Reliable Spreading of Messages in not Eponymous Systems

Sergio Arévalo
Dept. Sistemas Informáticos
Universidad Politécnica de Madrid
Madrid, Spain
sergio.arevalo@eui.upm.es

Carlos Herrera
Dept. de DETRI
Escuela Politécnica Nacional
Quito, Ecuador
carlos.herrera@epn.edu.ec

Ernesto Jiménez
Prometeo Researcher
Escuela Politécnica Nacional
and
Dept. Sistemas Informáticos
Universidad Politécnica de Madrid
Madrid, Spain
ernes@eui.upm.es

Jian Tang
Distributed System Laboratory (LSD)
Universidad Politécnica de Madrid
Madrid, Spain
tjapply@gmail.com

Rommel Torres
Dept. de C.C. y Electr.
Universidad Técnica Particular de Loja
Loja, Ecuador
rovitor@utpl.edu.ec

Abstract

The broadcast service spreads a message m among all processes of the system, such that each process eventually delivers m. A basic broadcast service does not impose any delivery guarantee in a system with failures. Fault-tolerant broadcast is a fundamental problem

This work has been partially funded by Prometeo Project of Secretaria de Educación Superior, Ciencia, Tecnología e Innovación de la República del Ecuador, by CEDIA, Ecuador, under project CEPRAVII-2013-05, and by Universidad Politecnica de Madrid, Spain, under project AL14-PID-30.

in distributed systems that adds certainty in the delivery of messages when crashes can happen in the system. Traditionally, the fault-tolerant broadcast service has been studied in classical distributed systems when each process has a unique identity (eponymous system).

In this paper we study the fault-tolerant broadcast service in anonymous systems, that is, in systems where all processes are indistinguishable.

1. Introduction

One of the most important communication abstractions for distributed systems is the *broadcast* service. This facility sends a message to all the processes of the system. However, it does not impose any fault-tolerant property. So, if a sender process crashes while it is broadcasting a message m, the delivery of m is not known a priori. To avoid this indeterminism in the delivery when processes may crash, the *reliable broadcast* (RB) service was introduced.

RB is another fundamental problem in fault-tolerant distributed computing that imposes delivery guarantees. In short, RB is a broadcast service that requires that all correct processes deliver the same set of messages, and that all messages sent by correct processes must be delivered to all correct processes [7].

All works that study the fault-tolerant broadcast service rely on distributed systems where processes are distinguishable because each one of them has a unique identity (called eponymous systems). In this paper we base our study in anonymous systems. In an anonymous system processes are not identifiable because all of them are coded identically (i.e, processes have no identity, and there is no a way to distinguish among them).

Anonymous processes are common in some practical distributed systems, such as sensor networks where a unique identity is not possible to be included in each device (due, for example, to small storage capacity, reduced computational capacity, or a huge number of elements to be identified) [1]. Another practical issue where anonymous processes are used is related with privacy (for example, to hide the user identity in a system) [5].

Our work Up to now, the fault-tolerant broadcast service has been studied in classical distributed systems when each process has a different identity. However, this is the first paper, to our knowledge, that is devoted to the broadcast service with fault-tolerant guarantees in anonymous systems.

In this paper we present an algorithm that implements fault-tolerant broadcast service in anonymous systems. We would like to remark that the aim of our algorithm is to prove that it is possible to implement them in anonymous systems instead of achieving an efficient solution.

This paper is organized as follows. The system model is presented in Section 2. Definitions are included in Section 3. In Section 4 we include an implementation of reliable broadcast. We finish our paper with the conclusion in Section 5.

2. The Anonymous System

The anonymous asynchronous system (denoted $AAS[\emptyset]$) is formed by a set of processes $\Pi = \{p_i\}_{i=1,\dots,n}$ such that its size $|\Pi|$ is n, and i is the index of each process p_i, $1 \leq i \leq n$.

Processes are anonymous [3]. Hence, they have no identity, and there is no a way to differentiate between any two processes of the system (i,e., processes have no identifier, and execute the same code). So, anonymity implies that process indexes are fictitious in the sense that each process $p_i \in \Pi$ does not know its index i. We only use process indexes from an observer point of view, and with the purpose of simplifying the notation.

A run R is formed by the set of steps taken by each process $p_i \in \Pi$. We assume that time advances at discrete steps in each run R, and there is a global clock T whose values are the positive natural numbers. Note that T is an auxiliary concept that we only use it for notation, but processes can not check or modify it. Processes are $asynchronous$, that is, the time to execute a step by a process in a run R is unbounded.

