Finally the Weakest Failure Detector for Non-Blocking Atomic Commit

Rachid Guerraoui Petr Kouznetsov

Distributed Programming Laboratory
EPFL

December 3, 2003

Abstract

Recent papers [7, 9] define the weakest failure detector for solving the Non-Blocking Atomic Commit problem (NBAC) in a message passing system where processes can fail by crashing and a majority of processes never crash.

In this paper, we generalize the result by presenting the weakest failure detector to solve NBAC in *any* environment, i.e., without any assumption on the number of processes that can crash. We present the result in a modular manner through determining first the weakest failure detector for *quittable consensus*, a variant of consensus introduced in [9].

1 Introduction

Problem. The *Non-Blocking Atomic Commit (NBAC)* [5, 8, 11] problem captures the notion of distributed transaction termination. The problem consists for a set of processes to reach a common *decision*, *commit* or *abort*, according to some initial votes of the processes, *yes* or *no*, such that the following properties are satisfied:

- *Agreement:* no two processes decide differently;
- *Termination:* every correct process eventually decides;
- *C-Validity: abort* can only be decided if some process votes *no* or crashes;
- *A-Validity: commit* can only be decided if all processes vote *yes*.

For brevity, we denote *yes* and *commit* by 1, while *no* and *abort* by 0.

In this note, we determine the weakest failure detector for solving NBAC in a *distributed crash-stop asynchronous message-passing* model. Informally, the model is one in which processes exchange messages through reliable communication channels, processes can fail by crashing, and there are no bounds on message transmission delay and relative processor speeds. Processes have access to failure detector modules that provide them with hints about failures.

The result of this note is a strict generalization of recently obtained results [7, 9] on the weakest failure detector for NBAC asuming an environment *with a majority of correct processes*.

Background. In [6], failure detector $?\mathcal{P}$, called the *anonymously perfect* failure detector, was introduced and shown to be *necessary* for solving NBAC. $?\mathcal{P}$ outputs 0 at every process as long as there are no failures. If a failure occurs, then $?\mathcal{P}$ eventually outputs 1 at every process. In other

words, $?\mathcal{P}$ correctly detects that *some* process has crashed, but does not tell *which* process has actually crashed.

In [9], a convenient modularization of NBAC was proposed, based on a variant of consensus, called *quittable consensus (QC)*. Solving quittable consensus consists for a set of processes to choose a common value in $\{0, 1, Q\}$, based on their input values in $\{0, 1\}$, so that the following properties hold:

- *Agreement:* no two processes decide differently;
- *Termination:* every correct process eventually decides;
- *QC-Validity:*
  (a) Q is decided only if a failure occurs;[1]
  (b) A value $v \in \{0, 1\}$ is decided only if some process proposes v.

It was shown in [9] that NBAC is equivalent to QC modulo $?\mathcal{P}$. Further, the weakest failure detector, denoted here by Ψ, to solve QC with a majority of correct processes was defined in [9]. Combined with [6], this implies that $?\mathcal{P} + \Psi$ is the weakest failure detector to solve NBAC with a majority of correct processes. (This result was concurrently obtained through directly attacking the NBAC problem [7].)

Failure detector Ψ [3] behaves as follows. Initially, Ψ outputs $\perp$ at every process. Eventually, Ψ either behaves like Ω [1] at all processes, or, only if a failure occurs, outputs a predefined value $\mathbf{F}$ at all processes. When $\mathbf{F}$ is output at every process, we say that *a crash is detected*.

We address the question of the weakest failure detector for solving NBAC in any environment using the result of [3] on the weakest failure detector for solving *consensus* in any environment. This failure detector was denoted by $\Omega + \Sigma$ where, roughly speaking, the liveness part of consensus is provided by Ω [1], and the safety part of consensus is provided the *quorum* failure detector Σ [3]. At every process and at each time, Σ outputs a set of processes, called *quorum*, so that (i) eventually, quorum contains only correct processes, and (ii) every two quorums intersect.

Contribution. In this note, we determine the weakest failure detector to solve QC in *any* environment, i.e., without any assumptions on the number of processes that can fail. In particular, we show that any failure detector that solves quittable consensus can, in some clearly defined scenarios, be used to implement both Ω [1] and Σ [3]. In all other scenarios, the processes uniformly detect a failure by outputting $\mathbf{F}$. As a result, we obtain the weakest failure detector for solving QC in any environment. This failure detector, denoted by $\mathcal{X}$, is similar to Ψ but, $\mathcal{X}$ behaves like $\Omega + \Sigma$ in cases when no crash is detected ($\mathbf{F}$ is not output). As a corollary, we show that $?\mathcal{P} + \mathcal{X}$ is the weakest failure detector for solving NBAC.

