7 Übungsaufgabe #7: Amazon Web Services

In dieser Aufgabe soll eine modifizierte Version des Web-Service aus der ersten Aufgabe mittels der *Amazon Web Services* in die Cloud-Computing-Umgebung von Amazon verlagert werden. Die Server-Implementierung `MWFacebookService` (wird bereitgestellt) und der Client `MWClient` sind zu erweitern. Der Server wird um Methoden für die Erfassung der Auslastung des Dienstes, der Client um Methoden für die Erzeugung hoher synthetischer Systemlast ergänzt. Diese Funktionalität wird im zweiten Teil der Übungsaufgabe dafür verwendet, den Dienst auf eine der Systemlast angemessene Anzahl von Server-Instanzen zu verteilen. Die Client-Anfragen werden hierbei von einem Load-Balancer gleichmäßig auf die Server-Instanzen verteilt.

7.1 Erzeugung und Kontrolle von Systemlast (für alle)

In der ersten Teilaufgabe gilt es, die Implementierung des Web-Service so zu erweitern, dass durch den Client eine Auswertung der Systemlast des Server möglich ist. Außerdem soll der Client `MWClient` um die Funktionalität erweitert werden, parallel mehrere Anfragen an den Server zu senden. Da die Implementierung des Web-Service `MWFacebookService` mehrfädig arbeitet, kann diese in der vorliegenden Implementierung bereits parallel mehrere Anfragen verarbeiten.

7.1.1 Server

Es ist die Auslastung des Server zu kontrollieren. Die Server-Implementierung von `MWFacebookService` gilt es hierfür um die Methode `getServerStatus()` zu erweitern. Diese Methode soll für ein vom Aufrufer bestimmtes Intervall (Granularität: Sekunden) die Anzahl der verarbeiteten Anfragen ermitteln. Für alle Server-Zugriffe muss daher ein Zeitstempel für die entsprechende Anfrage protokolliert werden. Die Methode soll als Web-Service-Schnittstelle des Dienstes `MWFacebookService` verfügbar sein.

```
public interface MWFacebookServiceInterface {
    public String[] searchIDs(String name);
    public String getName(String id) throws MWUnknownIDException;
    public String[] getFriends(String id) throws MWUnknownIDException;
    public String[][] getFriendsBatch(String[] ids) throws MWUnknownIDException;
    public int getServerStatus(int interval);
}
```

7.1.2 Client

Der Client `MWClient` ist um drei Funktionen zu erweitern. Bei Ausführung mit dem Parameter `status <i>` soll der Client die Auslastung des Facebook-Server in einem Intervall von i Sekunden abfragen und anschließend die Anzahl der in diesem Zeitraum verarbeiteten Server-Anfragen ausgeben.
Außerdem soll dem Client ermöglicht werden, die Antwortzeit des Facebook-Server zu messen. Für die Ermittlung der Antwortzeiten hat der Client die Bearbeitungszeit seiner Anfragen zu protokollieren. Hierfür wird gefordert, dass die Differenz Δt (Granularität: Millisekunden) zwischen Eintritt und Verlassen einer abfragenden Methode protokolliert wird.
Um eine hohe Server-Last synthetisch zu erzeugen, wird gefordert, dass der Client um einen Ausführungsmodus erweitert wird. Mit den Aufruf-Parametern `executor <m> <n> <query>` soll der Client einen Thread-Pool der Größe m erstellen und parallel die Suchanfrage `query` unter Zuhilfenahme des Thread-Pool genau n Mal ausführen. Diese Suche wird als *Executor*-Suche bezeichnet. Während der Ausführung solch einer Suche gilt es, in regelmäßigen Abständen die mittlere Bearbeitungsdauer der Anfragen auszugeben. Die Executor-Suche ist in der Klasse `MWClientThreadPool` zu implementieren.
Vorbereitend für die folgenden Teilaufgaben der Übung soll vor Instanziierung der Klasse `MWFacebookService` zur Laufzeit des Client überprüft werden, ob die Java-Umgebungsvariable `MW_SERVER_URL` gesetzt ist. Dies geschieht unter Verwendung der Klassenmethode `getProperty()` der Klasse `System` (Paket `java.lang`). Falls diese Umgebungsvariable gesetzt ist und eine korrekte URL beinhaltet, soll für die Instanziierung von `MWFacebookService` der Konstruktor mit der Signatur `(URL, QName)` verwendet werden. Im Folgenden wird beim Aufruf des Client mittels `java -DMW_SERVER_URL=<url> mw.MWClient <command>` der Client dynamisch an den mit der Adresse `<url>` spezifizierten Server gebunden.

