JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

GK9.2 High Availability

Systemtechnik Labor
5BHIT 2017/18

Martin Wölfer

Version 1.51
Begonnen am 9. November 2017
Beendet am 15. November 2017

Note:
Betreuer: MICHT

Inhaltsverzeichnis

1 Einführung

Als Lastverteilung (englisch Load Balancing) bezeichnet man in der Informatik der Verteilung von umfangreiche Berechnungen oder große Mengen von Anfragen auf mehrere parallel arbeitende Systeme. Dies kann sehr unterschiedliche Ausprägungen haben. Eine einfache Lastverteilung findet zum Beispiel auf Rechnern mit mehreren Prozessoren statt. Jeder Prozess kann auf einem eigenen Prozessor ausgeführt werden. Man unterscheidet eine Reihe von Algorithmen, genannt Load Balancing Methoden, um diese Verteilung durchzuführen.

1.1 Ziele

1.2 Voraussetzungen

- Grundlagen zu Load Balancing
- Java Programmierkenntnisse

1.3 Aufgabenstellung

Es soll ein Load Balancer mit mindestens 2 unterschiedlichen Load-Balancing Methoden implementiert werden. Eine Kombination von mehreren Methoden ist möglich. Die Berechnung bzw. das Service ist frei wählbar!

Folgende Load Balancing Methoden stehen zur Auswahl:

- Weighted Round-Round
- Least Connection
- Weighted Least Connection
- Agent Based Adaptive Balancing / Server Probes

Es sollen die einzelnen Server-Instanzen in folgenden Punkten belastet werden können:

- Memory (RAM)
- CPU Cycles

Bedenken Sie dabei, dass die einzelnen Load Balancing Methoden unterschiedlich auf diese Auslastung reagieren werden. Dokumentieren Sie dabei aufkommenden Probleme ausführlich.