2 Model

We consider in this note a crash-prone asynchronous message passing model augmented with the failure detector abstraction. We recall here what in the model is needed to state and prove our results. More details on the model can be found in [1, 2].

[1] In [9], QC has a slightly stronger definition in which Q can only be decided if a failure *previously* occurred. Accordingly, in the version of NBAC considered in [9], *abort* can be decided only because of *previously* occurred failure. In this note, we consider the original definition of NBAC given in [8]. The distinction between these definitions is not important here, since it does not affect the difficulty of determining the weakest failure detector.

System. We assume the existence of a global clock to simplify the presentation. The processes do not have *direct* access to the clock (timing assumptions are captured within failure detectors). We take the range $\mathcal{T}$ of the clock output values to be the set of natural numbers and the integer 0, ($\{0\} \cup \mathbb{N}$). The system consists of a set of n processes $\Pi = \{p_1, .., p_n\} (n > 1)$. Every pair of processes is connected by a reliable communication channel. The system is *asynchronous*: there is no time bound on message delay, clock drift, or the time necessary to execute a step [4].

Failures and failure patterns. Processes are subject to *crash* failures. A *failure pattern* F is a function from the global time range $\mathcal{T}$ to 2^{Π}, where $F(t)$ denotes the set of processes that have crashed by time t. Once a process crashes, it does not recover, i.e., $\forall t < t' : F(t) \subseteq F(t')$. We define $correct(F) = \Pi - \cup_{t \in \mathcal{T}} F(t)$ to be the set of *correct* processes. A process $p_i \notin F(t)$ is said to be *up* at time t. A process $p_i \in F(t)$ is said to be *crashed* (or *incorrect*) at time t. An *environment* $\mathcal{E}$ is a set of failure patterns. We denote by F_0 the failure-free failure pattern ($correct(F_0) = \Pi$).

Failure detectors. A *failure detector history H with range $\mathcal{R}$* is a function from $\Pi \times \mathcal{T}$ to $\mathcal{R}$. $H(p_i, t)$ is the output of the failure detector module of process p_i at time t. A *failure detector* $\mathcal{D}$ is a function that maps each failure pattern F to a set of failure detector histories $\mathcal{D}(F)$ with range $\mathcal{R}_{\mathcal{D}}$.

Every process p_i has a failure detector module $\mathcal{D}_i$ that p_i queries to obtain information about the failures in the system. We do not make any assumption a priori on the range of a failure detector. Among the failure detectors defined in the literature the following ones are of particular interest in this paper

Eventual leader Ω [1]: the output of each failure detector module Ω_i is a single process p_j, that p_i currently *trusts*, i.e., that p_i considers to be correct ($\mathcal{R}_{\Omega} = \Pi$). For every failure pattern, there is a time after which all correct processes always trust the same correct process.

Anonymously perfect failure detector $?\mathcal{P}$ [6]: the output of each module $?\mathcal{P}_i$ is either 0 or 1. When the failure detector module of $?\mathcal{P}$ at a process p_i outputs 1, we say that p_i *detects a crash*. $?\mathcal{P}$ satisfies the following properties: (*anonymous completeness*) If some process crashes, then there is a time after which every correct process permanently detects a crash; and (*anonymous accuracy*) No crash is detected unless some process crashes.

Quorum failure detector Σ [3]: the output of each failure detector module Σ_i at time t is a subset of processes $Q(i, t)$ so that (*quorum completeness*) Eventually, $Q(i, t)$ includes only correct processes; (*quorum accuracy*) For all p_i, p_j in Π and t, t' in $\mathcal{T}$, $Q(i, t) \cap Q(j, t') \neq \emptyset$.

Algorithms. We model the set of asynchronous communication channels as a message buffer which contains messages not yet received by their destinations. An algorithm A is a collection of n (possibly infinite state) deterministic automata, one for each of the processes. $A(p_i)$ denotes the automaton running on process p_i. In each step of A, process p_i performs atomically the following three actions: (receive phase) p_i chooses *non-deterministically* a single message addressed to p_i from the message buffer, or a null message, denoted λ; (failure detector query phase) p_i queries and receives a value from its failure detector module; (local state update phase) p_i changes its state; and (send phase) sends a message to all processes according to the automaton $A(p_i)$, based on its state at the beginning of the step, the message received in the receive phase, and the value obtained by p_i from its failure detector module.[2]

[2]The necessary part of our result also applies to weaker models where a step can atomically comprise at most one phase and where a process can atomically send at most one message to a single process per step. The sufficient part also holds in these models, since the algorithm of Section 3 is still correct there.

