ENSAI
École nationale de la statistique et de l'analyse de l'information

Paxos for Implementing Distributed Systems Algorithm
Masters Report

Thomas Zamojski
Anurag Miglani

Professor: Prof. R. Ludinard

Abstract

Distributed algorithms focuses on interactions and communication between independent systems and processors in diverse scenarios. These algorithms are usually written down in pseudo-code, and then transformed into the into high level languages to integrate them into the systems. We present Paxos, an algorithm for fault tolerant distributed systems, some of the most significant systems are created implementing paxos on top of it.

Introduction

Distributed systems are nowadays at the core of increasingly diverse applications, from highly scalable parallel systems to mobile data processing units. We implement paxos for a fault tolerant distributed systems. We have coded the concept of basic paxos with the help of the language Scala using Akka libraries & using Intellij IDE for the development.

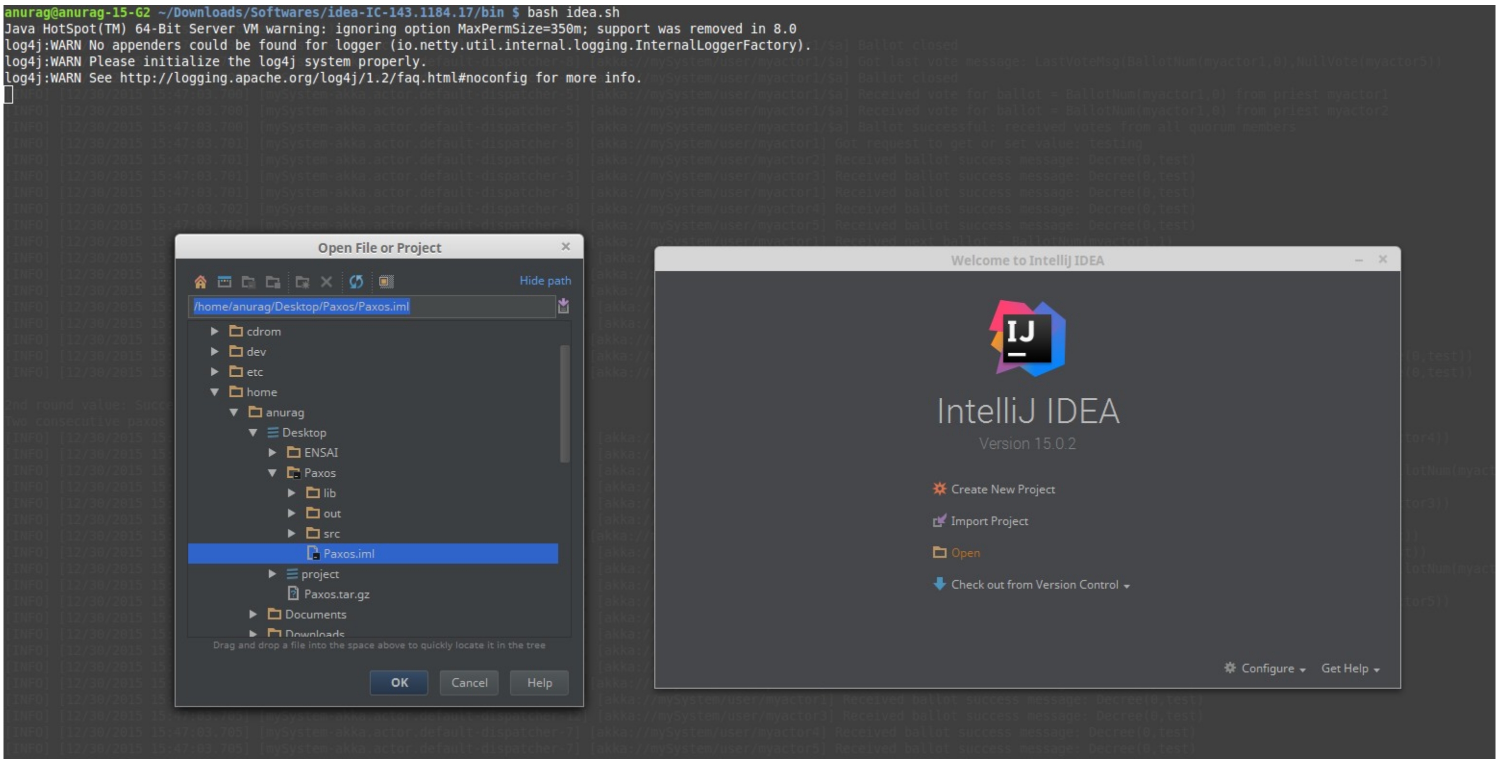
Implementation

Code can be downloaded from the shared folder:

