BBOBB: A total order broadcast algorithm achieving low latency and high throughput

Michel Simatic, Benoit Tellier

BBOBB: A total order broadcast algorithm achieving low latency and high throughput

Michel Simatic and Benoit Tellier
Télécom SudParis, Université Paris-Saclay, 9 rue Charles Fourier - 91011 Évry Cedex
Email: [First name].[Last name]@telecom-sudparis.eu

***Abstract*—Within data centers, many applications rely on a total order broadcast algorithm to achieve fault-tolerance. In this context, reducing latency and improving throughput are important issues. Current algorithms fail to optimize both latency and throughput at the same time. This paper presents BBOBB, a new total order broadcast algorithm. BBOBB offers simultaneously a low latency and a high throughput, especially for small application messages.**

I. Introduction

Total order broadcast is a fundamental group communication abstraction that lies at the core of different approaches to replication, such as state-machine replication [1], [2]. Total order broadcast guarantees that 1) every non-faulty replica receive all requests and 2) no two replicas disagree on the order in which requests are received. From a performance perspective, since 1984 when Chang and Maxemchuk published the historical first algorithm [3], reducing latency and improving throughput have been the two main driving forces for new algorithms. Reducing latency contributes to reducing application's response time. Improving throughput contributes to optimizing data center resources usage and reducing its energy consumption. Current algorithms fail to optimize both latency and throughput at the same time [4]. For instance, LCR [5] and TRAINS [6] are algorithms which optimize throughput to the detriment of latency. FastCast [7] is an attempt to combine low latency and high throughput, but its throughput is lower than the throughput of LCR and TRAINS.

This paper presents BBOBB (Broadcast Based On a Binary Behavior), a new total order broadcast algorithm inspired by the binary round-robin protocol [8] and the use of a broadcast tree in GentleRain [9]. BBOBB offers simultaneously a low latency and a high throughput, especially for small application messages: In this case, the average latency is $\frac{3n}{2\log_2(n)}$ instead of $\frac{3(n-1)}{2}$.

II. Model

Concerning the system model, we assume a small cluster of homogeneous machines interconnected by a local area network. Each machine hosts a process participating in the algorithm. BBOBB relies on a membership service. This service implements the abstraction of a perfect failure detector (P) [10] to which each process has access. In addition, it provides each process with the same ordered view of the processes participating in the algorithm.

Concerning the performance model, we assume that our LAN is based on a switch. Thus, we use the round-based model proposed in [5]. In one round: 1) a network card can send a message and simultaneously receive one; 2) a process can send a message to all or a subset of processes; 3) the network is able to carry several messages simultaneously.

III. Algorithm

When a process p_i wants to broadcast an application message `m` with total order guarantees, p_i stores `m` into lb_i, a local batch of application messages.

BBOBB is a distributed wave algorithm. During a step, each process sends its own batch and the batches received until now to a process chosen according to a binary scheme. Let us detail the algorithm. We assume there are n participating processes. When a new wave starts, its first step consists in each process p_i copying lb_i into a batch b_i and *fsending*[1] a set of batches containing only b_i to its successor $p_{(i+1) \bmod n}$. Step two of the wave consists in each process p_i *fsending* a set of batches containing the batch $b_{(i-1) \bmod n}$ received from p_i's predecessor and b_i to $p_{(i+2) \bmod n}$. More generally, step j ($j \geq 2$) consists in each process p_i *fsending* a set of batches containing $b_{u,u\in[(i+1-2^{j-1}) \bmod n,i]}$ to $p_{(i+2^{j-1}) \bmod n}$. When p_i receives messages of step k with 2^k greater or equal to the number n of participating processes, p_i has received all of the batches of this wave. We demonstrate this by proving by induction that: $\forall i \in [0,n), \forall j \in [1,k]$, p_i has received batches $b_{u,u\in[(i+1-2^j) \bmod n,i]}$ at the end of step j. By definition, at the end of step 1, any p_i has received $b_{(i-1) \bmod n}$ and b_i: The property holds for $j = 1$. Let us assume it is true for step $j \in [1,k)$. Then, $\forall i \in [0,n)$, p_i has received $b_{u,u\in[(i+1-2^j) \bmod n,i]}$ at the end of step j. In particular, $p_{(i-2^j) \bmod n}$ has received $b_{u,u\in[(i-2^j+1-2^j) \bmod n,(i-2^j) \bmod n]}$. During step $j+1$, $p_{(i-2^j) \bmod n}$ *fsends* a set containing $b_{u,u\in[(i+1-2^{j+1}) \bmod n,(i-2^j) \bmod n]}$ to p_i. Thus, at the end of step $j+1$, p_i has received batches $b_{u,u\in[(i+1-2^{j+1}) \bmod n,i]}$: The property holds for $j+1$. We conclude that, at step k, every p_i has received all of the batches. Moreover, thanks to the membership service, any p_i has the same ordered view of the processes participating in BBOBB: each p_i delivers application messages contained in batch b_0, then application messages in b_1, etc. The wave is finished.

