Reconfiguration in JPaxos

Maria Zerva
mzerva@cs.uoi.gr
Computer Science and Engineering
University of Ioannina

Ioannis Kouvatis
ikouvatis@cs.uoi.gr
Computer Science and Engineering
University of Ioannina

Abstract

The paxos algorithm, provides a way to build fault-tolerant distributed systems, using consensus via message exchange and guarantees safety. Jpaxos is a fully-functional implementation of state machine replication, which is a technique for making services fault-tolerant by replicating them over a group of machines. JPaxos is based on the Multi-Paxos protocol, that allows multiple instances of Paxos to run and is written in Java. Although JPaxos provides multiple functions, like state-transfer, recovery mechanisms and algorithms etc., it does not provide any reconfiguration service. In this paper we present our modification of JPaxos, which we extended to provide a reconfiguration service, that gives the client the ability to require a change to the configuration and choose the replicas that he wants to participate in Paxos. Our implementation uses messages between replicas and state transfer in order to complete this request.

1 Introduction

The need for trustworthy internet services is becoming more and more important over time. The users demand services that are reliable with no loss of information or money during their transactions. Another significant factor, is that of availability. A service is not considered credible, if one day it is available and the other is not.

State machine replication is a very reliable technique, for providing services that are fault-tolerant. What state machine replication does, is that it replicates the service to a number of machines, keeping each replica's state consistent. The latter happens, because the service is deterministic and so each replica executes the same sequence of requests. The physical isolation of machines in a distributed system ensures that failures of server replicas are independent, as required. As long as there are enough of non-faulty replicas, the service is guaranteed to be provided.

There are some systems that implement an efficient state machine replication. One of them which is a Java library and runtime system is JPaxos. We chose to extend JPaxos, as it is a full-fledged, high-performance Java implementation of state machine replication based on Multi-Paxos. JPaxos uses the Paxos algorithm and considers only non-byzantine faults, that is, replicas can only fail by crashing and can never perform actions that are not specified by the algorithm.

Changing the configuration, which means altering the set of machines that replicate a service, is of great importance. In general, replication allows a limited number of failures, so the replacement of failed machines, or machines with a huge load can be critical to the system. Although JPaxos is a fully-functional implementation of state machine replication, it does not support any re-configuration of its nodes. We thought that extending JPaxos and giving it an extra function of replacing the replicas as a service, would be really interesting.

Our main contributions are that we implemented a user requested service, that changes the configuration of JPaxos. Among the modifications we performed are a lot of messages that can be sent between replicas, state transfer of reconfiguration actions between replicas and a different definition of the nodes with extra fields, that were not initially in JPaxos.

The remaining of this paper is organized as follows. Related work that has been conducted in this area is presented in section 2. The background of the Paxos algorithm is given in section 3, while section 4 describes the reconfiguration procedure in Paxos. Section 5 analyses our implementation.

An evaluation and example of our system is shown in section 6. Conclusions and future work are found in sections 7 and 8 respectively.

2 Related Work

Konczak, Santos, Zurkowski, Wojciechowski and Schiper in [1] presented a fully functional state-machine replication Java library, JPaxos, which is based on the Multi-Paxos protocol. JPaxos handles unreliable network with message loss and delays and guarantees that each replica will be eventually up to date. It provides state transfer with the snapshot mechanism and some improvements like pipelining and batching. In order to handle failures in a proper way, they designed four different recovery algorithms to secure the system by tolerating crash-recovery faults. JPaxos was released as an open-source project.

Lorch, Adya, Bolosky, Chaiken, Douceur and Howell in [2] presented SMART, which is a technique for migrating replicated stateful services. SMART allows migrations to remove or replace a non-failed machine, while guarantees no overlap between consecutive configurations. It migrates services that can balance load by overlapping the processing of multiple requests. A new configuration starts with initializing a new replica of the service on each machine in the new configuration, while each replica of each configuration runs a per-configuration instance of Paxos. Until the new configuration is established, the new replicas with the old replicas run concurrently. The algorithm decides if a machine has failed and needs replacement, to prevent the risk of halting a service forever. The authors evaluated the performance of SMART with numerous experiments and found that SMART's ability to overlap processing of multiple requests reduces latency of requests when there are concurrent requests, while the migration has only a small temporary effect.

