INF-3203
Advanced Distributed Systems

Assignment 1

Andreas Brunes, Magnus Kanck

23.02.2022

Abstract

Multi-paxos is an improved version of the paxos consensus protocol that attempts to coordinate the sequence of operations between several nodes, or processes, in a network. There are three roles a process can have; *replicas*, *leaders* and *acceptors*. The distribution of these roles can be configured in order to maximize throughput, or to ensure reliability in case of failures. If the possibility of failing processes is fairly small or nonexistent then maximizing throughput becomes the main factor that determines the configuration.

1 Introduction

In this assignment an implementation of the multi-Paxos algorithm will be benchmarked with respect to various configurations such as an increasing number clients sending requests, frequency of sending requests and witch various configurations of leaders, replicas and acceptors. Throughput, which is the rate at which requests are processed, will be measured and an optimal configuration that maximizes the throughput will be found.

2 Technical background

The Paxos protocol is often said to be difficult to understand due to the various roles involved and their different states.[2][3] It is, however, important to understand how the protocol works in order to take measurements on it. The following two sections describes how Paxos, and its derivative Multi-Paxos works.

2.1 Paxos

The Paxos algorithm advances processes in a system to reach a consensus on a value proposed by one or more processes. Consensus is achieved by running the algorithm in two phases, namely the preparation phase and acceptance phase. The preparation phase is initiated by having the proposers index their request with a proposal number. The acceptors can then use this number to differentiate requests and promise to ignore any subsequent messages with a lower index. Essentially, the preparation phase is an election which decides who gets to propose. Once a promise has been made, the next phase can begin. The acceptance phase consists of having the proposer send their proposal to the majority of acceptors along with the highest index which the acceptors promised to receive. Ultimately, once a majority

of acceptors have chosen to accept a proposal, the decision is sent to the learners which will apply the proposal.

2.2 Multi-Paxos

Multi-Paxos is a modification of Paxos that aims to improve its efficiency. Instead of performing the preparation and acceptance phase for each proposal as in Paxos, Multi-Paxos allows a stable process to act as a leader that can perform multiple proposals in succession. The leader is elected by running at least one round of Paxos.

3 Design

For this assignment a finished implementation of Multi-Paxos will be further iterated upon to provide support for taking measurements.[1] Specifically, The time at which a request it gets delivered and the time at which it gets completed (when consensus is reached) will be taken and stored for later use.

A new message type is introduced that notifies the replicas how many requests they should expect to process. The environment will listen to the replicas and initiate an exit procedure that gracefully shuts down the process when all requests have been processed.

Clients that send requests are spawned on parallel threads so that requests can be sent simultaneously. This makes it possible to simulate a real-world environment where multiple communication channels can be active at the same time.

A separate test script has been implemented that configures how roles are distributed, which test to run, how many requests to send, etc. Any interaction with the Multi-Paxos implementation is made made through this script. When a given test is finished the average time and the standard deviation is calculated and plotted as a function of the given variable.

A simple test has also been implemented that performs a single run of the Multi-Paxos protocol and validates that the replicas reached a consensus.

4 Experiments

With respect to throughput bottlenecks, five main factors, or variables, have been identified. Tree of these are the roles a process can take; *leaders*, *replicas* and *acceptors*. In addition, the number of *clients* sending requests, and the total number

of requests to process are also factors. For each type of experiment one of these variables will be incremented while the others remain stable in order to isolate the effect each variable has on the throughput. The total amount of requests sent for each run is about 100 unless stated otherwise.

All tests presented in this section have been run on a computer with the specifications listed in Table 1.

Specifications
CPU: Intel Core i5-7200 CPU @ 2.50GHZ x 4
GPU: Mesa Intel HD Graphics 620 (KBL GT2)GB
RAM: 8 GB
Storage: 256 GB

Table 1: Computer specifications which are used to run tests and profile.

