Randomized Multivalued Consensus

Paul EZHILCHELVAN[†] Achour MOSTEFAOUI[⋆] Michel RAYNAL[⋆]

[⋆] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
[†] Computing Science Dept, University of Newcastle upon Tyne, NE1 7RU, UK
{achour|raynal}@irisa.fr paul.ezhilchelvan@newcastle.ac.uk

Abstract

The *Consensus* problem is a fundamental problem one has to solve to implement reliable services or applications on top of asynchronous distributed systems prone to failures. Unfortunately, this problem cannot be solved in those systems as soon as one process can crash (Fischer-Lynch-Paterson's impossibility result).

Two approaches have been investigated to circumvent this impossibility result. Both consist in enriching the underlying system with appropriate "oracles". The *Unreliable Failure Detector* concept proposed by Chandra and Toueg constitutes one family of such oracles. Since it has been proposed, the failure detector-based approach has given rise to several failure detector-based consensus protocols.

The other family of oracles consists in allowing each process to use a random number generator. In that case, the protocol termination is only probabilistic. A few randomized consensus protocols for message-passing asynchronous distributed systems have been proposed. Moreover, they consider that processes can only propose values from a binary set.

This paper proposes a new randomized consensus protocol that allows processes to propose arbitrary values. Contrarily to other randomized consensus protocols, the proposed protocol does not require the a priori knowledge of the set of values that can be proposed by processes. It relies on a relatively simple combination of randomization and reliable broadcast.

Keywords: Asynchronous Distributed System, Consensus Problem, Crash Failure, Fault-Tolerance, Message Passing, Random Number, Randomized Protocol, Unreliable Failure Detector.

Category submission: regular.

Contact author: Michel Raynal.

Approx. word count: 4300.

1 Introduction

The *Consensus* problem lies at the heart of a lot of agreement problems (e.g., Atomic Broadcast, Atomic Multicast, Weak Atomic commitment, etc.). This means that a solution to any of those problems can be expressed as a protocol that uses a solution to the consensus problem as an underlying building block [3, 8, 10, 19]. Actually, the consensus problem can be seen as the *greatest common sub-problem* of a family of agreement problems. This encourages the following system architecture advocated in [11]: first design a layer providing an efficient consensus protocol, and, on top of it, design protocols solving particular agreement problems.

The consensus problem can be informally stated as follows. Each process proposes a value and has to decide a value (termination property) such that (1) there is a single decided value and (2) the decided value is one of the proposed values (safety properties). This apparently simple problem is actually impossible to solve in a deterministic way in asynchronous distributed systems where processes may crash (even only one process). This is known as the Fischer-Lynch-Paterson (FLP) impossibility result [7]. Intuitively, this is due to the combination of asynchrony and process crashes that, in the worst case, can prevent the processes to get a consistent global state of the execution [13].

To circumvent this impossibility result two main approaches have been investigated. One lies in the *unreliable Failure Detector* concept proposed and investigated by Chandra, Hadzilacos and Toueg [3, 4]. In that case, each process has access to a FD-oracle (Failure Detector oracle) that provides it with a list of processes that it suspects of having crashed. According to the properties (completeness and accuracy) a failure detector is assumed to satisfy, several classes of FD-oracles have been defined [3]. It has been proved that the class denoted $\diamond\mathcal{S}$ is the weakest that allows to solve consensus with the help of a failure detector [4]. This class is defined by the following two properties: any process that crashes is eventually suspected (completeness), and there is a time after which there is a correct process that is no longer suspected (eventual weak accuracy). Several $\diamond\mathcal{S}$-based consensus protocols have been designed in the recent past years [3, 14, 15, 21].

Another approach (which actually has been the first to be investigated) consists in abandoning the determinism requirement of the protocol, and allowing processes to query an oracle (R-oracle) providing them with random values [2, 5, 6, 17, 18]. The price that has to be paid by this approach is that the termination of the randomized protocol is only *probabilistic*. Its main advantage lies in the robustness of the resulting protocol: its behavior does not depend on how the system actually behaves.

This paper focuses on the consensus problem in asynchronous distributed systems equipped with R-oracles. To our knowledge, the randomized consensus protocols studied so far consider that the values proposed by the processes are binary. Hence, they solve the *Binary Consensus* problem. This paper proposes a randomized protocol that allows processes to propose values from an arbitrary set. It is interesting to note that the approach proposed in this paper could be combined with the failure detector-based approach to give rise to *Hybrid Multivalued Consensus* protocols [1, 9, 16].