Step1:Download the project and Intellij from the below link *:https://www.jetbrains.com/idea/download/#section=linux*

Step2:Extract the IDE, go to the bin folder, from there open the terminal and give the command "bash idea.sh".

Step3:Intellij will open with options, download the Scala option and enter open the project just as the screen-shot below, it will open the project when you will click on "Paxos.iml" file.



```
anurag@anurag-15-G2 ~/Downloads/Softwares/idea-IC-143.1184.17/bin $ bash idea.sh
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=350m; support was removed in 8.0
log4j:WARN No appenders could be found for logger (io.netty.util.internal.logging.InternalLoggerFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

Step4:Now, the project has been opened, click on Build on the tool-bar, and build the module, "Make Module Paxos". After you have compiled it successfully, to the test the module there are two options:

- ➢ Either go to the directory "~/Downloads/Paxos/src/paxosbyanurag/" from the terminal, and run the script "bash WithCoverage.sh and get the output from there.
- ➢ Or in the IDE "Go to Run & go to Run PaxosTest" and get the output in the log in IDE.

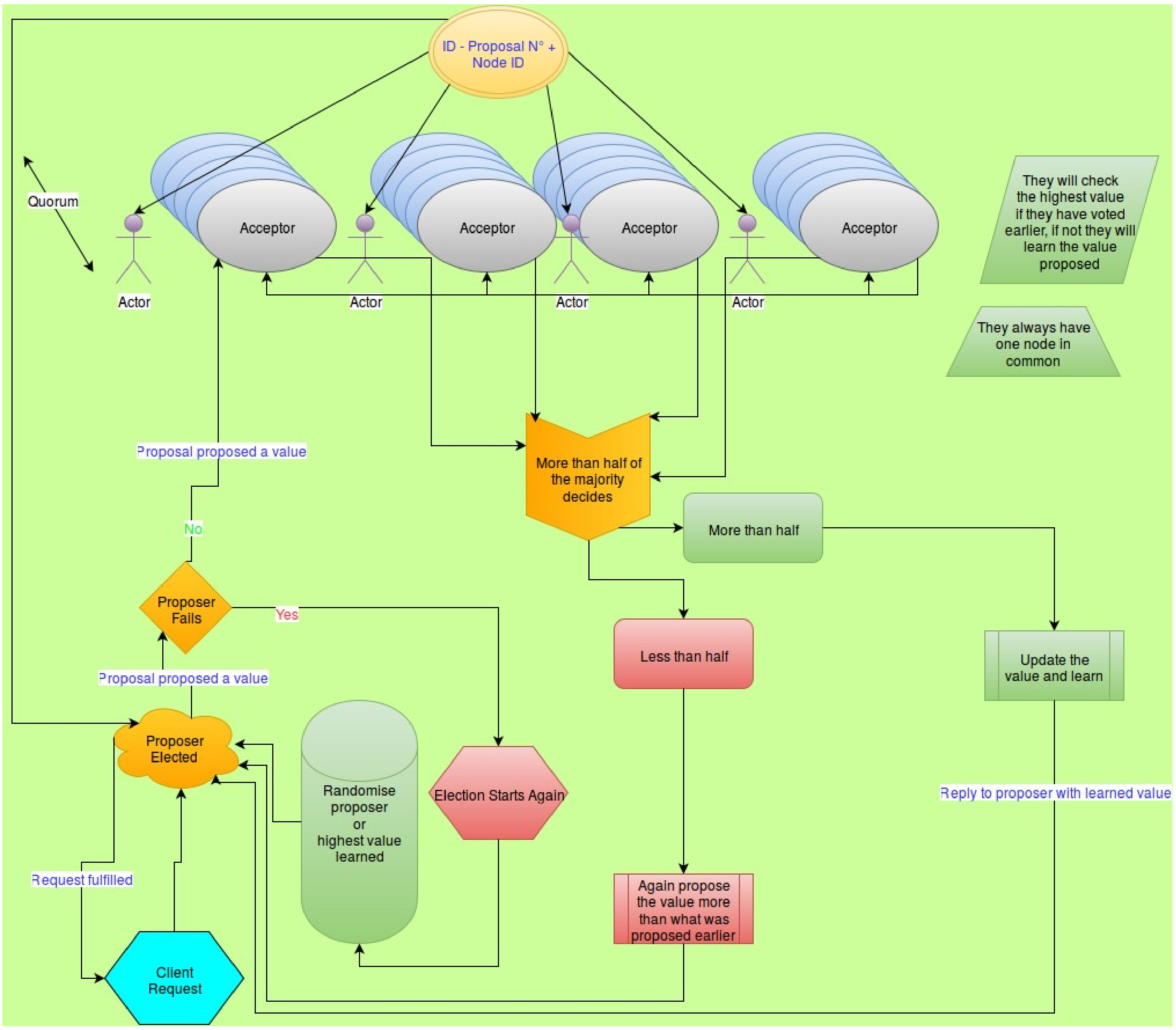
Step5:The execution of the Paxos has been done. As you will be able to see from the results.

Paxos

Paxos is a family of protocols for solving consensus in a network of unreliable processors. Paxos implementation has three main requirements:

- *Each proposal has a unique number.*
- *Two acceptors having a quorum will have at-least one node in common.*
- *The decree of the acceptor will be the decree of the node which has voted latest of the earlier in the acceptor's quorum.*

Project Diagram

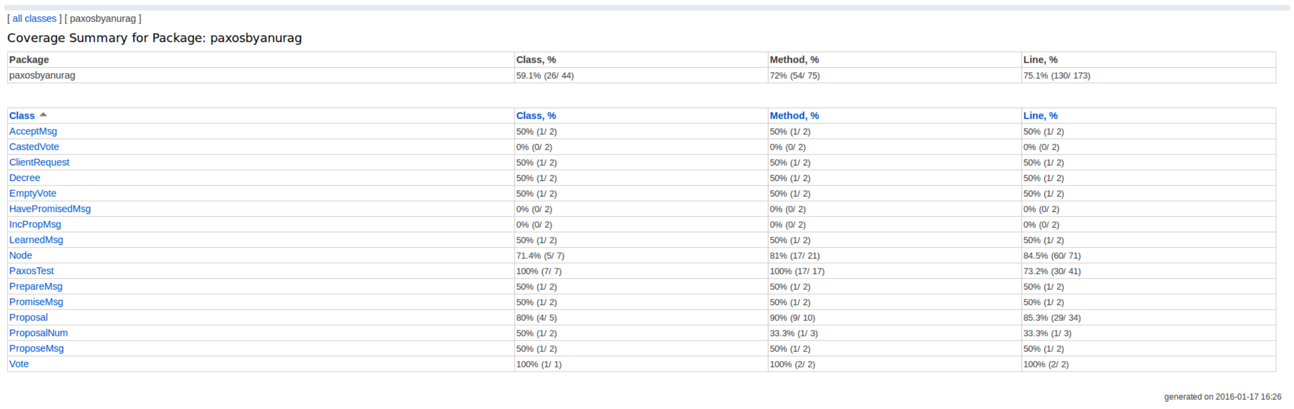

ID - Proposal N° + Node ID
Quorum
Acceptor
Acceptor
Acceptor
Acceptor
Actor
Actor
Actor
Actor
They will check the highest value if they have voted earlier, if not they will learn the value proposed
They always have one node in common
