

# *i*RBP - a Fault Tolerant Total Order Broadcast for Large Scale Systems

Luiz Angelo Barchet-Estefanel\*

Laboratoire ID - IMAG  
51, Avenue Jean Kuntzmann  
38330 Montbonnot St. Martin, France  
Luiz-Angelo.Estefanel@imag.fr

**Abstract** Fault tolerance is a key aspect on the development of distributed systems, but it is barely supported on large-scale systems due to the cost of traditional techniques. This paper revisits RBP, a Total Order Broadcast protocol known by its efficiency that presents some very interesting characteristics for scalable systems. However, we found a membership flaw on RBP that can lead to inconsistencies among correct processes. Hence, we propose *i*RBP, an improvement to the RBP algorithm that not only circumvents the membership weaknesses using recent membership techniques, but also improves its scalability aspects.

## 1 Introduction

Fault tolerance is a key aspect on the development of distributed systems, but it is barely supported on large-scale systems like GRIDs due to the cost of traditional techniques. A good alternative would be RBP (Reliable Broadcast Protocol), a well-known protocol for Total Order Broadcast [3,5]. Its structure around a logical ring allows the protocol to efficiently reduce the number of exchanged messages [11], which is very interesting in large-scale systems.

However, we observed that RBP is still vulnerable to some failure patterns. Correct processes may leave and rejoin the group due to incorrect suspicions, but nothing ensures that such processes will be kept consistent with the rest of the group members. This problem, known as *time-bounded buffering problem*, is not easy to solve (or has a high cost).

We propose *i*RBP [1], an improvement to the RBP protocol that integrates new membership and failure detection techniques to solve the *time-bounded buffering problem*. We also tried to improve its performance when the number of processes grows up. Based on the membership techniques implemented on *i*RBP, we propose a simple but efficient strategy to bound the latency levels of the protocol, an important scalability concern.

Our observations candidate *i*RBP as a good and efficient alternative to Total Order Broadcast protocols like Atomic Broadcast based on Consensus [2], that have scalability limitations due to the number of messages they exchange.

\* Supported by grant BEX 1364/00-6 from CAPES - Brazil

This paper is organized as follows. Section 2 defines the system model and properties considered through this paper. Section 3 presents RBP and how its membership flaws may lead to system inconsistency. Section 4 presents the techniques we used to solve the *time-bounded buffering problem*, and how they were integrated with *iRBP* internal structure. Section 5 analyses the *iRBP* performance through the number of exchanged messages and the delivery latency. Finally, Section 6 concludes the paper.

## 2 System Model and Definitions

We consider an asynchronous distributed system where a set of processes  $\pi(t) = \{p1;p2;...;pn\}$  interacts by exchanging point-to-point messages through unreliable (*fair-lossy*<sup>1</sup>) communication channels. For simplicity, **multicast primitives (including broadcast)** are constructed by sending **several point-to-point messages**.

Processes are only subjected to crash failures without recovery. A process that does not crash during all the execution is called “correct”. *Eventually strong* ( $\diamond S$ ) failure detectors, as defined by Chandra and Toueg [2], are used to suspect failed processes.

The Total Order Broadcast (Multicast) problem may be informally explained as how to ensure that remote processes deliver messages in the same order, while keeping a system consistent even if there are process failures. It is defined through four properties [6]:

- **VALIDITY** - If a correct process broadcasts a message  $m$  to  $Dest(m)$ , then some correct process in  $Dest(m)$  eventually delivers  $m$ .
- **AGREEMENT** - If a correct process in  $Dest(m)$  delivers a message  $m$ , then all correct processes in  $Dest(m)$  eventually deliver  $m$ .
- **INTEGRITY** - For any message  $m$ , every correct process  $p$  delivers  $m$  at most once, and only if (1)  $m$  was previously broadcast by  $sender(m)$ , and (2)  $p$  is a process in  $Dest(m)$ .
- **TOTAL ORDER** - If correct processes  $p$  and  $q$  both deliver messages  $m$  and  $m'$ , then  $p$  delivers  $m$  before  $m'$  if and only if  $q$  delivers  $m$  before  $m'$ .