2 Ergebnisse

2.1 Projektaufbau

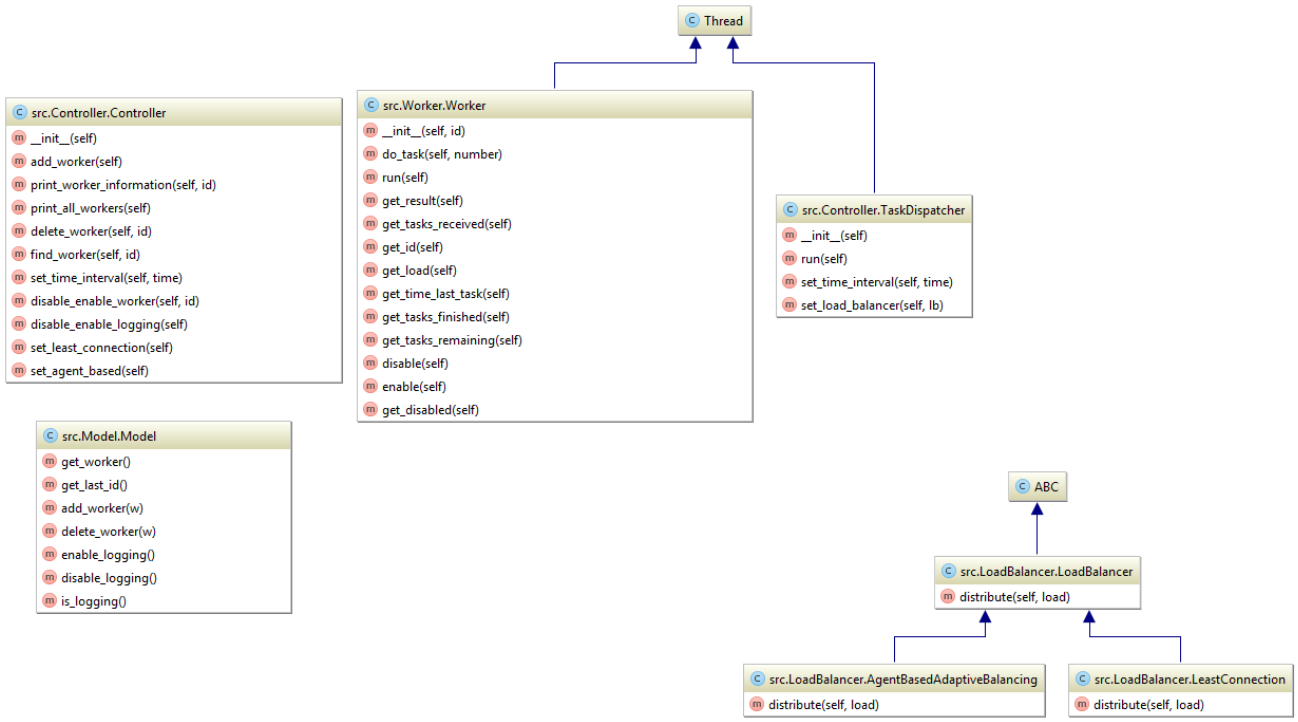

Abbildung 1: Klassendiagramm

Das Klassendiagramm zeigt die Relationen zwischen den einzelnen Klassen.

2.2 Controller

Der Controller kümmert sich um die Instanziierung der Klassen und um die User-Eingaben. Im Controller ist zusätzlich das User-Interface zu finden, welches in einer Endlosschleife User eingaben entgegennimmt und dem Controller übergibt und dieser dann an die zugehörigen Klassen weiterleitet.

2.3 CLI Interface

```
To add a new Worker, write 'A'
To get information of a Worker, write 'I #id'
To delete a Worker, write 'X #id'
To list all Worker's information, write 'S'
To set time interval between each task dispatch, write 'T #seconds'
To disable\enable a Worker, write 'D #id'
To enable\disable logging, write 'L'
To use the LeastConnection method, write 'LC'
To use the AgentBasedAdaptiveBalancing method, write 'AB'
To disable\enable a Worker, write 'D #id'
To exit the program, write 'E'
```

Abbildung 2: Ein Command Line interface ist bereitgestellt für den User

Der User kann folgende Tätigkeiten ausführen:

- einen neuen `Worker` hinzufügen
- die Information von einem bestimmten `Worker` ausgeben
- einen `Worker` löschen
- die Informationen von allen `Worker` ausgeben
- den Zeit-Interval bestimmten in welchen die Tasks an den Load Balancer übergeben werden
- einen `Worker` an oder ausschalten
- Logging an oder ausschalten
- die LeastConnection balancing methode verwenden
- die AgentBasedAdaptiveBalancing methode verwenden
- das Programm beenden

2.4 Model

Das Model hält alle relevanten Daten. Da es in Python keine statischen Klassen gibt, werden alle Methoden als statische methode mit der Annotation `@staticmethod` versehen, um Zugriff auf die Funktionen ohne Instanziierung zu ermöglichen.

2.5 TaskDispatcher

Der TaskDispatcher ist ein Thread welcher alle x Sekunden einen Task an den `LoadBalancer` sendet. Diese Klasse muss ein Thread sein da dieser parallel zu den anderen Klassen laufen muss um reibungslos einen praktischen Fall zu simulieren.

In diesem Fall ist der Task lediglich eine immer um 1 inkrementierte Zahl welche anschließend vom `Worker` abgearbeitet wird.

2.6 Worker

Der Worker ist ein Thread, welcher Tasks in eine `queue` übernimmt und diese dann Stück für Stück abarbeitet. Wie diese abgearbeitet wird, ist je nach Anwendungsfall anders. Z.B. könnte in einem realistischen Programm Verbindungen entgegenommen werden, welche anschließend Anfragen senden und vom Worker bearbeitet und zurückgesendet werden.

In dem Fall wird von der Zahl, welche vom `LoadBalancer` übergeben wurde, die Fakultät zu der davorigen Zahl dazu gerechnet. Grund für diese Berechnung ist einen Task darzustellen, welcher exponentiell immer länger dauert um tatsächlich den Sinn eines Load Balancers zu beweisen.

Um die Load Balancing Methoden implementieren zu können, muss der Worker bestimmte Statistiken bereitstellen können. Er muss bekannt geben können wieviele Aufgaben er bereits zugeteilt bekommen hat, und unter welcher Belastung er gerade steht.

2.6.1 get_load()

Diese Methode gibt aus, unter welcher Belastung der Arbeiter gerade steht. Der Worker besitzt das Attribut `progress`, mit welchem mit der hilfe von `desired_number` ausgerechnet werden kann zu wieviel Prozent der Task abgeschlossen wurde.

```
1 load = int((self.progress / self.desired_number) * 100)
```

Nun wird mit einem provisorischem Switch-Case überprüft ob der Arbeiter `idle`, `overloaded` oder `disabled` ist.