4.0.1 Variable number of clients

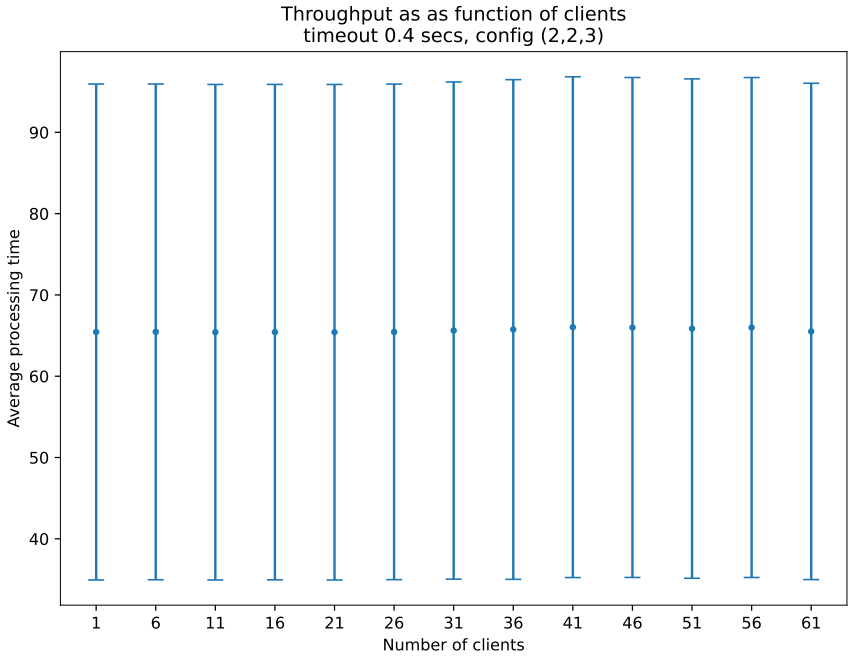

Figure 1: Plot of average throughput and standard deviation in seconds for a variable number of clients.

The timeout between sending requests is set to 0.4 seconds and the configuration of roles is kept constant with 2 *replicas*, 2 *leaders* and 3 *acceptors*.

4.0.2 Variable number of requests

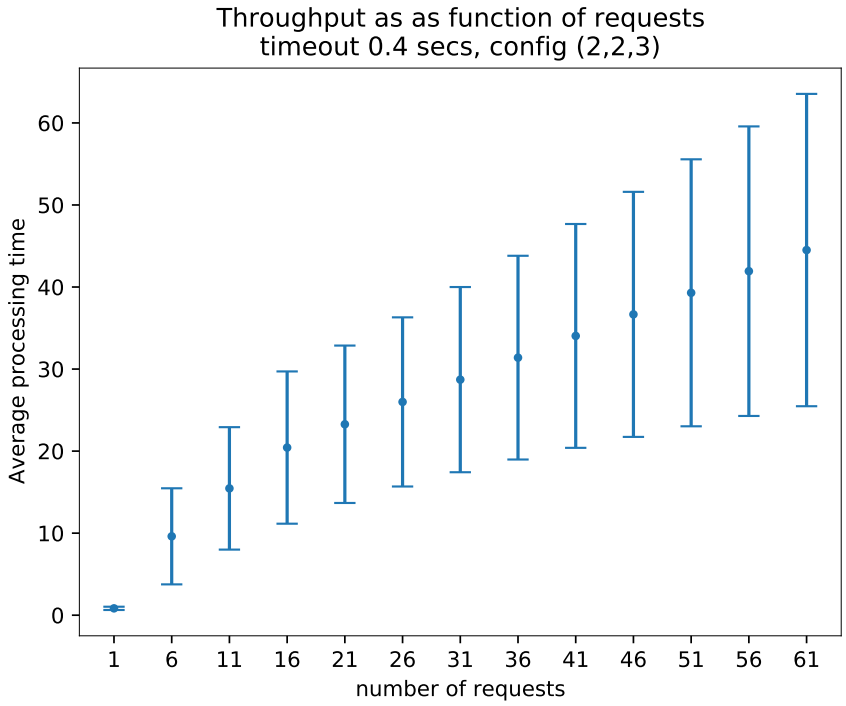

Figure 2: Plot of average throughput and standard deviation in seconds for a variable number of requests, using 3 clients.

The results on the throughput by varying the number of requests is shown in Figure 2. The number of clients is kept constant at 3 for all data points.

