Flexible Paxos: An Industry Perspective

Max Meldrum
Blekinge Institute of Technology
max@meldrum.se

Supervisors
Emiliano Casalicchio, Blekinge Institute of Technology
Ruwan Lakmal Silva and Pär Karlsson, Ericsson AB

June 6, 2017

Abstract

Paxos is an algorithm for implementing fault-tolerant distributed systems. The core of Paxos is found in many consensus algorithms. Raft and Zab are two prominent protocols that are used in the industry. They serve as the foundation of distributed key-value stores and coordination services such as Consul, Etcd and ZooKeeper. In distributed consensus, the most common way for servers to agree over a value is to use majority quorums. However in 2016, FPaxos (Flexible Paxos) was published. The authors make the observation that majority quorums are not required as intersection is mandatory only across the two phases of Paxos. By taking advantage of this, developers will have more control over the choice between performance and availability. In this paper we will look at how FPaxos can be adopted into existing systems and demonstrate the advantages through a ZooKeeper modification running on City Cloud.

Contents

1 Introduction

Building large-scale distributed systems that are reliable and performant is challenging. In order to make systems highly available we need to replicate the data across servers so that we can tolerate the loss of machines and still continue to serve requests. This is where consensus algorithms play a key role. Paxos [1], one such consensus algorithm has been very influential since its publication. Paxos is a perfect match for applications that needs to have its data available and consistent on multiple machines at the same time. We will not go deep into the Paxos protocol, but rather give a simple introduction of a more efficient version called Multi-Paxos [2] so that the practical implications of FPaxos [3] can be understood. Multi-Paxos allows us to reach agreement between entities while being able to handle failures when they occur. At its heart, there are two phases: in the first phase a leader needs to be chosen. As soon as it has been established, the system is ready for the second phase, called replication phase. In this phase the leader replicates client requests onto other nodes in the cluster. The leader keeps track of connected peers by using heartbeats, if it does not receive one in a certain time it will assume that the connection has been lost. In order to have a functioning system, a majority of the nodes would need to be up and running. This is because both phases require an acknowledgement of [(N/2)+1] nodes. This ensure that as long as a majority of the nodes are functioning, one will be able to learn past commands that were committed. For example, if we have a cluster of five nodes then three of them would always have to be up. The majority scheme is heavily used in consensus protocols that are found in production systems. Zab [4], which Apache ZooKeeper [5] uses, or Raft [6], that is applied by Consul[1] and Etcd[2].

Note: A quorum is the minimum number of votes that a distributed transaction has to obtain in order to be allowed to perform an operation in a distributed system. In this paper, the term refers to the number of votes needed for either the leader election or replication phase.

One of the limitations using a majority quorum system is that if the system needs to scale out with more nodes, there will be a performance penalty

[1]https://github.com/hashicorp/consul
[2]https://github.com/coreos/etcd

as one would have to wait for more replicas to get onboard for the respective phases. FPaxos introduces a more flexible quorum system. What Howard et al.[3] discovered is that majority quorums are not crucial. Instead of demanding that all quorums intersect, it is enough for the quorums from the different phases to intersect. The consequence is that many new quorum systems are now possible. One interesting example is what the authors refer to as Simple Quorum. It gives the developer the option to use different quorums for the two phases as long as the size of both quorums exceed the total number of nodes. Electing a leader is a rare event when compared to replicating data, so by lowering the quorum for the second phase the overall performance is enhanced. This solution also makes it possible to improve availability by choosing the opposite strategy and increasing the quorum for replication.

In the remainder of this paper we revisit FPaxos with the goal of bringing awareness of its practical implications. This is done by implementing FPaxos to a popular open source project and running benchmarks to compare the original system versus the prototype.

2 Research Questions

2.1 RQ1

Which of these three systems is more suitable for an adoption of FPaxos: ZooKeeper, Etcd or Consul?

Motivation: ZooKeeper, Etcd and Consul could be called the big three as they are the most prominent in the industry when it comes to distributed services such as key-value stores, distributed locks and service discovery. ZooKeeper is for example used at: Facebook [7], Yahoo [8] and Rackspace [8]. While Etcd is the backbone of a popular container orchestration engine called Kubernetes [9]. Raft (Etcd, Consul) is based on Multi-Paxos and Zab (ZooKeeper) share similarities with Multi-Paxos. Therefore adopting the more flexible quorum system offered by FPaxos should be feasible. Depending on the complexity of modifying the systems mentioned above to use FPaxos, companies could find it interesting as it allows for more control and flexibility over the consensus algorithm.

