ESLE 21/22 Final Project Report - Group 5

João Caldeira[93729] - `joao.ferreira.caldeira@tecnico.ulisboa.pt`, Diogo Santos[91054] - `diogo.bernardo.dos.santos@tecnico.ulisboa.pt`, and Sebastião Sotto Mayor[102146] - `sebastiao.sotto.mayor@tecnico.ulisboa.pt`

Instituto Superior Técnico, Taguspark 2744-016, Portugal

Abstract. Analysis of Zookeeper scalability and performance characteristics under specific YCSB workloads.

Keywords: Zookeeper · YCSB · Scalability · Performance · Bottleneck.

1 Introduction

The goal of this paper is to analyze Zookeeper scalability and performance characteristics by benchmarking the system using specific YCSB workloads discussed in the following sections.

1.1 System Presentation

ZooKeeper is a distributed, open-source coordination service for distributed applications. It exposes a simple set of primitives that distributed applications can build upon to implement higher level services for synchronization, configuration maintenance, groups and naming. [1]

1.2 System Choice

Zookeeper was chosen for this project because of its core qualities. Primarily, it is distributed and has a client-server architecture, which makes it a very interesting system to study. Also, it is open-source, thus facilitating the process of familiarization.

The Git commit to consider for evaluation purposes can be found in this link.

2 System Description

In this section, the main system characteristics of Zookeeper are discussed, along with the experiment parameters and environment. Also, the choice for the submitted workload is explained.

2.1 Main System Characteristics

Except for the Request Processor, each of the servers that make up the ZooKeeper service replicates its own copy of each of the components. As long as a majority of the servers are available, the ZooKeeper service will be available. Clients connect to a single ZooKeeper server. The client maintains a TCP connection through which it sends requests, gets responses, gets watch events, and sends heart beats. If the TCP connection to the server breaks, the client will connect to a different server.

Replicated Database The replicated database is an in-memory database containing the entire data tree. Updates are logged to disk for recoverability, and writes are serialized to disk before they are applied to the in-memory database.

Clients Every ZooKeeper server services clients. Clients connect to exactly one server to submit n requests. Read requests are serviced from the local replica of each server database. Requests that change the state of the service, write requests, are processed by an agreement protocol.

Servers As part of the agreement protocol, all write requests from clients are forwarded to a single server, called the leader. The rest of the ZooKeeper servers, called followers, receive message proposals from the leader and agree upon message delivery. The messaging layer takes care of replacing leaders on failures and syncing followers with leaders.

Messaging protocol ZooKeeper uses a custom atomic messaging protocol. Since the messaging layer is atomic, ZooKeeper can guarantee that the local replicas never diverge. When the leader receives a write request, it calculates what the state of the system is and when to apply the write operation, and transforms this into a transaction that captures this new state. [1]

Leader Election All the nodes create a sequential, ephemeral znode with the same path, /app/leader_election/guid_. ZooKeeper ensemble will append the 10-digit sequence number to the path (e.g., /app/leader_election/guid_0000000001). For a given instance, the node which creates the smallest number in the znode becomes the leader and all the other nodes are followers. Each follower node watches the znode having the next smallest number. If the leader goes down, then its corresponding znode /app/leader_electionN gets deleted. The next in line follower node will get the notification through watcher about the leader removal. The next in line follower node will check if there are other znodes with the smallest number. If none, then it will assume the role of the leader. Otherwise, it finds the node which created the znode with the smallest number as leader. Similarly, all other follower nodes elect the node which created the znode with the smallest number as leader. [2]

2.2 Experiment Parameters

The metric used was the throughput of the system (operations per second) because this is the best metric to evaluate the system's performance and scalability. The team also considered latency, but later verified that this metric was not relevant as it was not sensitive to changes in the system, and it maintained stable during the experiments.

First Stage The first stage experiments were conducted in six steps. In the first step, one Zookeeper server and one YCSB client were used. Each time, the servers and the clients are increased proportionally (i.e., 1, 2, 3, 4, 6, 8), in order to analyze the system's scalability and performance properties. For two or more servers, the throughput collected to build the graphs is an average of the results obtained in the respective clients.