Figure 1 presents an example: Participating processes E, F, G and H have built local batches le, lf, lg and lh containing application messages to be TO-broadcast. At the end of the wave, each process has received all of the batches e, f, g and h: It delivers application messages contained in batch e, then application messages in f, etc.

[1] `fsend()` is a primitive for sending messages reliably and in FIFO order. TCP is an example of a protocol providing `Fsend()`.

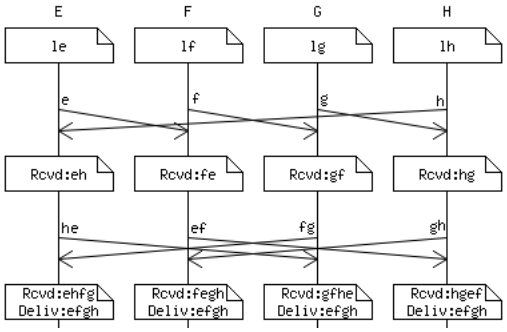

Fig. 1. Message sequence chart of a wave executed by participating processes E, F, G and H

Group membership changes: When a new process arrives or a participating process leaves (intentionaly or not), the membership service generates a `view_change` event on all processes. Upon receiving this event, each process p_i executes a view change procedure similar to the one of LCR protocol [5]. Once this procedure is done, p_i starts a new wave.

IV. Performance evaluation

The latency of broadcasting a message is the number of rounds that are necessary from the initial broadcast of message `m` until the last process delivers `m` [5]. In BBOBB, unlike other total order broadcast algorithms, it is not sufficient to count the number of network messages to determine the number of rounds. This is because the network messages sent during the different steps of a wave do not have the same size. There are 2 cases. First case: The size of the header of network messages (e.g. 40 bytes for TCP and IPv4, 60 bytes for TCP and IPv6) is negligible against the average size of batches. Then, sending a network message containing a set of batches takes as long as sending one network message per batch. In this case, a wave lasts $n-1$ rounds (where n is the number of participating processes). The average latency is $\frac{3(n-1)}{2}$. Second case: The size of the header of network messages is *not* negligible against the average size of batches. Then, sending a network message containing a set of batches is less sensitive to the number of batches inside the set. In this case, a wave lasts $\frac{n}{\log_2(n)}$ rounds. The average latency is $\frac{3n}{2\log_2(n)}$, which is remarkably lower than $\frac{3(n-1)}{2}$ as soon as $n \geq 4$.