Aufgaben:

→ Anpassung der Schnittstelle `MWFacebookServiceInterface`
→ Erweiterung der Klassen `MWMyFacebookService` und `MWClient`
→ Implementierung der Klasse `MWClientThreadPool`

Hinweise:

- Nach Modifikation der Web-Service-Schnittstelle muss erneut `wsimport` aufgerufen werden, damit die Generierung des Client `MWClient` gelingt.
- Die Implementierung der Executor-Suche soll unter Verwendung des Java-`ExecutorService` geschehen.

7.2 Verlagerung des Dienstes in die Cloud-Umgebung Amazon EC2 (für alle)

Der Server und der Client sollen in der *Amazon Elastic Computing Cloud* (Amazon EC2) ausgeführt werden. Hierfür wird das Maschinen-Abbild *ami-e43ccd93* bereitgestellt. Der Programmcode von Server und Client soll in Java-Archive (JAR) gekapselt auf dem *Amazon Simple Storage Service* (Amazon S3) hinterlegt werden. Bei der Instanziierung einer virtuellen Maschine vom Typ *ami-e43ccd93* wird beim Start des Betriebssystems das Start-Skript `http://i4mw.s3.amazonaws.com/rc.local.i4mw` ausgeführt, das eine Reihe von Aufruf-Parametern auswertet. Diese Aufruf-Parameter müssen beim Start der virtuellen Maschine spezifiziert werden, da sie darüber entscheiden, welcher Dienst auf der Instanz letztendlich ausgeführt wird (Server oder Client).
Für die Übergabe der Aufruf-Parameter wird das Nutzdatenfeld `user-data` von Amazon EC2 verwendet. In diesem Feld sind die Aufruf-Parameter als Schlüssel-Wert-Paare mit dem Format `key=value` verzeichnet, die *ausschließlich* durch einzelne Semikola voneinander getrennt sind. Das Start-Skript wertet diese Meta-Daten aus und prüft, ob folgende Schlüssel existieren: `group`, `jar`, `parameters` (optional), `java_defines` (optional) und `hook` (optional); Beispiel: `group=gruppe0-bucket;jar=MWFacebookServer.jar;parameters="42 4711"`.
Aus den Angaben für die Schlüssel `group` und `jar` erzeugt das Start-Skript die URL `http://<group>.s3.amazonaws.com/<jar>`. Diese URL verweist auf Daten, die auf Amazon S3 hinterlegt sind. Der Wert des Schlüssels `<group>` spezifiziert hierbei den *Bucket*, der Wert des Schlüssels `<jar>` den Dateinamen des in diesem Bucket abgelegten Java-Archives. Wenn das Start-Skript in der Lage ist, erfolgreich auf die Daten zuzugreifen, wird im Folgenden das spezifizierte Java-Archiv auf die Instanz geladen und mit den optionalen Parametern (`java_defines`, `parameters`) wie folgt ausgeführt:

```
$ java -D<java_defines> -jar <jar> <parameters>
```

Die Instanziierung der virtuellen Maschinen kann wahlweise über das Web-Interface, die Kommandozeilen-Tools von Amazon oder die Java-Bibliotheken geschehen. In jedem Fall ist es notwendig, dass bei der Instanziierung der Name eines EC2-Schlüsselpaares angegeben wird, das für den späteren Zugriff auf die Instanz mittels `ssh` dient.

Aufgaben:

→ Generierung der Java-Archive `MWFacebookServer.jar` und `MWClient.jar`
→ Erstellen eines Bucket auf Amazon S3, dortiges Hinterlegen der Java-Archive
→ Betrieb einer Server- und einer Client-Instanz in Amazon EC2

Hinweise:

- Zur Laufzeit des Start-Skriptes (z. B. beim Start von Java) ist die Host-Adresse der aktuellen Instanz in der Umgebungsvariable `I4MW_HOSTNAME` vermerkt.
- Beim Zugriff auf die in Amazon S3 hinterlegten Daten durch die Instanzen wird auf eine Authentifizierung verzichtet, daher müssen die Java-Archive unter Kenntnis der URL öffentlich zugänglich sein.