Falls der Arbeiter `idle` ist, also momentan keinen Task abzuarbeiten hat, wird der Wert **0** zurückgegeben.

Falls der Arbeiter `overloaded` ist, also noch zusätzliche Arbeit zu verrichten hat nach dem momentanen Task welcher abgearbeitet wird, wird der Wert **99** zurückgegeben. Es wird überprüft ob in der `queue` mehr als 1 Wert vorhanden ist.

Falls der Arbeiter `disabled` ist, also extern abgeschalten wurde, wird der Wert **101** zurückgegeben.

```
1 if load == 100:
  return 0
3 elif self.is_disabled:
  return 101
5 elif self.__q.qsize() > 1:
  return 99
7 else:
  return load
```

2.7 LoadBalancer

Da es in Python keine Interfaces gibt, wird die LoadBalancer klasse als `Abstract Base Class`(ABC) definiert. Es wird die methode `distribute()` vorgegeben, in welcher bereits Code für das Logging vorhanden ist und vererbt werden kann.

Im Grunde genommen geht es beim Load Balancer darum, dass anhand von bestimmten Kriterien (je nach Methode unterschiedlich) einem bestimmten Worker der Task zugeteilt wird.

2.7.1 LeastConnection

Implementiert `LoadBalancer`. Es wird die Methode `distribute()` überschrieben, um die Funktionalität der Least Connection Methode zu implementieren.

Diese Methode teilt jenem Worker den Task zu, welcher bisher die wenigstens Tasks zugeteilt bekommen hat.

```
determined_worker = min(Model.get_worker(), key=lambda x: x.get_tasks_received())
determined_worker.do_task(load)
```

2.7.2 AgentBasedAdaptiveBalancing

Laut folgender **Seite** müssen folgende Bedingung gegeben sein vom Worker:

> Each server machine has to provide a file that contains a numeric value in the range between 0 and 102 representing the actual load on this server (0 = idle, 99 = overload, 101 = server down, 102 = administratively disabled).

Nun wird dieser Wert nicht über ein File übergeben, sondern über `get_load()`.

Diese Methode teilt jenem Worker den Task zu, welcher momentan der niedrigsten Belastung untersteht.

```
determined_worker = min(Model.get_worker(), key=lambda x: x.get_load())
```

Zusätzlich wird überprüft, falls jener Worker mit der niedrigsten `load`, eine load von `99` oder `101` besitzt, bedeutet dies dass jeder Server überlastet bzw. abgeschalten ist!

```
if determined_worker.get_load() == 101 or determined_worker.get_load() == 99:
print(''Task was dismissed because all Workers are overloaded or disabled!'')
return
else:
determined_worker.do_task(load)
```

2.8 Belastungstests

Es wurde beim Logging die CPU-Usage, die generelle RAM-Usage und die RAM-Usage vom Programm hinzugefügt. Dafür wurde das Package `psutil verwendet`:

```
print(''CPU Usage: '' + str(psutil.cpu_percent()))
print(''General RAM Usage: '' + str(psutil.virtual_memory()))  # physical memory usage
pid = os.getpid()
py = psutil.Process(pid)
memoryUse = py.memory_info()[0] / 2. ** 30  # memory use in GB
print('Process RAM Usage: ', memoryUse)
```

```
Tasks finished by workers: [2, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Tasks received by workers: [2, 2, 2, 1, 1, 1, 1, 1, 1, 1]
CPU Usage: 41.9
General RAM Usage: svmem(total=8474607616, available=984494080, percent=88.4, used=7490113536, free=984494080)
Process RAM Usage:  0.01114654541015625
Current Mode: Least Connection
```

Abbildung 3: Belastungstest von CPU und RAM

Es ist zu erkennen, je länger das Programm läuft desto mehr RAM wird verbraucht vom Programm.

```
Tasks finished by workers: [7, 7, 7, 7, 6, 6, 6, 6, 6, 6]
Tasks received by workers: [63317, 63317, 63317, 63317, 63317, 63316, 63316, 63316, 63316, 63316]
CPU Usage: 18.7
General RAM Usage: svmem(total=8474607616, available=998584320, percent=88.2, used=7476023296, free=998584320)
Process RAM Usage:  0.02791595458984375
Current Mode: Least Connection
```

Abbildung 4: Belastungstest von CPU und RAM nach längerer Belastung

Zu beachten: es wurde die Least Balancing Methode verwendet, deswegen wurde einige Mehr tasks zugeteilt als abgearbeitet. Darauf ist zurückzuführen dass mehr RAM in Verwendung ist, da jeder einzelner Worker den Task in der `queue` speichern muss, was natürlich im RAM Speicher verbraucht.

