Near-Optimal Total Order Message Scattering in Data Center Networks

Gefei Zuo
USTC and Microsoft Research

1. Motivation & Related work

Distributed transaction processing is an important workload in data centers. One line of work [5, 7, 15] use locks to protect from conflicts among concurrent transactions. Clearly, for a contented object, the lock must be held for a network RTT per transaction, limiting the transaction throughput to $1/RTT$ [8]. The other line of work, optimistic concurrency control [9] assigns a *logical timestamp* to each transaction and orders transactions by the timestamps. A transaction needs *rollback* when a node receives a late write that conflicts with a previous read [4]. If the network has an ordering property to ensure transactions are received in strict timestamp order, single round-trip transactions can be processed without locks and rollbacks (in the absence of failure) [2].

Our work follows the trend of co-designing distributed systems with network layer [10, 12]. We propose *total order message scattering* to ensure each receiver processes transaction messages in timestamp order, so that each transaction is processed atomically. On the sender side, *send(messages)* scatters a group of messages with an automatically generated unique timestamp. Each message is a tuple of application data and a recipient. Transaction processing requires scattering different messages to different receivers atomically, and each message may span multiple packets. Mostly-Ordered Multicast [12] and NOPaxos [10] cannot guarantee ordering of multi-packet messages. Furthermore, [10, 12] require a centralized switch or sequencer, while we hope to leverage multiple shortest paths to improve network utilization. On the receiver side, *recv()* receives a message and returns data and sender. The messages are guaranteed to be delivered to the application in increasing timestamp order.

Because packet loss in data center networks with carefully engineered hop-by-hop flow control or end-to-end congestion control are rare [12], we follow the end-to-end principle in system design [13] and leave packet loss detection and recovery to applications, as in [10, 12].

2. Design

Delivering messages in timestamp order would be trivial if the network has *bounded delay* and the clocks are perfectly synchronized. Each sender can assign message timestamp by physical time and each receiver can buffer incoming messages, wait this bounded delay, reorder incoming messages and deliver them to the application. However, bounded delay in decentralized data center networks is impractical due to traffic burstiness [11]. Instead of *static* bounded delay, we provide *dynamic* bounded delay by providing a dynamic *lower bound* of message timestamps in the network.

When a receiver R receives an updated *timestamp lower bound* (TSLB) M, it can safely deliver messages with timestamps below M, because R won't receive any messages below M. To derive the TSLB, we consider an example with two senders S_1, S_2 and one receiver R connected via a network switch S. When R receives timestamp T_1 from S_1 and T_2 from S_2, it can set TSLB $M = min(T_1, T_2)$ and safely process messages with timestamp below M, because timestamps from each sender grow monotonically. A liveness problem would occur if S_1 keeps sending and S_2 has nothing to send. T_2 would stop increasing, block M as well. To make sure receivers can distinguish delayed messages and idle senders, an idle sender needs to send *beacon* messages to receivers periodically.

For a network with N senders and N receivers, it would result in N^2 beacon messages and clearly cannot scale to large distributed services. Fortunately, network switches in modern data centers have some computation power. P4 programmable switch [3] can do customized processing on each network packet; Many commodity switches [1] also have general-purpose CPUs to do control-plane processing.

We aggregate *timestamp lower bounds* hierarchically in the data center network: beacon messages are sent to the switches instead of receivers. As long as there is no loop in the network, the timestamp information can be propagated layer-to-layer. See Figure 1 for an example. When switch receives a packet from S_1 or S_2, T_1 or T_2 is updated accordingly, so is TSLB. Switch then sends the updated TSLB to receivers via beacon packets. We assume that each network path has FIFO property. Based on that, when a beacon packet M is transferred on a path P_i between switch and R_i, all data packets on P_i transferred after M is guaranteed to have a higher timestamp. This ordering property between beacon and data packets is preserved hop-by-hop in the network.

Switch processing. With programmable switches, TSLB can be an additional field in the packet header. Since each data packet carries both message timestamp and TSLB on the network link, dedicated beacon packets are no longer needed when network link is not idle.

For switches without per-packet processing capability, the end hosts can send beacon messages periodically to

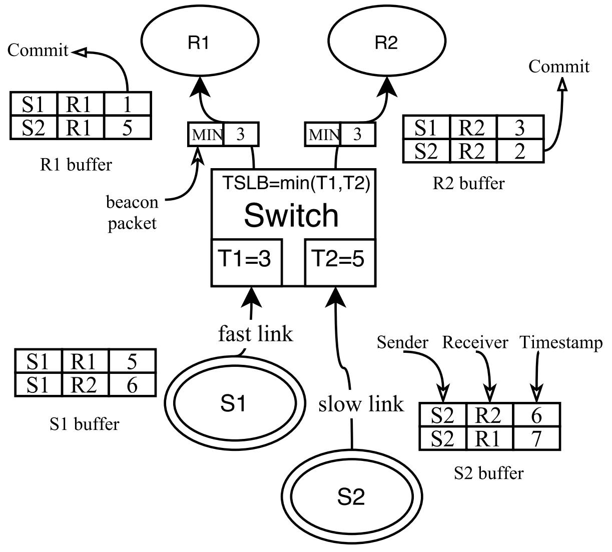

Figure 1: Two senders(S_1,S_2) and two receivers(R_1,R_2) connect via a switch. The switch maintains the TSLB and one timestamp register per port (T_1, T_2). Beacon packets are labeled **MIN** here.

the CPU port of switches. The switch CPU can process the beacon messages and send to next-hop switches. The data packets are forwarded by the switch fabric directly. As long as we ensure the ingress and egress queues for data and beacon packets are shared, the ordering property among beacon and data packets is preserved. The processing delay introduced by switch CPU would slack TSLB and increase message delivery latency.