Second Stage In the second stage, six factors were considered for the experiments. Those were:

- A-Machine network
- B-Size of ZNodes
- C-Number of server replicas
- D-Sync limit
- E-CPU limitation
- F-Memory limitation

From these six factors, only the first three were selected to be used in the experiments because of limited resources. For each one of them, two levels of experiments were conducted. The levels were as follows:

- Machine network
  - level -1: Powerline network (38.8 DL and 20.6 UL)
  - level 1: Wired network (158.6 DL and 20.9 UL)
- Size of ZNodes
  - level -1: 1MB ZNodes (Zookeeper default)
  - level 1: 4MB ZNodes
- Number of server replicas
  - level -1: 1:1
  - level 1: 1:2

Furthermore, from the three selected factors, one was judged as most impactful and was chosen to be examined in more detail. This factor was the number of server replicas, and in order to examine it, the system was scaled in a 1:2 ratio of servers and clients (i.e., 1:2, 2:4, 3:6, 4:8, 5:10, 6:12).

2.3 Experiment Environment

First Stage The main experiment machine had a Linux (Ubuntu 20.04.2 LTS 64 bits) OS, 7.7 GB of RAM, an i7-7500U 2.7ghz×4 CPU, HD graphics 620 GPU, and 512 GB SSD. The deployment consisted of a Docker Swarm service, with n Zookeeper server replicas and YCSB client containers in each step.

Second Stage The setup of the main experiment machine for the second stage was similar to that of the first stage before being scaled. The experiments at scale were conducted in a virtual machine hosted in GCP with a Linux (Ubuntu 20.04.2 LTS 64 bits) OS, 16 virtual CPUs and 64 GB SSD. The deployment consisted of a Docker Swarm service, with n Zookeeper server replicas and 2n YCSB client containers in each step.

2.4 Submitted Workload

The suggested YCSB benchmark workload was judged as adequate, considering the main system characteristics, the experiment parameters and environment mentioned in the subsections above. We started by using a workload of 10000 insert operations however, we saw that this number of operations was not enough to evaluate the system was not used at full capacity, so we had to increase the number of operations. After testing the system we concluded that 20000 operations was the best number to have the system at full capacity. The workload used had a ratio of 95 reads to 5 updates which is the ratio that best represents a real utilization of Zookeeper. The default data size was 1 KB records (10 fields of 100 bytes each, plus key). [3]

2.5 Stage Pipeline

Read and Write Requests Read requests are forwarded to the Request Processor module of the system, while write requests proceed directly to the Replicated Database.

Inter-service Communication The Request Processor sends the request data to the Atomic Broadcast module, where it is transmitted to the Replicated Database.

Responses Responses are produced in the Replicated Database.

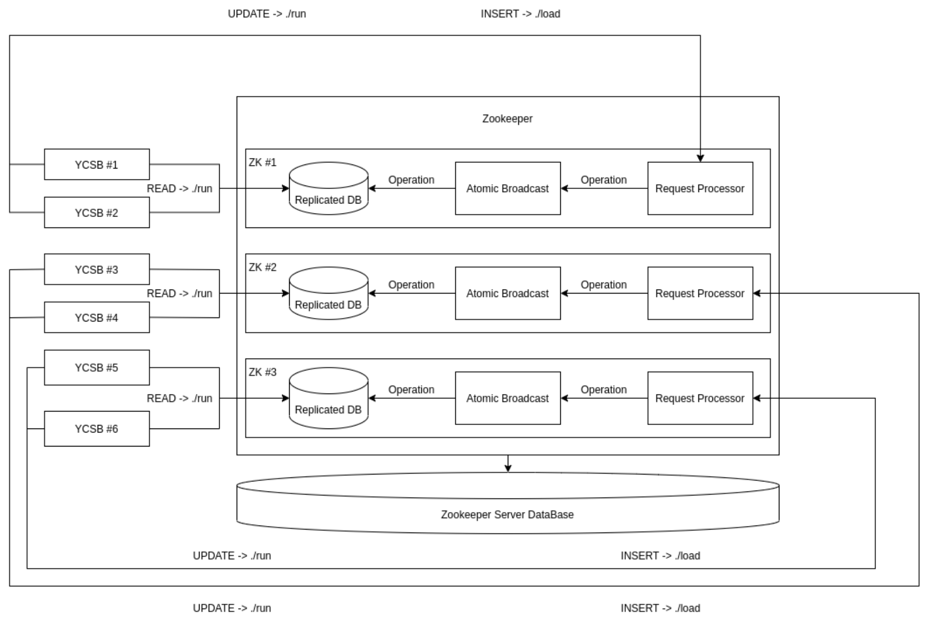


Fig. 1. This figure depicts the system's stage pipeline.