2.2 RQ2

What are the advantages in performance when using a FPaxos adopted system versus the original consensus algorithm?

Motivation: By comparing the performance of the two we contribute to the community and show that applying FPaxos is of benefit.

2.3 RQ3

Is the FPaxos prototype production ready?

Motivation: Does the FPaxos adopted system lack anything in correctness compared to the original implementation? Can it handle the typical workload seen in production environments? By answering these questions we validate the full ability of the prototype.

3 Method

To answer the research questions, we have first investigated which system that is more appropriate for an extension of FPaxos. FPaxos requires splitting up the quorum variable into two, one for the leader election and the other for replication. The Raft protocol has an additional election restriction that ZooKeeper does not have. In section 5.4.1 in [6], it is explained that Raft prevents any candidate from winning a leader election unless it has the latest committed entries. This means that for FPaxos to be adopted to Raft, the leader election quorums have to intersect [10]. In ZooKeeper, it is enough by having the two different phases intersect. Hence why it was chosen over Etcd And Consul for RQ1. Secondly, we have modified ZooKeeper to use FPaxos. By running experiments and comparing the performance between the original implementation and the prototype, we are able to answer RQ2 and RQ3.

3.1 ZooKeeper

ZooKeeper is an open source project under the Apache Software Foundation. It can be described as a distributed coordination service that is used by other distributed applications. It runs on the JVM and is developed using the programming language Java. ZooKeeper has a hierarchal data model similar to the one used in the UNIX file system.

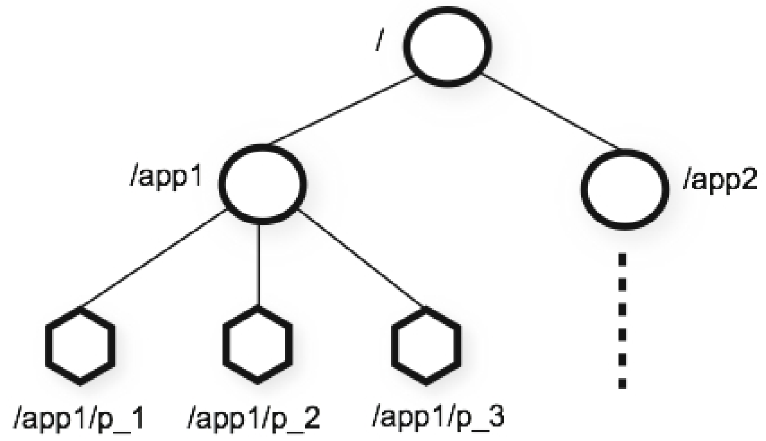


Figure 1: Overview of the ZooKeeper data model [11]

Each node in the tree structure represents a znode (ZooKeeper node) which is a data register comparable to files and directories. When a client requests something from the ZooKeeper ensemble[3], it is often an operation on a znode. The following operations can be executed: CREATE, SET, DELETE and READ. ZooKeeper ensures that each node in the ensemble has the same version of znodes. The data stored in a znode is kept in-memory, allowing the system to serve very fast reads. As with Multi-Paxos, ZooKeeper has a phase for leader election and replication (Atomic Broadcast) where majority quorums are utilised. However, even though they share similarities, these two protocols are clearly different. In ZooKeeper there are two roles, leader and follower. The former is responsible of managing and propagating incoming state changes, both to followers and itself. Followers receive write requests from clients and forward the new state change to the leader, which then broadcasts the transaction[4] to all the followers.

[3]The name ZooKeeper uses to refer to a cluster of nodes

[4]Client state change that is broadcasted

3.2 Zab

ZooKeeper Atomic Broadcast is the protocol that ZooKeeper relies on for crash recovery, propagating state changes to replicas and leader election.

Zab Guarantees [12]

- Reliable delivery

  If a message, m, is delivered by one server, then it will be eventually delivered by all correct servers.

- Total order

  If message a is delivered before message b by one server, then every server that delivers a and b delivers a before b

- Causal order

  If message a causally precedes message b and both messages are delivered, then a must be ordered before b.

At the core of Zab, there are three phases: discovery, synchronization and broadcast.