The paper is composed of six sections. Section 2 presents the system model. Then, Section 3 describes the protocol. Section 4 proves it solves the consensus problem. Then, Section 5 discusses some features of the protocol. Finally, Section 6 concludes the paper.

2 Distributed Systems, Random Oracles and Consensus Problem

2.1 Asynchronous Distributed Systems with Process Crash Failures

The computation model follows the one described in [3, 7]. We consider a system consisting of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, \ldots, p_n\}$. A process can fail by *crashing*, *i.e.*, by prematurely halting. It behaves correctly (*i.e.*, according to its specification) until it (possibly) crashes. By definition, a *correct* process is a process that does not crash. A *faulty* process is one that is not correct. Let f denote the maximum number of processes that may crash. We assume $f < n/2$, i.e., a majority of processes is correct. (This requirement is necessary and sufficient for randomized consensus protocols [2].) Processes communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected by a channel. Channels are not required to be FIFO, but are assumed to be reliable: they do not create, duplicate, alter or lose messages. There is no assumption about the relative speed of processes nor on message transfer delays (i.e., the system is *asynchronous*).

2.2 Random Oracles

A random oracle consists of a set of R-oracle modules, each attached to a process. The R-oracle module attached to p_i provides it with a value $x \in \{1, \ldots, n\}$ each time p_i invokes the primitive *random*. A uniform distribution is assumed; this means that each value x ($1 \leq x \leq n$) has probability $1/n$ to be returned when p_i invokes *random*.

2.3 The Consensus Problem

In the Consensus problem, every correct process p_i *proposes* a value v_i and all correct processes have to *decide* on the same value v, that has to be one of the proposed values. More precisely, in an asynchronous distributed system equipped with R-oracles, the *Consensus problem* is defined by two safety properties (Validity and Uniform Agreement) and a probabilistic Termination Property. These properties are:

- Validity: If a process decides v, then v was proposed by some process.
- Uniform Agreement: No two processes decide differently.
- Termination: With probability 1, every correct process eventually decides some value.

Let V be the set of values that can be proposed by the processes to an instance of the consensus problem. A consensus is *binary* when the set V consists of only two values [2, 18]. It is *multivalued* when the set V can be arbitrarily large.

3 A Randomized Multivalued Consensus Protocol

3.1 Preliminary: Reliable Broadcast

The proposed randomized protocol uses the *Reliable Broadcast* communication primitives [12], namely, *R_Broadcast*(m) and *R_Deliver*(m). When a process issues *R_Broadcast*(m), we say that it "*R_broadcasts*" m. Similarly, when a process issues *R_Deliver*(m), we say that it "*R_delivers*" m. Reliable Broadcast is defined by the following set of properties [12]:

- Termination: If a correct process *R_broadcasts* m, then any correct process *R_delivers* m (no message from a correct process is lost).
- Uniform Agreement: If a process *R_delivers* m, then any correct process *R_delivers* m (no message R_delivered by a -correct or not- process is missed by a correct process).
- Validity: If a process *R_delivers* m, then m has been *R_broadcast* by some process (no spurious message).
- Integrity: A process *R_delivers* a message m at most once (no duplication).

Implementations of Uniform Reliable Multicast can easily be designed for asynchronous systems. A very simple (but inefficient) one, that works in fully connected networks, is the following : when a process receives a message m for the first time, it first forwards m to all the other processes, and only then considers the delivery of m [12]. According to the underlying network topology, more efficient implementations can be designed [20].

3.2 Underlying Principles

The underlying principle of the protocol (Figure 1) is the combination of reliable broadcasts to disseminate the values proposed by processes, with the use of random numbers to ensure that the agreement will be "eventual".

Each process first reliably broadcasts the value v_i it proposes. This is done at lines 2 and 3. The array $val_i[1:n]$ allows p_i to keep the proposed values it receives. Then, the processes proceed by executing asynchronous consecutive rounds [2, 3]. The local variable r_i denotes the round number p_i is currently involved in. The local variable est_i keeps p_i's current estimate of the decision value; initially, est_i is set to v_i, the value proposed by p_i. The protocol strives for the processes to have the same estimate value when they start a round. When this occurs, the processes converge during that round and this single estimate becomes the decided value. The use of random numbers allows this "best effort strategy" to provide the Termination property with probability 1.

