UNIVERSITY OF APPLIED SCIENCES RAPPERSWIL
DEPARTMENT OF COMPUTER SCIENCE

STUDY PROJECT

Redbackup: A Redundant Distributed Backup System Prototype

Authors:
Fabian HAUSER and
Raphael ZIMMERMANN

Advisor:
Prof. Dr. Farhad MEHTA

Autumn Term 2017

DON'T PANIC

"It looked insanely complicated, and this was one of the reasons why the snug plastic cover it fitted into had the words DON'T PANIC printed on it in large friendly letters."

The Hitchhiker's Guide to the Galaxy

Abstract

Fabian HAUSER and Raphael ZIMMERMANN

Redbackup: A Redundant Distributed Backup System Prototype

Today, most individuals and small companies have a limited choice with regards to how and where they back up their data.

One possibility is via local storage media, for instance using external hard disk drives. This requires manual effort and may lead to a single point of failure, since location redundancy requires extra effort. A second possibility is via some publicly provided cloud service. This may lead to issues of privacy and a high dependency on third party providers.

No easy to use, distributed backup systems with private data storage is available on the market today.

We propose a redundant distributed backup system to address this issue.

The architecture of this system consists of backup nodes which exchange data using a peer-to-peer protocol, as well as a client application that creates and restores backups. A management system is introduced to allow users to manage multiple backup nodes.

As a proof of concept, a prototype of the proposed client and node applications with a reduced feature set has been implemented in the Rust programming language.

Management Summary

Motivation

Today, most individuals and small to medium enterprises make backups on cloud environments or local storage media as e.g. hard disk drives or network attached storage systems (NAS).

These solutions require either high personal efforts to maintain local storage media or a high level of trust in a third party storage provider.

Currently, there are no backup systems available on the market which are both easy to use and provide the user with a high level of data security and privacy.

Project Goals, Approach and Technology

A backup system which solves this issues must not only provide a secure and reliable application to create and store backups, but also permit users without further domain knowledge to install and configure the application.

To meet this requirements, we further analysed and created a comprehensive architectural design.

The subsequent implementation of an architecture prototype took place in the ***Rust*** system programming language[1], which we learned during the course of this project. Rust enabled us to create a very stable yet efficient backup prototype.

Results

The architecture consists of backup nodes, which store and distribute data directly over a network connection and a client application that creates and restores backups to or from nodes. Lastly, a management system is introduced to allow users to manage multiple backup nodes.

The presented prototype demonstrates the viability of our proposed architecture, introducing a reduced feature set. The prototype can create, distribute and restore unencrypted backups.

Prospects

To extend the prototype into a fully functional backup system, there are multiple functionalities and improvements that may be implemented. The main missing parts are backup encryption, splitting of backup data, advanced data distribution strategies and the management application.

With our prototype, we demonstrate the viability of the architecture and pave the way for further implementations.

[1] For more information, see https://www.rust-lang.org/

Acknowledgements

We would like to thank our advisor, Prof. Dr. Farhad Mehta, for his continuous support and helpful comments.

Furthermore, we would like to thank Andrea Jurt Massey for her feedback regarding writing and language use.

Contents

Chapter 1

Introduction

1.1 Motivation

In this section, we legitimate this work and explain the value and applicability of our proposed solution.

Bibliography

List of Figures

List of Tables

Appendices

A.1 Task Description

TODO

A.2 Project Plan

Redbackup: Project Plan

Authors:
Fabian HAUSER and
Raphael ZIMMERMANN

Advisor:
Prof. Dr. Farhad MEHTA

Autumn Term 2017

Contents

Chapter 1

Project Overview

The goal of the study project is to provide a theoretical description of an append-only, distributed peer-to-peer data storage as well as a working prototype as described in the problem statement [**?**].

Chapter 2

Project Organization

All team members have the same strategic rights and duties. Prof. Dr. Farhad Mehta is our project advisor as visible in Figure 6.1.

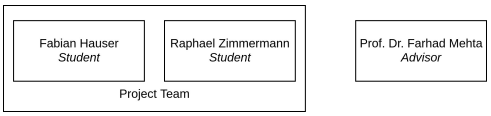

FIGURE 2.1: Project organization chart

2.1 Roles

Due to the small team size, most roles are performed by both team members.

Raphael Zimmermann project management, software engineering, quality assurance.