Phase 1 - Discovery: Before the Zab process enters the discovery phase, it executes the leader election algorithm. At the start of the phase, the decision is made whether the peer should become leader or follower. The leader election is thus sometimes called Phase 0 [13]. The purpose of the first phase is for the leader to gather information on most recent transactions.
Phase 2 - Synchronization: In this phase Zab handles the synchronisation of all replicas, making sure that the followers have the same history as the elected leader. It also enables recovery, as peers can crash and get back on track through the synchronization.
Phase 3 - Broadcast: In the broadcast phase the system is considered stable and the requests written by clients are turned into transactions.

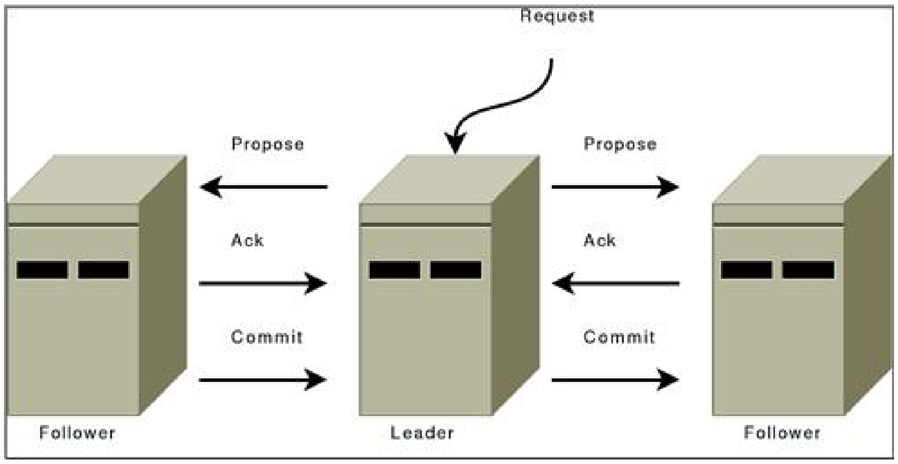


Figure 2: ZooKeeper messaging [14]

As seen in Figure 2, the phase works in a similar way to a two-phase commit. The leader sends out a proposal to all the followers in the ensemble. The leader receives an ACK from a follower when it has recorded the proposal to persistent storage. When the leader has received ACKs from a majority of the ensemble, it commits the transaction.

For a more in depth analysis of Zab and its phases, we refer to [4, 13].

3.3 FPaxos Adaptation

The source code of the modified ZooKeeper implementation utilised in this thesis is hosted on Github and can be found here[5].

In this section we will look at how FPaxos can be applied to a ZooKeeper[6] implementation. ZooKeeper uses majority quorums by default but also supports weighted voting [15] and hierarchical quorums [16]. ZooKeeper 3.5.2-alpha was selected as the development version. As mentioned earlier, to apply FPaxos in ZooKeeper, it is necessary to split up the quorum variable into two. We borrow the terminology from the FPaxos paper [3] for this next part. The amount of quorums required for leader election is specified as Q1 and Q2 for the atomic broadcast. We need to differentiate between the

[5]https://github.com/Max-Meldrum/zookeeper
[6]https://github.com/apache/zookeeper

quorum checks in the Zab protocol. The following is the original majority check method in ZooKeeper which is used for both phases:

```
// Majority
public boolean containsQuorum(Set<Long> ackSet) {
    return (ackSet.size() > half);
}
```

In the modified version, we split the method into two.

```
// Q1
public boolean containsElectionQuorum(Set<Long> ackSet) {
    return (ackSet.size() >= electionQuorum);
}
// Q2
public boolean containsAtomicBroadcastQuorum(Set<Long> ackSet) {
    return (ackSet.size() >= atomicBroadcastQuorum);
}
```

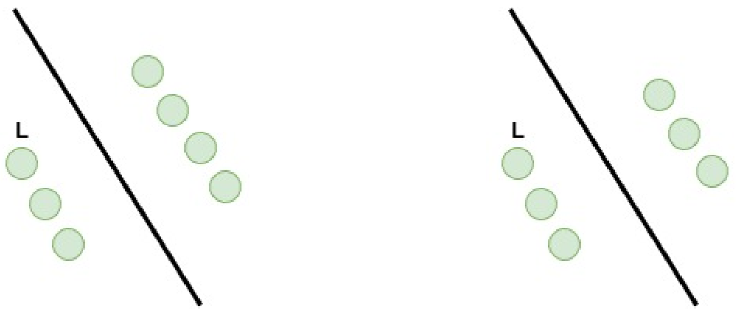
For FPaxos to work together with Zab, the following checks are now required:

Leader election: When the prospective leader is checking if it has enough followers to form a quorum, we now require it to check against Q1.