A process crashes when it stops taking steps. We assume that a crashed process never recovers. A process $p_i \in \Pi$ is *correct* if it does not crash, and *faulty* if it crashes. Let $Correct$ be the set of correct processes, and let $Faulty$ be the set of faulty processes. We denote by f the maximum number of processes that may crash. Unless otherwise is stated, we consider that this maximum number is $n-1$ (i.e., $f \leq n-1$).

In $AAS[\emptyset]$ processes communicate among them sending and receiving messages through links. Each pair of processes is connected by a link. We assume that links neither duplicate nor create spurious messages. We consider that links are $reliable$. A link l is reliable if it is guaranteed that every message sent using l is eventually received as long as sender and receiver are correct processes. Note that messages can be lost in a reliable link if either sender or receiver is a faulty process. Unless otherwise is stated, links do not enforce any restriction with respect to the order in which messages are sent or received[1].

The system $AAS[\emptyset]$ has two primitives to send and receive messages: $bcast(m)$ and $del(m)$. We say that a process p_i broadcasts a message m when it invokes $bcast_i(m)$. Similarly, a process p_i delivers a message m when it invokes $del_i(m)$. We omit the index i in these primitives when the process p_i that invokes these primitives is not important.

With $bcast_i(m)$ process p_i sends a copy of message m to each process $p_k \in \Pi$, and $del_i(m)$ reports to the invoking process p_i that m is the received message which is delivered. To preserve the anonymity of the system, we also consider that delivering processes can not identify the link through which a broadcast message is received.

In the literature is traditionally considered that broadcast and delivered messages are unique. It is assumed that every broadcast message m includes the different sender's process identity as part of the content of m to distinguish it [2, 9, 6, 7]. Since in $AAS[\emptyset]$ processes are anonymous, we have to consider that messages are not unique. Let $\mathcal{B}_i$ be the multiset of all messages broadcast by process p_i, and let $\mathcal{D}_i$ be the multiset of all messages delivered by process p_i. Let $\mathcal{B}$ be the multiset of all messages broadcast in the system, i.e., $\mathcal{B} = \bigcup_{p_i \in \Pi} \mathcal{B}_i$. Similarly, $\mathcal{D} = \bigcup_{p_i \in \Pi} \mathcal{D}_i$ is the multiset formed by all messages delivered in the system. Hence, for instance, if we have the following five primitves with the same message: $bcast_i(m)$, $bcast_j(m)$, $del_i(m)$, $del_j(m)$, and $del_k(m)$, then the multiset $\mathcal{B}$ has two instances of m, and $\mathcal{D}$ have three instances.

We assume that broadcast and deliver primitives of $AAS[\emptyset]$ dont not give any fault-tolerant guarantees if a process crashes. Specifically, if a process crashes while it is executing $bcast(m)$, a copy of the message m can be received by any subset of processes, and, hence, $del(m)$ only can be invoked by this subset of processes. Therefore, the system $AAS[\emptyset]$, with these two communication primitives, offers an *unreliable broadcast* service.

[1]This assumption happens, for example, if we consider that a link has, for each sender/receiver process, a internal buffer that stores sent/received messages not satisfying FIFO order.

3. Definitions

Now, we define broadcast services that include fault-tolerance.

Three properties have to be satisfied by the broadcast and deliver primitives to provide a $reliable\ broadcast$ (RB) service:

- *Integrity*: All instances of all messages delivered by any process have to be broadcast.
- *Validity*: All instances of all messages broadcast by correct processes have to be delivered by each correct process.
- *Agreement*: All correct processes deliver the same number of instances of each message.

Let us define the RB service more formally.

Definition 1 *The RB service has to preserve the following two properties:*

1. *Integrity:* $\forall p_i \in \Pi$, $\mathcal{D}_i \subseteq \mathcal{B}$.
2. *Validity:* $\forall p_i \in Correct$, $\bigcup_{p_j \in Correct} \mathcal{B}_j \subseteq \mathcal{D}_i$.
3. *Agreement:* $\forall p_i, p_j \in Correct$, $\mathcal{D}_i = \mathcal{D}_j$.

4. Implementing RB in $AAS[\emptyset]$.

We show in this section that the algorithm of Figure 1 implements the RB service in an anonymous asynchronous system $AAS[\emptyset]$ independently of the number of faulty processes.

Description of the algorithm. We say that process p_i RB-broadcasts a message m, if it invokes RB_bcast$_i(m)$ (line 3). Similarly, we say that process p_i RB-delivers a message m, if it invokes RB_del$_i(m)$ (line 15). When process p_i invokes RB_bcast$_i(m)$, it sends $(m, seq_i[m])$ to every process of the system $AAS[\emptyset]$, such that m is the message to spread, and $seq_i[m]$ is the p_i's number of sequence of this message m (line 5). The variable $seq_i[m]$ allows each process p_j to distinguish among several identical messages m RB-broadcast by process p_i (initially, $seq_i[m]$ is 0, line 1).

