On Exploiting Locality for Generalized Consensus

Sebastiano Peluso
Virginia Tech
peluso@vt.edu

Alexandru Turcu
Virginia Tech
talex@vt.edu

Roberto Palmieri
Virginia Tech
robertop@vt.edu

Binoy Ravindran
Virginia Tech
binoy@vt.edu

***Abstract*—Single leader-based Consensus protocols are known to stop scaling once the leader reaches its saturation point. On the other hand, establishing Consensus of commands by taking into account only their dependencies (as specified by Generalized Consensus) is appealing because of the potentially higher parallelism and lower latency. However, current solutions have well-known pitfalls due to the higher quorum size, which is required to exploit low-latency fast decisions, and the need for tracking dependency relations. In this paper we briefly introduce M^2PAXOS, a new implementation of Generalized Consensus that provides a fast decision of commands by leveraging a classic quorum size, which matches just the majority of nodes deployed. M^2PAXOS does not establish command dependencies based on conflicts; rather it associates accessed objects with nodes, so that the delivery decision of commands operating on the same objects is made by a common node. The evaluation study of M^2PAXOS confirms its effectiveness by showing an improvement up to 7× over state-of-the-art (Generalized) Consensus protocols.**

***Keywords*—Generalized Consensus; Locality; Scalability.**

I. Overview of the Achieved Results

Paxos [1] is a very popular protocol for implementing Consensus [2] among participants interconnected by asynchronous networks, even in presence of faults, and it is generally leveraged for building strongly consistent transactional systems (e.g., [3], [4], [5]). Despite its widespread use, Paxos suffers from performance bottlenecks when deployed on networks with a large amount of nodes. As an example, the version of Paxos mostly deployed due to its desirable progress guarantees is Multi-Paxos, where a single designated node (i.e., the elected leader) is responsible for deciding the order of all proposed commands (i.e., client requests). The performance of Multi-Paxos is great until the leader saturates its resources; then the whole system slows down due to the overloaded leader.

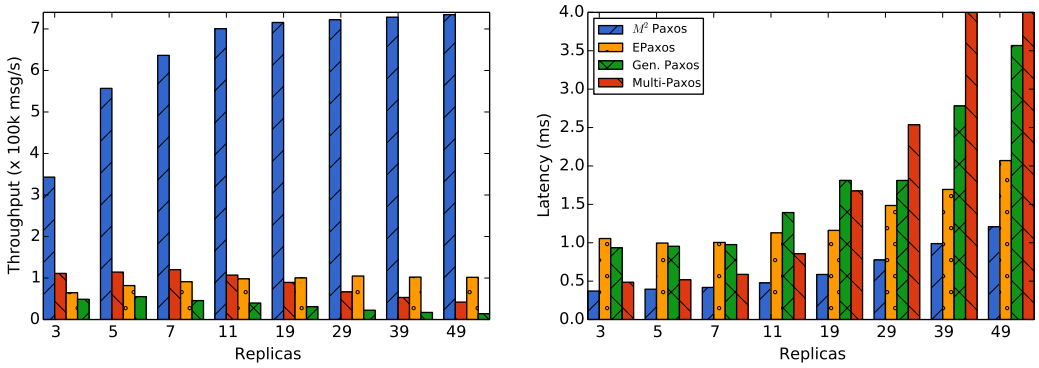
To overcome this limitation, a number of proposals aimed at eliminating the need for a single leader by allowing multiple nodes to operate as a leader at-a-time [6], [7], [8]. However, since multiple leaders now compete for the decision of their concurrent commands, those protocols relax the requirement of providing a unique agreement on all proposed commands, and instead they build a set of partial orders where only conflicting commands are totally ordered. Such solutions provide implementations of Generalized Consensus [9], [10], and they leverage the fact that defining a total order only among non-commutative (conflicting) commands is enough to provide strong consistency on top of replicated storage and transactional systems. However, they come with additional limitations: to avoid a designated leader and to guarantee a fast decision in two communication delays for every uncontended command, the size of the adopted quorums is bigger than the one required by Paxos, i.e., $\lfloor \frac{N}{2} \rfloor + 1$, where N is the total number of nodes. For instance, during some steps in the execution of those protocols, a node might have to wait for replies from $\lceil \frac{3}{4}N \rceil$ other nodes [9], or $\lceil \frac{3}{4}N - 1 \rceil$ other nodes [6], [8]. Also, unlike (Multi-)Paxos, they need to exchange commands' dependencies during the interactions among nodes, because the knowledge of commands' relations is mandatory for the generation of partial orders in Generalized Consensus.

Nevertheless we still agree that avoiding a designated leader with the possibility of reaching consensus in the minimum number of communication delays is fundamental for providing scalability and high performance in current implementations of replicated storage, services, and transactional systems. But we truly believe that current solutions for Generalized Consensus still pay some large but *generally* unnecessary costs, e.g., bigger quorums, processing of dependencies.

In this paper we answer the following question: can we sometimes allow a more expensive agreement process, even in case of no conflicts, in order to guarantee a *generally faster* agreement? Therefore we present M^2PAXOS, an implementation of Generalized Consensus that *generally* decides commands in only two communication delays (i.e., with a fast decision), without exchanging dependencies among conflicting commands and relying on the same quorum size as that used by Paxos, i.e., $\lfloor \frac{N}{2} \rfloor + 1$. We achieve this goal by exploiting the locality of application accesses. One of the widely used techniques for allowing the performance of a system to scale in presence of massive amount of requests is optimizing the placement of the accessed data and the synchronization mechanisms adopted according to the application access patterns [11], [12].