A round is made up of two communication phases. During the first phase of a round r (lines 7-9), the processes exchange their current estimates. If a process p_i discovers that there is a majority of estimates that have the same value v, it updates est_i to v; otherwise, it updates est_i to $\perp$. Consequently, at the end of the first phase, we have the following property: $(est_i = v \neq \perp) \wedge (est_j = w \neq \perp) \implies (v = w) \wedge (v$ was a majority value among the set of estimates at the beginning of the round).

Then, the processes enter the second phase of the round during which they again exchange the new content of their est_i variables: the communication pattern of this phase (lines 10-11) is similar to the one of the first phase. If a process p_i receives the same value v such that $v \neq \perp$ (hence it is a proposed value) from a majority of processes it decides on it (line 13). Otherwise there are two cases.

- (1) If it received a value v different from $\perp$ (line 14-15), it adopts it as its new estimate value. Let us note that, in this case, this value v was a majority value among the estimates at the beginning of the round.
- (2) If p_i received only $\perp$ during the second phase, it adopts an estimate value by selecting randomly a value from its array val_i. (Let k be the randomly selected entry. Note that the

value of $val_i[k]$ is either the value proposed by p_k, or $\perp$). The proof will show that, if processes do not decide, they will eventually select the same entry and this entry will necessarily be different from $\perp$.

Then, the processes that have not decided during r, start $r+1$. Let us remark that if a process decides during r, the other processes decide during the same round, or at the latest during $r+1$.

3.3 The Protocol

The protocol is described in Figure 1. Each process p_i starts a randomized multivalued consensus by invoking the function RM_Consensus(v_i) which returns the decided value. The decided value v is returned when the process invokes *return* at line 4 or 13. The execution of this invocation terminates the participation of p_i to the consensus protocol.

To prevent a process from blocking forever (*i.e.*, waiting for a value from a process that has already decided), a process that decides, uses again a reliable broadcast (lines 4 and 13) to disseminate the decision value[1].

```
Function RM_Consensus(v_i)
(1)  val_i ← (⊥, ..., ⊥);
(2)  R_Broadcast VAL(v_i); activate task {T_1, T_2}
-------------------------------------------------------------------------
task T_1:
(3)  when VAL(v)   is R_Delivered from p_j: do val_i[j] ← v enddo
(4)  when DEC(v)   is R_Delivered from p_j: do return(v) enddo
-------------------------------------------------------------------------
Task T2:
(5)  r_i ← 0; est_i ← v_i;
(6)  while true do r_i ← r_i + 1; % round r_i = r %
       ---------- Phase 1 of round r ----------
(7)    broadcast PHASE1(r_i, est_i);
(8)    wait until (PHASE1(r_i, est) messages have been received from a majority of processes);
(9)    if (all those messages carry the same value v) then est_i ← v else est_i ← ⊥ endif;
       ---------- Phase 2 of round r ----------
(10)   broadcast PHASE2(r_i, est_i);
(11)   wait until (PHASE2(r_i, est) messages have been received from a majority of processes);
(12)   if (all those messages carry the same value v ≠ ⊥)
(13)     then est_i ← v; R_Broadcast DEC(est_i); return (est_i)
(14)     else if (at least one message carries a value v ≠ ⊥)
(15)               then est_i ← v
(16)               else est_i ← val_i[random]
(17)   endif endif
(18) endwhile
```

Figure 1: A Randomized Multivalued Consensus Protocol ($f < n/2$)

[1] A similar dissemination of a decided value is done in all the failure detector-based consensus protocols that we know [3, 14, 15, 21].

4 Proof

The proof assumes $f < n/2$ (which has been shown to be a necessary requirement for randomized consensus protocols [1, 2, 18]). Let us note that at least $(n - f)$ processes are correct. Moreover any set of $(n - f)$ processes is a majority set.

The proof of the Validity property is left to the reader (hint: note that a decided value is different from $\perp$, and any estimate variable est_i can only contain a proposed value or $\perp$).

