$

(b-fork)
$$\frac{}{\Gamma \triangleright l\llbracket P|Q\rrbracket \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l\llbracket P\rrbracket \mid l\llbracket Q\rrbracket}$$

(b-eq)
$$\frac{}{\Gamma \triangleright l\llbracket \text{if } u = u \text{ then } P \text{ else } Q\rrbracket \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l\llbracket P\rrbracket}$$

(b-neq)
$$\frac{}{\Gamma \triangleright l\llbracket \text{if } u = v \text{ then } P \text{ else } Q\rrbracket \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l\llbracket Q\rrbracket}\; u \neq v$$

(b-ngo)
$$\frac{}{\Gamma \triangleright l\llbracket \text{go } k.P\rrbracket \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright k\llbracket \mathbf{0}\rrbracket}\; \Gamma \nvdash k : \textbf{alive}$$

(b-nping)
$$\frac{}{\Gamma \triangleright l\llbracket \text{ping } k.P \text{ else } Q\rrbracket \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l\llbracket Q\rrbracket}\; \Gamma \nvdash k : \textbf{alive}$$

(b-new)
$$\frac{}{\Gamma \triangleright l\llbracket (\nu\, n : \mathtt{T})P\rrbracket \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright (\nu\, n : \mathtt{T}) l\llbracket P\rrbracket}$$

(b-rest)
$$\frac{\Gamma, n : \mathtt{T} \triangleright N \stackrel{\tau}{\longmapsto}_\beta \Gamma', n : \mathtt{W} \triangleright N'}{\Gamma \triangleright (\nu\, n : \mathtt{T})N \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright (\nu\, n : \mathtt{W})N'}$$

(b-par)
$$\frac{\Gamma \triangleright N \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright N'}{\begin{array}{c}\Gamma \triangleright N|M \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright N'|M \\ \Gamma \triangleright M|N \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright M|N'\end{array}}$$

extrude public values. Contrary to [5], the lts does not allow external killing of locations (through the label kill : l) since public locations are reliable and never fail. Finally, the transition rule for internal communication, (l-par-comm), uses an overloaded function $\uparrow()$ for inferring input/output capabilities of the sub-systems: when applied to types, $\uparrow(\mathtt{T})$ transforms all the type tags to public ($\mathtt{p}$); when applied to environments, $\uparrow(\Gamma)$ changes all the types to public types in the same manner. All the remaining rules are a simplified version of the rules in [5].

Definition 6 (Weak bisimulation equivalence). *This is denoted as $\approx$, and is defined to be the largest typed relation over configurations such that if $\Gamma \triangleright M \approx \Gamma' \triangleright N$ then*

- $\Gamma \triangleright M \xrightarrow{\mu} \Gamma'' \triangleright M'$ *implies* $\Gamma' \triangleright N \stackrel{\hat{\mu}}{\Longrightarrow} \Gamma''' \triangleright N'$ *such that* $\Gamma'' \triangleright M' \approx \Gamma''' \triangleright N'$
- $\Gamma' \triangleright N \xrightarrow{\mu} \Gamma'' \triangleright N'$ *implies* $\Gamma \triangleright M \stackrel{\hat{\mu}}{\Longrightarrow} \Gamma'' \triangleright M'$ *such that* $\Gamma'' \triangleright M' \approx \Gamma''' \triangleright N'$

Theorem 1 (Full Abstraction). *For any DπLoc configurations $\Gamma \triangleright M$, $\Gamma' \triangleright N$:*

$$\Gamma \triangleright M \cong \Gamma' \triangleright N \;\; \textit{if and only if} \;\; \Gamma \triangleright M \approx \Gamma' \triangleright N$$

Theorem 1 allows us to prove *positive* fault tolerance results by giving a bisimulation for every reduction barbed congruent pair required by Definitions 4 and 5. We next develop up-to bisimulation techniques that can relieve some of the burden of exhibiting the required bisimulations. We identify a number of τ actions, which we refer to as β-actions or β-moves, inspired by the work in [3]. These are denoted as $\Gamma \triangleright N \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright N$ and are defined in Table 6. With these β-moves we develop up-to bisimulation techniques, by showing that our witness bisimulations can abstract away from matching configurations that denote β-moves. Our details are more complicated than in [3] because we deal with failure: apart from local rules ((b-eq) and (b-fork)) and context rules ((b-rest) and (b-par)), Table 6 includes rules dealing with failed locations such as (b-ngo)