Atomic broadcast: When the leader awaits ACKs from followers we now require it to check against Q2.

Leader sync: The leader periodically checks if it is connected to a quorum of followers. We now require it to check against Q2. The reason for this is that if the leader is still alive and it is possible to form a quorum for Q2, the system can continue to function as long as replicas are recovered until a new leader election is required.

3.3.1 Correctness and Safety



(a) 7 node split (b) 6 node split

Figure 3: Network partitions

If a system applying majority quorums is subject to a partition, it will still be able to operate as long as one of the sides can acquire a majority. In figure 3a, the left side will be unable to function as it would need 4 nodes to find a majority. The right side, however, will be able to elect a new leader for the ensemble. The FPaxos prototype is also able to withstand network partitions. Let's assume that the quorum setup for figure 3a is Q1=5 and Q2=3. The left side of the split will still be operating as it still meets the Q2 quorum requirement. But, if it loses one more node the system will lock up. In figure 3b a normal majority-based system would not be able to handle the partition as a majority of 6 is 4. In a system running FPaxos with Q1=4 and Q2=3, which side the leader ends up on will never prevent the operation (replication) to continue.

However, it is important to note that this system might instead increase the quorum for replication and decrease the leader election quorum. It is therefore possible to end up with a partition that is able to elect a leader but not capable of replicating.

ZooKeeper by design does not offer strong read consistency. In order to ensure high read-availability, stale read returns are allowed. ZooKeeper provides an API call named sync which will make sure that the latest version of the data is being read. More on this can be found in [5], section 4.4. This has however no implication on the FPaxos implementation.

The guarantees of Zab that are described in section 3.2 are still valid in the new FPaxos adopted version.

3.4 Benchmark Setup

In this section we describe the tools and environment that has been used to evaluate the performance (latency and throughput) of the original ZooKeeper 3.5-2 alpha and the adopted FPaxos version. The results from the benchmarks can be found here[7]. To perform the benchmarks two tools have been used, zookeeper-benchmark[8] and zk-smoketest[9]. Credit goes out to the authors who created such great tools.

zookeeper-benchmark: This tool will be used to measure the throughput over time. We chose to apply the default configurations offered by the project and therefore execute it in a synchronous fashion.

zk-smoketest: This tool was used to analyse the latency and throughput of different znode operations such as CREATE, SET, GET and DELETE.

ZooKeeper ensemble setup: To perform the benchmarks, City Cloud (public cloud based on OpenStack) was used to set up virtual machines in a data center in Stockholm, Sweden. The following are specifications of the virtual machines that were utilised in the ZooKeeper ensemble:

- Debian 8.7.2, 2 Cores, 2GB RAM and 20GB Disk.
- Java: Oracle 1.8.0_121
- Heap size 512m (25% of the total 2048m)

A dedicated log for each server was put in place as recommended for production environments. In addition, the benchmarks were executed on a separate machine with the following specifications: Debian 8.7.2, 2 Cores, 4GB RAM and 20GB Disk.

[7]https://github.com/Max-Meldrum/zookeeper-fpaxos-benchmarks
[8]https://github.com/brownsys/zookeeper-benchmark
[9]https://github.com/phunt/zk-smoketest

4 Related Work

The performance of FPaxos [3] was previously evaluated using a modified version of LibPaxos3 [17]. Similar to the situation in this paper, the full potential is not shown as both LibPaxos3 and ZooKeeper send transactions to all the replicas. Very little follow-up work has been published on FPaxos. As we were writing this work, WPaxos [18] was published. It is a version of Paxos aimed at wide area networks. It utilises flexible quorums from FPaxos to achieve high-throughput and low latency. Additionally, Trex-paxos [19], an implementation of Multi-Paxos on the JVM, added support for flexible quorums in its 0.2 release. Trex-paxos is a project in progress and not yet ready for production. It is in any case very interesting to see that projects are starting to adapt FPaxos. In the ZooKeeper paper [5] several performance evaluations can be found where different workloads are encountered. Similar to this paper, a strong focus is put on throughput and latency performance.