4.1 Preliminary Lemmas

Lemma 1 *If no process decides during $r' \leq r$, then all correct processes will start the round $r + 1$.*

Proof The proof is by contradiction. Let r be the first round during which a correct process blocks forever. It does it at line 8 or 11 (**wait** statement). As there is a majority of correct processes and as (due to the definition of r) no correct process is blocked forever during $(r - 1)$, they all send a PHASE1$(r, -)$ message. Due to the "reliable channels" assumption, each correct process receives a majority of PHASE1$(r, -)$ messages, and does not block forever at line 8. The same argument applies for the **wait** statement of line 11. It follows that no correct process can blocks forever during a round. $\square_{Lemma\ 1}$

Lemma 2 *If all the processes that start a round r do it with their estimates equal to the same value $v \neq \perp$, then their estimates remain equal to that value v.*

Proof As, due to the lemma assumption, all the processes that start executing r have their estimates equal to the same value v, they can only exchange that value at line 7. Hence a process p_i that updates its estimate est_i at line 9, updates it to v. It follows that only v can be exchanged by the processes at line 10. Hence, as due to the lemma asumption $v \neq \perp$, according to the tests of lines 12 and 14, a process p_i can only execute line 13 or line 15. In both cases, it again updates est_i to v. $\square_{Lemma\ 2}$

Lemma 3 *If no process decides during $r' < r$, and all the processes that start r have the same estimate value $v \neq \perp$ when they start r, then each of them decides during r unless it crashes.*

Proof First of all, due to the Lemma 1, all correct processes start r. Consequently, they send PHASE1 and PHASE2 messages. So, no process can block forever during any round $r' \leq r$. The lemma follows from this observation and Lemma 2. As all the processes that execute the second phase of round r have the same estimate value (v) after line 9, they receive the same value v from all the processes that sent a PHASE2$(r, -)$ message. According to the test of line 12, it follows that they execute line 13 and decide. $\square_{Lemma\ 3}$

Lemma 4 *After the first phase of any round r (i.e., after line 9), an est_i variable is equal to $\perp$, or to an estimate value v that was a majority value among the estimates at the beginning of r (note that such a majority value can be $\perp$).*

Proof This lemma follows directly from the **wait** condition of line 8, the test of line 9, and the fact that any PHASE1$(r, -)$ message carries a value that an estimate had at the beginning of r. $\square_{Lemma\ 4}$

4.2 Uniform Agreement

Theorem 1 *No two processes decide distinct values.*

Proof Let us first observe that a process that decides at line 4, decides a value that has been decided by another process at line 13. Hence, we only consider the values that are decided at line 13.

Let r be the first round during which processes decide (at line 13). We consider two cases.

- Let p_i and p_j be two processes that decide during r. They decide v and w respectively (note that, due to the test of line 12, v and w are different from $\perp$). Due to the lines 11-13 we conclude that p_i received the same message PHASE2(r, v) from a majority of processes. Similarly, p_j received the same message PHASE2(r, w) from a majority of processes. As a process sends a single PHASE2 message during a round, it follows that there is a process p_k that sent the same PHASE2(r, v') to p_i and p_j. Consequently, $v' = v = w$.

- Let us now consider the case where p_i decides v during r, while p_j decides during a later round $r' > r$ (note that $v \neq \perp$). We claim that, from $r + 1$, the only estimate value present in the system is v. Hence, no other value can be decided.

  Proof of the claim. As p_i decides $v \neq \perp$ during r, it received a PHASE2(r, v) message from a majority of processes. Let us consider any process p_j that does not decide during r and progresses to $r + 1$. As, while executing r, p_j received at line 11 PHASE2 messages from a majority of processes, it received at least one PHASE2(r, v) message. From Lemma 4 we conclude that $v \neq \perp$ was a majority estimate value at the beginning of r. It follows from this lemma that it is not possible to have a PHASE2(r, w) message with $w \neq v$ or $w \neq \perp$. Hence, p_j can receive only v or $\perp$ in a PHASE2 message. As there is a majority of PHASE2(r, v) messages, p_j received at least one PHASE2(r, v) message. Hence, according to the test of line 14, p_j updates est_j to v at line 15. *End of the proof of the claim.*

$\square_{Theorem\ 1}$

4.3 Termination

Theorem 2 *Every correct process eventually decides with probability 1.*

Proof Let us remark that if a process decides then all correct processes decide: this is due to the Reliable Broadcast primitive used to disseminate a decided value (lines 13 and 4). The proof is by contradiction. Let us assume that no process decides. There is a time t after which:
- (H1) There are only correct processes executing the protocol, and
- (H2) The *val* arrays of the correct processes are equal. This is due to the fact these arrays are filled in with values that are disseminated with a Reliable Broadcast primitive. If p_i and p_j are both correct, then if the value v_k is R_delivered by p_i, it is also R_delivered by p_j. Hence after t, $val_i[k] = v_k$ implies $val_j[k] = v_k$.