Table 7. *β-Equivalence Rules for Typed DπLoc*

(bs-comm)	$\Gamma \models N \vert M \equiv_f M \vert N$	
(bs-assoc)	$\Gamma \models (N \vert M) \vert M' \equiv_f N \vert (M \vert M')$	
(bs-unit)	$\Gamma \models N \vert l[\![\mathbf{0}]\!] \equiv_f N$	
(bs-extr)	$\Gamma \models (\nu\, n : \mathtt{T})(N \vert M) \equiv_f N \vert (\nu\, n : \mathtt{T}) M$	$n \notin \mathbf{fn}(N)$
(bs-flip)	$\Gamma \models (\nu\, n : \mathtt{T})(\nu\, m : \mathtt{U}) N \equiv_f (\nu\, m : \mathtt{U})(\nu\, n : \mathtt{T}) N$	
(bs-inact)	$\Gamma \models (\nu\, n : \mathtt{T}) N \equiv_f N$	$n \notin \mathbf{fn}(N)$
(bs-dead)	$\Gamma \models l[\![P]\!] \equiv_f l[\![Q]\!]$	$\Gamma \nvdash l : \mathbf{alive}$

and (b-nping). To obtain the required results for β-moves with failure, we define a new structural equivalence ranging over *configurations*, denoted as $\equiv_f$ and defined by the rules in Table 7, which takes into consideration *location status* as well. This enables us to obtain confluence for β-moves with respect to actions that change the status of locations. The only rule worth highlighting is (bs-dead), which allows us to ignore dead code.

Lemma 1 (Confluence of β-moves). $\overset{\tau}{\longmapsto}_{\beta}$ *observes the diamond property:*

$$\begin{tikzcd}
\Gamma \triangleright N \arrow[r, mapsto, "\tau", "\beta"'] \arrow[d, "\mu"'] & \Gamma \triangleright M & \text{implies} & \Gamma \triangleright N \arrow[r, mapsto, "\tau", "\beta"'] \arrow[d, "\mu"'] & \Gamma \triangleright M \arrow[d, "\mu"'] & \text{or } \mu = \tau \text{ and } \Gamma \triangleright M = \Gamma' \triangleright N' \\
\Gamma' \triangleright N' & \Gamma' \triangleright M' & & \Gamma' \triangleright N' \arrow[r, Rightarrow, "\tau", "\beta"'] & \equiv_f \Gamma' \triangleright M' &
\end{tikzcd}$$

Proof. The proof proceeds by case analysis of the different types of μ and then by induction on the derivation of the β-move.

Proposition 1. *Suppose $\Gamma \triangleright N \mathrel{|}\!\!\Longrightarrow_{\beta} \Gamma' \triangleright M$. Then $\Gamma \triangleright N \approx \Gamma' \triangleright M$.*

Proof. We prove the above statement by defining $\mathcal{R} = \left\{ \Gamma \triangleright N\ , \Gamma' \triangleright M\ \ \vert \Gamma \triangleright N \mathrel{|}\!\!\Longrightarrow_{\beta} \Gamma' \triangleright M \right\}$ and showing that $\mathcal{R}$ is a bisimulation, which follows as a consequence of Lemma 1.

Definition 7 (Bisimulation up-to β-moves). *Bisimulation up-to β-moves, denoted as $\approx_\beta$, is the largest typed relation between configurations such that $\Gamma_1 \triangleright M_1 \approx_\beta \Gamma_2 \triangleright M_2$ and*

- $\Gamma_1 \triangleright M_1 \xrightarrow{\mu} \Gamma_1' \triangleright M_1'$ *implies* $\Gamma_2 \triangleright M_2 \overset{\hat{\mu}}{\Longrightarrow} \Gamma_2' \triangleright M_2'$ *such that* $\Gamma_1' \triangleright M_1' \mathcal{A}_l \circ \approx_\beta \circ \approx \Gamma_2' \triangleright M_2'$
- $\Gamma_2 \triangleright M_2 \xrightarrow{\mu} \Gamma_2' \triangleright M_2'$ *implies* $\Gamma_1 \triangleright M_1 \overset{\hat{\mu}}{\Longrightarrow} \Gamma_1' \triangleright M_1'$ *such that* $\Gamma_2' \triangleright M_2' \mathcal{A}_l \circ \approx_\beta \circ \approx \Gamma_1' \triangleright M_1'$

where $\mathcal{A}_l$ is the relation $\mathrel{|}\!\!\Longrightarrow_{\beta} \circ \equiv$.