7.3 Lastverteilung und dynamische Skalierung

Der erweiterte Facebook-Service wird im abschließenden Teil der Übung um zwei Komponenten ergänzt (siehe Abbildung 1). Ein Load-Balancer leitet eingehende Client-Anfragen an die ihm zugeordneten Server-Instanzen im *Round-Robin*-Verfahren weiter und übermittelt deren Antwort anschließend an die anfragende Client-Instanz. Die Anzahl der im Betrieb befindlichen Server-Instanzen soll sich dynamisch an die aktuelle Systemlast der Server-Instanzen anpassen. Die Anzahl der verarbeiteten Client-Anfragen *einer* Server-Instanz innerhalb eines bestimmten Zeitintervalls gilt als Maß für die aktuelle Last des Gesamtsystems.

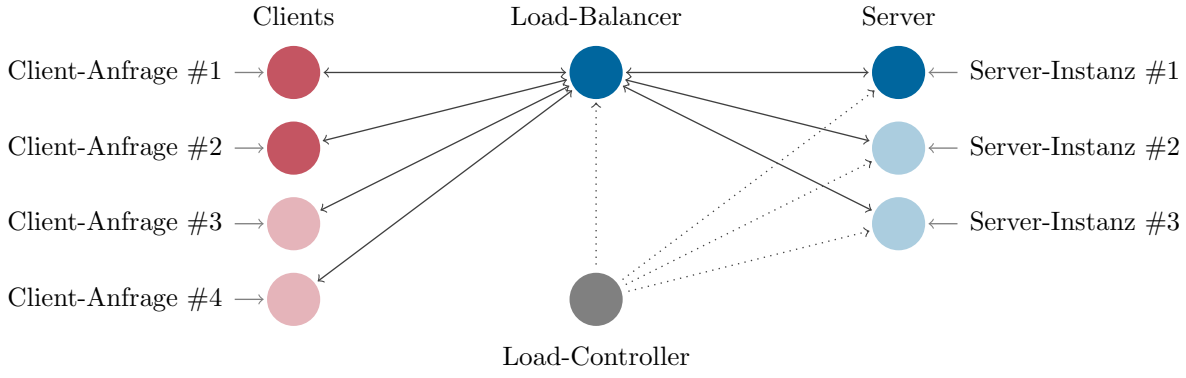

Abbildung 1: Systemübersicht

Das System wird durch den Load-Balancer auf TCP-Ebene entkoppelt. Dies geschieht für die Kommunikationspartner Client und Server völlig transparent. Zum Lösen dieser Aufgabe ist es daher *nicht* notwendig, Programmcode des Client oder Server zu modifizieren. Die einzige aktive Aufgabe des Load-Balancer besteht in der

Zuordnung und Weiterleitung von Verbindungsanfragen und den dazugehörigen Antworten. Der Load-Controller hingegen ist eine aktivere Komponente und zeichnet sich für Erstellung des Load-Balancer, Instanziierung und Terminierung von Server-Instanzen sowie die Lastüberwachung verantwortlich.