Let us first note that, as no process decides, no correct process blocks forever in a round (Lemma 1). Moreover, no process executes line 13. Hence, at each round r after t, a process executes line 15 or line 16. There are three cases.

- All the processes that execute r, execute line 15.
  Due to Lemma 4, all the processes set their estimates to the same value $v \neq \perp$. Hence, they all have the same estimate value when they start $r + 1$. Due to Lemma 3 they decide.

- During r at least one process executes line 15.
  Due to Lemma 4, all the processes that execute line 15 set their estimates to the same value $v \neq \perp$. This value v is equal to v_k, the initial value of some process p_k. The other processes execute the line 16. Let us consider one of them, say p_j. There is a probability (equal to $1/n$) that the invocation of *random* by p_j returns k, and that consequently, p_j updates est_j to v_k (due to H2).

- During r, no process executes line 15.
  In that case, all processes execute line 16. There a probability (strictly greater than 0) that they all get the same random value k, and that the corresponding entry of the *val* arrays be different from $\perp$ (and hence equal to v_k).

So, during any round after t, there is a probability $p > 0$ that all estimates are different from $\perp$ and equal to a same proposed value. Hence, there is a probability $P(\alpha) = p + p(1-p) + p(1-p)^2 + \cdots + p(1-p)^{\alpha-1} = 1 - (1-p)^\alpha$ that all processes have the same estimate after at most α rounds. As $\lim_{\alpha \to \infty} P(\alpha) = 1$, it follows that, with probability 1, all processes will start a round with the same estimate. Then, according to Lemma 3, they will decide.

$\square_{Theorem\ 2}$

5 Discussion

5.1 Cost of the Protocol

The cost of the protocol is the cost of the reliable broadcasts, plus the cost of the task $T2$. To analyze the protocol, we consider that each message takes one time unit to be communicated and processed (by its destination process).

In such a context, the most favorable scenario for processes to converge occurs when all processes propose the same value[2]. It is interesting to notice that this most favorable scenario does not require the Reliable Broadcasts! In that scenario, the decision is obtained during the first round which is made up of two communication steps. Moreover, the number of broadcasts per round is equal to $2n$.

5.2 An Improvement

When it executes line 16, a process p_i can get the $\perp$ value, and consequently start a new round with $est_i = \perp$. This can prevent a value different from $\perp$ to be a majority value among the estimates at the beginning of the next round, thereby delaying the decision.

A way to prevent this "bad" situation is to force any process p_i to have an estimate value est_i different from $\perp$ when it starts a new round. This can be obtained by replacing line 16 (namely, $est_i \leftarrow val_i[random]$) with the following sequence of statements:

```
k ← random;
if val_i[k] = ⊥ then
        for ℓ = k+1, ... , n, 1, ... , k−1:
                if val_i[ℓ] ≠ ⊥ then exit_for_loop endif
        endfor;
```

[2]This scenario occurs frequently in practice with agreement problems such as Atomic Commitment [10]. In this problem, a process can propose COMMIT or ABORT and, most of the time, all processes propose COMMIT [10, 19].

$k \leftarrow \ell$;
endif;
$est_i \leftarrow val_i[k]$

These statements force a process to always start a round with an estimate value (est_i) different from $\perp$.

Furthermore, let us note that the use of Reliable Broadcasts to disseminate proposed values guarantees that there is a time t after which all the val_i arrays will eventually be equal. Let us consider the case where no process has decided before t. If the processes a priori agree on a sequence of random numbers [18][3], after t they will eventually enter a sequence of rounds such that all the processes will select the same array entry. If no decision has been obtained before t, this will expedite the decision after t.

5.3 The Case of Binary Consensus

Let us consider the case where only the values 0 and 1 can be proposed by the processes. Then, all the processes a priori know (1) the set of the values that can be proposed, and (2) the fact that this set has only two values. This common knowledge allows to simplify the protocol in the following way.

- The lines 1, 2 and 3 are suppressed. This means the reliable broadcasts are no longer necessary to disseminate the proposed values.
- The line 16 is replaced by $est_i \leftarrow$ *random01* (where *random01* provides 0 or 1, each with probability 1/2). This means that the R-oracle is used to select an estimate value, while it was used to select a process identity in the general protocol. Let us remark that *random01* is always invoked by a process p_i in a context where p_i knows both values have been proposed.

Interestingly, the protocol that is obtained from these modifications is the binary consensus protocol proposed by Ben-Or [2]. This shows that the general protocol we have presented includes [2] as a particular case, and hence can be seen as a generalization of it.