Proposition 1 provides us with a powerful method for approximating bisimulations. In the approximate bisimulation $\approx_\beta$, an action $\Gamma_1 \triangleright M_1 \xrightarrow{\mu} \Gamma_1' \triangleright M_1'$ can be matched by a β-derivative of $\Gamma_1' \triangleright M_1'$, that is $\Gamma_1' \triangleright M_1' \mathrel{|}\!\!\Longrightarrow_{\beta} \Gamma_1' \triangleright M_1''$, and a weak matching action $\Gamma_2 \triangleright M_2 \overset{\hat{\mu}}{\Longrightarrow} \Gamma_2' \triangleright M_2'$ such that, up to structural equivalence on the one side and up-to bisimilarity on the other, the pairs $\Gamma_1' \triangleright M_1''$ and $\Gamma_2' \triangleright M_2'$ are once more related. Intuitively then, in any relation satisfying $\approx_\beta$, a configuration can represent all the configurations to which it can evolve using β-moves. We justify the use of $\approx_\beta$ by proving Proposition 2.

Proposition 2 (Inclusion of bisimulation up-to β-moves). *If $\Gamma_1 \triangleright M_1 \approx_\beta \Gamma_2 \triangleright M_2$ then $\Gamma_1 \triangleright M_1 \approx \Gamma_2 \triangleright M_2$*

Proof. We prove the above proposition by defining the relation $\mathcal{R}$ as

$$\mathcal{R} = \Big\{ \Gamma_1 \triangleright M_1 \ , \Gamma_2 \triangleright M_2 \quad \Big| \Gamma_1 \triangleright M_1 \approx \circ \approx_\beta \circ \approx \Gamma_2 \triangleright M_2 \Big\}$$

and show that $\mathcal{R} \subseteq \approx$. The required result can then be extracted from this result by considering the special cases where the $\approx$ on either side are the identity relations.

Example 3. We are now in a position to prove positive fault tolerance result. For instance to show that $\Gamma \triangleright \mathsf{sPassive}$ is 1-static fault tolerant we just need to provide 3 witness bisimulations up-to β-moves to prove

$$\textstyle\prod_{i=1}^{3} \ \Gamma \triangleright \mathsf{sPassive} \ \cong \ (\Gamma - k_i) \triangleright \mathsf{sPassive}$$

We here give the witness relation for the most involving case (where $i = 1$), and leave the simpler relations for the interested reader. Thus, the witness relation is $\mathcal{R}$ defined as

$$\mathcal{R} \stackrel{\text{def}}{=} \{\langle \Gamma \triangleright \mathsf{sPassive}, \Gamma - k_1 \triangleright \mathsf{sPassive} \rangle\} \ \cup \ \left(\bigcup_{u,v \in \textsc{Names}} \mathcal{R}'(u,v) \right)$$

$$\mathcal{R}'(u,v) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \Gamma \triangleright (\nu d) l\llbracket \mathsf{Png}(u,v) \rrbracket \mid \mathsf{R}_1 \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) l\llbracket \mathsf{Png}(u,v) \rrbracket \mid \mathsf{R}_1 \mid \mathsf{R}_2 \\ \Gamma \triangleright (\nu d) l\llbracket \mathsf{Q}_1(u,v) \rrbracket \mid \mathsf{R}_1 \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) l\llbracket \mathsf{Q}_2(u,v) \rrbracket \mid \mathsf{R}_1 \mid \mathsf{R}_2 \\ \Gamma \triangleright (\nu d) k_1\llbracket d!\langle u,v,l \rangle \rrbracket \mid \mathsf{R}_1 \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) k_2\llbracket d!\langle u,v,l \rangle \rrbracket \mid \mathsf{R}_1 \mid \mathsf{R}_2 \\ \Gamma \triangleright (\nu d) k_1\llbracket \mathsf{go}\ l.v!\langle f(u) \rangle \rrbracket \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) \mathsf{R}_1 \mid k_2\llbracket \mathsf{go}\ l.v!\langle f(u) \rangle \rrbracket \\ \Gamma \triangleright (\nu d) l\llbracket v!\langle f(u) \rangle \rrbracket \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) \mathsf{R}_1 \mid l\llbracket v!\langle f(u) \rangle \rrbracket \\ \Gamma \triangleright (\nu d) \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) \mathsf{R}_1 \end{array} \right\}$$

