Fast Total Ordering for Modern Data Centers
Extended Abstract

Amy Babay, Yair Amir
Department of Computer Science at Johns Hopkins University
{babay, yairamir}@cs.jhu.edu

I. Introduction

Data center applications rely on messaging services that guarantee reliable, ordered message delivery for a wide range of distributed coordination tasks. Totally ordered multicast, which (informally) guarantees that all processes receive messages in exactly the same order, is particularly useful for maintaining consistent distributed state in systems as diverse as financial systems, distributed storage systems, cloud management, and big data analytics platforms.

Défago et al. survey the many existing total ordering protocols and classify them based on their ordering mechanisms [1]. A particularly successful type of ordering protocol is the class of *token-based* protocols. Token-based protocols typically arrange the processes participating in the protocol in a logical ring and order messages using a special control message (called the token) that carries the information needed to order new messages and is passed around the ring. Token-based protocols are attractive because of their simplicity; a single mechanism, the token, provides ordering, stability notification, flow control, and fast failure detection.

Token-based protocols also achieved high network utilization at the time they were introduced; for example, the Totem Ring protocol [2] achieved about 75% network utilization on 10-megabit Ethernet using processors standard for 1995. The simplicity, flexibility, and high performance of token-based protocols led to their use in practical messaging services, including the Spread toolkit [3], the Corosync cluster engine [4], and the Appia communication framework [5].

When Fast Ethernet replaced 10-megabit Ethernet, network speed increased by a factor of ten, and network span shrank by the same factor (from 2000 to 200 meters) so that basic network characteristics remained essentially the same. This allowed the same protocols to continue to utilize the network well. However, on networks common in today's data centers, these protocols do not reach the same network utilization as in the past while maintaining reasonable latency.

1-gigabit and 10-gigabit networks could not use the same techniques as Fast Ethernet to scale throughput (this would have required a 1-gigabit network span to be limited to 20 meters). Moving to these faster networks required changing the network architecture and adding buffering to switches. This changed networking trade-offs. While throughput increased by a factor of 10, 100, or more and latency was substantially reduced, the decrease in latency was significantly lower than the corresponding improvement in throughput.

This change in the trade-off between a network's throughput and its latency alters the performance profile of token-based protocols, as these protocols are particularly sensitive to latency. The ability to multicast new messages rotates with the token, so no new messages can be sent from the time that one participant finishes multicasting to the time that the next participant receives the token, processes it, and begins sending new messages.

The gap between the performance of existing protocols and the performance that is possible in modern environments led us to design the Accelerated Ring protocol. The Accelerated Ring protocol compensates for, and even benefits from, the switch buffering that limits the network utilization of other token-based protocols.

II. Protocol Overview

Similarly to other token-based protocols, the Accelerated Ring protocol passes a token around a logical ring, and a participant is able to begin multicasting upon receiving the token. The key innovation is that, unlike in other protocols, a participant may release the token before it finishes multicasting. Each participant updates the token to reflect all the messages it will multicast during the current rotation of the token around the ring before beginning to multicast. It can then pass the token to the next participant in the ring at any point during the time it is multicasting. Since the token includes all the information the next participant needs, the next participant can begin multicasting as soon as it receives the token, even if its predecessor on the ring has not yet completed its multicasting for the current token rotation.

However, the fact that the token can reflect messages that have not yet been sent requires careful handling of other aspects of the protocol. For example, messages cannot be requested for retransmission as soon as they are reflected in the token, since this may result in many unnecessary retransmissions.

The ability to send the token before all multicasts are completed allows the token to circulate the ring faster, reduces or eliminates periods in which no participant is sending, and allows for controlled parallelism in sending. As a result, the Accelerated Ring protocol is able to simultaneously provide higher throughput and lower latency than a standard token-based protocol.

Subtly different total ordering semantics exist and the Accelerated Ring protocol offers multiple service levels. Here we consider *Agreed delivery*, which guarantees that messages delivered within a particular membership are delivered in the same total order by all members of that membership. The total order respects causality. A formal specification of Agreed delivery is given in [2].

III. Evaluation

We evaluate the performance profile of the Accelerated Ring protocol and compare it to the performance of the original

