Verteiltes Programmieren mit Space Based Computing Middleware – SS 2013 Übungsbeispiel 2

Die Aufgabe des zweiten Beispiels ist die Erweiterung der ersten Abgabe. Die Aufgabe dient dem Vergleich der Technologien. Sie sollen erkennen, welche Technologien eine leichte Erweiterbarkeit erlauben und wo eventuell Architecture Limiter/Breaker versteckt sind.

ALLE Beispiele müssen für eine positive Beurteilung rechtzeitig gelöst werden.

Keine Plagiate! Es ist verboten, Lösungen von vorherigen Semestern zu verwenden. Wir werden dies, teilweise automatisiert, überprüfen. Sollten Teile der Übung abgeschrieben sein, wird ein negatives Zeugnis ausgestellt.

Abgaben:

Die Deadline für das zweite Beispiel ist am **16. Juni 2013 23:55**.

Ihre Abgabe umfasst eine ZIP-Datei, inklusive aller verwendeten Sourcen, Build-Scripts und geforderten Dokumente.

Die Abgabe ist zwingend über einen Upload im TUWEL (vor dem eigentlichen Abgabegespräch) zu tätigen! Beim Abgabegespräch soll dann die Funktionsweise der Lösung demonstriert und erklärt werden.

Beispiel 2 - Erweiterte Pizzeria

Folgende Erweiterungen/Änderungen sind durchzuführen:

Lieferservice:

Um sich auf dem Markt besser zu etablieren, startet die Pizzeria ein Lieferservice. Die Bestellungen werden per Telefon angenommen. Die maximale Anzahl der Pizzen pro Bestellung ist 4. Für die Zustellung ist ein(e) neue(r) Akteur/-in zuständig und zwar ein(e) Fahrer/-in. Der Ablauf einer Bestellung mit Lieferung ist folgender:

- Die Bestellung und die Lieferadresse werden von einem/einer Kellner/-in notiert und an die Küche mit einer Lieferungsnummer weitergeleitet.
- Dort werden die Pizzen wie üblich von einem/einer oder mehreren Köchen/-innen zubereitet und auf die Theke gelegt.
- Wenn alle Pizzen einer Bestellung fertig sind, holt sie der/die Fahrer/-in ab, erstellt dafür eine Rechnung und liefert an den/die Kunden/-in.
- Der/die Fahrer/-in meldet den Erfolg (oder Misserfolg) der Lieferung an die Pizzeria.

Unter der Lieferadresse ist eine dem/der Kunden/-in zugeordnete virtuelle Adresse zu verstehen (z.B. eine Container-URI), an die der/die Fahrer/-in die Pizzen schickt. Die Bezahlung der Rechnung muss nicht modelliert werden, d.h. der/die Fahrer/-in wartet nach

der Lieferung auf keine Bestätigung des/der Kunden/-in. Eine Lieferung kann scheitern, wenn die Lieferadresse nicht erreichbar ist. In diesem Fall werden die Pizzen durch den Fahrer vernichtet (oder gegessen ;-)). Da die Lieferung auch eine gewisse Zeit dauert (3 Sekunden), haben solche Bestellungen in der Küche den Vorrang gegenüber internen Bestellungen von Gästen der Pizzeria. Das heißt es darf keine neue Gastbestellung abgearbeitet werden, falls es eine offene Lieferbestellung gibt. Wenn mit der Abarbeitung einer Gastbestellung in der Küche schon begonnen wurde, wird diese allerdings fortgesetzt und darf nicht zur Seite gelegt werden.

Load Balancing:

Es wurde eine zweite Pizzeria-Filiale eröffnet, die unabhängig von der ersten Filiale läuft. Um die Filialen in Stoßzeiten zu entlasten, sollen Lieferbestellungen (aber keine Gastbestellungen) bei Bedarf an Partnerfilialen umgeleitet werden können, wenn es ansonsten zu unnötig hohen Wartezeiten kommen könnte (für Lieferungen und Gäste). Implementieren Sie einen Load-Balancer als eigenen Prozess, der die Last regelmäßig unter allen Filialen verteilt. Wählen Sie dabei eine sinnvolle Strategie, bei der die Auslastung einigermaßen gleichmäßig erfolgt, ohne dass der Load-Balancer einen zu hohen Overhead erzeugt. Lieferbestellungen dürfen nach der Aufnahme durch den Kellner zwischen den Filialen transferiert werden, aber nur wenn mit der Zubereitung der Pizzen noch nicht begonnen wurde.