3 Paxos Protocol

Paxos is a well known family of protocols that solves consensus, which is the state of a group of participants to agree in the same result, in unreliable networks. The Paxos protocol, which was first published in 1989, is named after a a fictional legislative consensus system used on the Paxos island in Greece. Paxos is the basis of state machine replication in distributed systems and guarantees consistency, while there is a very small possibility, that makes Paxos not progress.

The processes in Paxos can have different roles, and a single process may have more than one role simultaneously. The role of the client, is to request some service from the distributed system and wait for an answer. The acceptor, is the voter, that can form a quorum with other acceptors. The proposer is the coordinator, that advocates the client request in order to make acceptors vote for it. The role of the learner, is to learn about an agreement, once there is one, and to sometimes execute or inform the client. A last and very important role, is that of the leader, which is a distinguished proposer, that guarantees the progress of the algorithm.

In general, the paxos algorithm achieves consensus via message exchange, it is asynchronous, with no timing guarantees and can handle a network that delays, while losing but not corrupting message packets. Paxos guarantees the safety, that each replica will eventually agree on a single value.

JPaxos, which we modified and will present in following sections, is based on Multi-Paxos, which is the most common deployment of the Paxos family. Multi-Paxos allows multiple instances of Paxos to run and uses a leader to coordinate an infinite stream of commands. Each command is the result of a single instance of the basic Paxos protocol.

4 Reconfiguration in Paxos

The configuration, in a Paxos protocol, is the set of processes (replicas) that participate in the Paxos algorithm. A certain state machine may allow reconfiguration, which is the process of changing the configuration, or not allow it and only let specified processes to execute the protocol. The reconfiguration function is of great importance, as it can be performed to reduce the vulnerability to further failures after a process has failed or to replace hardware without shutting down the system.

The most common way to implement a reconfigurable state machine, is to let the machine itself order and execute the reconfiguration. This usually happens, when the protocol suspects that a process has failed, after a certain amount of time. The machine chooses the new configuration and then resumes execution with a new state machine that uses the new configuration. This way of reconfiguration is implemented in SMART[2].

Our implementation that achieves the reconfig-

uration of a state machine, handles reconfiguration as a command requested by the client. The system itself does not guess when to execute a change of the configuration, but lets the client choose the replicas, that he wants to participate in Paxos. Then for a certain amount of time, until the new configuration is established, the old with the new processes co-exist until all replicas are up to date and the algorithm then continues its normal course.

5 Implementation

The reconfiguration service, that we implemented, is an extension of the JPaxos Java library and runtime system for state machine replication. JPaxos achieves failure-tolerant user-provided services by replicating them on separate machines and coordinating client interactions with these replicas. The failures of server replicas can be independent by the physical isolation of machines in a distributed system. As long as there are enough of non-faulty replicas, the service is guaranteed to be provided. The replicated service that JPaxos provides assumes deterministic behavior and non-Byzantine failures. It also supports the crash-recovery model, which tolerates message loss and communication delays, which means that after crash, the service can be restarted with the same IP address.

JPaxos' simple API provides to the user a *Service* interface, that the the programmer has to implement with the service code using methods and abstract classes that the library provides, in order to make the service he wants available to the client. It also provides a *Replica* class, that needs instantiation and implements many required functions for each server. Finally the *Client* class is provided, which sends the requests to be executed. JPaxos guarantees that if the client sends a request, it'll eventually get the answer, every replica will execute the request exactly once and any two replicas will execute the requests in the same order.

Jpaxos is a fully distributed system, and thus it implements the Paxos algorithm. Many optimizations to the basic Paxos algorithm are also offered in JPaxos, in order to deliver requests efficiently despite any failure that might happen.