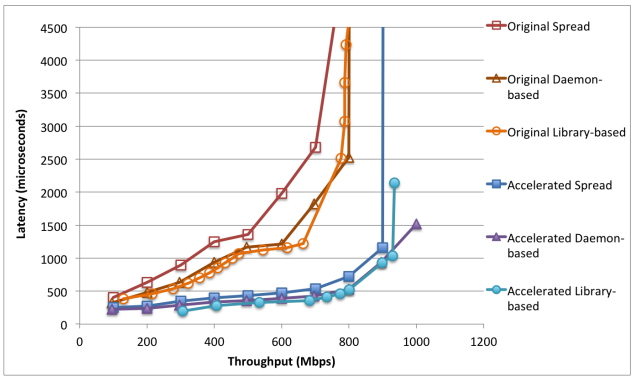

Fig. 1. Agreed delivery latency vs. throughput, 1-gigabit

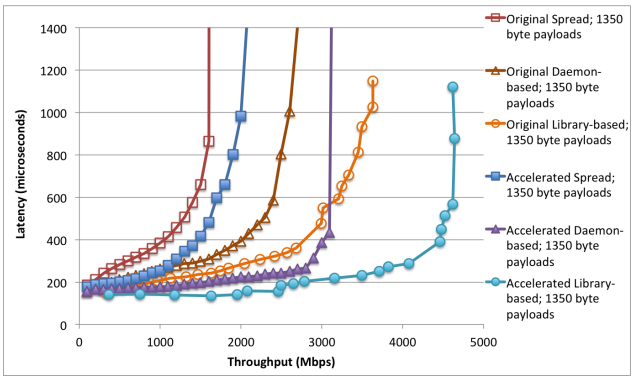

Fig. 2. Agreed delivery latency vs. throughput, 10-gigabit

Totem Ring protocol [2]. We evaluate both protocols in library-based and daemon-based prototype implementations as well as in complete production implementations in Spread. We ran the system at different throughput levels and measured the average latency to deliver a message at each throughput. Each server sent the same number of messages at a fixed rate and received all the messages sent by all the servers.

All benchmarks use eight Dell PowerEdge R210 II servers, with Intel Xeon E3-1270v2 3.50 GHz processors and 16 GB of memory. The servers were connected using a 1-gigabit Catalyst 2960 Cisco switch and a 10-gigabit 7100T Arista switch. All data messages contained a 1350 byte payload. This size allows the entire message to fit in a single IP packet with a standard 1500 byte MTU with sufficient space for protocol headers.

A. 1-gigabit Experiments

Figure 1 shows the difference between the original Ring and the Accelerated Ring protocols. When Spread uses the original protocol, its latency for Agreed delivery is at least 400 microseconds, even for the lowest throughput level tested (100 Mbps). In contrast, with the Accelerated Ring protocol, Spread is able to reach 400 Mbps throughput with latency below 400 microseconds. At 900 Mbps, Spread's latency is under 1.2 milliseconds using the accelerated protocol, which is about the same as the latency for the original protocol at 400 Mbps. With the original protocol, Spread supports up to 500 Mbps, with latency around 1.3 milliseconds, before latency begins to climb rapidly. The accelerated protocol is able to support 800 Mbps with latency around 720 microseconds, simultaneously improving throughput by 60% and latency by over 45%.

The accelerated protocol also improves maximum throughput compared to the original protocol. Using the accelerated protocol, Spread is able to reach over 920 Mbps. Since we only measure clean application data, and Spread adds substantial headers, this is practically saturating the 1-gigabit network.

B. 10-gigabit Experiments

Figure 2 shows the benefit of the Accelerated Ring protocol on a 10-gigabit network. Using the original protocol, Spread can provide throughput up to about 1 Gbps before the protocol starts to reach its limits and latency climbs. Its average latency at this throughput is 385 microseconds. Using the accelerated protocol, Spread can provide 1.2 Gbps throughput with an average latency of about 310 microseconds, for a simultaneous improvement of 20% in both throughput and latency. The maximum throughput Spread reaches with latency under 1 millisecond using the original protocol is 1.6 Gbps, but using the accelerated protocol, Spread is able to reach 2 Gbps with similar latency, for a 25% improvement in throughput.

Unlike on 1-gigabit networks, on 10-gigabit networks, processing is slow relative to the network. Therefore, the differing overheads of the different implementations have a significant impact on performance. While Spread can support 1.2 Gbps throughput with average latency around 310 microseconds using the accelerated protocol, the daemon-based prototype supports up to 2.9 Gbps with the same latency, and the library-based prototype reaches 3.5 Gbps at that latency.

Because processing is a bottleneck for Spread on 10-gigabit networks, we consider the prototype implementations to see the full power of the protocol. For the daemon-based prototype, the original protocol supports 2 Gbps with latency around 390 microseconds. The accelerated protocol supports 2.8 Gbps throughput with latency around 265 microseconds, for a simultaneous improvement of 40% in throughput and over 30% in latency.

IV. Conclusion

The Accelerated Ring protocol is implemented in open-source prototypes suitable for research. A production implementation has been adopted as the default protocol for local area networks and data center environments in Spread.

Acknowledgment

This work was supported in part by DARPA grant N660001-1-2-4014. Its contents are solely the responsibility of the authors and do not represent the official view of DARPA or the Department of Defense.

References

[1] X. Défago, A. Schiper, and P. Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Comput. Surv.*, vol. 36, no. 4, pp. 372–421, Dec. 2004.

[2] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella, "The totem single-ring ordering and membership protocol," *ACM Trans. Comput. Syst.*, vol. 13, no. 4, pp. 311–342, Nov. 1995.

[3] Spread Concepts LLC, "The Spread Toolkit," http://www.spread.org, retrieved March 23, 2015.

[4] "The Corosync cluster engine," http://corosync.github.io/corosync, retrieved March 23, 2015.

[5] "Appia communication framework," http://appia.di.fc.ul.pt, retrieved March 23, 2015.