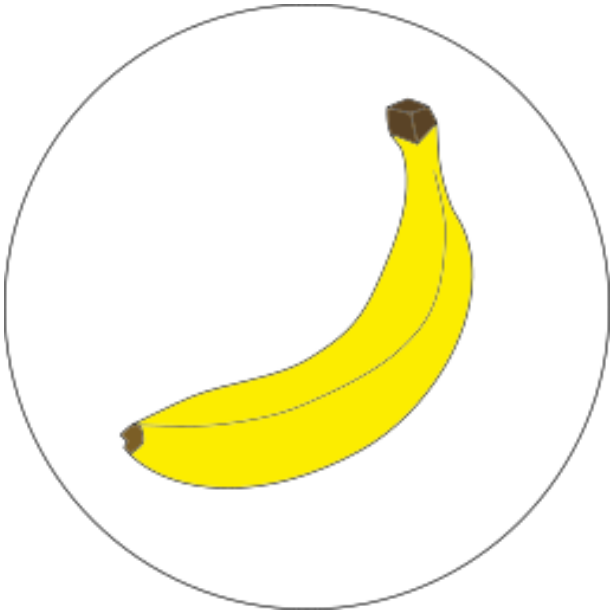

Balanced Banana

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 gegenüber dem ursprünglichen Pflichtenheft und Entwurf vorgenommen haben. Dieser 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, Änderungen am Pflichtenheft und 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. Durch teilweiße fehlender Synchronisierung mit der master Branch, entstanden vermeidbare Merge Konflikte. Wegen vieler Zyklen zwischen den Paketen, wurde die Task klasse in communication verschoben. Jedoch wurden weitere größere Paketänderungen in die nächste Phase verschoben, um die Implementierung und ausstehende Merges nicht verkomplizieren.

2.2 Änderungen am Pflichtenheft

2.2.1 CommandLineInterface

Befehle wurden restrukturiert, benutzen jetzt unterbefehle (wie bei git) Verkürzung der Abkürzungen: -minc -> -c, -maxc -> C, minr -> -r, -maxr -> -R, -ri -> I, -ai -> i, weil CLI11 für Options Abkürzungen einen einzelnen Buchstaben erfordert.

2.2.2 Communication

Automatische Verbindung des Communicators mit dem Schedulers, nur mit gespeicherter Adresse des Schedulers möglich.

2.3 Änderungen am Entwurf

2.3.1 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.

2.3.2 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.

2.3.3 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

3.1 Im Programm

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.

3.2 Auf GitHub

CircleCI Ein Continuous-Integrationsprogramm(CI). CI ist eine Softwareentwicklungsstrategie, die die Entwicklungsgeschwindigkeit erhöht und gleichzeitig die Qualität des Codes, den die Teams einsetzen, sicherstellt. Die Entwickler commiten kontinuierlich Code in kleinen Schritten (zumindest täglich oder sogar mehrmals täglich), der dann automatisch erstellt und getestet wird, bevor er mit dem gemeinsamen Repository zusammengeführt wird.

TravisCI Ein weiteres Continuous-Integrationsprogramm.

Coveralls Programm, um die Testabdeckung unseres Codes zu zeigen.

4 Statistik

4.1 Allgemein

Source 4452 LOC

TestSource 8649 LOC

Include 19994 LOC

Insgesamt 53307 LOC (inklusive CMake, SQL, usw.)

4.2 Testfälle

3 Tests von 265 schlagen fehl (master) Coverage 92%

4.2.1 Nicht Abgedeckt

Docker Checkpoint Tests brechen frühzeitig ab und bleiben unberücksichtigt. Unvollständige Einstiegspunkte, werden nicht durch Tests ausgeführt und sind negativ bewertet. Externe Bibliotheks schnittstellen kann man nicht einfach fehlschlagen lassen.

5 Implementierungsplan

5.1 Geplant

	Woche 1	Woche 2	Woche 3	Woche 4	Woche 5?
Authenticator	Christopher				
Queue	Jonas				
MsgProcessor		Niklas			
Datenbank-Schnittstelle	Rakan				
Commandlineinterface	Thomas				
Docker Kernfunktionalität		Christopher			
DB Workerlistener		Jonas			
Communicator	Niklas				
DB SQL		Rakan			
Client, DB Bedienung		Thomas			
Netzwerksocket			Christopher		
Worker Kernfunktionalität			Jonas		
DB HTTP&SMTP			Niklas		
HTTP&SMTP Server			Rakan		
Timer			Thomas		

5.2 Realität

						80% Testabdeckung	
		Woche 1	Woche 2	Woche 3	Woche 4	Woche 5?	Woche 6 (Durch integration gesunkene Testabdeckung)
Angefangen	Authenticator	Christopher			Christopher	Nur der MP ist nicht da	Fertig
Verzug	Queue	Jonas				ok	Fehlerbereinigung ausstehend
Fertig ohne Tests					Get jobids of last n hours		Fertig
Fertig mit Tests					Get Jobids of worker id		Fertig
Integration	MsgProcessor		Niklas	Niklas	Niklas		Alle notwendigen Messages sind / api ist implementiert
	Datenbank-Schnittstelle	Rakan				Höhere Testabdeckung	Anleitung zum aufsetzen fehlt
	Commandlineinterface	Thomas					Fertig
	Docker Kernfunktionalität		Christopher		*Tests implementiert / Fehlerkorrektur*		
	Docker Snapshot					unstabiles docker feature, online Test fails	Checkpoints (experimentell funktioniert nicht garantiert)
	DB Workerlistener	←Arbiter	Jonas		Jonas	Aufgelöst -> Listener	
	Communicator	Niklas					Praxis test ausstehend
	Communicator Listener	Christopher					Praxis test ausstehend
	DB SQL		Rakan		Aufteilen von Repository und Gateway	add credentials to constructor	Fertig
	Client, DB Bedienung		Thomas		Thomas		Fehlerbereinigung ausstehend
	Worker Kernfunktionalität			Jonas	Jonas		Bibliotheken sind schon alle da
	DB HTTP&SMTP			Niklas	Niklas		Fertig (Rakan)
	HTTP&SMTP Server			Christopher	Christopher	Convert Method to class	Testen gegen über dem KIT smtp server fehlt
						needs to get (poll) data	Callbacks über die Repository erstellen
	Timer			Thomas			Mögliche Fehlerbereinigung ausstehend
	Configfiles	Niklas/Christopher					
	DB SQL für Webserver (Callbacks)				Rakan		Callbacks über die Repository erstellen
	Scheduler/worker					Get current worker load	
						constructor	
	Workerstatuschangelistener						Observer
	Factory						2/3 Worker Factory braucht anpassungen
	Task				wie (de)serialisiert man		Fertig
	Worker					get current load / ram ..	Schedulerseitig fertig
	User					constructor/getjobids per user	