## 3 Revisiting RBP

RBP (Reliable Broadcast Protocol) is a well-known Total Order Broadcast [3,5]. It uses the moving sequencer strategy [6] to agree with the Total Order property: only one single process can define the delivery order, while the role of sequencer moves around the processes. It can be easily implemented using token passing: the process that holds the token is the only one that can assign sequence numbers to the messages. Once it has sequenced a message (or a group of messages, if

<sup>1</sup> **Fair-lossy channel** - if  $p$  send a message  $m$  to  $q$  an infinite number of times and  $q$  is correct, then  $q$  receives  $m$  from  $p$  an infinite number of times.

---

there are many in the buffer [11]), the token is passed to the next process in the ring using unreliable communication primitives.

When a process receives the token (i.e., it was indicated to be the next sequencer), it *accepts* the token if and only if it has all data messages previously sequenced. If the process has already all previous messages, it simply assigns a sequence number to a new message (this message acts as an acknowledgement to the previous sequencer). However, if a process detects that it has lost some messages, it uses negative acknowledgements to explicitly request retransmission. In fact, if channels “behave well”, i.e., they do not lose many messages, the protocol can efficiently deliver messages with a minimum of message exchange [11].

However, the use of fair-lossy channels has a drawback: processes are not allowed to deliver messages immediately because a simple crash may violate the *Agreement* property. Instead, messages should be delivered only after the token was passed among an arbitrary number of processes, what is called  $k$ -resiliency. Actually, RBP proposes the use of  $n$ -resiliency, where the token must pass among all processes (sometimes called “total resiliency”).

### 3.1 Fault Tolerance, Membership and RBP Limitations

Fault tolerance is essential on RBP, because a single failure may prevent the token passing, blocking the protocol. Thus, at the occurrence of a simple failure, the token list should be reconstructed, removing suspect processes, in a procedure called “Reformation Phase” [3].

During the Reformation Phase, each process that detects a failure (which is called “originator”) contacts the other processes to start a new token list. The originator proposes a new token list composed by all processes that acknowledged its call. A three-phase commit (3PC) protocol coordinated by the originator [8] conducts the commitment of the new token list, which should be composed by a majority of processes.

However, as detection is not simultaneous to all processes, multiple processes can invoke the reformation concurrently, and a correct non-suspected process may be excluded from the token list just because it was the originator of a concurrent token list. In [3] it is assumed that such excluded processes may rejoin the token list in a later Reformation, and eventually the token list will contain all correct processes. We observed that this assumption is not enough to guarantee the *Agreement* property, leading to inconsistencies among the processes. In fact, such assumption induces the *time-bounded buffering problem* [4], where a process  $p$  cannot safely remove the message  $m$  unless if it knows that all processes in  $Dest(m)$  have either received  $m$  or crashed.

## 4 *i*RBP, an *improved* RBP Protocol

### 4.1 Solving the *time-bounded buffering problem*

The *time-bounded buffering problem* represents a serious flaw on RBP. Hopefully, this problem was already studied in the context of the Primary-Backup replica-

tion model with View Synchronous Communication (VSC) [5]. Two techniques to solve the *time-bounded buffering problem* revealed to be highly adapted to the RBP structure. These techniques, presented below, were integrated on the protocol that we called *iRBP*, for *improved* RBP.

**Program-Controlled Crash** In [4] it was claimed that no implementation of Reliable Broadcast (a problem weaker than Total Order Broadcast, see [7]) over fair-lossy links could solve the *time-bounded buffering problem* unless it uses a *Perfect* failure detector ( $P$ ) [2]. As  $P$  failure detectors are impossible to be implemented in asynchronous systems, a practical solution is to use *program-controlled crash*. Program-controlled crash gives the ability to kill other processes or to suicide, ensuring time-bounded buffering (if all suspect processes will “crash”, then message  $m$  can be safely discarded from the buffers).

