Ergebnisbericht

Verteilte Systeme - Übungsblatt 2

Maxim Hotz (1338419)

Dieser Ergebnisbericht dokumentiert die Implementierung und die dabei aufgetretenen Probleme/Fragestellungen der Aufgaben 2 und 3.

Da das zu implementierende verteilte System auf Grundlage des Simulationsframeworks „sim4da-v2“ entwickelt werden sollte, musste ich mich zunächst mit den grundlegenden Strukturen und Funktionsweisen des Frameworks vertraut machen. Test-Klassen wie der „TokenRingTest“ waren hierfür eine gute Hilfestellung. Im Rahmen der Implementierung wurde nichts an dem Code des Frameworks verändert.

Aufgabe 2

Zur Lösung von Aufgabe 2 wurde die Klasse „Aufgabe2“ entwickelt, welche einen Aktor als Erweiterung der Klasse org.oxoo2a.sim4da.Node beschreibt. Zusätzlich zu den in der Aufgabe gegebenen Anforderungen wurden bei der Implementierung folgende Annahmen getroffen:

1. Die initiale zufällige Wartezeit aller Knoten beträgt maximal 1 Minute
2. Die Wahrscheinlichkeit, bei der ersten Aktivierung durch eine Nachricht erneut Nachrichten an eine Teilmenge des Netzwerks zu versenden, beträgt 80%. Diese halbiert sich bei jeder weiteren Aktivierung.

Ausgehend von diesen Eigenschaften ist es klar, dass die Dynamik im Netzwerk nach einiger Zeit ausstirbt. Jedoch gibt es hier noch keine Möglichkeit festzustellen, wann das System terminiert und keine weiteren Nachrichten mehr unterwegs sind. Die Aktoren selbst haben keine Information über die Zustände der anderen Aktoren im System oder darüber, welche Nachrichten gesendet und/oder empfangen wurden. Daher resultiert die Ausführung von Aufgabe 2 in einer Endlosschleife der Aktoren, die nach einer gewissen Zeit alle im passiven Zustand verweilen und auf eine eingehende Nachricht warten.

Aufgabe 3

Aufgabe 3 adressiert dieses Problem durch die Implementierung einer nachrichtenbasierten Terminierungserkennung. Dazu existieren einige Möglichkeiten, die unterschiedlich gut geeignet sind, anhängig von der Topologie des Netzwerks als auch der Anzahl der Initiatoren. Nach Betrachtung einiger Algorithmen, wie beispielsweise: Schnappschuss-Algorithmen, Dijkstra-Scholten, Huang´s Algorithmus, oder Mattern´s Algorithmen, bin ich wiederum zurück zu dem in der Vorlesung behandelten naiven Ansatz des „Nachrichten-Zählens“ gekommen.

Grundannahme hierbei ist, dass alle Aktoren Statistiken über die Anzahl empfangener und gesendeter Nachrichten führen. In der Theorie sollte nun folgende Annahmen gelten: Wenn

1. Alle Aktoren im Netzwerk passiv sind und

2. Die Summe aller gesendeten Nachrichten der Summe aller empfangenen entspricht,

dann sollte das System terminiert sein. Dieser Mechanismus kann auf zwei Arten gelöst werden, abhängig davon, ob das System vollvermascht ist, oder über eine andere Topologie verfügt:

1. Der Observer adressiert alle Aktoren im Netzwerk direkt und fordert diese periodisch auf, ihm ihre jeweiligen Status und Nachrichtenstatistiken zukommen zu lassen (vollvermascht).

2. Der Observer initiiert eine „Wave“ an einen verbundenen Knoten, über den gemäß dem Flooding-Prinzip die Anfrage des Observers durch das Netz propagiert wird. Die somit erreichten Knoten senden ihre Statistiken an den jeweiligen Nachbarknoten, von dem sie die „weitergeleitete“ Anfrage erhalten haben. Auf diese Weise können die jeweiligen Statistiken durch das System zum Observer zurückpropagiert werden (nicht vollvermascht oder Topologie unbekannt).

In dieser konkreten Implementierung habe ich angenommen, dass das System vollvermascht ist, und der Observer somit jeden Aktor direkt ansprechen kann.

Allerdings kann nun auf Grund der physikalischen Begrenzung der Geschwindigkeit bei der Nachrichtenübermittlung der Fall eintreten, dass es zu dem „Bermuda Dreieck der verteilten Terminierung“ kommt. Um nicht die Latenzen der beteiligten Knoten künstlich manipulieren zu müssen, habe ich mich dazu entschieden, dieses Problem mit Hilfe des Doppelzählverfahrens zu lösen. Konkret wurde dieses Verfahren so implementiert, dass der Observer permanent die Nachrichtenstatistiken aller Aktoren im System anfragt. Dies geschieht auf Runden-Basis, was in diesem Fall bedeutet, dass erst alle Aktoren geantwortet haben müssen, bevor die nächste Runde beginnt. Wenn nun die aggregierten Statistiken aller Knoten als auch ihr Status (in beiden Runde alle passiv) in zwei konsekutiven Runden übereinstimmt, kann garantiert werden, dass das System terminiert ist. Dabei zählen die Kontrollnachrichten selbst nicht mit in die Nachrichtenstatistik der Knoten. Sobald der Oberserver feststellt, dass das System terminiert ist, benachrichtigt er mit einer weiteren Kontrollnachricht alle Aktoren, damit diese aus der Schleife in ihrer *engage()* – Methode herausspringen können.

Die Durchführung einiger Testläufe mit variierender Anzahl an Aktoren hat gezeigt, dass die Terminierung des Systems in jedem Fall unverzüglich erkannt wurde. Ein konkreter Nachteil dieses Implementierungsdesigns besteht allerdings darin, dass durch den permanenten Verkehr von Kontrollnachrichten eine Menge Overhead entsteht, der in der echten Welt möglicherweise einen negativen Effekt auf die Systemperformance hat. Dem könnte entgegengewirkt werden, indem z.B. der Observer die Anfrage in fixen periodischen Intervallen ausführt, insofern eine gewisse zeitliche Toleranz bei der Erkennung der Terminierung akzeptiert wird. Alternativ könnte der Observer auch eine komplett passive Rolle einnehmen, bei der die Aktoren eigenständig in gewissen Abständen Bericht an den Observer erstatten. So könnte der Nachrichtenverkehr deutlich reduziert werden.