Paenko Endbericht

Das Projekt dauerte von Juli 2016 bis Mai 2017.

Projektverlauf

Die AP entsprechen denen aus der Projektübersicht.

Startphase

AP 1.1 und *AP 1.2*

In den Sommerferien haben wir uns intensiv mit Rust auseinandergesetzt. Wir haben mehrere kleinere Projekte und Übungen dazu gemacht. Gleichzeitig haben wir uns nach ähnlichen Projekten umgesehen und sind dabei auf *Raft-rs*, eine Raft Implementation für Rust, gestoßen.

Entwurfsphase

AP 2.1

Daraufhin haben wir den Raft-Algorithmus studiert und entschieden uns unser Projekt auf der Raft-rs Implementation aufzubauen.

AP 2.2

In diesem Arbeitspaket haben wir eine Architektur erstellt. Diese umfasst die Teile aus denen die Datenbank besteht bis hin zu der Interaktion zwischen den Komponenten.

Entwicklungsphase

AP 3.1

Die verwendete Implementation ist nur eine Libary. Als Grundlage für die Datenbank mussten Interfaces(=Traits in Rust) implementiert werden für Statemachine und Log. Außerdem fällt in dieses AP, die Einbindung von Docker, die Entwicklung der Konfigurationen und Refactorings, falls es letzteres erfordert hat.

AP 3.2

Im Arbeitspaket *AP 3.2* haben wir dann die REST CRUD Aktionen für die Datenbank entwickelt.

AP 3.3

Es wurden zwei Security Features entwickelt.

1. Community Strings

Dadurch können sich nicht alle Peers ungehindert verbinden, sondern beide müssen den gleichen String konfiguriert haben. Dieses Features wird notwendig, wenn Server(=Peers) sich zur Laufzeit zu dem Cluster hinzufügen wollen.

1. Username & Passwort

In der Konfigdatei vom Server lässt sich ein User definieren, nur mit diesen Zugangsdaten können Clients Aktionen durchführen.

AP 3.4

Dieses Arbeitspaket war das Schwierigste und ist deswegen verzögert fertig geworden. Rust verfügt über starke Memory-Garantien. Das war problematisch, da jeder REST-Route asynchron abgearbeitet wird. Wir mussten nicht nur Daten vom Server anzeigen, sondern diese mussten immer die Aktuellsten sein. In anderen Programmiersprachen hätten wir dazu einen Pointer bzw. Referenz verwendet. In Rust gibt es keinen einfachen Pointer bzw. keinen den wir verwenden hätten können, da nicht garantiert werden kann, ob das Objekt(Daten) bereits *free* sind und der Kompiler eine Kompilierung nicht erlaubt hat. Um diese Herausforderung zu lösen, haben wir einen Reference-Counter (Arc) und ein Rw-Lock eingesetzt. Dadurch hat der Server das notwendige Schreibrecht auf die Daten bekommen und die einzelnen Meta-REST-Routen konnten die Daten lesen.

AP 3.5

Das Webinterface kann die Meta-Daten abrufen und auf der Webseite darstellen. Die Meta-Daten werden vom Server in einem Json Format ausgegeben.

AP 3.6

Ein Client kann Transaktionskommandos dem Server senden. Das umfasst ein erweiteres CRUD und Begin, Commit und Rollback. Diese werden dann vom Leader an alle Follower verteilt. Alle Client-Nachrichten, die nicht die TransaktionId der Transaktion haben, befinden sich in einer Warteschlange und werden fortgesetzt sobald die Transaktion commited wurde. Wenn ein Client ein Rollback sendet, werden alle Aktionen die während der Transaktion geschehen sind, zurückgenommen.

AP 3.7

Bei “Multiple Instances von Statemachine und Log” oder auch MultiLog genannt wurde implementiert damit Anwender ihre Daten logisch trennen können. Dabei läuft nicht wie gewöhnlich ein Log auf einem Node(=Peer, Server), sondern mehrere. Ein Vorteil, ist das pro Log eine Transaktion rennen kann.

AP 3.8

In diesem Arbeitspaket wurde ein Rest-Client in der C# Library implementiert. Damit ist es möglich aus C# den Rust-Server zu erreichen und CRUD- oder

Transaktionen-Befehle abzusetzen. Außerdem wurde in diesem Arbeitspaket an der Architektur der C# Library gearbeitet.

AP 3.9

Im Arbeitspaket 3.9 wurde ein C# GUI implementiert. Datenbanknodes werden graphisch auf einer Google Maps Karte visualisiert, über intuitive interaktionen mit dem GUI können CRUD Operationen durchgeführt werden.

AP 3.10

Bei einer normalen Konfiguration müssen alle Peers vorher definiert sein. In diesem AP wurde "dynamic Peering" entwickelt. Damit soll es möglich sein, dass sich neue Peers zur Laufzeit in den Cluster beitreten können. Der Anwärter konfiguriert dazu einen "dynamic Peer" in seiner Konfiguration und startet den Server. Wenn der CommunityString gleich ist, sendet der DynamicPeer dem Anwärter für den Cluster alle seine Peers und der neue Peer versucht sich mit allen anderen auch zu verbinden. Bei der Löschung eines Peers aus dem Cluster hatten wir einige Probleme und konnten es nicht fertigstellen.

Testen und Dokumentation

AP 4.1

Im `test` Ordner von PaenkoDb befinden sich einige Test-Shellskripte mit denen wir die Funktionalität gestestet haben. Die REST-Schnittstelle haben wir über das `http_test.sh` Skript getestet. Bei den einzelnen curl Requests aus dem Skript gibt es farblichen Output, um den Tester darüber zu informieren, welche Aktionen erfolgreich waren und welche nicht.