Zusammenfassung der Aktivitäten:

Liefer-Kunde:

- Sendet eine Bestellung mit Lieferadresse an eine Pizzeria-Filiale.
- Wartet auf die Lieferung.

Fahrer:

- Erstellt eine Rechnung, wenn die Pizzen für eine Lieferung fertig sind.
- Stellt die Lieferung an den/die Kunden/-in zu (schreibt Pizzen und Rechnung an die virtuelle Lieferadresse).
- Meldet eine erfolgreiche Lieferung an die Pizzeria. Falls die Lieferung scheitert, wird dies ebenfalls gemeldet.

Kellner (zusätzlich zu bisherigen Aufgaben):

- Nimmt die Lieferung auf und übermittelt sie an die Küche.

Load-Balancer:

- Überwacht die Last in allen Filialen.
- Wenn nötig transferiert er die Lieferbestellungen zwischen den Filialen.

Persistenz & Recovery:

Analyse:
Überlegen Sie, wie Sie für beide Lösungen Persistenz und Recovery umsetzen würden. Damit sind die nötigen Maßnahmen gemeint, um alle Programmteile (Akteure/-innen und/oder die Pizzeria selbst) jederzeit abstürzen lassen zu können. Nach dem Neustart der Pizzeria sollen alle Daten wiederhergestellt werden.

Implementierung (Bonusaufgabe):
Implementieren Sie Persistenz und Recovery für eine oder beide Implementierungen. Sie dürfen die eingebauten Persistenzmöglichkeiten der von Ihnen verwendeten Technologien ausnutzen, falls es diese gibt. Diese Aufgabe ist optional.

Allgemeines:

Es soll jederzeit nachvollziehbar sein, welche(r) Akteur/-in welche Aufgaben ausgeführt hat. Jede(r) Akteur/-in wird dabei durch eine eigene ID identifiziert. Alle Beteiligten können dynamisch hinzugefügt und weggenommen werden.

Jede(r) Akteur/-in stellt für sich ein kleines unabhängiges Programm dar (mit eigener main-Methode) mit der Ausnahme von Gruppen und Liefer-Kunden, die von anderen Programmen erzeugt und verwendet werden können.

Deadline: 16. Juni 2013 23:55 im TUWEL

Task 2.1 – GUI

Erweitern Sie das Pizzeria-GUI entsprechend den zusätzlichen Anforderungen, damit auch die Bestellungen mit Lieferung überwacht werden können.

- Anzeige aller aktiven und abgeschlossenen Lieferungen
    - Lieferungs-ID
    - Lieferadresse
    - Status (bestellt, wird ausgeliefert, erfolgreich/gescheitert)
    - beteiligte Akteure/-innen
        - ID jeweils für Bestellung und Zustellung
    - Daten zu jeder bestellten Pizza
        - Pizza-Typ
        - Zustand (bestellt, in Zubereitung, fertig)
        - verantwortliche(r) Koch/Köchin (ID)

Falls eine Lieferung vom Load-Balancer an eine andere Filiale ausgelagert wird, soll dies auf beiden Seiten ersichtlich sein (inkl. der ID des Load-Balancers).

Erweitern Sie außerdem das Gruppen-GUI, um Lieferbestellungen unterstützen zu können:

- Erstellung von Lieferkunden/-innen. (Jede(r) Lieferkunde/-in muss als eigener, parallel laufender Thread gestartet werden!)
- Es sollen beliebig viele Lieferkunden/-innen gleichzeitig erzeugt werden können.
- Es sollen 1-4 Pizzen angegeben werden können.
- Es soll eine Lieferadresse angegeben werden können

Die Tätigkeit der Lieferkunden/-innen kann automatisch erfolgen, d.h. nach dem Erstellen wird keine weitere User-Interaktion benötigt. Außerdem soll das GUI eine Übersicht über die erstellten Lieferkunden/-innen zeigen. Für jede(n) Lieferkunden/-in sollen folgende Informationen angezeigt werden:

- ID
- Lieferadresse (z.B. Container-URI)
- Bestellte Pizzen
- Status (bestellt, geliefert, bezahlt)

Zusätzlich soll pro Gruppe bzw. Lieferkunden die Filiale angegeben (und in der Übersicht angezeigt) werden können, an die die Bestellung gerichtet ist. Dadurch benötigt nicht jede Filiale ein eigenes Gruppen-GUI.

Task 2.2 – Space-Implementierung

Erweitern Sie die Implementierung aus Task 1.2 um die neuen Anforderungen.