2.9 Konkrete Fragestellungen

2.9.1 Vergleichen Sie die verwendeten Load Balancing Methoden und stellen Sie diese gegenüber.

Der Vergleich ist zu sehen, wenn man sich anschaut welchen Workern wieviele Tasks zugewiesen wurden:

Es wurden **31** Tasks auf **10** Worker vom jeweiligen LoadBalancer verteilt:

Es ist zu sehen, dass das Agent Based Adaptive Balancing anfänglich nur einem Worker Tasks zuteilt. Dies liegt daran, dass die Tasks exponentiell länger werden und am Anfang sehr schnell abgearbeitet werden können. Erst wenn die Tasks länger zum abarbeiten brauchen, werden sie auch auch anderen Workern zugeilt.

Beim Least Connection Balancing ist die Funktionsweise klar zu sehen: Es wird systematisch den jeweiligen Arbeitern die Arbeit zugeteilt. Problem dabei besteht darin, dass beim Verteilen nicht auf die Belastung der einzelnen Arbeiter geachtet wird, das bedeutet dass wenn ein Arbeiter einen sehr aufwendigen Task zugeteilt bekommt, erhält er trotzdem gleich schnell wieder einen task wie jener Worker der einen sehr schnell abgearbeiteten Task zugeteilt bekommen hat.

```
Tasks received by workers: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [3, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [4, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [5, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [6, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [7, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [7, 2, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [8, 2, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [8, 3, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [9, 3, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [9, 4, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [9, 4, 1, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [10, 4, 1, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [10, 5, 1, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [10, 5, 2, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [11, 5, 2, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [11, 6, 2, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [11, 6, 2, 1, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [11, 6, 3, 1, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [12, 6, 3, 1, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [12, 7, 3, 1, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [12, 7, 3, 2, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [12, 7, 3, 2, 1, 0, 0, 0, 0, 0]
Tasks received by workers: [12, 7, 4, 2, 1, 0, 0, 0, 0, 0]
Tasks received by workers: [13, 7, 4, 2, 1, 0, 0, 0, 0, 0]
Tasks received by workers: [13, 8, 4, 2, 1, 0, 0, 0, 0, 0]
Tasks received by workers: [13, 8, 4, 3, 1, 0, 0, 0, 0, 0]
Tasks received by workers: [13, 8, 4, 3, 2, 0, 0, 0, 0, 0]
Tasks received by workers: [13, 8, 5, 3, 2, 0, 0, 0, 0, 0]
```