where d stands for *data* and

$$\begin{aligned} \mathsf{Png}(x,y) &\Leftarrow \mathsf{ping}\ k_1.\mathsf{Q}_1(x,y)\ \mathsf{else}\ \mathsf{Q}_2(x,y) \\ \mathsf{Q}_i(x,y) &\Leftarrow \mathsf{go}\ k_i.d!\langle x,y,l \rangle \\ \mathsf{R}_i &\Leftarrow k_i\llbracket d?(x,y,z).\mathsf{go}\ z\ .y!\langle f(x) \rangle \rrbracket \end{aligned}$$

5 Generic Techniques for Dynamic Fault Tolerance

Despite the fault tolerance proof techniques developed in Section 4, proving positive fault tolerance results entails a lot of unnecessary repeated work because Definition 4 and Definition 5 quantify over all valid fault contexts: to prove that server_3 is 2-dynamic fault tolerant, we need to provide 6 relations, one for every different case in

$$\textstyle\prod_{i \neq j=1}^{3} \ \Gamma \triangleright \mathsf{server}_3 \ \cong \ \Gamma \triangleright \mathsf{server}_3 | k_i\llbracket \mathsf{kill} \rrbracket | k_j\llbracket \mathsf{kill} \rrbracket$$

A closer inspection of the required relations reveals that there is a lot of overlap between them: these overlapping states would be automatically circumvented if we require a single relation that is somewhat the merging of all of these separate relations. Hence we reformulate our fault tolerance definition for dynamic fault tolerance (the most demanding), to reflect such a merging of relations; a similar definition for the static case should not be more difficult to construct. The new definition is based on the actions given earlier in Section 4 together with a new action, fail, defined as

$$\text{(l-fail)} \quad \frac{}{\Gamma \triangleright N \xrightarrow{\mathsf{fail}} (\Gamma - l) \triangleright N}\ \Gamma \vdash l : \mathtt{loc}^{\mathsf{a}}_{\mathsf{c}}$$

permitting external killing of confined locations. Intuitively, this action allow us to *count* the number of failures, but prohibits us from determining which specific location failed.[3] The asymmetric relation $\preceq^n_D$, defined below, is parameterised with an integer n, denoting the number of confined locations that can still be killed on the right hand side: the additional third clause states that a fail move on the right hand side may be matched by a weak τ-move on the left hand side and the two residuals need to be related in $\preceq^{n-1}_D$.

Definition 8 (Dynamic Fault Tolerance Simulation). *Dynamic n-fault tolerant* simulation, *denoted $\preceq^n_D$, is the largest* asymmetric *typed relation over configurations such that whenever $\Gamma_1 \triangleright M_1 \preceq^n_D \Gamma_2 \triangleright M_2$,*

- $\Gamma_1 \triangleright M_1 \xrightarrow{\gamma} \Gamma_1' \triangleright M_1'$ *implies* $\Gamma_2 \triangleright M_2 \overset{\widehat{\gamma}}{\Longrightarrow} \Gamma_2' \triangleright M_2'$ *such that* $\Gamma_1' \triangleright M_1' \preceq^n_D \Gamma_2' \triangleright M_2'$
- $\Gamma_2 \triangleright M_2 \xrightarrow{\gamma} \Gamma_2' \triangleright M_2'$ *implies* $\Gamma_1 \triangleright M_1 \overset{\widehat{\gamma}}{\Longrightarrow} \Gamma_1' \triangleright M_1'$ *such that* $\Gamma_1' \triangleright N_1' \preceq^n_D \Gamma_2' \triangleright M_2'$
- *if* $n > 0$, $\Gamma_2 \triangleright M_2 \xrightarrow{\mathit{fail}} \Gamma_2' \triangleright M_2'$ *implies* $\Gamma_1 \triangleright M_1 \Longrightarrow \Gamma_1' \triangleright M_1'$ *such that* $\Gamma_1' \triangleright M_1' \preceq^{n-1}_D \Gamma_2' \triangleright M_2'$

Before we can use Definition 8 to prove dynamic fault tolerance, we need to show that the new definition is sound with respect to Definition 5.