3 Results

First Stage The results obtained are shown in the following table and figure. The table contains important metrics from the conducted experiments, and the figure shows the graph of the Universal Scalability Law and throughput of the system. The results are analyzed in depth in the next subsections.

Table 1. Gathered metrics

Metric	Value
Coefficient of performance	2845.2309970244
Coefficient of serialization	0.2692010226
Crosstalk penalty factor	0.0310961856

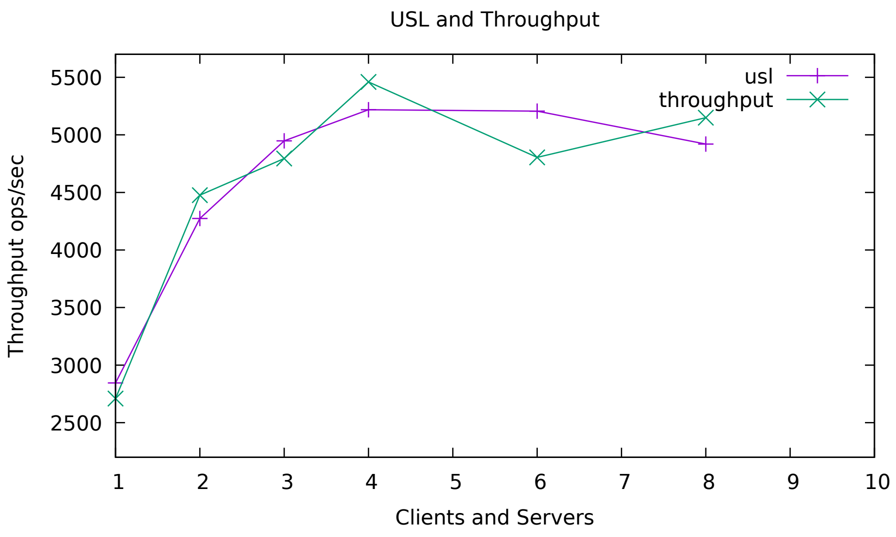
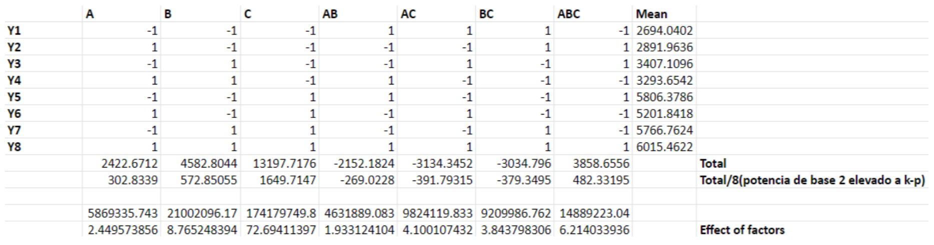


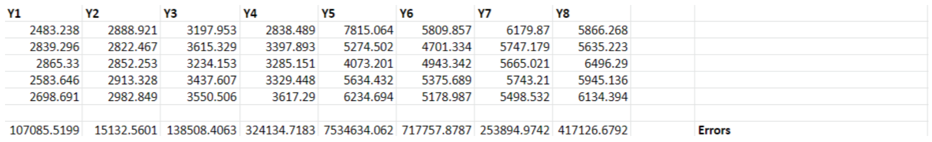
Fig. 2. The x-axis is the clients and servers, and the y-axis is the throughput as operations per second.