Jede Bestellung/Lieferung und jede Pizza soll durch (mindestens) ein eigenes Objekt im Space repräsentiert sein (und nicht etwa in einer Liste, die als Ganzes in den Space geschrieben wird). Jede(r) Fahrer/-in, jede(r) Koch/Köchin, jede(r) Kellner/-in und auch der Load-Balancer wird dabei als eigenständiger Prozess gestartet. Mit Ausnahme des Load-Balancers sind die Akteure/-innen dabei immer genau einer Filiale (und damit einem Space) zugeordnet. Diese Prozesse können als einfache Konsolenapplikationen implementiert werden, bei denen die ID als Argument übergeben wird. Gruppen und Lieferkunden/-innen sind ebenfalls als unabhängige Threads (mit gemeinsamer GUI) zu realisieren.

Jede Filiale muss in einem eigenen Space gestartet werden. Die Space-URI kann beim Start als Parameter übergegeben werden. Für die Lieferungen benötigen Lieferkunden auch einen eigenen Space. Für jede Lieferadresse soll ein eigener Container erstellt werden.

Die Kommunikation zwischen allen Beteiligten (Köche/-innen, Kellner/-innen, Gruppen, Fahrer/-innen, Lieferkunden/-innen) darf nur über Spaces erfolgen, die vor dem Starten der Prozesse evtl. initialisiert werden müssen. Die Anzeigetafel aus Task 2.1 erhält alle Informationen nur über den jeweiligen Space. Prozesse für alle Akteure/-innen sollen jederzeit dynamisch gestartet oder geschlossen werden können, ohne dass die Funktionalität beeinträchtigt wird. Achten Sie auch auf Fehlerfälle, falls ein oder mehrere Spaces offline sind (Transaktionen funktionieren in derzeitigen Space-Versionen meist nur innerhalb eines Spaces!).

Task 2.3 – Alternativ-Implementierung

Erweitern Sie die Implementierung aus Task 1.3 um die neuen Anforderungen.

Die einzelnen Prozesse können als einfache Konsolenapplikationen implementiert werden, bei denen die ID als Argument übergeben wird. Mit Ausnahme des Load-Balancers sind die Akteure/-innen dabei immer genau einer Filiale zugeordnet. Prozesse für Akteure/-innen sollen jederzeit dynamisch gestartet oder geschlossen werden können, ohne dass die Funktionalität beeinträchtigt wird.

Analog zur Space-Lösung muss jede Filiale voneinander unabhängig sein (ohne gemeinsamen Server). Die mit den Lieferadressen verknüpften Ressourcen sollen unter Kontrolle der Lieferkunden/-innen stehen.

Die Lösungen aus Task 2.2 und 2.3 müssen nicht kompatibel sein, d.h. Akteure/-innen des Space-basierten Programms müssen nicht mit Akteuren/-innen des Alternativ-Programms interagieren können.

Benchmark

Um die Skalierbarkeit Ihrer Implementierungen zu testen, werden Benchmarks durchgeführt.

Task 2.4

Sie erhalten Zugang zu vier Servern. Diese sind mit einer, zwei, vier und acht CPU(s) ausgestattet. Auf den Servern ist Linux mit dem OpenJDK 7 installiert. Eventuelle Abhängigkeiten wie Datenbanken müssen vorher mit den Tutoren abgeklärt werden.

Auf jedem dieser Server ist folgender Benchmark sowohl mit der Space-, als auch mit der Alternativimplementierung durchzuführen (also pro Implementierung 4 Benchmarks):

Starten von 2 Pizzerien (pro Test jeweils auf demselben Server):

Vorproduzierte Bestellungen:

100 Lieferbestellungen an Filiale 1 mit Konfiguration:

- 2x Margarita
- 1x Salami
- 1x Cardinale

100 Lieferbestellungen an Filiale 2 mit Konfiguration:

- 1x Margarita
- 1x Salami
- 1x Cardinale

100 Lieferbestellungen an Filiale 2 mit Konfiguration:

- 1x Margarita
- 1x Salami

Die Lieferadresse soll bei allen Bestellungen gleich sein und auf eine von den Pizzerien unabhängigen Ressource (z.B. einen Container in einem separaten Space am selben Server) referenzieren. Threads für Lieferkunden, die auf die Bestellungen warten, sollen nicht gestartet werden.

Akteure/-innen pro Filiale:

2 Kellner/-innen

2 Köche/-innen

2 Fahrer/-innen

1 Load-Balancer

Wichtig: Die Bestellungen sollen vor dem Benchmark erzeugt werden.