The use of program-controlled crash together with membership control is not a new technique (it was already employed on the ISIS system [5]), but it is not very popular due to its non-negligible cost. Every time a process  $q$  is forced to crash due to an incorrect suspicion, a membership change is executed to update the view. Hence, is common to rely on failure detectors with conservative timeout values, which can block the system for a long period.

**Two views** In typical group membership architectures, solving efficiently both problems of time-bounded buffering and blocking prevention at the same time is not possible, because they are linked to the same failure detection scheme. In order to better explore both issues, Charron-Bost [4] proposed the use of two levels of Group Membership Service (GMS).

There, *membership views* (or simply *views*) are identical to the views of View Synchronous Communication, while *ranking views* (or *rk-views*) are installed between membership views. If membership views are denoted by  $v_0, v_1, \dots$ , the *rk-views* between  $v_i$  and  $v_{i+1}$  are denoted as  $v_i^0, v_i^1, \dots, v_i^{last}$ . The membership of all ranking views  $v_i^0, \dots, v_i^{last-1}$  is the same as the membership of  $v_i$ , differing only in the order that processes are listed in the view. For example,  $v_i = v_i^0 = \{p, q, r\}$ ,  $v_i^1 = \{q, r, p\}$ , etc. As *rk-views* are composed by the same set of processes, they do not require program-controlled crash.

This model in two layers allows us to solve the dilemma between fail over time and program-controlled crash. Membership views are generated by suspicions resulting from conservative timeouts, while *rk-views* are generated by suspicions resulting from aggressive timeouts. This way, *rk-views* contribute to avoid blocking situations, while membership views ensure time-bounded buffering of messages.

## 4.2 *iRBP* Membership

To solve the membership problems from RBP, we constructed a Two-Level View-Synchronous membership algorithm to replace the *Reformation Phase*. We did

---

**Algorithm 1.** Membership View Change with *program-controlled crash*  
 Upon suspicion of some process in  $TL_i$  by a conservative Failure Detector  
 RBroadcast (reformation, i)  
 Upon R-Deliver (reformation, i) by  $p_k$  for the first time

1. send  $seqQ_k$  to all /\* sends the "unstable" list of messages \*/
2.  $\forall p_i$ , wait until receive  $seqQ_i$  from  $p_i$  or  $p_i$  suspected
3. let  $initial_k$  be the tuple  $(TL, Msgs_k)$  s.t.
  - $TL$  means  $\{(core_k), ()\}$ , and  $core_k$  contains all processes that sent their  $seqQ$
  - $Msgs_k$  is the union of the  $seqQ$  sets received
4. execute Consensus among  $TL_i$  processes, with  $initial_k$  as the initial value
5. let  $(TL, Msg)$  be the consensus decision
6.  $stableQ \leftarrow Msg, seqQ \leftarrow \{\}$  /\* the set  $Msg$  becomes stable \*/
7. if  $p_k \in TL$ , then "install"  $TL$  as the next view  $TL_{i+1}$   
     else suicide

---

not rely on the model suggested by Charron-Bost [4] to represent the views and ranking views ("move the suspect coordinator to the end of the list") because it was not suitable for our ring-based protocol. Instead, we suggest a different approach to implement *rk*-views, where a view is composed by two subsets of processes:

$$\{("core\ group"),("external\ group")\}$$

Broadcasts are sent to the whole group, but the token is passed only among the "*core group*". Once a process is suspected, an *rk*-view change moves it from the *core group* to a contiguous set, called "*external group*". As a suspected process does not participate in the sequencing process (it is not in the token ring), it does not block the token passing. As it still belongs to the view, it receives all broadcasts, can send messages to the group and request retransmissions.

While algorithm for Membership View Change is only a modified version of the traditional View Synchronous Membership that includes *program-controlled crash* (Algorithm 1), the *rk*-view change was optimized, as an *rk*-view change does not need to exchange information about delivered messages (processes just need to know who is active), and no *program-controlled crash* is required. Consequently, Algorithm 2 presents an optimized version of the View Change algorithm, adapted to *rk*-view changes.