Configurations, schedules and runs. A *configuration* defines the current state of each process in the system and the set of messages currently in the message buffer. Initially, the message buffer is empty. A step (p_i, m, d, A) of an algorithm A is uniquely determined by the identity of the process p_i that takes the step, the message m received by p_i during the step (m might be the null message λ), and the failure detector value d seen by p_i during the step. We say that a step $e = (p_i, m, d, A)$ *is applicable* to the current configuration if and only if $m = \lambda$ or m is a message from the current message buffer destined to p_i. $e(C)$ denotes the unique configuration that results when e is applied to C. A *schedule* S of algorithm A is a (finite or infinite) sequence of steps of A. $S_\perp$ denotes the empty schedule. We say that a schedule S *is applicable to a configuration* C if and only if (a) $S = S_\perp$, or (b) $S[1]$ is applicable to C, $S[2]$ is applicable to $S[1](C)$, etc. For a finite schedule S applicable to C, $S(C)$ denotes the unique configuration that results from applying S to C.

A *partial run of algorithm A in an environment $\mathcal{E}$ using a failure detector $\mathcal{D}$* is a tuple $R = \langle F, H_\mathcal{D}, I, S, T\rangle$, where $F \in \mathcal{E}$ is a failure pattern, $H_\mathcal{D} \in \mathcal{D}(F)$ is a failure detector history, I is an initial configuration of A, S is a *finite* schedule of A, and $T \subset \mathcal{T}$ is a *finite* list of increasing time values, such that $|S| = |T|$, S is applicable to I, and for all $t \leq |S|$, if $S[t]$ is of the form (p_i, m, d, A) then: (1) p_i has not crashed by time $T[t]$, i.e., $p_i \notin F(T[t])$ and (2) d is the value of the failure detector module of p_i at time $T[t]$, i.e., $d = H_\mathcal{D}(p_i, T[t])$.

A *run of algorithm A in an environment $\mathcal{E}$ using a failure detector $\mathcal{D}$* is a tuple $R = \langle F, H_\mathcal{D}, I, S, T\rangle$, where S is an *infinite* schedule of A and $T \subseteq \mathcal{T}$ is an *infinite* list of increasing time values indicating when each step of S occurred. In addition to satisfying the properties (1) and (2) of a partial run, run R should guarantee that (3) every correct process in F takes an infinite number of steps in S and eventually receives every message sent to it (this conveys the reliability of the communication channels).

Weakest failure detector. A *problem* (e.g., NBAC, QC or consensus) is a set of runs (usually defined by a set of properties that these runs should satisfy). We say that a failure detector $\mathcal{D}$ *solves a problem M in an environment $\mathcal{E}$* if there is an algorithm A, such that all the runs of A in $\mathcal{E}$ using $\mathcal{D}$ are in M (i.e., they satisfy the properties of M).

Let $\mathcal{D}$ and $\mathcal{D}'$ be any two failure detectors and $\mathcal{E}$ be any environment. If there is an algorithm $T_{\mathcal{D}'\rightarrow\mathcal{D}}$ that emulates $\mathcal{D}$ with $\mathcal{D}'$ in $\mathcal{E}$ ($T_{\mathcal{D}'\rightarrow\mathcal{D}}$ is called a *reduction* algorithm), we say that $\mathcal{D}$ *is weaker than $\mathcal{D}'$ in $\mathcal{E}$*, or $\mathcal{D} \preceq_\mathcal{E} \mathcal{D}'$. If $\mathcal{D} \preceq_\mathcal{E} \mathcal{D}'$ but $\mathcal{D}' \not\preceq_\mathcal{E} \mathcal{D}$, we say that $\mathcal{D}$ *is strictly weaker than $\mathcal{D}'$ in $\mathcal{E}$*, or $\mathcal{D} \prec_\mathcal{E} \mathcal{D}'$.[3] Note that $T_{\mathcal{D}'\rightarrow\mathcal{D}}$ does not need to emulate *all* histories of $\mathcal{D}$; it is required that all the failure detector histories it emulates be histories of $\mathcal{D}$.