JPaxos contains a configuration file, that defines the nodes (replicas), the crash model, network options etc. In our implementation we transformed this file in order to change the node configuration. Each replica is defined by the process id, the hostname, the port that other replicas use to communicate with it and the port that the client uses for their communication. To create the reconfiguration service, we added an extra field to the nodes' definition, called active, that represents the participation of a replica to the Paxos algorithm. Field active is of type boolean, if a replica is active, this means that the replica will participate immediately to the Paxos protocol, when it is initialized. In the case that a replica is defined as inactive in the configuration file, this means that the replica will be initialized when started, but wait for a join message from the other replicas to join Paxos.

In order to make the system recognize this node definition change, we modified the configuration classes *Configuration* and *PID*, to also contain the active field and handle it properly.

The *Client* class that JPaxos provides sends the request to the replicas and waits for their answer. It contains methods for connection with the replicas and execution of requests. We created a new class that uses JPaxos' *Client* class, handles and launches the reconfiguration request. The reconfiguration command that the client gives consists of three integers, each representing the process id of the replica, that he wants in the new configuration.

The *Replica* class uses the Paxos algorithm to order the client requests and after deciding them it executes them on service, always with the proper order. All the methods from the *Service* class are called from one *Replica* thread, so that no two will be called concurrently on the service. We also modified the *Replica* class in order to take into consideration the active field that we added. When an active replica gets the reconfiguration request, it sends join messages to all replicas in the new configuration and tries to also send a snapshot to the replicas that were initially inactive. The snapshot contains information about decisions of any other reconfiguration services that happened so far. The replica also has to check whether it is in the new configuration, or whether it is not. In the latter case, the replica has to wait for all the other replicas to be ready for the new configuration, and have the most recent state and then it leaves paxos, becomes inactive and waits for a join message to reenter the configuration. The reconfiguration command is given by the *Replica* class to the *ServiceProxy* class, which dispatches it to the *Service*.

As we mentioned before, the reconfiguration that we implemented is in the form of a client-requested service. JPaxos' *Service* interface, requires implementation with a service class that executes all the requests of the clients in a deterministic way. This class also required some additional methods to save and restore its state and all the data that are exchanged between the service implementation class that we created with the *Replica* class is in the form of byte arrays. Extra functions that convert the bytes to integer or objects in both classes where also implemented.

6 Evaluation

In order to evaluate the reconfiguration modification of JPaxos that we implemented, we performed numerous experiments in a single machine, where each replica and client are executed as threads. These experiments took place for different number of replicas (default number in JPaxos was three) and different number of active and inactive ones.

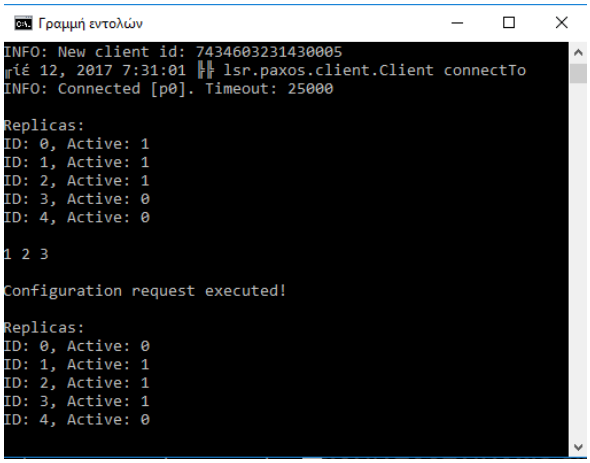

Figure 1. Client thread.