M^2PAXOS does not assume well partitioned accesses but it finds its sweet spot in this scenario. The intuition is the following: when a command is submitted, M^2PAXOS inspects the data to be accessed by the command and determines the *owner* node responsible for ordering the command. This way, all commands accessing the same data will be implicitly ordered by the same node. Clearly, commands accessing different data may have different owner nodes, thus enabling high concurrency. Given this relation between data and owners, and assuming that all data to be accessed by a proposed command is already associated with a unique owner, the decision on that command is a step executed locally at the owner and entails making that decision stable in the system. The latter task is accomplished by paying an optimal latency of two communication delays, namely one delay for sending the message to all nodes and another one for receiving a number of acknowledgments equal to a majority of nodes, thus the same quorum size required by Paxos.

The establishment of the ownership relation between nodes



(a) Throughput (b) Latency

Fig. 1. Scalability plot where command locality is 100%. (a) Maximum attainable throughput. (b) Median latency when the system is underloaded.

and data is part of M^2PAXOS, and it is based on the idea of asynchronous leases previously introduced and explored for transactional systems, e.g., in [13]. Therefore, when a node has to become the owner of a subset of data, it executes a distributed coordination to acquire a lease, i.e., the ownership of that data. Object ownership is asynchronous, meaning that it is released by a node only if it is requested by another node.

II. Evaluation

We implemented M^2PAXOS and all competitors within a unified framework[1], written in the Go programming language, version 1.4rc1. We evaluated M^2PAXOS by comparing it against two Generalized Consensus implementations, i.e., Generalized Paxos [9] and EPaxos [6], and one Consensus implementation, i.e., Multi-Paxos [1]. We used up to 49 nodes on Amazon EC2 infrastructure. Each node is a c3.4xlarge instance (Intel Xeon 2.8GHz, 16 cores, 30GB RAM) running Amazon Linux 2014.09.1, and all nodes were deployed under a single placement group, where the network bandwidth was measured in excess of 7900 Mbps.

To load the system, we injected commands in an open-loop using up to 64 client threads at each node. Commands are accompanied by a 16-byte payload. After issuing each command, a client thread goes to sleep for a configurable amount of time, i.e., think time. To prevent overloading the system, we limited the number of commands still in-flight. The limit is configured for best performance under each deployment, and when it is reached, a node skips issuing new commands. Each data-point represents the average of 10 measurements. We report on M^2PAXOS evaluation under its most favorable conditions: all commands touch a single object, and a command proposed by a node can only conflict with commands proposed by the same node. This scenario is representative for strongly partitioned data, where replication is only employed for fault-tolerance.

We assessed the scalability of all protocols as we scaled the system up from 3 to 49 nodes. For each deployment we gradually increased the workload until the saturation point is reached, and we report these results in Figure 1. From a throughput perspective (see Figure 1(a)), M^2PAXOS observes a 3-7$\times$ improvement when compared to the nearest competitor. It exhibits almost linear scalability up until 11 replicas, and its throughput keeps increasing past 11 nodes, albeit at a slower rate. Multi-Paxos is a distant runner-up at 11 nodes and below. After that, Multi-Paxos' performance degrades, leaving way for EPaxos, which almost manages to maintain its throughput up to the full 49 nodes. Figure 1(b) shows the median command latency with an underloaded system and aggressive batching disabled. With a low number of nodes, the M^2PAXOS narrowly wins over Multi-Paxos, having its latency lower by 23%. As the number of nodes is increased, M^2PAXOS remains the fastest to deliver, with up to 41% better latency than EPaxos.

[1]The code is publicly available at https://bitbucket.org/talex/hyflow-go.

Acknowledgment

This work is supported in part by US National Science Foundation under grant CNS-1217385.

References

[1] L. Lamport, "Paxos made simple," *ACM Sigact News*, 2001.

[2] B. Charron-Bost and A. Schiper, "Uniform Consensus is Harder Than Consensus," *J. Algorithms*, vol. 51, no. 1, pp. 15–37, Apr. 2004.

[3] J. C. Corbett *et al.*, "Spanner: Google's Globally Distributed Database," *ACM TOCS*, 2013.

[4] S. Hirve, R. Palmieri, and B. Ravindran, "Archie: A Speculative Replicated Transactional System," in *Middleware 2014*.

[5] H. Mahmoud *et al.*, "Low-latency Multi-datacenter Databases Using Replicated Commit," *Proc. VLDB Endow.*, 2013.

[6] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is More Consensus in Egalitarian Parliaments," in *SOSP 2013*.

[7] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: Building Efficient Replicated State Machines for WANs," in *OSDI 2008*.

[8] A. Turcu, S. Peluso, R. Palmieri, and B. Ravindran, "Be General and Don't Give Up Consistency in Geo-Replicated Transactional Systems," in *OPODIS 2014*.

[9] L. Lamport, "Generalized Consensus and Paxos," Microsoft Research, Tech. Rep. MSR-TR-2005-33, March 2005.

[10] P. Sutra and M. Shapiro, "Fast genuine generalized consensus," in *SRDS 2011*.

[11] D. Sciascia, F. Pedone, and F. Junqueira, "Scalable Deferred Update Replication," in *DSN 2012*.

[12] S. Peluso, P. Romano, and F. Quaglia, "SCORe: A Scalable One-copy Serializable Partial Replication Protocol," in *Middleware 2012*.

[13] D. Hendler, A. Naiman, S. Peluso, F. Quaglia, P. Romano, and A. Suissa, "Exploiting Locality in Lease-Based Replicated Transactional Memory via Task Migration," in *DISC 2013*.