We say that a failure detector $\mathcal{D}$ is *the weakest failure detector to solve a problem M in an environment $\mathcal{E}$* if two conditions are satisfied: (1) Sufficiency: $\mathcal{D}$ solves M in $\mathcal{E}$, and (2) Necessity: if a failure detector $\mathcal{D}'$ solves M in $\mathcal{E}$ then $\mathcal{D} \preceq_\mathcal{E} \mathcal{D}'$.

3 The weakest failure detector for solving QC

Candidate. Our candidate to be the weakest failure detector to solve QC in any environment, denoted by $\mathcal{X}$, behaves as follows. For any failure pattern F, $\mathcal{X}$ eventually either outputs value $\mathbf{F}$ at every process or outputs a history H' of failure detector $\Omega + \Sigma$, $H' \in (\Omega + \Sigma)(F)$. When $\mathcal{X}$ outputs $\mathbf{F}$ we say that $\mathcal{X}$ *detects a crash*. When $\mathcal{X}$ outputs $H' \in (\Omega + \Sigma)(F)$, we say that $\mathcal{X}$ *behaves like* $\Omega + \Sigma$. $\mathcal{X}$ guarantees that no crash is detected unless some process crashes.

Now we give a sketch of the proof that $\mathcal{X}$ can be emulated from any failure detector that solves QC. Let $\mathsf{QC}_\mathcal{D}$ be any QC algorithm using a failure detector $\mathcal{D}$. In this section, we present a reduction algorithm that transforms $\mathcal{D}$ into $\mathcal{X}$.

[3] Later we omit $\mathcal{E}$ in $\prec_\mathcal{E}$ and $\preceq_\mathcal{E}$ when there is no ambiguity on the environment $\mathcal{E}$.

Multivalued quittable consensus. So far we considered the *binary* version of quittable consensus: the set of non-Q decision values is $\{0, 1\}$. A *multivalued* quittable consensus is a generalization of binary quittable consensus where processes propose values in a given set V and eventually choose a common value in $V \cup \{Q\}$ so that, in addition to agreement and termination of QC, the following property is satisfied:

- *Multivalued QC-Validity:*
  (a) Q is decided only if a failure occurs;
  (b) A value $v \in V$ is decided only if some process proposes v.

By applying the transformation algorithm proposed in [10], we can easily transform any solution of quittable binary consensus into a multivalued quittable consensus.[4]

Agreement on a "good" simulation forest First we observe that, in failure-free executions, QC behaves exactly like consensus. Thus, as long as there are no failures, we can use QC to make processes agree on a consistent system state.

Let $\mathcal{I}$ be the set of all possible initial configurations of $\mathsf{QC}_{\mathcal{D}}$.

In the first phase of our reduction algorithm, every process p_i periodically samples the current failure detector history, advertises its current state and, by simulating partial runs of $\mathsf{QC}_{\mathcal{D}}$, constructs an ever-growing *simulation forest* [1], denoted by Υ_i, until, for each input configuration in $\mathcal{I}$, Υ_i contains a schedule in which some process decides (this is going to happen eventually for every correct process p_i [1]). Then p_i proposes Υ_i to an instance of QC. If Q is returned, p_i outputs $\mathbf{F}$ ($\mathcal{X}$ detects a crash). Otherwise, let $\bar{\Upsilon}$ be the resulting simulation forest. For a given input configuration $I \in \mathcal{I}$, consider the *first* schedule in $\bar{\Upsilon}$ in which processes start from I and at least one process decides (the *deciding* schedule for I in $\bar{\Upsilon}$). Again, if Q is decided, p_i outputs $\mathbf{F}$ ($\mathcal{X}$ detects a crash).

Now assume that, for each $I \in \mathcal{I}$, the deciding schedule for I in $\bar{\Upsilon}$ returns a value in $\{0, 1\}$. (Note that the processes either uniformly agree on such a "good" simulation forest $\bar{\Upsilon}$ or uniformly agree that a crash is detected and output $\mathbf{F}$ as a history of $\mathcal{X}$.)

Quorum emulation Now we show that, having agreed on a "good" simulation forest $\bar{\Upsilon}$, the processes can emulate a history of the quorum failure detector Σ [3].

Let $I_l \in \mathcal{I}$ be an initial configuration in which teams (non-empty partitions of Π) Q_{l_0} and Q_{l_1} propose, respectively 0 and 1. Let S_l be the corresponding schedule in $\bar{\Upsilon}$ in which a process decides on a value in $\{0, 1\}$. Let $\sigma_0, \ldots, \sigma_m$ be the sequence of "subsechedules" of S_l ($\sigma_0 = I_l$ and $\sigma_m = S_l$). Clearly, some process decides in σ_m.

