ESLE 21/22 Project Report - Checkpoint 1 - 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

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

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]

2.2 Experiment Parameters

The experiments were conducted in six stages. In the first stage, 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.

2.3 Experiment Environment

The main experiment machine had a Linux (Ubuntu 20.04.2 LTS 64bits) 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 stage.

2.4 Submitted Workload

The suggested YCSB benchmark workload was proven adequate, considering the main system characteristics and the experiment parameters and environment mentioned in the subsections above. More precisely, an initial load of 10000 insert operations was issued, the read and update ratio was 95 to 5, and the default data size was 1 KB records (10 fields of 100 bytes each, plus key). [2]

3 Results

The results obtained are shown in the following table and graph. The table contains important metrics from the conducted experiments, and the graph shows 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	995.8079689984
Coefficient of serialization	-0.0350111598
Crosstalk penalty factor	0.4570330893

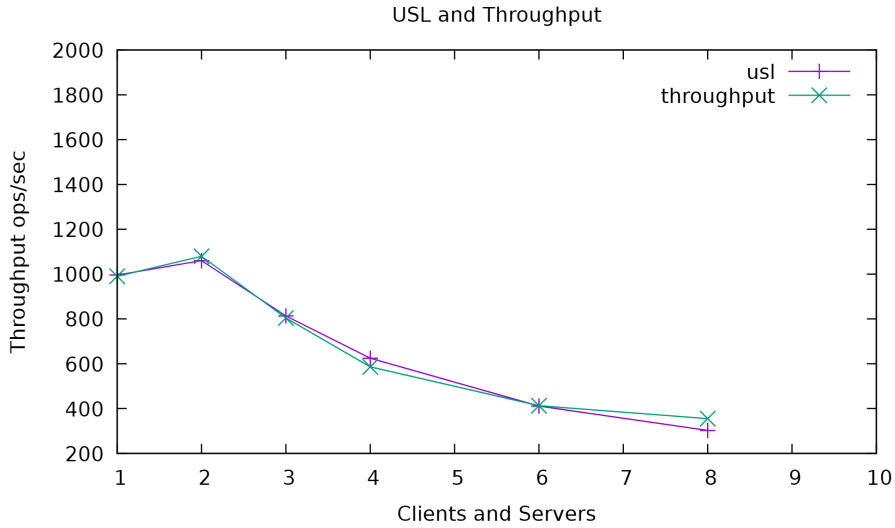

Fig. 1. The x-axis is the clients and servers, and the y-axis is the throughput as operations per second.

3.1 Scalability and Performance Properties

Coefficient of serialization A negative coefficient of serialization was identified. As a consequence, the speedup is negative, and the system tends to perform worse as the number of servers increases. This happens because Zookeeper will always have only one server attached to the client to provide the service and if more servers are added for one client, the result is the same because it only

benefits in the fail case, in which Zookeeper transfers the client to the second replica. Hence, it was decided to have a one-to-one relation between servers and clients. This means that when there are more servers, there also need to be more clients in the experiment environment. Therefore, the Zookeeper service as a whole will be under more pressure and work, resulting in the negative coefficient of serialization and speedup (Amdahl's Law).

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.

3.2 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.

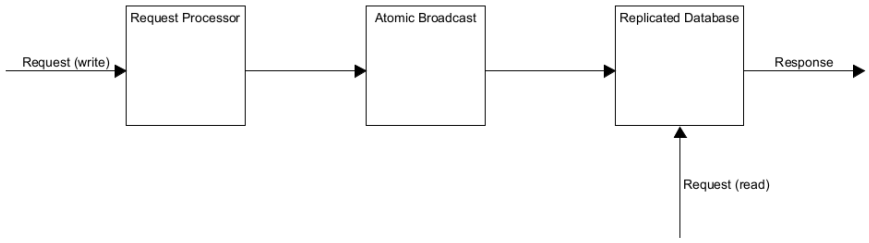

Fig. 2. This figure depicts the system's stage pipeline.

3.3 System Bottlenecks

ZooKeeper is intended to be replicated over a set of hosts called an ensemble. The servers that make up the ZooKeeper service must all know about each other. They maintain an in-memory image of state, along with transaction logs and snapshots, in a persistent store.

It became apparent during the experiments that the main system bottleneck was in the Replicated Database module of Zookeeper. As more servers were added, more load was processed by the Database, resulting in decreased throughput and ultimately poor system performance.

4 Conclusion

4.1 Main Insights

The results showed that as the system scaled, it did not achieve perfect scalability, as the third stage measurement was significantly lower than the optimal value. This can be explained from the crosstalk penalty being quadratic, thus growing very fast, and affecting the system's scalability. The coefficient of performance is satisfactory and behaves as expected. This means that without the large amount of crosstalk in the system, Zookeeper would have better scalability properties. A possible mitigation technique for the identified bottleneck could be the use of dedicated Databases for each server instead of the use of a Replicated Database, to achieve concurrency in the system, thus increasing its throughput.

4.2 Limitations

Significant performance limitations were observed among the machines in the experiment environment, which weighed greatly on the throughput of Zookeeper.

4.3 Future Work

In a later stage of the project, it might be interesting to evaluate the system's scalability and performance measurements in larger scenarios, for example, by being deployed in the cloud, as opposed to the current local experiment environment used.

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. A series of tools for ZooKeeper, `https://zookeeper.apache.org/doc/current/zookeeperTools.html` Last accessed 22 Oct 2021