Proposition 3 (Soundness of $\preceq^n_D$). *If $\Gamma \models M_1 \ \preceq^n_D \ M_2$ then for any dynamic n-fault context $F^{\tilde{l}}_D$ that is valid with respect to Γ we have $\Gamma \models M_1 \ \cong \ F^{\tilde{l}}_D(M_2)$*

Proof. Let $\mathcal{R}_n$ be a relation parameterised by a number n and defined as

$$\mathcal{R}_n \stackrel{\text{def}}{=} \left\{ \Gamma_1 \triangleright M_1 \ , \Gamma_2 \triangleright M_2 \,|\, F^i_D \quad \middle| \Gamma_1 \triangleright M_1 \preceq^i_D \Gamma_2 \triangleright M_2,\ \prod_{j=1}^{2} \Gamma_j \vdash F^i_D \text{ and } 0 \leq i \leq n \right\}$$

By showing $\mathcal{R}_n \subseteq\approx$ we prove that $\preceq^n_D$ is sound with respect to n-dynamic fault tolerance

It would be ideal if we could reuse up-to techniques and give relations satisfying $\preceq^n_D$ that abstract away from β-moves. Similar to Section 4, we define a fault tolerance simulation up-to β-moves and show that this is sound with respect to $\preceq^n_D$. This definition uses a weak bisimulation (Definition 6) that ranges over α actions, that is μ *and* the new action fail. We refer to this bisimulation as a *counting* bisimulation over configurations, denoted as $\approx_{cnt}$, because it allows us to count failing confined locations on each side and match subsequent observable behaviour.

Definition 9 (Fault Tolerant Simulation up-to β-moves). *An n-fault tolerant simulation up-to β-moves, denoted as $\preceq^n_\beta$, is the largest typed relation $\mathcal{R}$ between configurations parameterised by the number n, such that whenever we have $\Gamma_1 \triangleright M_1 \preceq^n_\beta \Gamma_2 \triangleright M_2$*

[3] This point differs from [5], where labels for external killing carried the location name, kill:l.

- $\Gamma_1 \triangleright M_1 \xrightarrow{\mu} \Gamma_1' \triangleright M_1'$ *implies* $\Gamma_2 \triangleright M_2 \overset{\hat{\mu}}{\Longrightarrow} \Gamma_2' \triangleright M_2'$ *such that* $\Gamma_1' \triangleright M_1' \mathcal{A}_l \circ \leq_\beta^n \circ \approx_{cnt} \Gamma_2' \triangleright M_2'$
- $\Gamma_2 \triangleright M_2 \xrightarrow{\mu} \Gamma_2' \triangleright M_2'$ *implies* $\Gamma_1 \triangleright M_1 \overset{\hat{\mu}}{\Longrightarrow} \Gamma_1' \triangleright M_1'$ *such that* $\Gamma_2' \triangleright M_2' \mathcal{A}_l \circ \leq_\beta^n \circ \approx \Gamma_1' \triangleright M_1'$
- *If* $n > 0$ *then* $\Gamma_2 \triangleright M_2 \xrightarrow{\mathsf{fail}} \Gamma_2' \triangleright M_2'$ *implies* $\Gamma_1 \triangleright M_1 \Longrightarrow \Gamma_1' \triangleright M_1'$ *such that* $\Gamma_2' \triangleright M_2' \leq_\beta^{n-1} \circ \approx \Gamma_1' \triangleright M_1'$

where $\mathcal{A}_l$ *is the relation* $\models\!\!\!\Longrightarrow_\beta \circ \equiv$. *We highlight the use of* $\approx_{cnt}$ *for matching configurations in the first clause.*