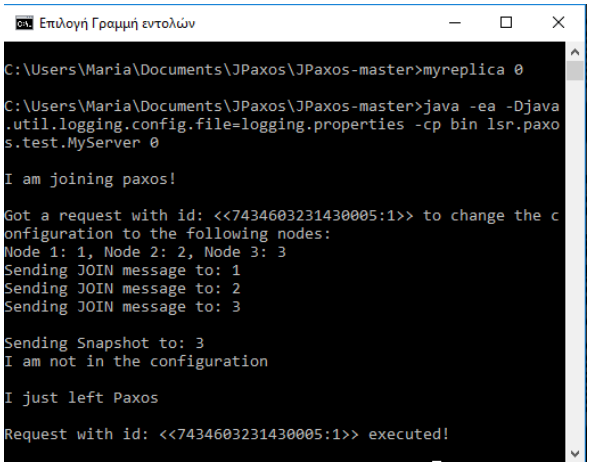

Figure 2. Active replica with process id 0.

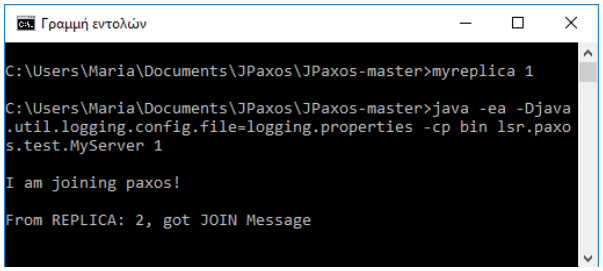

Figure 3. Active replica with process id 1.

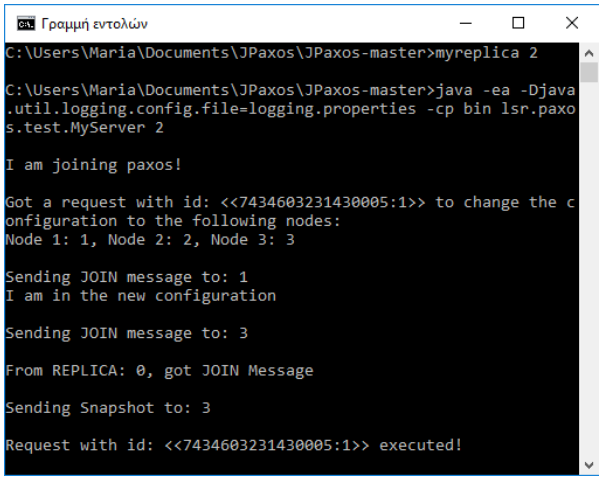

Figure 4. Active replica with process id 2.

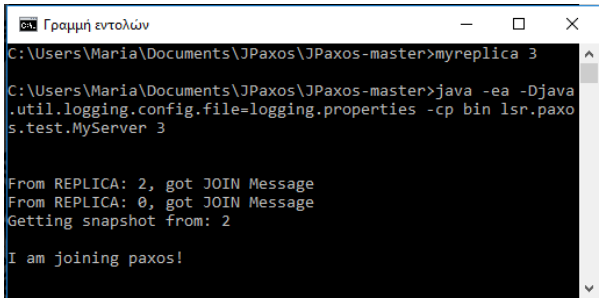

Figure 5. Inactive replica with process id 3.

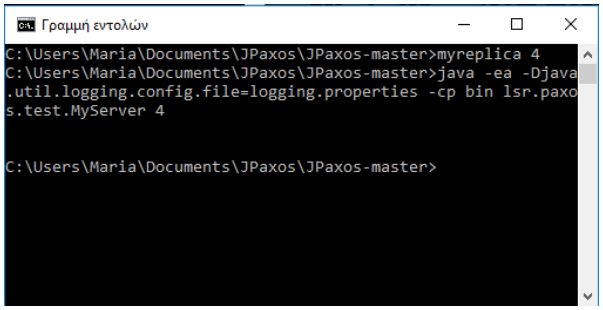

Figure 6. Inactive replica with process id 4.

In the figures above we show an example of a test run of our system with five replicas, three active and two inactive. In this example, the printed messages in each command line show the state of each process.

