Verteilte Primzahlfaktorisierung im Aktormodell– Konzept

In der zweiten Aufgabe soll eine Anwendung entwickelt werden, welche eine Primfaktorzerlegung für große Zahlen durchführt. Da dies ein sehr großer Rechenaufwand ist, soll die Rechenlast aufgeteilt werden, wofür das Aktormodell verwendet wird. Mehrere Aktoren tauschen Nachrichten asynchron aus, um koordiniert und im „Wettbewerb“ das Problem zu lösen.
Die Faktorisierung wird mit Hilfe der Pollard Rho Methode durchgeführt. Dabei wird eine Folge von Pseudozufallszahlen erzeugt und die Perioden der Folge gesucht. Mit dem Floyd Zyklenfindungsalgorithmus lässt sich ein Teiler der zu behandelnden Zahl finden und dieser wird gegeben falls noch weiter behandelt, bis man einen Primfaktor erhält.

Aktormodell:

Aktoren sind nebenläufige Einheiten, welche ausschließlich über Nachrichtenaustausch kommunizieren. Da die Aktoren nicht auf den gleichen Speicherbereich zugreifen sind sie parallel ausführbar und kommunizieren asynchron miteinander. Dies bedeutet, dass ein Aktor nach dem Versenden einer Nachricht direkt weiterarbeiten kann und nicht darauf warten muss, bis der Empfänger die Nachricht empfängt. Außerdem kommen sich die Aktoren dadurch auch nicht in die Quere und „Race Conditions“ gibt es dementsprechend nicht. Daher kommt es nicht dazu, dass Aktoren, im Gegensatz zu Threads, aufeinander warten müssen, wenn sie auf gewisse Speicherbereiche zugreifen wollen oder dass ein Aktor das komplette System aufhält. Wenn ganz viele Aufgaben parallel erledigt werden müssen, wird es schwieriger Threads, welche auf den gleichen Speicherbereich zugreifen, zu koordinieren ohne dass diese sich „in die Quere kommen“, weshalb sich das Aktorenmodell mehr als die Nutzung von Threads eignet, da dessen Skalierbarkeit besser ist.
Die Aktoren sind nachrichtengetrieben, was bedeutet, dass sie beim Eintreten einer Nachricht reagieren. Dabei kann ein Aktor entweder eine Nachricht senden, neue Aktoren starten oder sein Verhalten ändern. Diese Nachrichten gehen in die Mailbox jedes Aktors, was der einzige Synchronisationspunkt ist.[1] [2]

Programmablauf:

Ein Client erhält die zu zerlegende Zahl ‘N‘ und delegiert diese in Form eines Tasks an die Worker. Jeder Worker führt dann den Rho Algorithmus aus, bei dem jeder Worker mit einer zufälligen Zahl ‘x‘, im Bereich 1 bis N, startet. Der Unterschied zwischen den Workers ist das Inkrement ‘a‘, wodurch die Worker nie den Algorithmus mit genau gleichen Variablen ausführen. Jeder Worker bekommt eine Iterationszahl, welche festlegt, wie lange dieser maximal den Algorithmus ausführt. Die Iterationszahl berechnet sich durch M/L. M ist die durchschnittliche Schrittdauer $M = 1{,}18 \sqrt{(\sqrt{N})}$ und L die Anzahl an Teile, in die die Aufgabe unterteilt ist. Bei jeder

Iteration prüft der Worker seine Mailbox, um gegebenfalls zu unterbrechen, wenn beispielsweise alle Primfaktoren gefunden wurden. Wenn ein Worker vor dem Erreichen der Iterationszahl einen Faktor findet, wird dies dem Client mitgeteilt, welcher daraufhin prüft, ob der gefundene Faktor ein Primfaktor ist oder das Produkt mehrerer Primfaktoren. Daher muss die gefundene Zahl gegebenfalls weiterbehandelt werden.
Wenn ein Primfaktor gefunden wurde, wird dieser gespeichert und jedes Mal, wenn ein neuer eingetragen wird, wird ‘N‘ so lange durch den gefundenen Primfaktor geteilt und geprüft, ob das Produkt die Zahl ‘N‘ ergibt und das neue ‘N‘ wird an die Worker geschickt. Dazu wird auch die aufgewendete CPU-Zeit dem Client mitgeteilt. Nebenbei protokolliert der Client die Summe der Rho Zyklendurchläufe und die verstrichene Zeit vom ersten Versenden bis zum Erhalt des letzten Faktors. Damit sichergestellt wird, dass gewisse Worker für einen Client arbeiten und nicht von anderen „weggenommen“ werden, bekommt jeder Client eine ID, welche mit der Aufgabe an den Worker gesendet werden. Die ID wird lokal im Worker gespeichert und wenn noch keine vorhanden ist, wird die erste Empfangene als lokale Client ID gesetzt und bei jedem Erhalten einer Nachricht wird geprüft, ob diese von dem zuständigen Client gesendet wurde und nur dann ausgeführt.

Fehlerbehandlung:

Der Ausfall eines Workers keinen Effekt auf andere Workers und den Client.
Es gibt zwei Varianten zur Erkennung von Fehlern: Linking und Monitoring.
Das Linking ist bidirektional und es werden zwei Aktoren lediglich miteinander verbunden und sie senden einander Signale. Wenn einer der Aktoren sich beendet wird automatisch ein exit-Signal gesendet. Im Standardfall führt dies dazu das der verbundene Aktor auch terminiert, außer dieses Signal wird gefangen. Dann kann der Aktor, der das exit-Signal erhalten hat, daraufhin anders agieren. Anders als beim Linking, wird beim Monitoring keine Verbindung hergestellt und es werden nur Nachrichten versandt, die den anderen Aktor nicht beeinflussen. Daher können Aktoren die „beobachtet“ werden sich beenden, ohne den beobachtenden Aktor direkt zu beeinflussen. Der Aktor sendet einfach eine „down“ Nachricht an den beobachtenden Aktor und dieser erhält diese Nachricht und entscheidet, wie er darauf reagiert, was das Monitoring unidirektional macht[3].
Da ein Client mehrere Worker hat und nicht direkt von ihnen abhängt bietet sich hierfür die Monitoring Strategie besser an, da das Linking auch nur eine Verbindung zwischen zwei Aktoren erlaubt. Der Client bekommt von seinen Workers eine „down“ Nachricht, wenn sie nicht mehr arbeiten, woraufhin der Client dann einen neuen Worker startet. Der Worker bekommt von den anderen Workern eine aktualisierte Liste, wenn ein Faktor gefunden wurde und kann sich somit der Berechnung anschließen. Da dies aber bei großen Zahlen lange dauern kann, muss der Worker schon vorher anfangen zu arbeiten. Deshalb wird er nach einer Zeit dazu aufgefordert nicht mehr zu warten, sondern anfangen zu arbeiten.

The Kiet Dang
Sofian Wüsthoff

Quellen:

1 – Verteilte Systeme: C++ Message Passing & Actor Programming mit CAF
http://inet.haw-hamburg.de/teaching/ws-2019-20/verteilte-systeme-1/copy_of_03_Programmieren_im_Aktormodell.pdf

2 – The actor model in 10 minutes
https://www.brianstorti.com/the-actor-model/

3- Link and Monitor differences in Erlang
https://marcelog.github.io/articles/erlang_link_vs_monitor_difference.html