The work required to show that $\leq_\beta^n$ is sound with respect to $\leq_D^n$ is similar to earlier up-to β-moves work discussed in Section 4: we have to show that β-move confluence (similar to Lemma 1) is also preserved for the new action fail; we also have to show that after a β-move, the redex and reduct configurations are counting-bisimilar (similar to Proposition 1). Finally we prove the following proposition

Proposition 4 (Inclusion of fault tolerant simulation up-to β-moves).
If $\Gamma_1 \triangleright M_1 \leq_\beta^n \Gamma_2 \triangleright M_2$ *then* $\Gamma_1 \triangleright M_1 \leq_D^n \Gamma_2 \triangleright M_2$

Proof. We prove the above proposition by defining the relation $\mathcal{R}_n$ as

$$\mathcal{R}_n = \left\{ \Gamma_1 \triangleright M_1 \,, \Gamma_2 \triangleright M_2 \quad \middle| \Gamma_1 \triangleright M_1 \approx \circ \leq_\beta^i \circ \approx_{cnt} \Gamma_2 \triangleright M_2 \text{ and } 0 \leq i \leq n \right\}$$

and show that $\mathcal{R}_n \subseteq \leq_D^n$. The required result can then be extracted from this result by considering the special cases where $\approx$ and $\approx_{cnt}$ on either side are the identity relations.

Example 4. The results of Proposition 3 and Proposition 4 allow us to prove that the configuration $\Gamma \triangleright \mathsf{server}_2$ is 1-dynamically fault tolerant by providing a *single* witness fault tolerance simulation up-to β-moves showing that $\Gamma \triangleright \mathsf{server}_2 \leq_\beta^1 \Gamma \triangleright \mathsf{server}_2$ Due to lack of space, we relegate the presentation of this relation to the full paper [4].

6 Conclusions and Related Work

We adopted a subset of [5] and developed a theory for system fault tolerance in the presence of fail-stop node failure. We formalised two definitions for fault tolerance based on the well studied concept of observational equivalence. Subsequently, we developed various sound proof techniques with respect to these definitions.

Future Work The immediate next step is to apply the theory to a wider spectrum of examples, namely using replicas with state and fault tolerance techniques such as lazy replication: we postulate that the existing theory should suffice. Another avenue worth considering is extending the theory to deal with link failure and the interplay between node and link failure [5]. In the long run, we plan to develop of a compositional theory of fault tolerance, enabling the construction of fault tolerant systems from smaller component sub-systems. For both cases, this paper should provide a good starting point.

Related Work To the best of our knowledge, Prasad's thesis [9] is the closest work to ours, addressing fault tolerance for process calculi. Even though similar concepts such as redundancy (called ”duplication”) and failure-free execution are identified, the setting and development of Prasad differs considerably form ours. In essence, three new operators (”displace”, ”audit” and ”checkpoint”) are introduced in a CCS variant; equational laws for terms using these operators are then developed so that algebraic manipulation can be used to show that terms in this calculus are, in some sense, fault tolerant with respect to their specification.

References

1. Roberto M. Amadio and Sanjiva Prasad. Localities and failures. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 14, 1994.
2. Flavin Christian. Understanding fault tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.
3. Alberto Ciaffaglione, Matthew Hennessy, and Julian Rathke. Proof methodologies for behavioural equivalence in Dπ. Technical Report 03/2005, University of Sussex, 2005.
4. Adrian Francalanza and Matthew Hennessy. A theory for observational fault tolerance. `www.cs.um.edu.mt/˜ afran/`.
5. Adrian Francalanza and Matthew Hennessy. A theory of system behaviour in the presence of node and link failures. In *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 368–382. Springer, 2005.
6. Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 322:615–669, 2004.
7. Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14:651–684, 2004.
8. Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.
9. K. V. S. Prasad. *Combinators and Bisimulation Proofs for Restartable Systems*. PhD thesis, Department of Computer Science, University of Edinburgh, December 1987.
10. James Riely and Matthew Hennessy. Distributed processes and location failures. *Theoretical Computer Science*, 226:693–735, 2001.
11. Davide Sangiorgi and David Walker. *The π-calculus*. Cambridge University Press, 2001.
12. Richard D. Schlichting and Fred B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *Computer Systems*, 1(3):222–238, 1983.
13. Paulo Verissimo and Luis Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, 2001.