Second Stage The sign table with the results of the conducted experiments can be seen in the below figures and tables, along with other important metrics. The last figure shows the graph of the Universal Scalability Law and throughput of the system. The results are analyzed in depth in the next subsections:



	A	B	C	AB	AC	BC	ABC	Mean	
Y1	-1	-1	-1	1	1	1	-1	2694.0402	
Y2	1	-1	-1	-1	-1	1	1	2891.9636	
Y3	-1	1	-1	-1	1	-1	1	3407.1096	
Y4	1	1	-1	1	-1	-1	-1	3293.6542	
Y5	-1	-1	1	1	-1	-1	1	5806.3786	
Y6	1	-1	1	-1	1	-1	-1	5201.8418	
Y7	-1	1	1	-1	-1	1	-1	5766.7624	
Y8	1	1	1	1	1	1	1	6015.4622	
	2422.6712	4582.8044	13197.7176	-2152.1824	-3134.3452	-3034.796	3858.6556		**Total**
	302.8339	572.85055	1649.7147	-269.0228	-391.79315	-379.3495	482.33195		**Total/8(potencia de base 2 elevado a k-p)**
	5869335.743	21002096.17	174179749.8	4631889.083	9824119.833	9209986.762	14889223.04		
	2.449573856	8.765248394	72.69411397	1.933124104	4.100107432	3.843798306	6.214033936		**Effect of factors**

Fig. 3. This figure depicts the sign table of the conducted experiments along with the effects of each factor.



Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8		
2483.238	2888.921	3197.953	2838.489	7815.064	5809.857	6179.87	5866.268		
2839.296	2822.467	3615.329	3397.893	5274.502	4701.334	5747.179	5635.223		
2865.33	2852.253	3234.153	3285.151	4073.201	4943.342	5665.021	6496.29		
2583.646	2913.328	3437.607	3329.448	5634.432	5375.689	5743.21	5945.136		
2698.691	2982.849	3550.506	3617.29	6234.694	5178.987	5498.532	6134.394		
107085.5199	15132.5601	138508.4063	324134.7183	7534634.062	717757.8787	253894.9742	417126.6792		Errors

Fig. 4. This figure depicts the sign table of the conducted experiments along with the errors.

Table 2. Experiment statistics

Metric	Value
Average of Mean	4626.167486
SST	239606400.5

Table 3. Experiment error statistics

Metric	Value
Effect of Errors	3.968289152
Sum of Errors	9508274.799

Table 4. Gathered metrics

Metric	Value
Coefficient of performance	3183.0449249965
Coefficient of serialization	0.2053051581
Crosstalk penalty factor	-0.0159011355

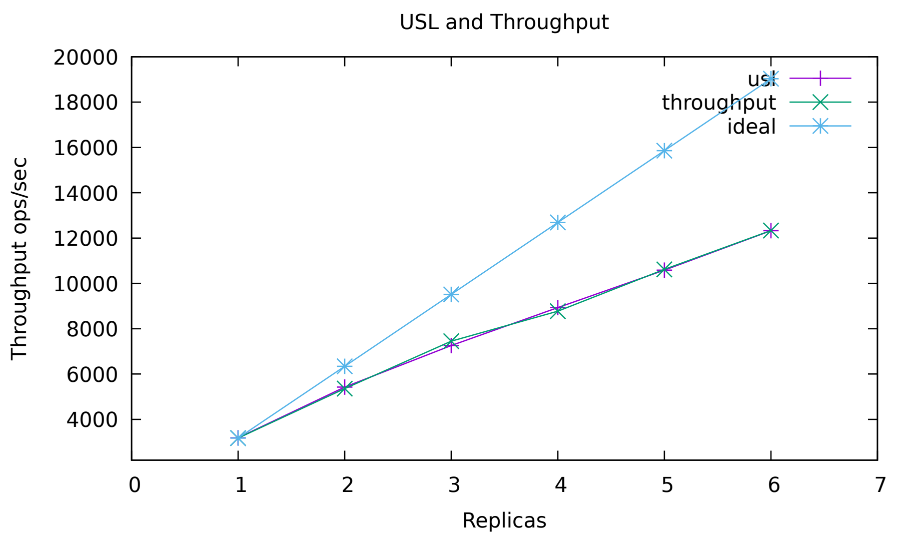


Fig. 5. The x-axis is the server replicas, and the y-axis is the throughput as operations per second.

3.1 Scalability and Performance Properties

First Stage Two important metrics gathered from the first stage experiments are discussed below:

Coefficient of serialization The team observed a significant coefficient of serialization, which means that a real workload has a big serial fraction, and this is a potential bottleneck of the system because having a significant serial portion of the work means that an instruction needs to wait for the instruction ahead to finish before it starts. This means that in significantly large workloads, the system will end up clogged.

