Hochschule Bremerhaven
Prof. Dr. Thomas Umland

Bremerhaven, 15. Juni 2016

Programmierung paralleler/verteilter Systeme
Abschlussaufgabe

1 Vorbemerkungen

In dieser Abschlussaufgabe sollen Algorithmen implementiert werden, die häufig in verteilten Systemen angewendet werden.[1]

Unter einem „verteilten System“ sei hier eine Menge von Prozessen mit folgenden Eigenschaften verstanden:

- Das System besteht aus endlich vielen, voneinander unabhängigen, asynchron arbeitenden Prozessen.
- Jedem Prozess ist nur eine kleine Teilmenge aller Prozesse des Systems bekannt; diese werden als seine „Nachbarn“ bezeichnet.
- Jeder Prozess kann mit seinen Nachbarn kommunizieren, indem er Methoden von ihnen aufruft, ihnen dadurch also „Nachrichten“ zuschickt.
- Die Nachbarschaftsbeziehung der Prozesse lässt sich als ungerichteter Graph interpretieren. Dieser „Nachbarschaftsgraph“ muss nicht vollständig, zusammenhängend oder schlicht sein!
- Die Prozesse besitzen keinerlei gemeinsam nutzbare globale Variablen, sondern kommunizieren allein durch den Austausch von Nachrichten zwischen Nachbarn.
- Keiner der Prozesse besitzt eine globale Sicht auf das Gesamtsystem; insbesondere ist ihnen weder der komplette Nachbarschaftsgraph bekannt, noch wissen sie, wie viele Prozesse insgesamt im System vorhanden sind. Jeder Prozess kennt nur seine eigenen Nachbarn.
- Es gibt auch keine Instanz, die auf alle Prozesse zugreifen bzw. mit allen direkt kommunizieren kann.
- Die Prozesse besitzen zur Identifikation eine eindeutige Kennung.

2 Der Echo-Algorithmus

2.1 Das Problem

Um eine geschlossene Sicht auf den Gesamtumfang eines verteilten Systems zu ermitteln oder Botschaften an alle Knoten zu verteilen, stellt sich oft ein Problem der folgenden Art:

> Ein verteiltes System mit den in Abschnitt 1 beschriebenen Einschränkungen soll auf Veranlassung eines beliebigen Knotens (Initiator) nebenläufig durchlaufen werden, so dass jeder erreichbare Knoten des Systems besucht wird. Implizit wird dabei ein spannender Baum des Nachbarschaftsgraphen aufgebaut.[2] Wenn alle erreichbaren Knoten besucht wurden, wird der Initiator vom Abschluss des Verfahrens unterrichtet.

[1] Diese Aufgabenstellung ist eng an [UV92, Kapitel 5] angelehnt; die Bezeichnung der Algorithmen entstammt[Mat89].

[2] Genau genommen wird ein spannender Baum der Komponente des Nachbarschaftsgraphen aufgebaut, in welcher der Initiator enthalten ist!

2.2 Die Lösung

Um das geschilderte Problem etwas zu vereinfachen, soll nur ein Prozess zur Zeit das Verfahren als Initiator starten dürfen. Außerdem nehmen wir zunächst an, dass der Nachbarschaftsgraph zusammenhängend ist. Dann lässt sich das vereinfachte Problem mit folgendem Echo-Algorithmus lösen:

> Zunächst „schlafen“ alle Prozesse des Systems. Der Initiator „erwacht“ und weckt jeden seiner Nachbarn durch Übersenden einer Wakeup-Nachricht auf.
>
> Wird ein schlafender Knoten durch ein Wakeup aufgeweckt, so schickt er selbst nebenläufig Wakeup-Nachrichten an alle seine Nachbarn, mit Ausnahme des Knotens, der ihn selbst aufgeweckt hat. Erhält ein bereits aufgewachter Knoten von einem weiteren Nachbarn eine Wakeup-Nachricht, so ignoriert er diese. Die Wakeup-Nachrichten breiten sich so im System aus, bis schließlich alle erreichbaren Prozesse aufgewacht sind.
>
> Um den Initiator über den Abschluss des Verfahrens zu informieren, werden außerdem Echo-Nachrichten verschickt. Dazu zählt jeder Knoten, wie viele Nachrichten – Wakeup oder Echo – er bereits empfangen hat. Stimmt die Anzahl der empfangenen Nachrichten mit der Anzahl seiner Nachbarn überein, ist das Verfahren für den Knoten beendet. Handelt es sich bei dem Knoten um den Initiator, kann er sicher sein, dass alle Knoten besucht wurden. Handelt es sich nicht um den Initiator, so verschickt er vor dem Beenden des Algorithmus noch eine Echo-Nachricht an den Knoten, von dem er selbst zuvor aufgeweckt wurde.

