Sizing Tasks for Scientific Workflows with Reinforcement Learning

Nicolas Zunker

n.zunker@campus.tu-berlin.de

September 30, 2021

BACHELOR'S THESIS

Mobile Cloud Computing Chair

Technische Universitaet Berlin

Examiner 1: Prof. Dr. Odej Kao

Advisor: Jonathan Bader

Declaration

I hereby declare that I have created the present work independently and by my own without illicit assistance and only utilizing the listed sources and tools.

Hiermit erklaare ich, dass ich die vorliegende Arbeit selbststaandig und eigenhaandig sowie ohne unerlaubte fremde Hilfe und ausschliesslich unter Verwendung der aufgefaehrten Quellen und Hilfsmittel angefertigt habe.

Die selbstaandige und eigenstaandige Anfertigung versichert an Eides statt:

DATE

NAME

Acknowledgment

I would like to thank the following persons for their great support when writing my thesis at the Mobile Cloud Computing Research Group (MCC) at Technische Universitaet Berlin.

INITIALS

Abstract

The growth of the Internet of Things (IoT) will challenge our existing approaches of storing, processing, and presenting data substantially.

Zusammenfassung

Der Aufstieg des Internet of Things (IoT) stellt uns vor neue Herausforderungen bezueglich der Speicherung, Verarbeitung und Darstellung von Daten.

Table of Contents

Declaration I

Acknowledgment III

Abstract V

Zusammenfassung VI

List of Figures IX

List of Tables X

List of Listings XI

1 Introduction **1**

1.1 Motivation & Problem Description 1

1.2 Goal of the Thesis 1

1.3 Structure of the Thesis 2

A Technologies and Concepts of a Larger Context **3**

A.1 Finding Consensus in a Distributed System 3

A.2 Detecting Mutual Inconsistency 5

B Acronyms **7**

C Lexicon **9**

D Listings **11**

D.1 Configuration for Node A 11

List of Figures

A.1 Example Setup of Three Machines Agreeing on Values Based on the Paxos Algorithm 4

List of Tables

A.1 Influence of Operations on a Data Item's Version Vector 6

List of Listings

Chapter 1

Introduction

This chapter gives a general introduction into the content of and the motivation for this thesis. This is done by initially outlining the motivation and problem in section 1.1, defining the goal in section 1.2, and explaining the structure of the thesis in section 1.3.

1.1 Motivation & Problem Description

Thus, many executives feel fear to become unconscious [**TheInquisitiveMind**]. Also, I want to buy **smart home devices** such as the ones listed in table **??**.

1.2 Goal of the Thesis

The goal of this thesis is to XXXXX. The contributions are:

1. Analysis ...
2. Design ...

3. Prototypical implementation of the design.

1.3 Structure of the Thesis

The remainder of the thesis is structured as follows ... .

Appendix A

Technologies and Concepts of a Larger Context

This appendix chapter contains definitions for technologies and concepts that are mentioned in the thesis, but are not of higher importance for it. Section A.1 introduces a consensus algorithm for distributed systems.

A.1 Finding Consensus in a Distributed System

The Paxos algorithm has first been described by Leslie Lamport in his work about the parliament on the island of Paxos [**paxos1998**]. The parliament's part-time legislators had been able to maintain consistent copies of their records by following the algorithm protocol. While the original work's main focus has not been the application in computer science, the author simplified his explanations later in [**paxos2001**] and described how Paxos can be used to find consensus in fault-tolerant distributed systems.

Even though the Paxos algorithm cannot mitigate Byzantine failures (see appendix C), it

can mitigate the effects of different processing speeds of participants and reordered, delayed, or lost messages. The only requirement is that at least half of the participants can somehow communicate.

In Paxos, four different roles exist. The **client** issues requests and waits for responses. Based on the client's request, multiple **proposer** propose a value. All these proposals are processed by the **acceptors** which will agree upon one at the end. Finally, there are the **learners** which guarantee persistence of agreed decisions and respond to client reads. The usual setup used in practice, in which every machine participating fulfills each of the roles besides the client role, is shown in figure A.1.