Abbildung 5: Agent Based Adaptive Balancing

```
Tasks received by workers: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
Tasks received by workers: [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
Tasks received by workers: [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
Tasks received by workers: [1, 1, 1, 1, 1, 1, 1, 0, 0, 0]
Tasks received by workers: [1, 1, 1, 1, 1, 1, 1, 1, 0, 0]
Tasks received by workers: [1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
Tasks received by workers: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Tasks received by workers: [2, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Tasks received by workers: [2, 2, 1, 1, 1, 1, 1, 1, 1, 1]
Tasks received by workers: [2, 2, 2, 1, 1, 1, 1, 1, 1, 1]
Tasks received by workers: [2, 2, 2, 2, 1, 1, 1, 1, 1, 1]
Tasks received by workers: [2, 2, 2, 2, 2, 1, 1, 1, 1, 1]
Tasks received by workers: [2, 2, 2, 2, 2, 2, 1, 1, 1, 1]
Tasks received by workers: [2, 2, 2, 2, 2, 2, 2, 1, 1, 1]
Tasks received by workers: [2, 2, 2, 2, 2, 2, 2, 2, 1, 1]
Tasks received by workers: [2, 2, 2, 2, 2, 2, 2, 2, 2, 1]
Tasks received by workers: [2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
Tasks received by workers: [3, 2, 2, 2, 2, 2, 2, 2, 2, 2]
Tasks received by workers: [3, 3, 2, 2, 2, 2, 2, 2, 2, 2]
Tasks received by workers: [3, 3, 3, 2, 2, 2, 2, 2, 2, 2]
Tasks received by workers: [3, 3, 3, 3, 2, 2, 2, 2, 2, 2]
Tasks received by workers: [3, 3, 3, 3, 3, 2, 2, 2, 2, 2]
Tasks received by workers: [3, 3, 3, 3, 3, 3, 2, 2, 2, 2]
Tasks received by workers: [3, 3, 3, 3, 3, 3, 3, 2, 2, 2]
Tasks received by workers: [3, 3, 3, 3, 3, 3, 3, 3, 2, 2]
Tasks received by workers: [3, 3, 3, 3, 3, 3, 3, 3, 3, 2]
Tasks received by workers: [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
Tasks received by workers: [4, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

Abbildung 6: Least Connection Balancing

2.9.2 Was kann als Gewichtung bei Weighted Round Robin verwendet werden?

Mit der Gewichtung geht es darum, welchem Server mehr oder weniger Tasks zugewiesen werden. Dabei geht es um Hardware-Spezifikationen (**Performance**), beispielsweise besitzt ein Server mit mehr RAM eine höhere Gewichtung als einer mit weniger Hauptspeicher. In den meisten Fällen wird geachtet auf:

- RAM
- CPU
- HDD oder SSD
- Übertragungsrate

2.9.3 Warum stellt die Hochverfügbarkeit von IT Systemen in der heutigen Zeit eine sehr wichtige Eigenschaft dar?

In einer Zeit wo Steuererklärungen und Banküberweisungen über das Internet ablaufen, müssen Webapplikationen immer zuverlässig verfügbar sein. Zusätzlich durch das immer mehr populär

werdende Online-Shopping werden sehr viele User-Anfragen an die Server gestellt, welche alle **zuverlässig** und **schnell** bearbeiten werden müssen.

2.9.4 Welche anderen Massnahmen neben einer Lastverteilung müssen getroffen werden, um die "Hochverfügbarkeit" sicher zu stellen?

Natürlich müssen genug Server zur Verfügung stehen, der beste Lastverteilungssystem ist unbrauchbar wenn es nichts gibt auf was die Last verteilt werden kann. Es muss auch die interne und die Verbindung mit dem Internet gewährleistet sondern, Kommunikation ist das wichtigste!

2.9.5 Was versteht man unter Session Persistenz und welche Schwierigkeiten ergeben sich damit?

Eine Lastverteilungsmethode bei welcher der Load Balancer die gesamte session information während der Applikationstransaktion speichern muss. Falls die Sitzung abbricht aus unerwarteten Gründen, kann es zu fatalen Datenverlusten kommen.

2.9.6 Nennen Sie jeweils ein Beispiel, wo Session Persistenz notwendig bzw. nicht notwendig ist.

Notwendig ist es bei Webanwendungen welche eine Sitzung aufbauen müssen. Es ist zum Beispiel auf **Amazon.de** nötig, ab dem Moment wo ein Artikel zum Warenkorb hinzugefügt oder man sich einloggt.

Nicht benötigt wird es bei Seiten welche lediglich eine Aufgabe erledigen und keine Sitzung aufbauen müssen, beispielsweise ein RGB-to-HEX Converter.

2.9.7 Welcher Unterschied besteht zwischen einer "server-side" bzw "client-side" Lastverteilungslösung?

Beim Client-seitigen Load Balancer ist lokal ein Register vorhanden, in welchem alle Services mit den jeweiligen Informationen bereitstehen. Bei einer Anfrage wird der URL des passenden Services abefragt.

Beim Server-seitigen Load Balancer ist eine zentrale Komponente vorhanden, welche alle Clients/Tasks zu den passenden Server/Abarbeitungskomponente verweist.

2.9.8 Was versteht man unter dem "Mega-Proxy-Problem"?

Die Mega Proxy Session, auch bekannt als Mega Proxy Problem, beschreibt jene Situation in welcher die IP des Users nicht korrekt herausgefunden werden kann. Dieses Problem tritt auf wenn die User hinter einem sogenannte Proxy-Server stehen, welcher die IP-Adresse des Users zu der des Proxy-Servers ändert. Grund für die Verwendung eines Proxy-Server ist der Wunsch von Anonymität bzw. Netz-Sicherheit.

Abbildungsverzeichnis