### 4.3 Resiliency Levels

As commented on Section 3, RBP [3] suggests the use of total resiliency, which guarantees the maximum level of fault tolerance. However, total resiliency induces a high latency, representing a time cost that should be reduced whenever it is possible. On *i*RBP, the resiliency level on depends on the number of processes in the *core group*, and consequently, to control the number of core processes is a simple technique to efficiently bounds the latency of the protocol.

In the next Section we evaluate how different resiliency levels influence the *delivery latency* from *i*RBP.

---

**Algorithm 2.** Optimised  $rk$ -view change

Upon suspicion of some process in  $TL_i^j$  by an aggressive Failure Detector

RBroadcast (rk-view, j)

Upon R-Deliver (rk-view, j) by  $p_k$  for the first time

1. send ack to all
  2.  $\forall p_i$ , wait until receive ack from  $p_i$  or  $p_i$  suspected
  3. let  $initial_k$  contain  $\{(core_k), (external_k)\}$  s.t.
    - $core_k$  is the list of all processes that sent ack
    - $external_k$  is the list of all processes that does not sent ack
  4. execute Consensus among  $TL_i^j$  processes, with  $initial_k$  as the initial value
  5. let (TL) be the consensus decision
  6. if  $p_k \in TL$ , then "install" TL as the next  $rk$ -view  $TL_i^{j+1}$
- 

## 5 Performance Analysis

### 5.1 Number of Messages

Due to the ability of RBP to piggyback acknowledgements and token passing in a single message,  $iRBP$  uses very efficiently the network resources. If no message is lost,  $iRBP$  needs only two broadcasts: the first one, sent by the message source, submits the message; the second one comes from the sequencer, which assigns a sequence number to that messages, at the same time that passes the token to the next sequencer. Thus, if no messages are losts, the protocol requires only  $2(n-1)$  messages (assuming that broadcasts transmit  $n-1$  point-to-point messages) for each message to be delivered.

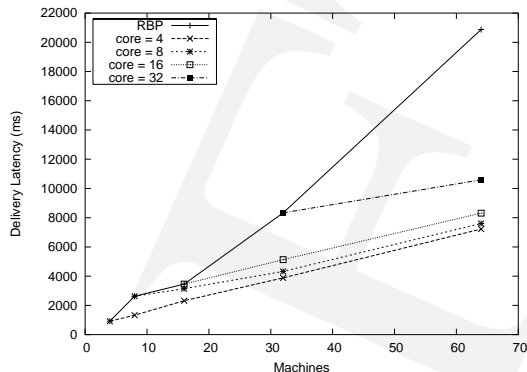
Comparatively, the Atomic Broadcast based on the Consensus [2] may exchanges up to  $(4+2n)(n-1)$  messages in a single execution step. There are other consensus algorithms than Chandra and Toueg Consensus, as for example the Early Consensus [9], but they mainly focus on the reduction of communication steps, while the number of exchanged messages remains high (still generating  $O(n^2)$  messages, against  $O(n)$  from  $iRBP$ ).

### 5.2 Latency, Core-resiliency and Scalability

One important element on  $iRBP$  performance analysis is its *latency*. We consider latency as the time elapsed between the broadcast of a message  $m$  by the source process and the first time it is delivered to the application, i.e., the *delivery latency*. On  $iRBP$ , latency depends on the resiliency level, because a message is only delivered after this resiliency level is achieved. If the number of processes grows up, the latency may reach undesirable levels.

To prevent such effects, we use the own structure of the Two-levels membership from  $iRBP$  to bound the latency. By controlling the number of processes in the *core* group, it is possible to limit the latency levels even if the system scales up.

We implemented *i*RBP in Java using the Neko framework [10]. The experiments were conducted on the **ID/HP i-cluster** from the ID laboratory Cluster Computing Center<sup>2</sup>, where machines are interconnected by a switched Ethernet 100 Mbps network. The tests were executed with 4, 8, 16, 32 and 64 machines (one process/machine), and we compared RBP's performance against *i*RBP with different *core*-resiliency levels. Figure 1 presents our results when messages arrive according to a Poisson process of rate 10 messages per second.