Processes proceed in rounds. In each round r, and for each $I_l \in \mathcal{I}$, every process p_i does some gardening. More precisely, p_i tags all its messages with r (the current round number), and, for each $k \in [0, m]$, p_i maintains a growing tree $\bar{\Upsilon}_{irlk}$ which has σ_k as a basis and, besides vertexes of σ_k, $\bar{\Upsilon}_{irlk}$ includes only vertexes tagged with r. Process p_i lets $\bar{\Upsilon}_{irlk}$ grow until $\bar{\Upsilon}_{irlk}$ includes a schedule in which p_i decides [1]. Let Q_{irlk} be the "quorum" of this decision, i.e., the set of processes which have vertexes in $\bar{\Upsilon}_{irlk}$ tagged with r. At this point p_i outputs $Q_{ir} = \cup_{l,k} Q_{irlk}$. In parallel, p_i computes its estimate of Ω_i as in [7, 9].

The reduction algorithm is presented in Figure 1.

Lemma 1 $\forall r, r' \in \mathbb{N}, \forall i, j \in [1, n] : Q_{ir} \cap Q_{jr'} \neq \emptyset$

[4]The transformation algorithm of [10] uses uniform reliable broadcast which is given for free in our model.

```
1: output_i ← ⊥
2: Maintain Υ_i until it contains a deciding schedule for each I_l ∈ 𝓘
3: Ῡ ← QC_𝒟(Υ_i)                                   { Agreement on a forest }
4: Υ_i ← Ῡ
5: if (Ῡ = Q) or (Q is decided in some schedule in Ῡ) then
6:    output_i ← F                                   { Crash is detected }
7: else
8:    r ← 0
9:    while true do
10:      r++
11:      Update simulation forest Υ_i with Ῡ taken as a basis
12:      p_c ← deciding process of the smallest decision gadget of Υ_i  [1, 7, 9]
13:      Compute Q_ir
14:      output_i ← (p_c, Q_ir)
```

Figure 1: Reduction algorithm $T_{\mathcal{D}\to\mathcal{X}}$ for process p_i.

Proof: By contradiction, assume that there exists a partition of Π into two teams Q_0 and Q_1, such that $Q_{ir} \subseteq Q_0$ and $Q_{jr'} \subseteq Q_1$.

Consider $I_l \in \mathcal{I}$ in which that all processes in Q_0 propose 0 and all processes in Q_1 propose 1. Let $\sigma_0, \ldots, \sigma_m$ be the sequence of vertexes of the deciding schedule for I_l in $\bar{\Upsilon}$. By construction, for each $k \in [0, m]$, there exist two schedules S_i^k and S_j^k, respectively, in $\bar{\Upsilon}_{irlk}$ and $\bar{\Upsilon}_{jr'lk}$, such that:

(i) both S_i^k and S_j^k extend σ_k;

(ii) after σ_k, no process in Q_1 takes any step in S_i^k, and no process in Q_0 takes any step in S_j^k;

(iii) there exist v_i^k and v_j^k in $\{0, 1, Q\}$, such that p_i decides v_i^k in S_i^k and p_j decides v_j^k in S_j^k.

We proceed through the following claims:

Claim 1. For each $k \in [0, m]$, $v_i^k = v_j^k$.

Indeed, since, after σ_k, S_i^k and S_j^k include steps of two disjoint sets of processes, we can construct a schedule S^k in which all processes in Q_0 take the same steps as in S_i^k and all processes in Q_0 take the same steps as in S_i^k. Thus, in S^k, p_i decides v_i^k and p_j decides v_j^k. By the agreement property of QC, $v_i^k = v_j^k$.

Let $v^k = v_i^k = v_j^k$.

Claim 2. $v^0 = Q$

Assume, by contradiction, that $v_i^0 \in \{0, 1\}$. Since no process in Q_1 take any step in S_i^0 and all processes in Q_0 propose 0, p_i must decide 0 in S_i^0. Similarly, p_j either decides Q or decides 1 in S_j^0. Thus $v_i^0 \neq v_j^0$ — a contradiction with Claim 1.

Claim 3. There exists $k \in [0, m-1]$, such that $v^k \neq v^{k+1}$.

Assume, by contradiction, that $\forall k \in [0, m-1]$, we have $v^k = v^{k+1}$. By repeatedly applying Claim 2, we obtain $v^m = Q$.