More than half of the majority decides
More than half
Less than half
Update the value and learn
Reply to proposer with learned value
Again propose the value more than what was proposed earlier
Proposal proposed a value
No
Proposer Fails
Yes
Proposal proposed a value
Proposer Elected
Randomise proposer or highest value learned
Election Starts Again
Request fulfilled
Client Request

Problems faced

- Documentation available for same Paxos is so much, that it becomes strenuous which one is most appropriate.

- Apprehension with programming using Akka, neither Java nor Scala

- Maximum utilization of the code, maximizing the code performance was hard, as in most of us are prone to long writing methods of code, and Scala has the advantage of writing long codes in one line, so code utilization was an important factor. URL for downloading the performance & screen-shot below.



[all classes] [paxosbyanurag]

Coverage Summary for Package: paxosbyanurag

Package	Class, %	Method, %	Line, %
paxosbyanurag	59.1% (26/ 44)	72% (54/ 75)	75.1% (130/ 173)

Class ▲	Class, %	Method, %	Line, %
AcceptMsg	50% (1/ 2)	50% (1/ 2)	50% (1/ 2)
CastedVote	0% (0/ 2)	0% (0/ 2)	0% (0/ 2)
ClientRequest	50% (1/ 2)	50% (1/ 2)	50% (1/ 2)
Decree	50% (1/ 2)	50% (1/ 2)	50% (1/ 2)
EmptyVote	50% (1/ 2)	50% (1/ 2)	50% (1/ 2)
HavePromisedMsg	0% (0/ 2)	0% (0/ 2)	0% (0/ 2)
IncPropMsg	0% (0/ 2)	0% (0/ 2)	0% (0/ 2)
LearnedMsg	50% (1/ 2)	50% (1/ 2)	50% (1/ 2)
Node	71.4% (5/ 7)	81% (17/ 21)	84.5% (60/ 71)
PaxosTest	100% (7/ 7)	100% (17/ 17)	73.2% (30/ 41)
PrepareMsg	50% (1/ 2)	50% (1/ 2)	50% (1/ 2)
PromiseMsg	50% (1/ 2)	50% (1/ 2)	50% (1/ 2)
Proposal	80% (4/ 5)	90% (9/ 10)	85.3% (29/ 34)
ProposalNum	50% (1/ 2)	33.3% (1/ 3)	33.3% (1/ 3)
ProposeMsg	50% (1/ 2)	50% (1/ 2)	50% (1/ 2)
Vote	100% (1/ 1)	100% (2/ 2)	100% (2/ 2)

generated on 2016-01-17 16:26

- The fact that in Scala there are no means to use macros like in other languages such as Ruby, made the implementation limited, from a syntax point of view.