When process p_i delivers (m, s), that is, message m with number of sequence s (line 6), it uses $count\_msg_i[m, s]$ to increase the number of messages m with the same number of sequence s delivered by process p_i. Then, it sends $(ACK, m, s, count\_msg_i[m, s])$ to every process of the system $AAS[\emptyset]$ (line 8).

When process p_i delivers (ACK, m, s, c) for first time, that is, message m with number of sequence s and counter c of messages (m, s) (line 9), it relays this message (ACK, m, s, c) (line 10) to spread this message even if the sender process of (ACK, m, s, c) crashes. To avoid relaying a same message indefinitely, lines 9-11 are executed only the first time that a message is delivered (line 9).

To RB-deliver message m as many times as processes RB-broadcast a message m with a number of sequence s, process p_i uses $exec_i[m, s]$ and the function $apply\_msg(m, s, c)$. The variable $exec_i[m, s]$ remembers the number of times that process p_i executed RB_del$_i(m)$ due to the reception of $(ACK, m, s, -)$ (initially $exec_i[m, s]$ is 0, line 1). The function $apply\_msg(m, s, c)$ allows process p_i to execute RB_del$_i(m)$ from next time, indicated by $exec_i[m, s]$+1, to c (line 14). To avoid to RB-deliver messages due to outdated delivery of (ACK, m, s, c), c has to be greater than $exec_i[m, s]$ (line 13).

Correctness of the algorithm.

Lemma 1 *Integrity:* $\forall p_i \in \Pi$, $\mathcal{D}_i \subseteq \mathcal{B}$.

Proof: Let us consider, by the way of contradiction, that the claim is not true. Then, there is a process p_i such that $\mathcal{D}_i \supset \mathcal{B}$. Following the contradiction, we have that $RB\_bcast(m)$ is executed x times, and $RB\_del_i(m)$ is executed $y > x$ times. Note that in one extreme case x processes can execute $RB\_bcast(m)$ once, and, in the other, a same process can execute $RB\_bcast(m)$ x times.

A process p_k increments its local number of instance s of m by one (line 4) previously to execute $bcast(m, s)$ (line 5). Then, for each process p_k, the values of s for m that are broadcast are $1, 2, 3, \ldots$. So, in this case, these values of s for m that are broadcast by any process will be in the range from $1, 2, 3, \ldots$ up to x. On the other hand, each time that a process p_k delivers a number of instance s of m executing $del_k(m, s)$ (line 6), it counts this number of instances incrementing $count\_msg_k[m, s]$ by one (line 7). Hence, because links are reliable and neither duplicate nor create spurious messages,

```
(1) Init
(2)    arrays $seq_i$, $exec_i$ and $count\_msg_i$ have 0 in all positions;

(3) when RB_bcast$_i(m)$ is executed:
(4)    $seq_i[m] \leftarrow seq_i[m] + 1$;
(5)    $bcast_i(m, seq_i[m])$

(6) when $del_i(m, s)$ is executed:
(7)    $count\_msg_i[m, s] \leftarrow count\_msg_i[m, s] + 1$;
(8)    $bcast_i(ACK, m, s, count\_msg_i[m, s])$

(9) when $del_i(ACK, m, s, c)$ is executed for first time:
(10)   $bcast_i(ACK, m, s, c)$;
(11)   apply_msg($m, s, c$)

(12) function apply_msg($m, s, c$):
(13)   if ($exec_i[m, s] < c$) then
(14)       for ($j = exec_i[m, s] + 1$ to $c$) do
(15)           RB_del$_i(m)$
(16)       end for;
(17)       $exec_i[m, s] \leftarrow c$
(18)   end if.
```

Figure 1: RB service in $AAS[\emptyset]$ (process p_i' code).

if $r \leq x$ processes executes $bcat(m, s)$, then every process p_k broadcast the sequence of messages $bcast_k(ACK, m, s, 1)$, $bcast_k(ACK, m, s, 2)$, $\ldots$ $bcast_k(ACK, m, s, c)$, such that $c \leq r$. Thus, because links are reliable and neither duplicate nor create spurious messages, process p_i eventually receives the messages of these broadcast primitives, and executes their corresponding $del_i(ACK, m, -, -)$. Note that, because links do not force any delivery order, these executions may not be in the same order than their respective broadcast primitives were issued.