But by construction, at least one process decides on a value in $\{0, 1\}$ in σ_m. By agreement of QC, in any extension of σ_m, no process can decide Q. That is, $v^m \in \{0, 1\}$ — a contradiction.

Now we establish a final contradiction through the following claim:

Claim 4. Claim 3 is false.

Let $k \in [0, m-1]$ be such that $v^k \neq v^{k+1}$. By construction, σ_{k+1} is the result of applying a single step of some process p_k to σ_k. Without loss of generality, assume that $p_k \in Q_0$. Thus, the processes in Q_1 have the same states in σ_k and σ_{k+1}.

Denote by S the suffix of S_j^k consisting of steps of the processes in Q_1 taken after σ_k. Note that the steps of S are all taken *after* σ_m and, thus, after σ_{k+1}. Since no process in Q_0 takes any step in S, and the processes in Q_1 cannot distinguish between σ_k and σ_{k+1}, S also applies to σ_{k+1}.

Thus, there is an extension of σ_{k+1} in which only the processes in Q_1 take steps after σ_{k+1} and in which p_j decides v^k. On the other hand, S_i^k is an extension of σ_{k+1} in which only the processes in Q_0 take steps after σ_{k+1} and in which p_j decides v^{k+1}. The two subsets Q_0 and Q_1 are disjoint and, thus, we can construct a common schedule consisting of the steps of these two extensions in which agreement is violated — a contradiction. □

Lemma 2 *There exists $r' \in \mathbb{N}$ such that $\forall r > r'$, Q_{ir} includes only correct processes.*

Proof: By the algorithm, Q_{ir} includes only processes that take steps tagged with round number r. Since there is a round number r' in which no faulty process participates anymore, we have the lemma. □

The result. Lemmata 1 and 2 imply that, when no crash is detected, we can emulate the quorum failure detector Σ [3].

On the other hand, when no crash is detected, the algorithm of Figure 1 emulates Ω [1, 7, 9]. Thus, in the case when no crash is detected, $\Omega + \Sigma$ is emulated. By our construction, a crash can be detected only if some instance of $\mathsf{QC}_{\mathcal{D}}$ returns Q. By validity of QC, some process has crashed. Thus, no crash is detected unless some process crashes. $\mathcal{X}$ is emulated.

$\Omega + \Sigma$ solve consensus (and, hence, QC) using read-write memory [3]. Thus, if no crash is detected, we have an algorithm to solve QC. Otherwise, if a crash is detected (line 6 in Figure 1), then processes can safely decide Q.

Theorem 3 *$\mathcal{X}$ is the weakest failure detector to solve QC in any environment.*

Employing the reduction proposed by Hadzilacos and Toueg [9], we obtain the following corollary:

Corollary 4 *$?\mathcal{P} + \mathcal{X}$ is the weakest failure detector to solve NBAC in any environment.*

References

[1] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM (JACM)*, 43(4):685–722, July 1996.

[2] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, March 1996.

[3] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. Shared memory vs message passing. Technical Report IC/2003/77, EPFL, December 2003. Availabe at http://icwww.epfl.ch/publications/.

[4] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(3):374–382, April 1985.

[5] J. Gray. A comparison of the byzantine agreement problem and the transaction commit problem. In *Proceedings of the Workshop on Fault-Tolerant Distributed Computing*, volume 448 of *LNCS*, pages 10–17. Springer-Verlag, 1986.

[6] R. Guerraoui. Non-blocking atomic commit in asynchronous distributed systems with failure detectors. *Distributed Computing*, 15:17–25, January 2002.

[7] R. Guerraoui and P. Kouznetsov. The weakest failure detector for non-blocking atomic commit. Technical Report IC/2003/47, EPFL, May 2003. Availabe at http://icwww.epfl.ch/publications/.

[8] V. Hadzilacos. On the relationship between the atomic commitment and consensus problems. In *Proceedings of the Workshop on Fault-Tolerant Distributed Computing*, volume 448 of *LNCS*, pages 201–208. Springer-Verlag, 1986.

[9] V. Hadzilacos and S. Toueg. The weakest failure detector to solve quittable consensus and non-blocking atomic commit. Unpublished manuscript – private communication, May 2003.

[10] A. Mostefaoui, M. Raynal, and F. Tronel. From Binary Consensus to Multivalued Consensus in asynchronous message-passing systems. *Information Processing Letters*, 73(5–6):207–212, Mar. 2000.

[11] D. Skeen. NonBlocking commit protocols. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 133–142. ACM Press, May 1981.