- A challenging part of the implementation was to deal with the statically typed characteristic of Scala.

- The computing community has not developed the tools to make it easy to implement their algorithms & has not paid enough attention to testing, a key ingredient for building fault-tolerant systems.

- The most prominent problem faced was to write test cases to check different scenario getting accomplished by the algorithms.

Conclusion

Paxos is only a small piece of building a distributed systems: it only implements the process to write exactly one new thing to the system. Processes governed by an instance of Paxos can either fail, and not learn anything, or by the end of it have a majority having learned the same value such that there is consensus. Paxos doesn’t really tell us how to use this to build a database or anything like that, it is just the process which governs the individual communications between nodes as they execute one instance of deciding on one new value. So, for our purposes here, the thing we build with Paxos is a datumbase*(one durable write)* which can store exactly one value, and only once, such that you can’t change it after you’ve set it the first time.

We have described our implementation of a fault-tolerant system, based on the Paxos consensus algorithm. Despite the large body of literature in the field, algorithms dating almost decade ago, and experience of ours, it was significantly harder to build this system then originally anticipated.

Reference

Most of the help has been taken from the documents in priority:

Ref1:*http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf*

Ref2:*http://infoscience.epfl.ch/record/187164/files/MasterThesisReport_PamelaDelgado.pdf*

Ref3:*https://en.wikipedia.org/wiki/Paxos_%28computer_science%29*

Ref4:*https://github.com/Treode/store*

Ref5:http://harry.me/blog/2014/12/27/neat-algorithms-paxos/

Ref6:http://static.googleusercontent.com/media/research.google.com/en//archive/paxos_made_live.pdf