Man mache sich folgende Punkte klar:

1. Knoten mit dem Grad 1 (d. h. Blätter des Nachbarschaftsgraphen) verschicken sofort nach Erhalt eines Wakeup ein Echo an ihren Nachbarn.
2. In jedem Prozess trifft über jede Kante genau eine Nachricht ein; das ist entweder eine Wakeup- oder eine Echo-Nachricht.

   Sei q die Anzahl der Kanten des Nachbarschaftsgraphen, dann werden im Verlauf des Algorithmus genau $2q$ Nachrichten verschickt.
3. Die Kanten, über die Echo-Nachrichten verschickt wurden, bilden einen spannenden Baum des Nachbarschaftsgraphen.

3 Election-Algorithmen

3.1 Die Problemstellung

In einem verteilten System mit den Einschränkungen aus Abschnitt 1 sollen sich gleichartige Prozesse ohne Hilfe von außen auf einen Repräsentanten einigen und damit eine sogenannte „Leader-Election“ durchführen.

Dabei soll gelten:

- Jeder Prozess kann jederzeit unabhängig von den anderen das Verfahren starten – d. h. hier sind *mehrere* gleichzeitig aktive Initiatoren erlaubt.
- Alle Prozesse sind gleichberechtigt.
- Nach Abschluss des Algorithmus wurde aus der Menge der Initiatoren ein eindeutiger Repräsentant (der sogenannte „Leader“) ausgewählt.
- Der Repräsentant weiß, dass er ausgewählt wurde und unterrichtet bei Bedarf die anderen Prozesse von seiner Wahl.

Ein Algorithmus, der das Election-Problem löst, lässt sich anschließend einfach zu einem Echo-Algorithmus modifizieren, bei dem die vorherige Einschränkung auf einen einzelnen Initiators entfällt.

Ohne Beschränkung der Allgemeinheit sei angenommen, dass die Identifikationen der Prozesse total geordnet sind. Damit kann die Election-Aufgabe in ein anderes Problem transformiert werden, das darin besteht, den Initiator mit der größten Identifikation zu finden, d. h. eine verteilte Maximumbestimmung unter den Initiatoren in einem beliebigen Netzwerk durchzuführen.

3.2 Eine erste Lösungsidee

Jeder Prozess, der als Initiator das Verfahren einleiten möchte, sendet eine Anfrage an seine Nachbarn. Diese verbreiten die Anfrage an ihre Nachbarn weiter und schicken außerdem eine Nachricht mit ihrer eigenen Identifikation an den Initiator zurück, wenn sie selbst auch ein Initiator sind, d. h. an der Leader-Election teilnehmen möchten. Jeder Initiator kann nun aus den ankommenden Nachrichten der anderen Initiatoren das Maximum ermitteln und so feststellen, ob er der Gewinner ist.

Diese Idee ist gut, hat aber einen Haken: Wie stellt ein Initiator fest, dass er von allen Knoten des Netzes eine Nachricht empfangen hat? Ihm ist die Gesamtanzahl der Prozesse unbekannt!

Dieser Ansatz führt also auf das Problem der verteilten Terminierung. Eine Lösungsidee dafür ist z. B. die folgende: Das Netz wird mit dem Echo-Algorithmus aus Abschnitt 2 inspiziert. Mit dem Echo schickt jeder Knoten die ihm bekannte maximale Identität an den Initiator zurück. Anschließend verteilt der Initiator das Maximum und der Gewinner kann sich identifizieren.

Auch diese Lösung funktioniert nicht, da aufgrund des verwendeten Echo-Algorithmus nur ein Initiator erlaubt ist und die Wakeup-Wellen sich „vermischen“ würden.

3.3 Das Echo/Election-Verfahren

Die eben in Abschnitt 3.2 vorgestellte Lösungsidee war eigentlich gut, sie muss aber noch für die Zulassung mehrerer Initiatoren erweitert werden.