4.0.3 Variable number of replicas

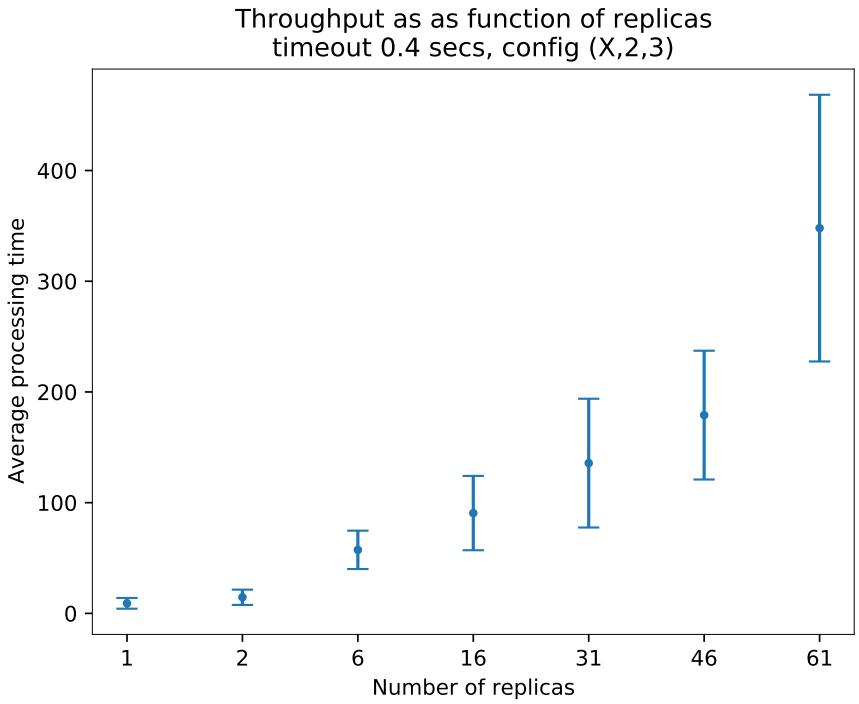

Figure 3: Plot of average throughput and standard deviation in seconds for a variable number of replicas, using 3 clients.

The results on the throughput by varying the number of acceptors is shown in Figure 4. The number of clients is kept constant at 3 for all data points.

4.0.4 Variable number of leaders

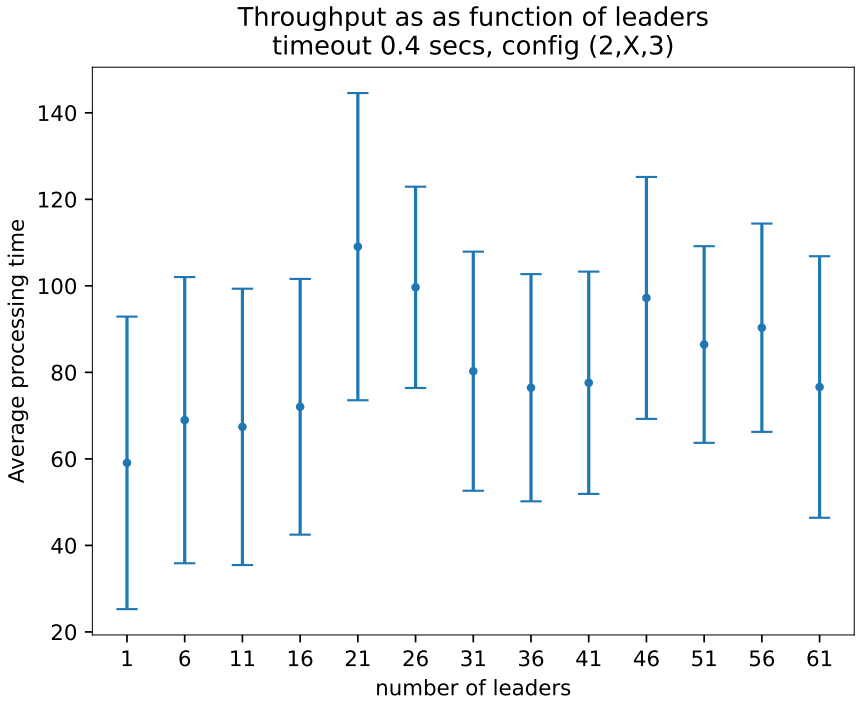

Figure 4: Plot of average throughput and standard deviation in seconds for a variable number of leaders, using 3 clients.