Fabian Hauser infrastructure management, software engineering, quality assurance.

Chapter 3

Project Management

3.1 Components

For a better overview and to allow us a sophisticated time assessment, we decided to group tasks into categories, i.e. JIRA components. Components represent processes, documents and products which are to be released.

Currently, tasks are separated into following components:

- Final Submission Document
- Concept Paper
- Management
- Poster
- Presentation
- Project Plan
- Prototype

3.2 Time Budget

The project started with the Kickoff Meeting on 18.09.2017 and will be completed after 14 weeks by 22.12.2017. The two team members are available for 240 hours each during this period which corresponds to a weekly time budget of 17.15 hour per person.

Apart from the statutory holidays, there are no further absences planned. Statutory holidays do not affect the weekly time budget.

3.3 Schedule

The project schedule is an iterative process based on elements of SCRUM.

Due to the explorative nature of the project, we decided on a sprint duration of one week.

3.3.1 Iterations & Milestones

The goals and milestones resulting from each sprint are shown in Figure 3.2.

3.3.2 Deviations from Original Schedule

The tasks and milestones were continuously updated during the project course, as is evident from Figure 3.2. The most notable additions and exclusions are noted hereafter.

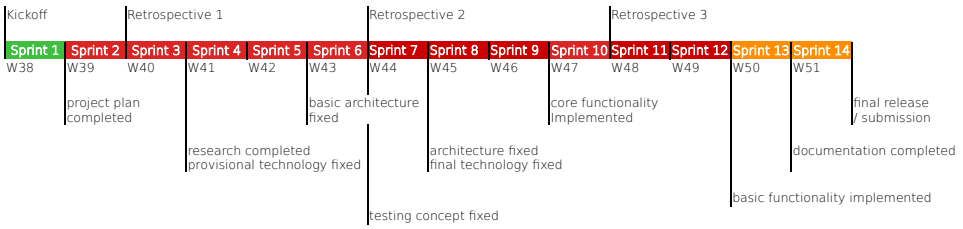

FIGURE 3.1: Overview of the 14 iterations

Research Report After the underlying research at the project start, we decided to not further research and summarise possible data distribution mechanisms, as it would not have been possible to finish the report and project in time otherwise.

Rust Programming Language As we settled for the Rust programming language in the language evaluation, we introduced the additional task to get familiar with rust an its network stack.

Retrospective 2 During the last standup meeting in sprint 6, we decided to not hold the retrospective meeting 2, as we both had no open topics to discuss.

Presentation We decided to not create a presentation during the project course as this is no mandatory part of the study project.

3.3.3 Meetings

The team works every Tuesday (10:00 - 17:00) together in the study project room and every Monday (08:00 - 17:00) remotely. Both days begin with a daily stand-up meeting taking no longer than 15 minutes. Sprint planning meetings are carried out on Tuesday at 10:00. Table 3.1 shows an overview of the total meeting time budget.

TABLE 3.1: Meeting Time Budget

Meeting Type	Total Duration per Person	Total Duration for the Team
Supervision Meetings	10 hours	20 hours
Standup Meetings	7 hours	14 hours
Sprint Planning Meetings	16 hours	32 hours
Retrospective	3 hours	6 hours
Total	36 hours	72 hours

Regular meetings with the project advisor take usually place on Friday in Prof. Dr. Mehta's office. The agenda must be sent via email to all participants at least 24 hours prior the meeting.

Raphael Zimmermann will take meeting notes for every meeting. Meeting minutes are published on the project website afterwards.

Sprint	Sprint 1	Sprint 2	Sprint 3	Sprint 4
Tag	0.1.0	0.2.0	0.3.0	0.4.0
Date	19.09.2017-25.09.2017	26.09.2017-02.10.2017	03.10.2017-09.10.2017	10.10.2017-16.10.2017
Milestones	~~*project plan completed*~~	*- project plan completed*	- provisional technology fixed - research completed	
Time Budget	34	34	34	34
Tasks	- write project plan - adapt Infrastructure	- academic research - brainstorm basic concept - define quality of service attributes - retrospective 1 (at the end of the sprint)	- write basic concept (including testing concept) - carry out language evaluation ~~*academic research*~~	- draft basic architecture & domain model - bootstrap project ~~*prototype architectural critical components*~~ *- get familiar with Rust and its Network Stack*
Documents / Artefacts	- problem statement - project plan draft	- first draft of concept paper - project plan finalized	~~research report~~ - language evaluation - second draft of concept paper	- third draft of concept paper with drafted architecture