Concerning BBOBB throughput, Guerraoui *et al.* prove that, given n computers involved in a total order broadcast algorithm, wired on a switch providing a throughput D_{link} on each link, the maximum throughput is $D_{link} \times \frac{n}{n-1}$ [5]. However, their demonstration of this upper bound is based on the number of messages. Using the same demonstration with BBOBB leads to the non pertinent upper bound $D_{link} \times \frac{n}{\log_2(n)}$. To prove that $D_{link} \times \frac{n}{n-1}$ is indeed the upper bound, we notice that, during a wave, any process p_i delivers $n \times S_{batch}$ bytes (where S_{batch} is the average total size of all application messages in a batch). But, p_i receives only $(n-1) \times S_{batch}$ bytes from the network (the bytes of all batches except the batch of p_i). We conclude that the maximum throughput is $D_{link} \times \frac{n \cdot S_{batch}}{(n-1) \cdot S_{batch}}$. In addition, the maximum throughput is reduced by the prediction-oriented throughput efficiency ($POTE$), i.e. the theoretical ratio between the number of bytes delivered per network message and the number of bytes of the network message [6]. When batches of messages are small, $POTE_{\text{BBOBB}}$ is better than $POTE_{\text{TRAINS}}$, TRAINS currently having the best $POTE$ [6]: BBOBB is closer to maximum theoretical throughput than TRAINS.

V. Related work

TRAINS [6] has better throughput than LCR [5]. BBOBB has a better throughput than TRAINS, and a lower latency. FastCast [7] has been designed to provide a low latency and a high throughput. For small messages, as soon as there are at least 4 processes, BBOBB has a lower latency than FastCast [7], with a better throughput.

Cason *et al.* present the importance of latency variability in nowadays applications [11]. We need a full implementation of BBOBB to measure this variability.

VI. Conclusion and perspectives

This paper presents BBOBB, a new total order broadcast algorithm. BBOBB offers simultaneously a low latency and a high throughput, especially for small application messages. We are currently implementing BBOBB to study experimentally its performance and in particular its latency variability.

References

[1] L. Lamport, "The implementation of reliable distributed multiprocess systems," *Computer Networks*, vol. 2, no. 2, pp. 95–114, May 1978.

[2] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Comput. Surv.*, vol. 22, pp. 299–319, December 1990.

[3] J.-M. Chang and N. F. Maxemchuk, "Reliable broadcast protocols," *ACM Trans. on Comput. Syst.*, vol. 2, no. 3, pp. 251–273, 1984.

[4] P. Urbán, X. Défago, and A. Schiper, "Contention-aware metrics for distributed algorithms: Comparison of atomic broadcast algorithms," in *Proceedings of the Nineth International Conference on Computer Communications and Networks*, October 2000, pp. 582–589.

[5] R. Guerraoui, R. R. Levy, B. Pochon, and V. Quéma, "Throughput optimal total order broadcast for cluster environments," *ACM Trans. on Comput. Syst.*, vol. 28, pp. 5:1–5:32, July 2010.

[6] M. Simatic, A. Foltz, D. Graux, N. Hascoët, S. Ouillon, N. Reboud, and T. Wang, "TRAINS: A Throughput-Efficient Uniform Total Order Broadcast Algorithm," in *Proceedings of the International Conference on New Technologies of Distributed Systems (NTDS)*, Paris, France, Jul. 2015, pp. 1–8.

[7] G. Berthou and V. Quéma, "FastCast: a Throughput- and Latency-efficient Total Order Broadcast Protocol," in *Proceedings of the International Middleware Conference*, 2013, pp. 1–20.

[8] S. Ranganathan, A. George, R. Todd, and M. Chidester, "Gossip-style failure detection and distributed consensus for scalable heterogeneous clusters," *Cluster Computing*, vol. 4, no. 3, pp. 197–209, 2001.

[9] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel, "Gentlerain: Cheap and scalable causal consistency with physical clocks," in *Proceedings of the ACM Symposium on Cloud Computing*, 2014, pp. 4:1–4:13.

[10] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of ACM*, vol. 43, pp. 225–267, March 1996. [Online]. Available: http://doi.acm.org/10.1145/226643.226647

[11] D. Cason, P. J. Marandi, L. E. Buzato, and F. Pedone, "Chasing the tail of atomic broadcast protocols," in *Proceedings of the 34th Symposium on Reliable Distributed Systems*, Sept 2015, pp. 286–295.