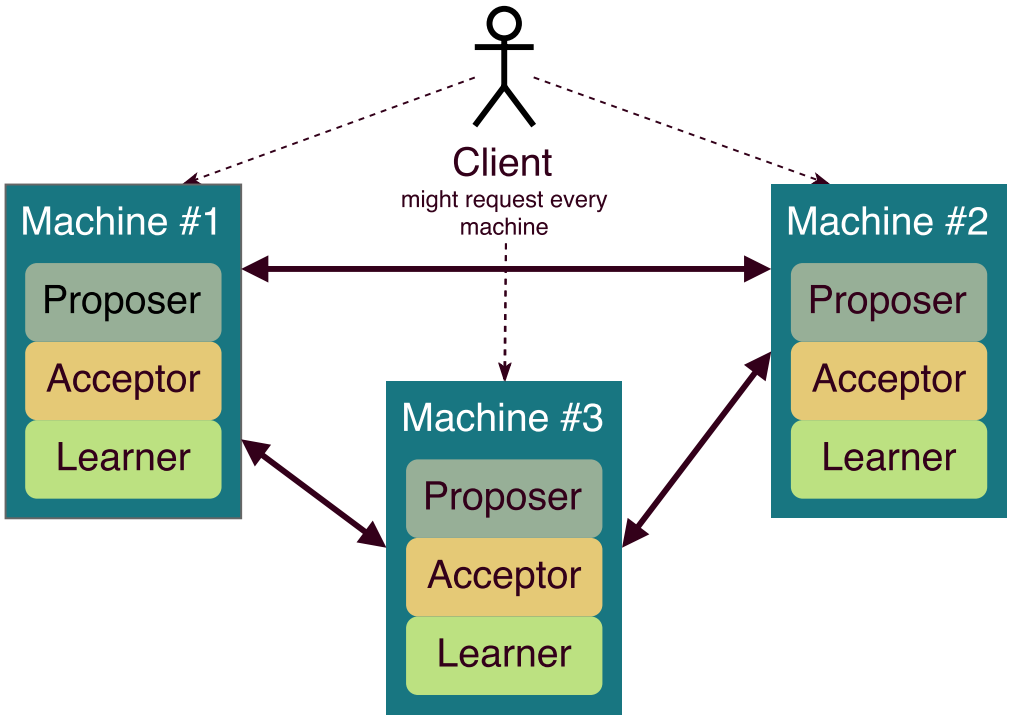

Figure A.1: Example Setup of Three Machines Agreeing on Values Based on the Paxos Algorithm

A.2 Detecting Mutual Inconsistency

Parker et al. claim that a system must ensure the mutual consistency of data copies by applying all changes made to one copy to all others correspondingly [**versionVektor**]. Each time two copies of the same original data item have a different set of modification applied to them, they become incompatible and this conflict must be detected. However, this is not trivial, because "network partitioning can completely destroy mutual consistency in the worst case". Nevertheless, Parker et al. state that the efficient detection of conflicts that lead to mutual inconsistency can be done by a concept they call *version vectors*. They define a version vector for an item f as "a sequence of n pairs, where n is the number of sites at which f is stored. The ith pair (S_i: v_i) gives the index of the latest version of f made at site S_i". An example vector for an item stored at the sites A, B and D is <A:2, B:4, D:3>, which translates in a file that has been modified twice on site A, four times on site B, and thrice on site D.

A set of vectors is "compatible when one vector is at least as large as any other vector in every site component for which they have entries". Otherwise the vectors conflict and are incompatible. E.g., the two vectors <A:2, B:4, D:3> and <A:4, B:5, D:3> are compatible because the second one dominates the other one. <A:3, B:4, D:3> and <A:2, B:5, D:3> conflict, because the first one indicates that the data item was modified one more time on node A, while the second one indicates one more modification occurred on node B. However, if we add a third vector <A:3, B:5, D:4>, no conflict exists anymore, because it dominates the two others. The consequences of an operation performed on a data item for a data item's vector is depicted in table A.1.

By using version vectors, one can detect version conflicts and initiate (automatic) reconciliation. However, Parker et al. warn that two identical updates made on separate partitions will result in a conflict, even though none is present. Thus, they recommend to additionally check two data items on differences before a conflict is raised in certain applications. Furthermore, the

Operation related to data item	Consequence for vector of data item
Update on site S_i	Increment v_i by one
Delete or rename on site S_i	Keep vector and increment v_i by one, remove data item value
Reconcile version conflict	Set each v_i to maximum v_i from all vectors used for reconciliation. In addition, increment v_i of site that initiated reconciliation by one
Copy to new site	Augment vector to include new site

Table A.1: Influence of Operations on a Data Item's Version Vector

reconciliation is most times not trivial, that's why tools such as Cassandra delegate this task to the application layer [**cassandra2010**].

Appendix B

Acronyms

AWS Amazon Web Services

IoT Internet of Things

Appendix C

Lexicon

Byzantine failure A malfunction of a component that leads to the distribution of wrong/conflicting information to other parts of the system is called Byzantine failure [**lamport1982byzantine**]. The name is based on the Byzantines Generals Problem, in which three Byzantine generals need to agree on a battle plan while one or more of them might be a traitor trying to confuse the others.

Appendix D

Listings

This is the appendix for code, that does not need to be provided directly inside the thesis.

D.1 Configuration for Node A