7.3.1 Implementierung des Load-Controller (für alle)

In dieser Teilaufgabe gilt es, den Load-Controller zu implementieren. Nach dem Programmstart ist der Load-Controller zunächst dafür verantwortlich, einen Load-Balancer in Amazon EC2 anzulegen und eine initiale Server-Instanz zu starten, auf der der Web-Service `MWFacebookService` ausgeführt wird. Diese initiale Server-Instanz muss dem Load-Balancer zugeordnet werden. Sobald die initiale Server-Instanz betriebsbereit ist, können von Clients unter Verwendung der Adresse des Load-Balancer transparent Anfragen an die Server-Instanz gestellt werden. Die Clients verbinden sich zu dem Web-Service über TCP-Port 18081, daher muss der Load-Balancer Verbindungsanfragen auf diesem Port annehmen und an die ihm zugeordneten Server-Instanzen weiterleiten. Nach erfolgreicher Erstellung des Load-Balancer soll der Load-Controller eine URL in der Form `http://<lb-name>.eu-west-1.elb.amazonaws.com:18081/MWFacebookService?wsdl` auf der Kommandozeile ausgeben. Diese Adresse wird im Folgenden beim Start an die Client-Instanzen übergeben, um die Lokation des Web-Service bekannt zu machen (siehe Teilaufgabe 7.1.2).
Im weiteren Betrieb des Systems ist der Load-Controller dafür zuständig, kontinuierlich die Systemlast zu überwachen und im Falle einer Über- oder Unterbeanspruchung der Server-Instanzen entsprechende Gegenmaßnahmen einzuleiten. Um festzustellen, ob der Web-Service über- oder unterbelastet ist, soll der Load-Controller durch periodische Aufrufe der vom Web-Service exportierten Methode `getServerStatus()` (siehe Teilaufgabe 7.1.1) die momentane Last von *einem* der Server in Erfahrung bringen und anschließend mit einem selbst zu bestimmenden Schwellwert vergleichen. Bei Überbeanspruchung gilt es, neue Server-Instanzen zu starten und diese dem Load-Balancer zuzuordnen. Stellt der Load-Controller hingegen eine Unterbeanspruchung des Dienstes fest, sollen nicht mehr benötigte Ressourcen freigegeben werden. Dies geschieht durch Beenden einer im Betrieb befindlichen Server-Instanz, darf aber nur dann durchgeführt werden, wenn mehrere Server-Instanzen aktiv sind.
Damit durch kurzfristige Lastspitzen bzw. kurzfristiges Ausbleiben von Client-Anfragen keine Server-Instanzen gestartet bzw. terminiert werden, gilt es, entsprechende Vorkehrungen zu treffen. Solch ein Schutz kann beispielsweise durch Zähler oder Timer realisiert werden.
Die Funktionsweise des Load-Controller wird überprüft, indem ein oder mehrere Clients gestartet werden, die mittels einer Executor-Suche (siehe Teilaufgabe 7.1.2) eine große Anzahl paralleler Anfragen über die Adresse des Load-Balancer an die Server-Instanzen schicken.

Aufgabe:

→ Implementierung der Klasse `MWLoadController`

Hinweise:

- Für diese Aufgabe benötigte externe Java-Bibliotheken sind unter `/proj/i4mw/pub/aufgabe7` hinterlegt.
- Load-Balancer und Server-Instanzen sollen in der gleichen *Region* von Amazon EC2 betrieben werden.
- Die lasterzeugenden Clients und der Load-Controller können wahlweise lokal oder in der Umgebung von Amazon EC2 gestartet werden.

7.3.2 Load-Controller mit Bereitschafts-Instanz (optional für 5,0 ECTS)

Im letzten Teil der Übungsaufgabe gilt es, den Load-Controller aus Teilaufgabe 7.3.1 um eine Bereitschafts-Instanz zu erweitern. Im Fall einer Überbeanspruchung des Dienstes verkürzt diese Instanz die Reaktionszeit.
Da die Instanziierung einer neuen Server-Instanz Zeit benötigt, ist es wünschenswert, beim Betrieb von n Server-Instanzen lediglich $n - 1$ Instanzen dem Load-Balancer zuzuweisen und eine einzelne, sogenannte Bereitschafts-Instanz *ohne* Zuweisung zu betreiben („hot standby“). Wird durch den Load-Controller im Folgenden eine Überlastung festgestellt, kann die einzelne Instanz ohne weitere Verzögerung dem Load-Balancer zugewiesen werden, um die Überlast abzufangen. Im direkten Anschluss an diese Zuteilung soll der Load-Controller eine neue Instanz starten, die fortan als Bereitschafts-Instanz genutzt wird.
Im Falle einer Unterbeanspruchung des Systems soll die Bereitschafts-Instanz unberührt bleiben und eine dem Load-Balancer zugeordnete Server-Instanz terminiert werden. Es gilt sicherzustellen, dass zu jedem Zeitpunkt mindestens eine dem Load-Balancer zugeordnete Server-Instanz *und* eine Bereitschafts-Instanz in Betrieb sind.

Aufgabe:

→ Erweiterung der Klasse `MWLoadController`

Abgabe: am 28.1.2015 in der Rechnerübung