Ablauf eines Benchmark-Durchlaufs:

- Starten aller Server und Akteure/-innen
- Erstellen von Bestellungen
- Senden eines Startsignals an alle Akteure/-innen
- 60 Sekunden warten
- Senden eines Stoppsignals an alle Akteure/-innen
- Zählung der abgearbeiteten Bestellungen

Beim Benchmark sollen alle Wartezeiten (z.B. fürs Zubereiten und Ausliefern) deaktiviert werden. Die Ergebnisse der Benchmarks sollen tabellarisch und grafisch dargestellt werden. Außerdem sollen eine Zusammenfassung und eine kurze Interpretation der Ergebnisse als eigenes Dokument erstellt werden. Falls das System schon vor Ablauf der Zeit mit der Auslieferung fertig ist, können Sie für den Vergleich die Anzahl der in 60 Sekunden abgearbeiteten Bestellungen entsprechend hochrechnen. Dokumentieren Sie aber jedenfalls auch die tatsächlich produzierte Anzahl und die benötigte Zeit.

Ablauf der Serverzuteilungen:

Jede Gruppe kann Timeslots ab dem 1. Juni 2013 per Mail beantragen.

Ein Timeslot dauert von **06:00 bis 14:00 Uhr** bzw. **16:00 bis 24:00 Uhr** eines Tages.

Eine Gruppe kann, sobald jede Gruppe mindestens einen Timeslot zugewiesen bekommen hat, weitere Timeslots beantragen.

Für jeden Timeslot können zwei Gruppen gleichzeitig einen Zugang beantragen. Die reservierten Timeslots sind unter folgender Adresse abrufbar:

https://spreadsheets.google.com/ccc?key=0An_dk-igNwkydEdPX09LT0hOQnlXaTdGcFp3YzZFb0E&hl=en&authkey=CJ-ojeYM

Eine Anleitung zum Verbinden mit den Servern finden Sie im TUWEL.

Benchmarken von zu Linux inkompatiblen Lösungen:

Sollte Ihr Programm nicht unter Linux lauffähig sein, führen Sie die Tests auf mindestens zwei PCs mit wenn möglich unterschiedlicher Konfiguration aus und dokumentieren Sie die Hardwareausstattung dieser PCs ausführlich.

Dokumentation

Deadline: 16. Juni 2013 23:55 im TUWEL

Task 3:

Erstellen Sie eine kurze Präsentation Ihrer Lösungen mit Architektur, Vorteilen, Nachteilen, Aufwandsabschätzung, etc. und vergleichen Sie dabei die von Ihnen evaluierten bzw. verwendeten Technologien. Vergessen Sie nicht folgende Punkte zu berücksichtigen:

- Welche Koordinationsmodelle haben Sie benutzt und warum?
- Lösungsvorschlag für Persistenz & Recovery für beide Lösungen
- Analyse der Benchmarkergebnisse
- Wie viele Zeilen Code umfasst jeder (Sub-)Task?
- Gesamter Arbeitsaufwand in Stunden pro (Sub-)Task

Dokumentieren Sie die Erfahrungen und Schwierigkeiten bei der Erweiterung Ihrer ursprünglichen Lösung, um den Anforderungen der zweiten Angabe gerecht zu werden und beschreiben Sie den Einfluss der von Ihnen gewählten Technologien.

Zu beachten:

Versuchen Sie alle Aufgaben möglichst effizient und einfach zu lösen. Für eine positive Beurteilung müssen alle Tasks rechtzeitig gelöst werden.

Sie sollen ein „Build-Tool“ für die Aufgabe verwenden. Hierfür kann zwischen Maven und Ant gewählt werden. Die Maven-Dateien (pom.xml usw.) bzw. Ant-Dateien, die zum Kompilieren und Ausführen des Programms benötigt werden, müssen Sie auch abgeben. Alternativ können auch Scripts zum Starten des Programms mitgeliefert werden (wenn möglich für Windows und Linux). Außerdem muss in einem Readme-File genau dokumentiert werden, wie die einzelnen Applikationen kompiliert und gestartet werden können.

Als Programmiersprache können Sie Java und/oder C# verwenden.

Hilfreiche Dokumentation:	Abgabe:
• VO-Folien • MozartSpaces Tutorial • XcoSpaces Tutorial	• Abgabe des Source Codes (kommentiert) und der restlichen Dokumente inkl. Präsentationsfolien im TUWEL zum angegebenen Termin. • Präsentation des Programms auf ihrem Rechner (Laptop) • Ausfüllen des Feedbackformulars (bei der Abgabe)