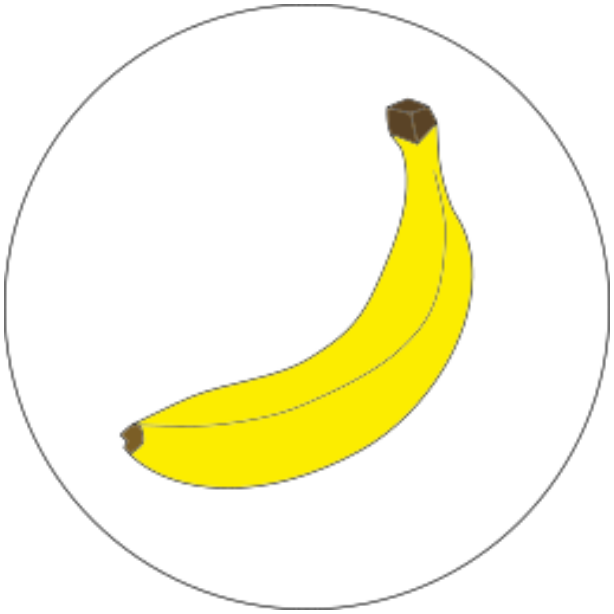

Balanced Banana

A Distributed Task Scheduling System

Implementierung

Niklas Lorenz, Thomas Häuselmann, Rakan Zeid Al Masri, Christopher Lukas Homberger und Jonas Seiler

6. März 2020

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument dient dazu, einen allgemeinen Überblick darüber zu geben, was wir, das BalancedBanana-Team, in der Implementierungsphase gemacht haben.

Das erste Abschnitt beschreibt die Probleme, die wir während dieser Phase hatten, und die Änderungen, die wir am ursprünglichen Entwurf vorgenommen haben. Das Abschnitt ist in mehrere Unterabschnitte unterteilt, die jeweils ein Modul unseres Programms beschreiben.

Das nächste Abschnitt listet kurz die verschiedenen externen Bibliotheken und Programme auf, die wir verwendet haben.

Danach folgt ein Abschnitt über allgemeine Statistiken über unser Programm einschließlich der Testabdeckung.

Das letzte Abschnitt gibt einen allgemeinen Überblick über unseren ursprünglichen Zeitplan und den tatsächlichen Zeitplan für das Projekt.

2 Probleme und Änderungen am Entwurf

2.1 Allgemein

Viele der Probleme zu Beginn der Phase standen im Zusammenhang mit der Einrichtung unserer einzelnen Systeme. Die Konfiguration der verschiedenen Entwicklungsumgebungen war zu Beginn sehr zeitaufwändig und die Fehlerbehebung war manchmal schwierig, da jede Person eine andere Einrichtung hatte. Es wäre viel einfacher gewesen, wenn wir alle die exakt gleiche Entwicklungsumgebung auf dem gleichen Betriebssystem verwendet hätten.

2.2 Datenbank

Viele der früheren Probleme mit der Implementierung der Datenbank waren mit dem Setup und meiner Unkenntnis von MySQL verbunden. Im Allgemeinen gab es jedoch keine großen Probleme mit der Implementierung des Datenbankteils unseres Programms. Ich habe die Zeit, die es nicht nur für die Implementierung der Datenbank, sondern auch für das Testen der Datenbank brauchte, grob unterschätzt. Das Testen der Datenbank nahm aufgrund der Natur relationaler Datenbanken enorm viel Zeit in Anspruch. Die aktuelle Implementierung folgt im Allgemeinen dem ursprünglichen Entwurf, es gab jedoch einige Änderungen.

Eine große Änderung war die Aufspaltung der Gateway-Klasse in drei kleinere Klassen (Job-, Worker- und UserGateway). Damit sollte vermieden werden, dass es eine große Götterklasse gibt, die schwer zu warten und zu testen ist. Alle Gateway-Klassen erben von IGateway. Worker- und UserGateway haben sich im Vergleich zum ursprünglichen Entwurf nicht viel geändert, jedoch wurden neue Methoden zu JobGateway hinzugefügt, die von anderen Teilen unseres Programms benötigt wurden.