In Figure 1 we can see the client process. The client tries to connect with an active replica and achieves to connect with the replica with id 0. Once the connection is done, it informs the user about the configuration that paxos has in that moment. We can see that replicas with ids 0, 1 and 2 are active, while replicas 3 and 4 are inactive. Then the client requests a new configuration with replicas with ids 1, 2 and 3 as active. Once the request is completed, the process informs the user and also shows the new state of the configuration, which is exactly the one that he requested. Then the client waits for another request.

In Figure 2 we can see the process of the active replica with id 0, which also is the leader in this execution. As soon as this replica is initialized, since it is active, joins Paxos immediately and informs the user about it. Once the reconfiguration request is delivered to the replica, the replica process prints the command, which as we can see is 1, 2 and 3 as mentioned in the client section above. Then, as we can see from the figure, it sends join message to the replicas in the new configuration and tries to send a snapshot to the inactive one (here the replica with id 3). Once the other replicas are ready for the new configuration, replica with id 0 realizes it is not in the new configuration, becomes inactive, leaves paxos, informs the user that the request is completed and waits for a join message from the other replicas to re-join Paxos if needed.

In Figure 3 we can see the process of the active replica with id 1. Once again, in the moment that this replica is initialized, since it is active, it joins Paxos immediately and informs properly the user. This replica did not need to execute the reconfiguration itself. It gets a join message from replica with id 2, so it knows that it is also in the new configuration and stays active and waits.

In Figure 4 we can see the process of the active replica with id 2. Again, as an active replica, it joins Paxos immediately, when it is initialized. This replica follows the same procedure with replica with id 0. It gets the command, sends join messages to the replicas that belong to the new configuration and tries to send the snapshot to the inactive replica that will be in the new configuration. It also, notices that it itself was in the request, so it stays active and waits for the next requests, while it informs the user that the request was completed.

In Figure 5 we can see the process of the inactive replica with id 3. Here, since the replica is inactive, when it is initialized, it does not join Paxos, but rather waits for an invitation from the active replicas. Because the client requested replica with id 3 to join Paxos, we can see that it gets join messages from replicas with id 0 and 2 that executed the request, gets the snapshot from replica with id 2 and then when all the other replicas are up to date with the current state, it also joins Paxos.

Figure 6 just shows us, that the replica with id 4 that was inactive and did not join Paxos, just waits for a message and does nothing, because the request did not include it.

The example described above is one of the numerous tests that we performed to evaluate our implementation. As the reader can see this implementation resembles SMART a lot. JPaxos is based on Multi-Paxos, so it allows many instances of Paxos to run. There is a leader that coordinates the algorithm. The reconfiguration service that we created follows the Paxos protocol and guarantees that the client request will be done eventually. After the first reconfiguration is done properly as required, there are some minor bugs during the second reconfiguration request, which we intend to solve as soon as possible.

7 Conclusions

In this report, we presented a modification of the JPaxos, that is a fully functional state machine replication Java library. Our implementation, except from the functions already supported by JPaxos, adds a few more important features. We offered a new structure of the node definition, that gives an extra role to each replica, that represents its participation to the protocol. Extra messages were implemented to make the reconfiguration service achievable. Finally, we accomplished state transfer between the replicas in the old configuration and the replicas in the new configuration, in order to keep each replica up to date.

8 Future Work

Although the reconfiguration process in JPaxos, was achieved, there are some minor bugs, that we are willing to solve in the near future. Furthermore, an interesting extension would be to allow services to communicate with each other and the state transfer implemented, to also contain logs

from different services, and not only from reconfiguration logs.

9 References

[1] Jan Konczak, Nuno Santos, Tomasz Zurkowski , Pawel T. Wojciechowski and Andre Schiper, 2011.
JPaxos: State machine replication based on the Paxos protocol.

[2] Jacob R. Lorch, Atul Adya, William J. Bolosky , Ronnie Chaiken, John R. Douceur, and Jon Howell, 2006.
The SMART Way to Migrate Replicated Stateful Services