Crosstalk penalty factor A significant crosstalk penalty factor was also identified, which was expected due to Zookeeper's architecture, which implies that the servers are all connected and results in a large amount of crosstalk. The system doesn't have a perfect scalability, as can be seen by the USL and throughput plot, reaching its peak at 4 servers.

Second Stage The experiment environment greatly impacted the system scalability and performance, as the main difference between the first and second stages was using Google Cloud Platform to test the system.

Coefficient of serialization The coefficient of serialization decreased, but the difference was not substantial. This variable was consistent with what was seen in the first stage and confirms the problem with the serialization of the workload.

Crosstalk penalty factor The crosstalk penalty factor changed to a negative value, which shows a very different behavior of the system. In the first stage, the team observed that adding replicas would negatively impact the system scalability because it increased crosstalk and that would mean loss of throughput per client and therefore deteriorate scalability. In the second stage, the crosstalk penalty is negative, which means that the increase of replicas will help the system perform better. In this stage, a near perfect scalability was identified.

3.2 System Bottlenecks

It became clear that the main bottleneck was the environment (hardware) where the system was being examined. The first stage experiment and second stage experiment are very identical, as the team ended up confirming that the number of replicas was the factor with the most significant effect in the system. In the first stage the number of replicas had already been scaled, the only difference being, in the first stage there was a 1:1 ratio of replicas and clients and in the second stage there was a 1:2 ratio. In the second stage experiment the system appears to be much more scalable than in the first stage experiment and having 1:2 ratio should have impacted the performance, but even so, better performance was identified. The only assumption that can be made is that the main bottleneck of this system is the hardware of the machine where it is deployed.

4 Conclusion

The team came to specific conclusions regarding the scalability and performance of the Zookeeper system. Those, are discussed in the following subsections.

4.1 Main Insights

The final results showed that the system didn't have perfect scalability, as the last measurement of the experiment was significantly lower than the optimal value. This can be explained by the proposed main system bottleneck, which is the hardware that limits the system performance and scalability, mainly affecting its crosstalk penalty. This can be proved by the large differences between the experiment of the first stage and second stage, where the impactful difference was the different hardware. The other system bottleneck is the big portion of serial workload that is seen in the USL coefficient calculations. The group had tested previously the CPU and Memory limitation factors as main factors, however ended up getting worse effect results than the machine network and size of znode factors, and hence used these last ones as the main factors. The team also identified a big effect of errors, even compared to the chosen factors (excluding the factor with the most effect). This can be explained because of the experiments being executed with ongoing voice calls between the group elements which will affect the machine performance and network usage, and the biggest error inducing factor is, in our view, the lack of automation in the experiments which leads to not having perfect synchronization with all the replicas and clients, meaning that they don't run all at the same time. This happens because when testing, we need to open n number of terminals to test the n/2 number of replicas and n number of clients. This means that we need to have one terminal for each YCSB client and one extra terminal to scale the Zookeeper service and getting the container IPs. To test the system we need to execute the commands manually on every terminal, meaning that the clients don't run all at the same time, and therefore the requests are not all at the same time. So, the scalability and performance results are somewhat more optimistic than what they actually are. In the last experiment, the group observed that the time-lapse between the first and last command executions, was of about 2 seconds. The sign table was built with these experiments, and it was observed that the number of replicas was the factor with the most effect by a significant margin, having about 72.694 percent of calculated effect in the system. Scaling this factor in the last experiment of the second stage helped understand the main system bottleneck.

Finally, the team concluded that the stage pipeline could be further improved by prioritizing common requests. This, would create a fast path to deal with those requests by utilizing the cache, and a slow path to handle the rest of the requests.

4.2 Limitations

Some performance limitations were observed among the machines in the experiment environment, which weighed on the throughput of Zookeeper. Also, the available hardware did not permit scaling with more replicas, thus limiting the experiments to a maximum of 6 server replicas to 12 clients in the last phase.

References

1. ZooKeeper: A Distributed Coordination Service for Distributed Applications, `https://zookeeper.apache.org/doc/r3.1.2/zookeeperOver.html` Last accessed 22 Oct 2021
2. Zookeeper - Leader Election, `https://www.tutorialspoint.com/zookeeper/zookeeper_leader_election.htm` Last accessed 31 Oct 2021
3. A series of tools for ZooKeeper, `https://zookeeper.apache.org/doc/current/zookeeperTools.html` Last accessed 22 Oct 2021