AP 4.2

Es wurden alle REST Funktionen der C# Library, durch das Aufrufen der Methoden mit Testdaten, getestet.

Abschlussphase

AP 5.1

Das Diplomarbeitsbuch wurde verfasst und abgeben.

Projektergebnisse

Projektendbericht

Lizenz: CC-BY-3.0 AT
Ort: https://paenko.github.io/PaenkoDb/documents/projektendbericht.pdf und

www.netidee.at/paenkodb/projektendbericht.pdf
Fertigstellungsgrad: fertig

Diplomarbeitsbuch

Lizenz: CC-BY-3.0 AT
Ort: https://paenko.github.io/PaenkoDb/documents/diplomarbeitsbuch.pdf und www.netidee.at/paenkodb/diplomarbeitsbuch.pdf
Fertigstellungsgrad: fertig

Anwendungsdokumentation

Lizenz: CC-BY-3.0 AT
Ort: https://paenko.github.io/PaenkoDb/documents/anwendungsdokumentation.pdf und www.netidee.at/paenkodb/anwendungsdokumentation.pdf
Fertigstellungsgrad: fertig

Zusammenfassung

Lizenz: CC-BY-3.0 AT
Ort: https://paenko.github.io/PaenkoDb/documents/zusammenfassung.pdf und www.netidee.at/paenkodb/zusammenfassung.pdf
Fertigstellungsgrad: fertig

Datenbank

Lizenz: MIT
Ort: https://github.com/paenko/paenkodb und www.netidee.at/paenkodb
Fertigstellungsgrad: fertig

C# Library

Lizenz: MIT
Ort: https://github.com/paenko/PaenkoDB_CSharp und www.netidee.at/paenkodb
Fertigstellungsgrad: fertig

C# Manager

Lizenz: MIT
Ort: https://github.com/paenko/PaenkoDB_CSharp_Client und www.netidee.at/paenkodb
Fertigstellungsgrad: fertig

Up & Running Video

Lizenz: CC-ND-3.0 AT
Ort: https://www.youtube.com/watch?v=H2JRI3eahCo
Fertigstellungsgrad: fertig

Öffentlichkeitsarbeit

Wir haben bereits Andrew Hobden eine Email geschrieben und haben ihn um Feedback gebeten. Leider haben wir bisher noch keine Antwort erhalten (Stand

21. April 2017). Außerdem haben wir ein Video Up&Running auf Youtube veröffentlicht. Des Weiteren werden wir einen Reddit Post in `/rust` schreiben und haben bei mehreren Wettbewerben teilgenommen wie: JugendInnovativ, ITS Award und AXA. (Stand 21. April 2017)

Zukünftige Erweiterungen

Query Language

Derzeit ist die einzige Möglichkeit Dokumente abzufragen, über die DocumentId. Das erfordert eine Überarbeitung des Payloads, denn zurzeit wird der Payload in Bytes gespeichert. Multi User Konfiguration. Die jetzige Implementation der Authentifikation unterstützt nur einschränken festgelegten Benutzer. Das kann ein Sicherheitsproblem darstellen, wenn man Dieses Rechte eines Benutzers nicht einschränken und keine weiteren Benutzer haben kann.

Client - Prioritäten

Nicht alle Clients sind gleich wichtig. Aus Quality of Service Gründen könnte es für einen Datenbank-Administrator relevant sein, Clients unterschiedliche Prioritäten zu zuteilen.

SSL - Http Clients

Daten sollten im Transit nicht Plaintext versendet werden. Denn es könnten in manchen PaenkoDocuments sensible Daten gespeichert sein. Aus diesem Grund wäre es sinnvoll für die HTTP-Schnittstelle, SSL zu implementieren.

SSL - Peers

In den meisten Fällen sind die Knoten nicht lokal, sondern auf mehrere Rechner verteilt, eventuell sogar über die ganze Welt. Nachrichten die über Capnproto serialisiert wurden, sind sie nicht verschlüsselt und damit nicht “sicher”. Denn Nachrichten der Peers werden über gewöhnliche TcpStreams gesendet. Es ist daher empfehlenswert, auch hier SSL zu implementieren.

Crosslog - Transactions

Wie bereits erwähnt wurde, kann nur eine Transaktion pro Log gestartet werden. Möchte aber ein Anwender Transaktionen über mehrere Logs hinweg machen, geht das leider nicht. Eine interessante Erweiterung wäre es, wenn man auf

mehreren Logs Aktionen durchführen könnte, mit der Möglichkeit, dass wenn etwas schief geht, die Transaktion wieder zurückgerollt werden kann.

RemoveServer

Derzeit ist nur das dynamische Hinzufügen von Peers zur Laufzeit möglich. Möchte man jedoch einen Node aus dem Cluster entfernen, geht das nicht. Sobald ein Node ausfällt, versuchen die anderen Nodes ihn immer wieder zu erreichen. Eine mögliche Implementation wäre es das Timeout auf mehrere Minuten zu beschränken. Sobald dieses Timeout erreicht wurde, löschen die Peers diesen Peer aus ihrer Liste.

Library

PaenkoDb ist klar als Binary konzipiert und implementiert. Jedoch da die PaenkoDb Raft-Implementation einige Erweiterungen besitzt, könnte man versuchen diese als Library zu implementieren. Eine Dritt-Applikation könnte dann diese Library implementieren. Indem man dann diese Library in einer Applikation implementiert, wird diese Applikation zu einer Peer-to-Peer Software mit fehlertoleranten Eigenschaften.