Sprint	Sprint 5	Sprint 6	Sprint 7	Sprint 8
Tag	0.5.0	0.6.0	0.7.0	0.8.0
Date	17.10.2017-23.10.2017	24.10.2017-30.10.2017	31.10.2017-06.11.2017	07.11.2017-13.11.2017
Milestones	- basic architecture fixed	- testing concept fixed	- architecture fixed - final technology fixed	
Time Budget	34	34	34	34
Tasks	~~*prototype architectural critical components*~~ *- get familiar with Rust and its Network Stack*	*- prototype architectural critical components* - prototype testing architecture ~~*finalize testing concept*~~ ~~*retrospective 2 (Skipped)*~~	- finalize architecture *- finalize testing concept* *- implement core functionality*	- implement core functionality - set up testing infrastructure
Documents / Artefacts	- fourth draft of concept paper with provisional architecture	- fifth draft of concept paper with final testing concept	~~*concept paper*~~	*- concept paper*

Sprint	Sprint 9	Sprint 10	Sprint 11	Sprint 12
Tag	0.9.0	0.10.0	0.11.0	0.12.0
Date	14.11.2017-20.11.2017	21.11.2017-27.11.2017	28.11.2017-04.12.2017	05.12.2017-11.12.2017
Milestones	- core functionality implemented			- basic functionality implemented
Time Budget	34	34	34	34
Time Spent	33.25	29.25	28	
Tasks	- implement core functionality	- implement basic functionality - retrospective 3 (at the end of the sprint)	- implement basic functionality - exhaustive testing	- improve code quality & documentation - exhaustive testing - performance testing *- begin with final submission document*
Documents / Artefacts				

Sprint	Sprint 13	Sprint 14
Tag	0.13.0	0.14.0
Date	12.12.2017-18.12.2017	19.12.2017-22.12.2017
Milestones	- documentation completed	- final release / submission
Time Budget	34	34
Tasks	- finalize release documents - prepare presentation	- prepare submission archive (e.g. JIRA export) - print documents
Documents / Artefacts	- management summary - abstract - final submission document ~~*poster*~~ ~~*presentation*~~	- final submission archive (CD) *- poster*

FIGURE 3.2: Detailed overview with tasks and milestones of all 14 sprints. *Italic* and ~~struck out~~ text mark deviations from the original project plan that occurred during the projects course.

Chapter 4

Risk Management

An assessment of the project-specific risks is carried out in Table 4.2 as time loss during the whole project. The risk matrix in Table 4.1 provides an overview of the risk weighting.

To account for these risks, we reduce our weekly sprint time by the total weighted risk applicable to the planned task topics (on average approximately 20%). We also review the risk assessment after every sprint, adapt it and take measures if necessary.

Probability \ Severity	High ($\geq 5d$)	Medium (2-5d)	Low ($\leq 2d$)
High ($\geq$ 60%)	1	2	
Medium (30-60%)	8	3, 4	
Low ($\leq$ 30%)		5, 6, 7	

TABLE 4.1: The risk matrix. Numbers reference to the risk assessment Table 4.2

TABLE 4.2: Risk assessment table. Time in hours over the total project duration.

#	Title	Description	Prevention / Reaction	Risk [h]	Probability	= [h]
1	Problems with technology stack	Parts of the selected technology stack are not well suited, incomplete or immature.	Reflect the suitability of the chosen technology during the architecture draft.	60	60%	36
2	Missing aspects in concept paper	The concept paper does not fully cover all necessary aspects of the prototype.	Invest an adequate amount of time in architecture design.	30	60%	18
3	The architecture/concept does not scale.	The chosen architecture/concept does not scale to the expected data volume or node size	Invest an adequate amount of time in architecture design.	30	40%	12
4	Communication errors	Errors due to miscommunication or misapprehension.	Maintain a high level of interaction, precise specification of tasks responsibilities, conduct meetings if ambiguities exist.	20	50%	10
5	Problems with project infrastructure	The used project infrastructure is not or only partially available, or data loss occurs within management software.	Clean setup and self-hosting of the tools to prevent third-party dependencies.	30	30%	9
6	Scope creep	The project's scope is extended over the project course.	Define the project scope and limitations precisely. Discuss changes with the project advisor.	30	30%	9
7	Dependency errors	There are errors/bugs in third-party dependencies, i.e. libraries.	Carefully select libraries and limit thid-party dependency to a minimum.	20	30%	6
8	Missing dependency documentation	Selected libraries are lacking proper documentation	The documentation quality of a library should be a selection criterion.	15	40%	6
	Total weighted risk					106

Chapter 5

Infrastructure

5.1 Project Management and Development

For project management, document/code storage and continuous integration/deployment we utilise the corresponding products by Atlassian (JIRA, BitBucket, Bamboo, Crowd)[**?**].

