SERVER-LOAD-SIMULATOR-MONITOR

DISTRIBUTED COMPUTING

SWD16

BENJAMIN MOSER, ANTON PIIROJA, ANDREAS RAINER

1 CONTENT / INHALT

2 EINLEITUNG

2.1 Ausgangssituation

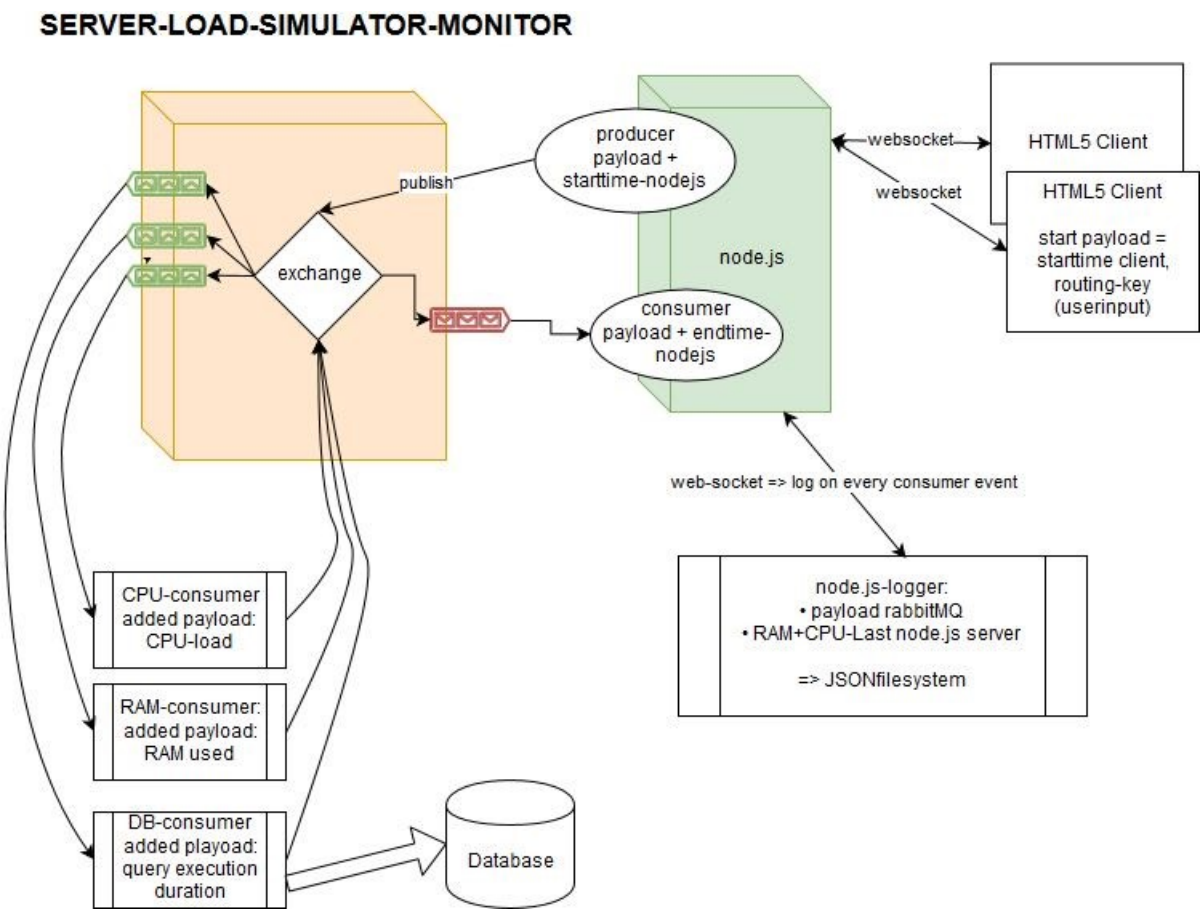
Die Aufgabe bestand in der Realisierung eines praktischen Anwendungsfalles für einen verteile Anwendung unter Verwendung eines Servers mit Clients, wobei die Verbindung über Websockets laufen sollte. Zusätzlich sollte ein Message-Queuing-System mit einem Exchange-Server und drei Queues für drei Consumer, die in drei unterschiedlichen Programmiersprachen geschrieben sind, basierend auf der Projektidee, implementiert werden. Außerdem sollte eine Datenbankverbindung und ein Logging-System geschaffen werden.

2.2 Projektübersicht

Bei unserem Projekt „Server-Load-Simulator-Monitor“, kurz *ServerLoadSimMon*, handelt es sich um eine Anwendung zur Simulation und Monitoring von Hardwarelast auf den verschiedenen Hostsystemen. Dabei können die jeweiligen Belastungswerte vom Benutzer oder der Benutzerin im Web-UI festgelegt werden.

Die Benutzereingaben werden über Websockets an den „Node.js“-Server und in weiterer Folge an das Message-Queuing-System „RabbitMQ“ übergeben. Der Exchange von RabbitMQ reicht die Anfragen an den zuständigen Consumer weiter, wo dann die eigentliche Auslastungssimulation stattfindet. Bei diesem Belastungstest durch den Consumer wird die benötigte Zeit, CPU-Verbrauch und RAM-Verbrauch aufgezeichnet und als Antwort in Form einer Nachricht an das Message-Queuing System zurückgeschickt. RabbitMQ stellt seinerseits diese Nachrichten dem Node.js-Server zur Verfügung, welcher diese schlussendlich über die Websocketverbindung dem Browser schickt und dort dem Benutzer oder der Benutzerin angezeigt wird.