Timestamp generation. To minimize reordering delay, received messages' timestamp should be as close as possible. The main reason of the outliers is the delay variance of network paths, especially the last-hop software processing delay introduced by OS network stacks and hypervisors. In modern data centers, the delay on end hosts is typically much larger and varying than the delay in network [10, 11].

We propose a logical timestamp generation scheme using feedback control, which does not require clock synchronization. When an end host joins the system, it asks its topological nearest neighbor in the system for a latest timestamp to register with the uplink switch. As long as the timestamp is greater than last TSLB on the switch, admission is granted and the switch will take the newcomer into account when calculating next TSLB.

When a message is received, the receiver measures the difference Δ between message timestamp and the receiver's TSLB. It also maintains an average Δ of recent messages. When the application sends an ACK for this message, the difference $\delta = \Delta - avg(\Delta)$ is piggybacked to the ACK message. If the application does not send an ACK in a certain time, the difference information is discarded. The sender then adjusts its logical clock speed c according to δ: $c = c - \alpha \cdot \delta$, where α is a positive parameter.

Beacon interval. The selection of beacon interval is a trade-off between network overhead and message delivery delay. We address this issue with three parameters.*Min beacon interval* is the minimum time between broadcasting two beacon messages. When the TSLB is kept unchanged for *max beacon interval*, it broadcasts a beacon message. The *timeout* is to prevent a *failure* node from blocking the whole network. Upon timeout, the switch would not take the ingress link's timestamp into account and block its traffic to ensure no data packets with inconsistent timestamps pass through.

Network overhead optimality. With a programmable switch, TSLB is piggybacked to data packets whenever the network link is not idle. Delay feedback is piggybacked to application-layer ACK messages. Except for new node joining handshakes and beacons on idle links, no additional packets are generated for total order message scattering.

Reordering delay optimality. Assuming hosts send packets continuously via programmable switches, the reordering delay, measured by the time from receiving a data packet to receiving a higher TSLB, converges to the amplitude of delay fluctuations among multiple network paths according to the timestamp feedback control. An idle host needs to send beacons for the network to know it is idle in the beacon period, increasing the reordering delay by beacon interval. With software processing on commodity switches, there is one more factor: switch software processing delay, which is a trade-off between delay and message complexity.

3. Preliminary Evaluation

We evaluate our design by a testbed emulating Figure 1, including two PowerEdge R730 servers with Mellanox ConnectX-4 NIC and an Arista DCS-7060CX-32S-F switch with AMD GX-424CC 4-core CPU. Using Linux kernel network stack, a single CPU core can process beacon messages at 70K packets per second (pps) and 100 μs delay. To process beacons from 32 ports, the minimal beacon interval is 1 ms. In the future, we plan to improve beacon processing performance with libpcap or Netfilter, and simulate performance with P4 switches.

4. Conclusion

This paper presents total order message scattering, an ordered unreliable distributed message delivery service for lockless transaction processing. This service works by hierarchically accumulating a timestamp lower bound on each network path. It has three assumptions on the network: (1) network paths should be loop-free; (2) packets on the same path should be transferred in order; (3) switches are programmable or have a CPU port. Our scattering is close to optimal in terms of both reordering delay and network overhead. We plan to implement lockless transaction processing with total order message scattering and run performance benchmarks [6, 14, 16].

References

[1] Arista eos. URL https://www.arista.com/en/products/eos.

[2] H. Attiya and J. L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems (TOCS)*, 12(2):91–122, 1994.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[4] M. J. Carey and W. A. Muhanna. The performance of multiversion concurrency control algorithms. *ACM Transactions on Computer Systems (TOCS)*, 4(4):338–378, 1986.

[5] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen. Fast and general distributed transactions using rdma and htm. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 26. ACM, 2016.

[6] T. T. P. COUNCIL. Tpc-c benchmark v5. URL http://www.tpc.org/tpcc/.

[7] A. Kalia, M. Kaminsky, and D. G. Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided (rdma) datagram rpcs. In *OSDI*, pages 185–201, 2016.

[8] A. K. M. Kaminsky and D. G. Andersen. Design guidelines for high performance rdma systems. In *2016 USENIX Annual Technical Conference*, page 437, 2016.

[9] H.-T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)*, 6(2):213–226, 1981.

[10] J. Li, E. Michael, N. K. Sharma, A. Szekeres, and D. R. Ports. Just say no to paxos overhead: Replacing consensus with network ordering. In *OSDI*, pages 467–483, 2016.

[11] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats, et al. Timely: Rtt-based congestion control for the datacenter. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 537–550. ACM, 2015.

[12] D. R. Ports, J. Li, V. Liu, N. K. Sharma, and A. Krishnamurthy. Designing distributed systems using approximate synchrony in data center networks. In *NSDI*, pages 43–57, 2015.

[13] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

[14] T. H.-S. TEAM. Smallbank benchmark. URL http://hstore.cs.brown.edu/documentation/deployment/benchmarks/smallbank/.

[15] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen. Fast in-memory transaction processing using rdma and htm. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 87–104. ACM, 2015.

[16] A. Wolski. Tatp benchmark description (version 1.0), 2009.