Logische Uhren

Peter Maximilian Kain

12. April 2020

Inhaltsverzeichnis

Kapitel 1

Einleitung

Der folgende Text handelt von Logische Uhren. Dabei werden wir besprechen, wofür man logische Uhren überhaupt braucht, wie sie funktionieren und auch, wo sie beispielsweise verwendet werden können. Am Schluss folgt eine Dokumentation über eine Implementierung einer Lamport Clock. Eine Lamport Clock ist eine simple logische Uhr, benannt nach ihrem Erfinder Leslie Lamport. Dieser hat im Juli 1978 ein Paper über "Time, Clocks, and the Ordering of Events in a Distributed System"geschrieben. Logische Uhren sind also eine etwas ältere Idee.
Der Titel von Lamports Arbeit sagt schon aus, worum es geht. Wir müssen in einem verteilen System die Reihenfolge von Events wissen. Verteiltes System? Was ist das?

1.1 Verteilte Systeme

Ein verteiltes System ist laut Lamport definiert als: "Eine Sammlung von verschiedenen, räumlich getrennten Prozessen, welche miteinander kommunizieren", oder generell: "Ein System, in dem die Verzögerungen beim Übertragen von Nachrichten nicht verwerflich klein sind". Beispielsweise wäre das Internet ein verteiltes System. Nach Lamport ist aber auch ein Multiprozess System (heutzutage auch Multithreading System) zwar nicht als verteiltes System zu sehen, jedoch gibt es ähnliche Probleme wie bei verteilten Systemen, sodass viele Grundsätze auch auf Multiprozess Systeme zutreffen.

Kapitel 2

Möglichkeiten der Bestimmung der Reihenfolge von Events

Nun, da wir kennengelernt haben, was ein verteiltes System überhaupt ist, werden wir uns einige Möglichkeiten, die es zur zeitlichen Zuordnung von Events gibt, genauer anschauen. Um Events in eine Reihenfolge zu bringen, müssen wir sagen können, dass das Event a zum Beispiel vor dem Event b eingetreten ist. Lamport hat dafür die Schreibweise "a $\rightarrow$ b"definiert - also quasi "Auf a folgt (irgendwann) b". Diese Schreibweise gibt an, dass Event a vor Event b eingetroffen ist. Es gibt dafür ein paar Regeln:

1. Wenn a $\rightarrow$ b und b $\rightarrow$ c, dann gilt klarerweise a $\rightarrow$ c
2. Wenn zwei Events gleichzeitig eintreten, gilt: a $\nrightarrow$ b und b $\nrightarrow$ a. Also ist a weder vor b und b weder vor a eingetreten.
3. Lamport hat definiert, dass für jedes Event a gilt: a $\nrightarrow$ a. Sonst würde das bedeuten, dass ein Event eintreten kann, bevor es eintritt, was natürlich keinen Sinn macht.

2.1 Physische Uhren

Wie können wir nun bestimmen, dass ein Event a vor einem Event b eintritt? Als erstes denkt man natürlich an die Zeit, also an physische Uhren. Physische Uhren haben den Vorteil, dass sie schon existieren. Heutzutage ist es üblich, dass man, im Fall eines Computers, (meist) in die untere rechte Ecke des Monitors schaut, und man weiß, wie spät es ist. Trifft also Event a um 12:00:00 und Event b um 12:00:01 ein, könnte man sagen, dass Event a vor Event b eingetroffen ist. Das Problem dabei ist jedoch, dass die physischen Uhren in einem Verteilten System standardmäßig nicht synchronisiert werden. Wir sprechen hier von lokalen Zeiten, die pro Prozess unterschiedlich sein können. Natürlich gibt es Algorithmen, wie zum Beispiel der Algorithmus von Christian, der mithilfe eines Zeitservers, also einer "globalen" Zeit im verteilten System alle Prozesse synchronisiert, indem sie sich an die Zeit des Zeitservers richten. Auch gibt es den Berkeley Algorithmus, der die Zeit von Prozessen auch ohne Zeitserver synchronisieren kann, man allerdings die wirkliche physische Zeit verliert.

2.2 Logische Uhren

Logische Uhren bieten einen Weg, eine zeitliche Reihenfolge festzulegen, ohne eine physische Uhr zu verändern. Mit dem Berkeley Algorithmus hat man beispielsweise die gespeicherte Zeit verändert, da egal ist, wie spät es eigentlich ist, wenn man nur die zeitliche Reihenfolge überprüfen will. Es ist nur wichtig, dass alle Prozesse, die sich an eine zeitliche Reihenfolge halten sollen, die gleiche Zeit haben. Logische Uhren verfolgen genau diesen Ansatz.