5 Results

5.1 RQ1

It is viable to adopt FPaxos to ZooKeeper. In section 3 we explain why ZooKeeper was most suited for this experimentation. Although Etcd and Consul requires the leader election quorums to intersect, it is still feasible to apply FPaxos. It is worth noting a few points:

- ZooKeeper will always by design commit transactions to all its followers. This is not mandatory for FPaxos. But, in the case of ZooKeeper, this is something that is wanted as it ensures high read-availability.
- One can maintain ZooKeepers read-availability while increasing the systems steady-state performance. This is done by lowering the amount of ACKs the leader has to wait for before committing the transaction.

5.2 RQ2

For the performance evaluation we will look into a situation where a majority quorum system is weak and how FPaxos can fix it. In practice, nodes are usually deployed in cluster sizes of three, five or seven. This is because when using a majority quorum system, there is no redundancy benefit in using an even number of servers. Let us assume that we either were given six machines or that we want to set two machines into three different racks, data centers or zones. If a majority system is installed it would require a quorum of $[(6/2)+1]$ nodes for both phases. By applying FPaxos it is possible to lower the quorum for replication by one and end up with $Q1=4$, $Q2=3$ and still satisfy $|Q1| + |Q2| > N$. We now achieve the write performance of a five node cluster and tolerate two failures at minimum and three at maximum. This is because as described earlier, as long as it is possible to form the quorum for the replication phase and to recover nodes until a new leader election is required, requests will still be served.

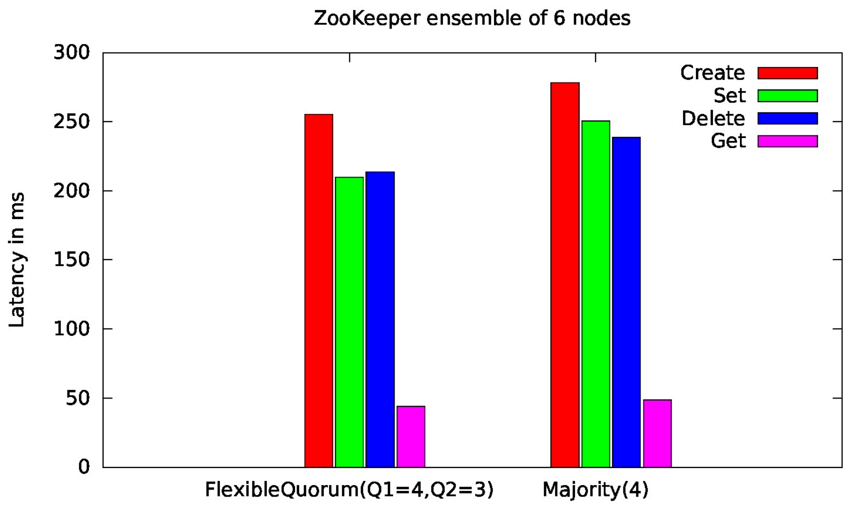


Figure 4: zk-smoketest - 100 requests per operation

The results from Figure 4 were taken from the average of ten runs. Each znode held a size of 100 bytes and the tests were executed synchronously. As expected, the latency for operations that require to go through the atomic broadcast (Create, Set and Delete) was decreased. The Get operation, being served locally, will always output a similar latency.