6 Conclusion

This paper has presented a new randomized consensus protocol that allows processes to propose values from an arbitrary set. The protocol combines the use of random number generators with reliable broadcasts. The reliable broadcasts are used to disseminate the values initially proposed by processes. The random numbers are used to entail the protocol termination with probability 1. It has been shown that in the most favorable scenario, the decision can be obtained in two communication steps. Interestingly and contrarily to previous randomized binary consensus protocols, the random number generators are independent of the set of values that can be proposed (they only depend on the number of processes).

References

[1] Aguilera M.K. and Toueg S., Failure Detection and Randomization: a Hybrid Approach to Solve Consensus. *SIAM Journal of Computing*, 28(3):890-903, 1998.

[3] This means that there is an "a priori agreement" on the sequence of random values. Hence, during a round, all processes that call *random* get the same result.

[2] Ben-Or M., Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. *2nd ACM Symposium on Principles of Distributed Computing, (PODC'83)*, Montréal (CA), pp. 27-30, 1983.

[3] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, 1996.

[4] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[5] Chor M., and Dwork C., Randomization in Byzantine Agreement. *Advances in Computing Research*, 5:443-497, 1989.

[6] Chor M., Merritt M. and Shmoys D.B., Simple Constant-Time Consensus Protocols in Realistic Failure Models. *Journal of the ACM*, 36(3):591-614, 1989.

[7] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, 1985.

[8] Fritzke U., Ingels Ph., Mostefaoui A. and Raynal M., Fault-Tolerant Total Order Multicast to Asynchronous Groups. *Proc. 17th IEEE Symposium on Reliable Distributed Systems*, Purdue University (IN), pp.228-234, October 1998.

[9] Goldreich O. and Petrank E., The Best of Both Worlds: Guaranteing Termination in Fast Randomized Byzantine Agreement Protocols. *Information Processing Letters*, 39:45-49, 1990.

[10] Guerraoui R., Revisiting the Relationship between Non-Blocking Atomic Commitment and Consensus. *Proc. 9th Int. Workshop on Distributed Algorithms (WDAG95)*, Springer-Verlag LNCS 972 (J.M. Hélary and M. Raynal Eds), pp. 87-100, September 1995.

[11] Guerraoui R. and Schiper A., Consensus Service: a Modular Approach for Building Fault-Tolerant Agreement Protocols in Distributed Systems. *Proc. 26t IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-26)*, pp. 168-177, June 1996.

[12] Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press (S. Mullender Ed.), New-York, pp. 97-145, 1993.

[13] Hélary J.-M., Hurfin M., Mostefaoui A., Raynal M. and Tronel F., Computing Global Functions in Asynchronous Distributed Systems with Process Crashes. *Proc. 20th IEEE Int. Conf. on Distributed Computing Systems (ICDCS'00)*, Taipei (Taiwan), April 2000.

[14] Hurfin M. and Raynal M., A Simple and Fast Asynchronous Consensus Protocol Based on a Weak Failure Detector. *Distributed Computing*, 12(4):209-223, 1999.

[15] Mostefaoui A. and Raynal M., Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors: a Generic Quorum-Based Approach. *Proc. 13th Int. Symposium on Distributed Computing (DISC'99), (Formerly WDAG)*, Springer-Verlag LNCS 1693, pp. 49-63, Bratislava (Slovakia), 1999.

[16] Mostefaoui A., Raynal M. and Tronel F., The Best of Both Worlds: A Hybrid Approach to Solve Consensus. *Proc. Int. Conference on Dependable Systems and Networks (DSN'00, formerly FTCS)*, IEEE Computer Society Press, New-York City, June 2000.

[17] Motvani R. and Raghavan P., Randomized Algorithms. *Cambridge University Press*, 476 p., 1995.

[18] Rabin M., Randomized Byzantine Generals. *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS'83)*, pp. 116-124, Los Alamitos (CA), 1983.

[19] Raynal M., Non-Blocking Atomic Commitment in Distributed Systems: A Tutorial Based on a Generic Protocol. *Journal of Computer Systems Science and Engineering*, 15(2):77-86, 2000.

[20] Rodrigues L. and Verìssimo P., Topology-Aware Algorithms for Large Scale Communication. *Advances in Distributed Systems*, Springer-Verlag, LNCS Series #1752, pp.1217-156, 2000.

[21] Schiper A., Early Consensus in an Asynchronous System with a Weak Failure Detector. *Distributed Computing*, 10:149-157, 1997.