We can observe that process p_i stores in $exec_i[m, s]$ the number of invocations of $RB\_del_i(m)$ for each instance s of m when $del_i(ACK, m, s, c)$ is executed (lines 14-17). We can also observe that process p_i only delivers the instance s of m, executing $RB\_del_i(m)$, when $del_i(ACK, m, s, c)$ is also executed, but if it has not been applied yet (line 13). Hence, if $RB\_bcast(m)$ is executed x times, then $RB\_del_i(m)$ is executed at most x times. So, we reach a contradiction, and, hence, $\forall p_i \in \Pi$, $\mathcal{D}_i \subseteq \mathcal{B}$.

Lemma 2 *Validity:* $\forall p_i \in Correct$, $\bigcup_{p_j \in Correct} \mathcal{B}_j \subseteq \mathcal{D}_i$.

Proof: A correct process p_j increments its local number of instance s of m by one (line 4) previously to execute $bcast(m, s)$ (line 5). So, its values of s for m that are broadcast are $1, 2, 3, \ldots$.

On the other hand, each time that a correct process p_j delivers a number of instance s of m executing $del_j(m, s)$ (line 6), it counts this number of instances incrementing $count\_msg_j[m, s]$ by one (line 7). Hence, because links are reliable and neither duplicate nor create spurious messages, if c correct processes executes $bcat(m, s)$, then every correct process p_j broadcast the sequence of messages $bcast_j(ACK, m, s, 1)$, $bcast_j(ACK, m, s, 2)$, $\ldots$ $bcast_j(ACK, m, s, c)$. Thus, because links are reliable and neither duplicate nor create spurious messages, every correct process p_i eventually receives the messages of these broadcast primitives, and executes their corresponding $del_i(ACK, m, -, -)$.

We can observe that any correct process p_i stores in $exec_i[m, s]$ the number of invocations of $RB\_del_i(m)$ for each instance s of m when $del_i(ACK, m, s, c)$ is executed (lines 14-17). We can also observe that process p_i only delivers the instance s of m, executing $RB\_del_i(m)$, when $del_i(ACK, m, s, c)$ is also executed, but if it has not been applied yet (line 13). Hence, if $RB\_bcast(m)$ is executed x times, such that c of them are correct processes, then $RB\_del_i(m)$ is executed at least c times.

Lemma 3 *Agreement:* $\forall p_i, p_j \in Correct$, $\mathcal{D}_i = \mathcal{D}_j$.

Proof: Let us consider, by the way of contradiction, that the claim is not true. Following the contradiction, let us consider, w.l.o.g., that a correct process p_i delivers s instances of a message m, and a correct process p_j delivers $x < s$ instances of this message m. If correct process p_i delivers s instances, it also executes s times $del_i(ACK, m, -, -)$ and, hence, $bcast_i(ACK, m, -, -)$ (lines 9-11). Thus, because links are reliable, all these messages $(ACK, m, -, -)$ will be received by correct process p_j. Hence, process p_j eventually has to deliver from x to s instances of m (lines 14-17). Therefore, we reach a contradiction, and $\forall p_i, p_j \in Correct$, $\mathcal{D}_i = \mathcal{D}_j$.

5. Conclusion

Fault-tolerant broadcast is a fundamental problem in distributed systems that includes several guarantees in the delivery of messages when crashes can happen in the system. Traditionally, the fault-tolerant broadcast service has been studied in classical distributed systems where each process has a unique identity (eponymous systems).

In this paper we have studied the fault-tolerant broadcast service in anonymous systems.

References

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. Distributed Computing, 18 (4), pp. 235–253, 2006.

[2] H. Attiya and J. Welch. Distributed Computing: Fundamentals, Simulations, and Advanced Topics. Wiley, 2004.

[3] F. Bonnet and M. Raynal. Anonymous Asynchronous Systems: The Case of Failure Detectors. Distributed Computing, 26 (3), pp. 141–158, 2013.

[4] T. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM, 43(2), pp. 225–267, 1996.

[5] T. Chothia and K. Chatzikokolaris. A survey of anonymous peer-to-peer file-sharing. In Proceedings Workshops on Embedded and Ubiquitous Computing (EUC), LNCS 3823, pp. 744–755, Springer, 2005.

[6] X. Défago, A. Schipe, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Computing Surveys, 36(4), pp. 372–421, 2004.

[7] V. Hadzilacos and S. Toueg. A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical Report of 94–1425, 83 pages, Cornell University, Ithaca (USA), 1994.

[8] A. Mostéfaoui, S. Rajsbaum, M. Raynal, and C. Travers. On the computability power and the robustness of set agreement-oriented failure detector classes. Distributed Computing, 21 (3), pp. 201–222, 2008.

[9] M. Raynal. Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems Morgan & Claypool, 2010.