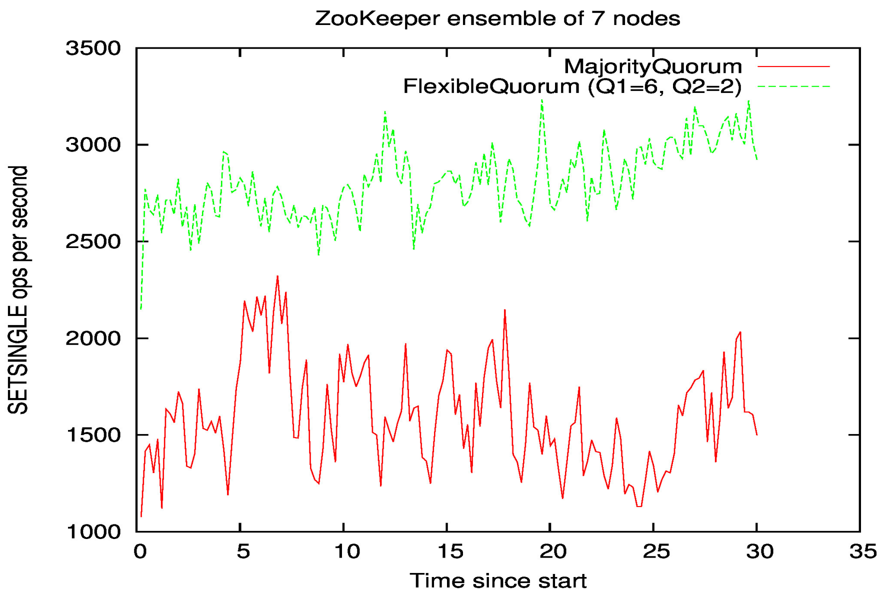


Figure 5: zookeeper-benchmark - repeated writes to a single znode

In figure 5 we demonstrate the use of the quorum system named Simple Quorum. In this case a ZooKeeper ensemble of seven nodes is used. Six nodes are required for the leader election phase while only two are needed for the atomic broadcast. The green curve shows the write performance of a 3 node ensemble as only 2 ACKs are required. As long as the leader and another node is up, loss of minimum 1 node and maximum 5 nodes still allows replication. The red curve shows the write performance of a majority quorum of 4 nodes. Most often when it comes to services such as ZooKeeper, availability is more crucial than write performance. So by attempting to gain performance at the cost of availability may not always be optimal.

We conclude that the performance advantage of applying FPaxos to ZooKeeper is that it is possible to achieve higher throughput and decreased latency.

5.3 RQ3

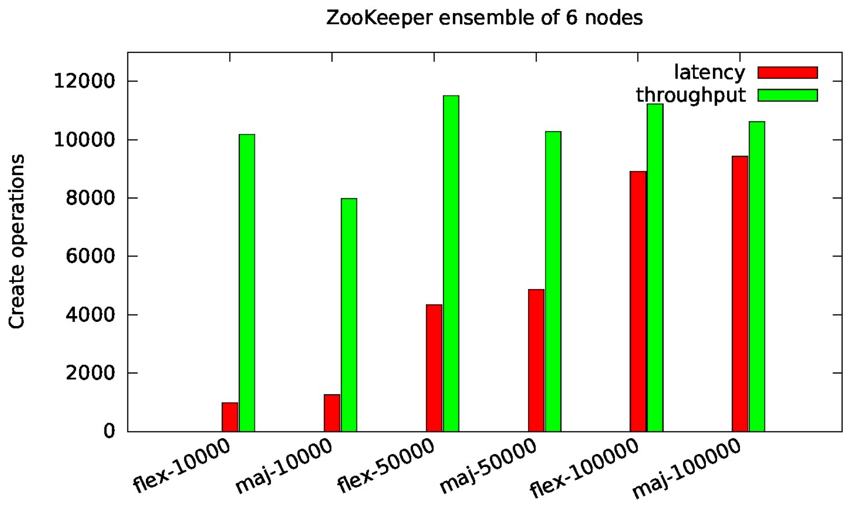


Figure 6: zk-smoketest - asynchronous create operations

Figure 6 shows the latency and throughput (req/s) for various workloads when using the same quorum setup for a six node ensemble (see section 5.2). We used 10000, 50000 and 100000 write requests as test cases. Flex has the quorum setup of Q1=4 and Q2=3, while maj is a majority quorum of 4. The size of each created znode is 100 bytes. The prototype is able to handle the same workload pressure as the original and at the same time experience higher throughput and decreased latency. In section 3.3.1 we conclude that the prototype is safe and that it can handle network partitions, keep Zab guarantees and maintain ZooKeeper's high read-availability.

The prototype is not battle-tested, and therefore should only be used as an experimental system until it has seen more exposure.

6 Conclusion

Distributed services such as ZooKeeper, Etcd and Consul can be found in cloud infrastructures of many organizations. In this paper we draw the conclusion that of the three services, ZooKeeper is the most suitable for adding FPaxos support. We successfully apply FPaxos to ZooKeeper. By utilising Simple Quorum when the number of nodes are even, we show that it is possible for ZooKeeper to achieve increased performance and at the same time tolerate more failures. We conclude that the modified version is able to handle high workload and maintain the correctness and safety of the original implementation. ZooKeeper is used by companies such as Facebook, Yahoo and Rackspace. Several popular open source projects such as Mesos, Kafka and Spark rely on ZooKeeper for coordination. As far as we know, this is the first publicly published contribution to add FPaxos to ZooKeeper.