Eine weitere Änderung war das Hinzufügen eines Caches zum Repository. Der Cache ermöglicht es unserem Programm, verschiedene Objekte zu speichern, so dass andere Teile unseres Programms die Datenbank nicht ständig abfragen müssen.

Eine kleinere Änderung bestand darin, dass die IGateway-Klasse die Verbindung zur Datenbank aufbauen konnte, anstatt sie von der Gateway-Klasse erstellen und ein QSqlDatabase-Objekt als Klassenmitglied speichern zu lassen. Diese Änderung wurde aufgrund einer genaueren Untersuchung der Dokumentation vorgenommen.

2.3 Config Files

Da es sich bei den Config Files um sehr einfache Container handelt, die keine große Funktion haben, musste hier nichts groß geändert werden. Lediglich die SchedulerConfig Klasse wurde in ApplicationConfig umbenannt, da sie auch in der Worker-Anwendung eingesetzt werden kann.

2.4 Communication

Das Communication Paket war im Entwurf noch sehr abstrakt gehalten. Daher ist während der Implementierung vieles dazugekommen, besonders neue Message-Typen. Mittlerweile gibt es:

- AuthResultMessage
- ClientAuthMessage
- HardwareDetailMessage
- PublicKeyAuthMessage
- RespondToClientMessage

- TaskMessage
- TaskResponseMessage
- WorkerAuthMessage
- WorkerLoadRequestMessage
- WorkerLoadResponseMessage

Eine weitere große Änderung ist die Aufteilung des Communicator in den Communicator und den CommunicatorListener. Letzterer übernimmt jetzt das Warten an einem Port auf Verbindungsanfragen, während der Communicator nur noch für das Empfangen und Senden von Messages verantwortlich ist. Für die Aufteilung haben wir und deshalb entschieden, da auch in allen Netzwerk-Bibliotheken zwischen Socket und Server unterschieden wird. Für das Verarbeiten von Messages sind weiterhin die Unterklassen des MessageProcessor verantwortlich. Diese befinden sich jetzt in dem entsprechenden Anwendungspaket (Scheduler, Worker oder Client) und nicht länger im Communication Paket, da dieses nur als Grundlage für die Kommunikation dienen soll, nicht jedoch die gesamte Verarbeitung für alle drei Anwendungen.

3 Externe Abhängigkeiten

QT Bibliothek für Netzwerkübertragung und Datenbanken. Wird verwendet, um die drei Programme (Client, Scheduler und Worker) miteinander kommunizieren lassen zu können. Wird außerdem dazu verwendet die MySQL Datenbank auf dem Scheduler zu verwalten und von innerhalb des Codes SQL Anfragen zu ermöglichen.

CLI11 Bibliothek zum einlesen der Befehlszeile. Wird verwendet um die Argumente beim Aufruf der drei Programme (Client, Scheduler und Worker) die der Benutzer angegeben hat einzulesen.

OpenSSL Bibliothek zur Verschlüsselung von Daten. Wird dazu verwendet einen Benutzer mithilfe eines Public-Private-Key Verfahrens mit dem Scheduler zu verbinden und zu identifizieren, wenn sich der selbe Benutzer erneut im System anmeldet.

GTest Bibliothek zum Testen von C und C++ Code. Wird verwendet, um die einzelnen Bestandteile der Programme zu testen.

SimpleHTTPServer Selbstentwickelte Bibliothek, die das Aufsetzten und verwalten eines HTTP Servers ermöglicht. Wird verwendet, um den HTTP Server zu verwalten, der auf HTTP Anfragen bezüglich des Systemstatus (z.B. Auslastung der Arbeiter Rechner, ...) antwortet.

Docker Bibliothek zum kapseln von Prozessen in Containern. Wird dazu verwendet, die Benutzer Aufträge voneinender zu trennen und die gesetzten Grenzen für die Hardwareressourcen einzubehalten.

Cereal Bibliothek zur Serialisierung.

criu Bibliothek zur Checkpoint-/Wiederherstellungsfunktionalität

python Programmiersprache für Scripts. Wird verwendet, um gewisse Tests auszuführen, die Netzwerkfunktionalität erfordern.

4 Statistik

4.1 Allgemein

4.2 Testfälle

5 Implementierungsplan

5.1 Geplant

5.2 Realität