Das ist z. B. dadurch zu erreichen, dass jeder Initiator mit der Wakeup-Nachricht seine eigene Identität verschickt. So breiten sich im Falle mehrerer Initiatoren verschiedene Wellen im Netzwerk aus. Treffen nun Wellen unterschiedlicher Initiatoren aufeinander, so setzt sich die „stärkere“ Welle durch, während die „schwächere“ aufgibt. Alle „schwachen“ Initiatoren werden im Verlauf des Algorithmus von einer „stärkeren“ Welle überlaufen, so dass am Ende alle Knoten von der insgesamt „stärksten“ Welle eingenommen werden und ihr Echo auch wirklich an den „stärksten“ Initiator gelangt.

3.4 Das Adoptionsverfahren

Das in Abschnitt 3.3 vorgestellte Echo/Election-Verfahren löst zwar die gestellte Aufgabe der verteilten Maximumbestimmung, es ist jedoch im Hinblick auf die Anzahl verschickter Nachrichten sicher nicht optimal. So kann es beispielsweise passieren, dass ein Knoten mehrfach ein Echo versenden muss, weil er im Verlauf des Algorithmus von immer „stärkeren“ Wellen überrollt wird.

Dieser Aufwand kann vermieden werden, wenn beim Aufeinandertreffen zweier Wakeup-Wellen das gesamte Gebiet des „schwächeren“ Initiators sofort der „stärkeren“ Welle zugeschlagen wird. Das lässt sich z. B. dadurch erreichen, dass nur die Wurzel des „schwächeren“ spannenden Baumes zum „stärkeren“ umgelenkt wird, die restlichen Knoten aber nicht erneut von einer Welle besucht, sondern sofort „adoptiert“ werden.

Hinweis: Das Adoptionsverfahren kann für mehr als zwei Initiatoren und beliebige Netzwerke sehr tückisch sein!

4 Aufgabenstellung

1. Implementieren Sie in Java den Echo-Algorithmus aus Abschnitt 2.2.

   Auf dem ELLI-Server sind die Dateien `Node.java` und `NodeAbstract.java` vorhanden, die Sie für die Implementierung Ihres Echo-Algorithmus verwenden sollten! `Node` beschreibt ein Interface, das jeder Prozessknoten implementiert, `NodeAbstract` ist eine abstrakte Klasse, die noch zur eigentlichen Knotenklasse erweitert werden muss. Wählen Sie eine sinnvolle `package`-Struktur für diese Klassen!

2. Benutzen Sie Ihren Echo-Algorithmus, um in einem verteilten System einen spannenden Baum zu bestimmen. Lassen Sie den ermittelten Baum in geeigneter Weise durch den Initiatorprozess ausgeben.

3. Formulieren Sie geeignete Sicherheits- und Lebendigkeitseigenschaften für Ihre Implementierung des Echo-Algorithmus und machen Sie plausibel, dass diese eingehalten werden.

4. Implementieren Sie in Java das Echo/Election-Verfahren aus Abschnitt 3.3.

   Den Echo/Election-Algorithmus sollen alle Prozesse immer wieder nach einer zufälligen Zeitspanne starten. Wurde ein Leader ermittelt, so soll dieser anschließend den Echo-Algorithmus ausführen und den entsprechenden spannenden Baum des Nachbarschaftsgraphen ausgeben.

5. Formulieren Sie auch für das Echo/Election-Verfahren geeignete Sicherheits- und Lebendigkeitseigenschaften und machen Sie plausibel, dass diese eingehalten werden.

Testen Sie Ihre Algorithmen mit verschiedenen Nachbarschaftsgraphen (Bäume, Kreise, vollständige Graphen, Graphen mit Schlingen, . . .)!

Die Programme für den Echo- und den Echo/Election-Algorithmus müssen durch Aufrufe über die Kommandozeile gestartet werden können. Werden dabei zusätzliche Aufrufparameter benötigt, muss beim Fehlen der Parameter ein „usage“-Hinweis ausgegeben werden!

Abgabe und Präsentation

Die Abgabe und Präsentation der Lösungen erfolgt am ??? im Raum Z2320 (ein Termin während der ersten Vorlesungswoche des Wintersemesters wird Mitte September über ELLI bekanntgegeben).

Arbeitsgruppen

Zur Bearbeitung der Aufgabenstellung sind 3-er-Gruppen erlaubt.

Viel Spaß beim Bearbeiten der Aufgabe!

Literatur

[Mat89] F. Mattern, *Verteilte Basisalgorithmen*, Springer-Verlag, Berlin, Heidelberg (1989).

[UV92] T. Umland, R. Vollmar, *Transputerpraktikum*, B. G. Teubner, Stuttgart (1992).