**Figure 1.** Comparison between RBP and *i*RBP delivery latencies

We can observe that in the RBP model, the latency increases almost linearly with the number of processes in the token list. By fixing the number of elements in the core group, we obtain latency levels smaller than those from RBP. This experiment shows that bounded delivery latency is possible with the use of simple techniques like the *core*-resiliency.

The choice of a small core-group allows the protocol to deliver messages with latency levels similar to conventional techniques (we should not forget that the Atomic Broadcast based on the Consensus requires from 6 to 8 communication steps), while the number of exchanged messages is far reduced. In addition, future implementations may take advantage of network low-level primitives, like Ethernet-Broadcast or IP-Multicast, optimizing all *i*RBP communications, which is not the case with the Atomic Broadcast based on the Consensus.

## 6 Conclusions

In the first part of this paper, we demonstrate that RBP, a well-known Total Order Broadcast protocol, allows some situations that can violate the consistency of processes. It is an interesting observation, because there are several works that

<sup>2</sup> <http://www-id.imag.fr/grappes.html>

explore RBP efficiency, but to our knowledge, none has examined the consistency problem as we did.

We looked for feasible solutions that could be applied to our “improved RBP” protocol, that we called *iRBP*. Specially, we identified the origins of the inconsistencies as the *time-bounded buffering problem*, and we focused on solving efficiently this problem. We replaced the original RBP membership control, allowing our protocol to deal with the *time-bounded buffering problem* while reducing the cost involved in the use of *program-controlled crash*.

In the second part of this paper, we focused on the performance of *iRBP*. The results from our practical experiments demonstrate that *iRBP* is a good candidate to large-scale distributed computing, like GRID environments, where fault tolerance is a major concern. *iRBP* is an efficient protocol for Total Order Broadcast, and can be used as a building block to provide fault-tolerant support on large scale systems.

## References

1. Barchet-Estefanel, L. A.: Analysing RBP, a Total Order Broadcast Protocol for Unreliable Networks. Technical Report, IC-EPFL - Switzerland. (2002)
2. Chandra, T. D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2). ACM Press New York, NY, USA (1996) 225-267
3. Chang, J., Maxemchuck, N.: Reliable Broadcast Protocols. *ACM Trans. on Computer Systems*, 2(3). ACM Press New York, NY, USA (1984) 251-273
4. Charron-Bost, B., Défago, X., Schiper, A.: Broadcasting Messages in Fault-Tolerant Distributed Systems: the benefit of handling input-triggered and output-triggered suspicions differently. *Proceedings of the 21<sup>st</sup> Int'l Symposium on Reliable Distributed Systems*, Osaka, Japan. (2002)
5. Chockler, G., Keidar, I., Vitenberg, R.: Group Communication Specifications: a comprehensive study. *ACM Computing Surveys*, 33(4). ACM Press New York, NY, USA (2001) 427-469
6. Défago, X.: Agreement-related Problems: from semi-passive replication to totally ordered broadcasts. PhD Thesis, EPFL - Switzerland. (2000)
7. Jalote, P.: *Fault Tolerance in Distributed Systems*. Prentice-Hall (1994)
8. Maxemchuck, N., Shur, D.: An Internet Multicast System for the Stock Market. *ACM Trans. on Computer Systems*, 19(3). ACM Press New York, NY, USA (2001) 384-412
9. Schiper, A.: Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3) Springer-Verlag, Berlin Heidelberg New York (1997) 149-157
10. Urbán, P., Défago, X., Schiper, A.: Neko: A single environment to simulate and prototype distributed algorithms. *Proceedings of the 15th Int'l Conf. on Information Networking*, Beppu City, Japan (2001)
11. Whetten, B., Montgomery, T., Kaplan, S.: A High Performance Totally Ordered Multicast Protocol. In: Birman, K. P., Mattern, F., Schiper, A. (eds.): *Theory and Practice in Distributed Systems: International Workshop*. Springer-Verlag, Berlin Heidelberg New York (1995) 33-57