The results on the throughput by varying the number of leaders is shown in Figure 4. The number of clients is kept constant at 3 for all data points.

4.0.5 Variable number of acceptors

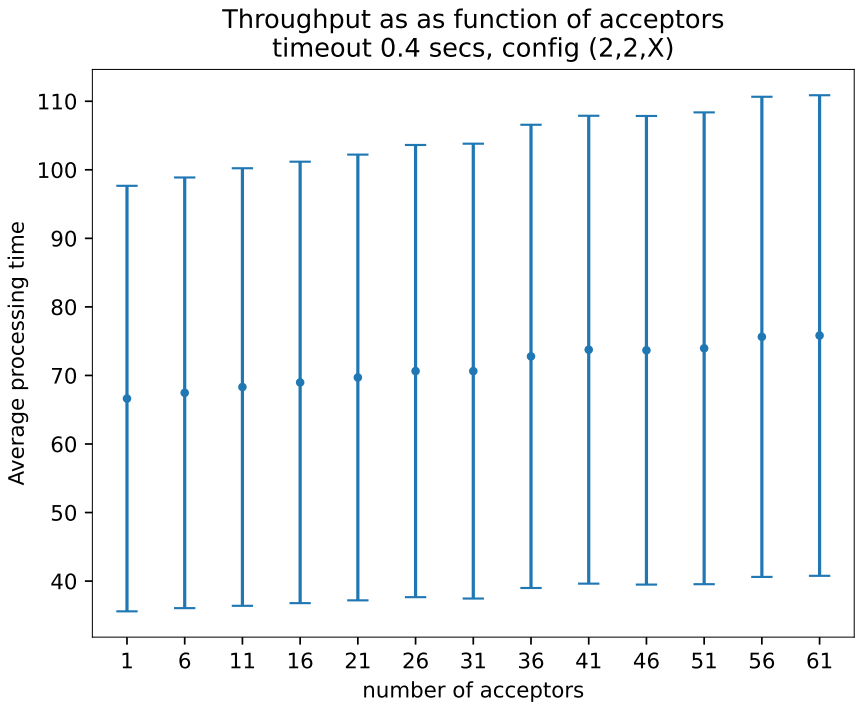

Figure 5: Plot of average throughput and standard deviation in seconds for a variable number of acceptors, using 3 clients.

The effects on the throughput by varying the number of acceptors is shown in Figure 4. The number of clients is kept constant at 3 for all data points.

5 Discussion

In Figure 1 it seems the number of clients simultaneously sending requests have little to no impact on the average throughput. The standard deviation also remains the same. This might be because the total number of requests remains the same (100) no matter how many clients are active. From these results it is reasonable to conclude that the Multi-Paxos implementation is able to handle receiving 61 virtually simultaneous requests well.

The total number of requests being handled has an detrimental effect on the average processing time (Figure 2). The relation between the processing time and the total requests sent seems to be linear.

The average processing time seems to increase exponentially, as seen in Figure 3 when the number of replicas increase. This might be due to communication overhead.

The average processing time varies with the highest process time with number of leaders being 21, 26 and 46 (Figure 4). This could be explained by there being a lot of competing leaders. However, a configuration of 61 leaders had a lower processing time which might be due to a leader being allowed to propose many operations sequentially.

Increasing the number of acceptors has a minor negative effect on the processing time (Figure 5). The negative effect might be due to communication overhead and the increasing difficulty in getting a majority.

The test that measures and plots the throughput as a function of replicas is not functioning properly due to the way in which time measurements are stored after each run. This problem was not fixed due to time constraints. The results shown in Figure 3 were therefore manually stored and a separate script was implemented to plot these results. This is also the reason for there being less data points in this plot.

6 Summary

In this assignment the throughput has been measured for various configurations of the Multi-Paxos protocol. Of the three roles a process can have it has been shown that the role XXXXXXX has the biggest effect on the throughput. Why? In what way?

The optimal configuration of the Multi-Paxos system, when ignoring the possibility of failing processes, is XXX.

References

[1] Deniz Altinbuken. Paxos made moderately complex.

[2] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, December 2001.

[3] Robbert Van Renesse and Deniz Altınbüken. Paxos made moderately complex. *ACM Computing Surveys*, 47, 02 2015.