These applications are hosted on our HSR project server (*sinv-56017.edu.hsr.ch*), which runs a standard Ubuntu Linux 17.04.

5.1.1 Development Tools

Since most development tools depend on the chosen technology and might change in the future, they are maintained separately on the project website [1].

5.2 Backup and Data Safety

An incremental backup of the project server including the source code and documentation is created on an independent system (*pin1262031.hsr.ch*) every night.

As our documents and code is stored in a git repository, they are also distributed on all development systems.

[1] https://www.redbackup.org/development/

Chapter 6

Quality Measures

To maintain a high standard of quality, we take the following measures:

- short sprint reviews
- three extended retrospectives
- code reviews
- automated unit and integration testing
- publish all documentation on the project website using continuous integration/delivery.
- using continuous integration for source code

6.1 Documentation

The official documents such as the final submission document, the theoretical concept as well as this project plan are written in LaTeXand published on the project website [1] as PDF documents. All other documents such as meeting minutes are written in Markdown and made available in HTML on the website as well.

The sources are in both cases kept under version control in the same repository, which allows us to use the same tools and processes for documentation and code. The continuous integration server builds and publishes the website whenever new changes are pushed to the repository.

6.2 Project Management

Because the project plan allows for an iterative process, we use JIRA with its SCRUM-Features (such as sprint creation or boards) for project management.

6.2.1 Sprint Planning

Each sprint is mapped to JIRA, which allows the project advisor to trace the project progress. Sprints are represented as boards on which the current state and assignee of any issue is easily visible ("To Do", "In Progress", "Review", "Done").

6.2.2 Definition of Done / Review of Pull Requests

An issue may be closed if ***all*** of the following conditions are met:

- All functionality conforms to the specification. Any deviations must be discussed and decided by the team.

[1] https://www.redbackup.org

- A review is performed and accepted in a pull request.
  - The source code is reasonably documented.
  - No code is commented out.
  - No warnings and errors by the compiler or any other quality tool.
  - Reasonable unit and integration tests exist and pass.
  - All documentations are up to date including the project website.
  - The complete continuous integration pipeline works.
  - The code is formatted according to the guidlines (i.e. according to RustFmt)
- The corresponding branch is merged into the stable branch (e.g. master).
- All time is logged.

6.3 Development

Since we are in a very early and agile stage we decided to use GitHub Flow[**?**], a straightforward development workflow.

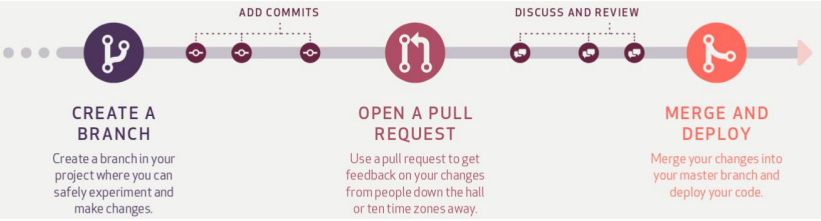

FIGURE 6.1: GitHub Flow illustrated (Source [**?**])

Since the effective technology will be fixed later in the project, concrete coding guidelines, tools, metrics and an error policy will be defined when appropriate.

6.4 Testing

All functionality must be automatically testable using continuous integration. Any non-trivial function/method must be verified with unit tests.

Integration tests verify extended test scenarios.

A minimal performance analysis will be carried out at the end of the project.

Bibliography

List of Figures

List of Tables

List of Abbreviations / Glossary

Declaration of Authorship

We, Fabian HAUSER and Raphael ZIMMERMANN, declare that this thesis and the work presented in it are our own, original work. All the sources we consulted and cited are clearly attributed. We have acknowledged all main sources of help.

Fabian Hauser

Raphael Zimmermann

Rapperswil, February 19, 2018