Die nachfolgende Grafik illustriert die grundlegende Idee und Struktur des Projekts „Server-Load-Simulator-Monitor“.



3 AUFGABENSTELLUNG

3.1 Client UI

Möglichkeit für die Eingabe der Belastungswerte für die entsprechenden Hostsysteme und Darstellung der Testergebnisse.

3.2 Message-Queuing

Es sind in Summe vier Message-Queues in Verwendung:

- Drei für die unterschiedlichen Consumer mit Direct-Exchange.
- Eine für die Rückgabe der Testergebnisse, ebenfalls Direct-Exchange.

3.3 Consumer

Die drei Consumer wurden in den folgenden Sprachen geschrieben:

- *RAM – Belastung*: .NET Core 2.1
- *DB – Belastung*: Java
- *CPU – Belastung*: Python 3

3.4 Web-Socket-Kommunikation

Zwischen Client-UI und Node.js-Server wird eine Web-Socket-Verbindung etabliert, welche die Anfragen zur Simulation an den Server sendet und die Ergebnisse daraus empfängt.

4 INSTALLATION

Die Installation ist ausführlich im readme.md-File des Repositories beschrieben:

https://git-iit.fh-joanneum.at/raineran16/ServerLoadSimMon/blob/master/README.md

5 ANLEITUNG / BEDIENUNG

Um den vollen Funktionsumfang der Applikation zu gewährleisten, müssen zuerst der Node.js-Server mittels Ausführen der "server.js"-Datei, RabbitMQ und die drei Consumer gestartet werden. Das Client-UI wird durch einen Browser, welcher die public/index.html-Datei ausführt, angezeigt.

Im Client-UI können einzelne Belastungstests mithilfe von Bedienelementen ein- und ausgeschaltet werden. Aktive Felder benötigen einen Wert, welcher dann beim Belastungstest als Parameter verwendet werden. Mittels Klick auf den "Start simulation"-Button wird der Test gestartet. Nach erfolgreicher Ausführung der Consumer werden die Endergebnisse im Client-UI angezeigt.

6 ZUSAMMENFASSUNG

6.1 Ergebnis

Ergebnis dieses Projekts ist eine Applikation, die eine vollständige Message-Queuing-Funktionalität besitzt und alle restlichen Anforderungen, wie zum Beispiel Web-Sockets, drei Consumer, Logging usw., erfüllt. Der komplette Nachrichtenweg läuft über:

Client-UI → Node.js → RabbitMQ-Exchange → Consumer → RabbitMQ-Exchange → Node.js → Client-UI

Der tatsächliche Output besteht aus der für die Belastungstests benötigten Zeit/RAM/CPU-Last und wird der Benutzerin oder dem Benutzer im Client-UI angezeigt.

Weiters besteht die Möglichkeit die Tests von mehreren Clients gleichzeitig auszuführen um auch die Auswirkungen auf den Node.js-Server evaluieren zu können. Hierzu gibt es derzeit noch keine grafische Auswertung, es kann aber die JSON-Datei „logs/node.log“ dazu verwendet werden.

6.2 Bekannte Probleme

Alle Tests wurden auf Windows 10 / 64-bit durchgeführt. Bei der Ausführung der Applikation kann es auf anderen Betriebssystemen unter Umständen zu Einschränkungen der Funktionalität kommen. Bei unseren Tests funktionierte RabbitMQ unter MacOS nicht korrekt, der Node.js-Server konnte sich nicht darauf verbinden.

Werden mehrere Prozesse pro Consumer angefordert, kann es passieren, dass nicht alle Return-Messages beim Benutzer ankommen und somit die UI scheinbar einfriert und in einen Timeout läuft.

6.3 Ausblick / weitere Möglichkeiten

Es können beliebig viele zusätzliche Consumer mit eigenen Message-Queues hinzugefügt werden, damit weitere Teile des Hostsystems getestet werden können. Außerdem besteht die Möglichkeit des Einsatzes auf mehreren Endgeräten zur Nutzung in der Infrastruktur eines Unternehmens-netzwerkes als verteiltes System mit grafischer Darstellung (derzeit nur gespeichert in „logs/node.log“).

6.4 Persönliche Meinung / Erkenntnisse

Unterschiedliche Programmiersprachen können in einem verteilten System problemlos kombiniert werden, dadurch wird die Funktionalität in keinster Weise eingeschränkt. Mithilfe von Message-Queuing Systemen kann erreicht werden, dass eine Applikation durch die Technologieunabhängigkeit sehr gut skalierbar ist.

So gut die prinzipielle Einbindung mehrerer Systeme und Programmiersprachen auch funktioniert, die Komplexität und die Anzahl der Fehlerquellen steigt hingegen in gleichem Maße. Vor alle die Fehlersuche kann sich schwierig gestalten und ist ohne ausführliches Logging praktisch nicht bewerkstelligen. Weitere technische Herausforderungen wie zum Beispiel unterschiedliche Version der verwendeten Technologien auf unterschiedlichen Betriebssystemen erhöhen ebenfalls die Komplexitätsebene.

7 ANHANG

7.1 Code Repository / Sources

https://git-iit.fh-joanneum.at/raineran16/ServerLoadSimMon.git

7.2 Links

Links für die benötigte Software befinden sich in der "readme.md" Datei im Code-Repository:

https://git-iit.fh-joanneum.at/raineran16/ServerLoadSimMon/blob/master/README.md