7 Future Work

In this paper we have shown that it is possible to reduce the phase two quorum by one when the number of servers is even and how the quorum system named Simple Quorum can be used. Many more quorum systems could be applied. The grouped example described by Howard (main author of FPaxos) in her blog post [10] sounds promising. If we can instruct the consensus protocol to be more specific of which nodes can form quorums and thereby making it easier for quorums to intersect, the quorum for both phases can be reduced. In the blog post mentioned above this is done by putting nodes into groups and requiring them to intersect in certain ways. ZooKeeper supports hierarchical quorums where servers can be split in to groups. By using the implementation of hierarchical quorums, it should be trivial to modify the prototype so that it can support the grouped quorum system. Dynamic reconfiguration was added to the 3.5.0 release of ZooKeeper. Which for instance makes it possible to reconfigure quorum values during runtime, add nodes and delete nodes. This type of feature has been seen before in Vertical Paxos [20]. By applying FPaxos and dynamic reconfiguration together, ZooKeeper could be enhanced even more. Further investigation of applying FPaxos to Raft needs to be done. Finally, to enable the integration of FPaxos in future versions of ZooKeeper, it must be able to support both the existing quorum systems as well as FPaxos.

8 Acknowledgements

I would like to thank Ruwan Lakmal Silva and Pär Karlsson at Ericsson for bringing FPaxos to my attention, supporting me with their guidance, and giving me access to a CityCloud account. I would also like to thank Emiliano Casalicchio for supervising my thesis at Blekinge Institute of Technology.

References

[1] Leslie Lamport. The part-time parliament. May 1998.

[2] Leslie Lamport. Paxos made simple. pages 51–58, December 2001.

[3] Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman. Flexible paxos: Quorum intersection revisited. *CoRR*, abs/1608.06696, 2016.

[4] Flavio P. Junqueira, Benjamin C. Reed, and Marco Serafini. Zab: High-performance broadcast for primary-backup systems. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks*, DSN '11, pages 245–256, Washington, DC, USA, 2011. IEEE Computer Society.

[5] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[6] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 305–320, Berkeley, CA, USA, 2014. USENIX Association.

[7] Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, Rodrigo Schmidt, and Amitanand Aiyer. Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1071–1080, New York, NY, USA, 2011. ACM.

[8] Apache Foundation. Apache zookeeper, powered by. `https://cwiki.apache.org/confluence/display/ZOOKEEPER/PoweredBy`, 2014. Accessed: 2017-04-27.

[9] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *ACM Queue*, 14:70–93, 2016.

[10] Heidi Howard. Majority agreement is not necessary for consensus. `http://hh360.user.srcf.net/blog/2016/08/majority-agreement-is-not-necessary`, 2016. Accessed: 2017-05-04.

[11] Apache Foundation. Zookeeper overview. `https://zookeeper.apache.org/doc/trunk/zookeeperOver.html`, 2016. Accessed: 2017-04-15.

[12] Benjamin Reed and Flavio P. Junqueira. A simple totally ordered broadcast protocol. In *Proceedings of the 2Nd Workshop on Large-Scale Distributed Systems and Middleware*, LADIS '08, pages 2:1–2:6, New York, NY, USA, 2008. ACM.

[13] Andre Medeiros. Zookeeper, atomic broadcast protocol: Theory and practice. `http://www.tcs.hut.fi/Studies/T-79.5001/reports/2012-deSouzaMedeiros.pdf`, 2012. Accessed: 2017-04-16.

[14] Apache Foundation. Zookeeper internals. `https://zookeeper.apache.org/doc/r3.5.2-alpha/zookeeperInternals.html`, 2016. Accessed: 2017-04-15.

[15] David K. Gifford. Weighted voting for replicated data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, SOSP '79, pages 150–162, New York, NY, USA, 1979. ACM.

[16] Akhil Kumar. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Trans. Comput.*, 40(9):996–1004, September 1991.

[17] Daniele Sciascia. Libpaxos3, multi-paxos. `https://bitbucket.org/sciascid/libpaxos`, 2016. Accessed: 2017-04-28.

[18] A. Ailijiang, A. Charapko, M. Demirbas, and T. Kosar. Multileader WAN Paxos: Ruling the Archipelago with Fast Consensus. *ArXiv e-prints*, March 2017.

[19] Simon Massey. Multi-paxos for the jvm. `https://github.com/trex-paxos/trex`, 2017. Accessed: 2017-04-29.

[20] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical paxos and primary-backup replication. Technical report, May 2009.