2.3 Lamport Uhr

Im Fall einer Lamport Uhr wird dieser Ansatz dadurch erreicht, dass beim Senden und Empfangen von Nachrichten, sowie bei auftretenden Events, eine Zählervariable inkrementiert wird. Da ja sowohl beim Senden als auch beim Empfangen diese Variable inkrementiert wird, könnte man schon schlussfolgern, dass beim Empfangen einer Nachricht der eigene Zähler geupdated werden könnte. Das ist korrekt! Jeder Prozess hat zwar seinen eigenen Zähler, aber dieser wird von den Zählern der anderen Prozesse beeinflusst. Im Folgenden finden sich die Regeln einer Lamport Uhr:

1. Generell wird der Zähler bei jedem intern auftretenden Event (also im eigenen Prozess) um Eins erhöht.

2. Wird eine Nachricht an einen anderen Prozess gesendet, wird der Zähler um Eins erhöht und mitgesendet.

3. Wird eine Nachricht empfangen, vergleicht man die mitgesendete Zeit mit der eigenen. Der lokale Zähler wird dann der höhere Wert der beiden Zeitpunkte, wiederum erhöht um Eins. Allgemein kann man sagen:

   ```
   eigeneZeit = max(eigeneZeit, empfangeneZeit) + 1
   ```

Was haben wir nun davon? Da der Zähler nur erhöht und nicht vermindert wird, ist die Reihenfolge von Events logisch gesehen klar. Man weiß zwar nicht, wann genau sie passiert sind, aber man kann definitiv sagen, das ein Event, wo der Zähler zum Beispiel Drei war, vor einem Event aufgetreten ist, wo der Zähler zum Beispiel Sieben war.
Man kann also sagen, dass, wenn Clock(Event) einen Zeitpunkt zurückgeliefert hat, dass wenn:

```
Clock(a) < Clock(b)
```

dann gilt: a $\rightarrow$ b
Nicht ganz! Man kann nur sagen, dass a $\rightarrow$ b, wenn a ein Event ist, das eine Nachricht sendet und b ein Event ist, das diese Nachricht empfängt, oder a und b zwei Events des selben Prozesses sind. Warum?
Stellen wir uns folgende Situation vor: Es gibt zwei Prozesse: P und Q. Im Prozess P passieren irgendwann zwei Events, im Prozess Q eines. Das heißt, das zweite Event in P, P2, hat als Zähler 2 und das erste Event in Q, Q1, hat als Zähler 1. Aber Q1 könnte auch erst nach P2 passieren, da Q1 $\rightarrow$ P2 keine Vorraussetzung ist, ebensowenig wie P2 $\rightarrow$ Q1! Die Events sind unabhängig voneinander.

Wir können jedoch sicher sagen, dass P1 → P2, da P1 und P2 am selben Prozess passiert sind. Wenn P3 ein Event wäre, dass eine Nachricht an Q schickt und Q2 ein Event, welches diese empfängt, dann wäre P3 → Q2 auch sicher, da man eine Nachricht erst empfangen kann, wenn sie gesendet wurde.
Nun, da wir über die Theorie gehört haben, schauen wir uns doch an, wie sowas in der Praxis aussieht.

2.4 Lamport Uhren in C++

Das C++ Beispiel erwartet sich drei gestartete Prozesse. Es wird der Port angegeben, auf dem der Prozess listenen soll, sowie die zwei Ports, wo die zwei anderen Prozesse listenen. Ein Beispielaufruf wäre:

```
$ ./process -p 1234 -c 1235 1236
Listening. Press any key to connect...
```

Danach noch zwei Prozesse starten, die auf den Ports 1235 und 1236 listenen, es wird gewartet, bis der Nutzer dies getan hat. Danach kann man bei einem beliebigen (oder allen drei) Prozessen einfach Enter drücken, woraufhin Events simuliert werden. Pro Event wird die Lamport Zeit ausgegeben. Man kann die Ausgabe auch bei anderen Prozessen beobachten. Drücken wir mal bei einem Prozess Enter:
Prozess 1 (Enter gedrückt):

```
$ ./process -p 1234 -c 1235 1236
Listening. Press any key to connect...

[2020-04-12 15:26:50.674] [info] Yet another event... / Lamport Time: 1
[2020-04-12 15:26:50.674] [info] One event... / Lamport Time: 2
[2020-04-12 15:26:51.674] [info] Sent a message. / Lamport Time: 3
[2020-04-12 15:26:53.674] [info] Another event... / Lamport Time: 4
[2020-04-12 15:26:53.675] [info] Sent a message. / Lamport Time: 5
```

Prozess 2:

```
$ ./process -p 1235 -c 1234 1236
Listening. Press any key to connect...
[2020-04-12 15:26:53.675] [info] Received Message: Hello from 1234 / Lamport Time: 6
```

Prozess 3:

```
$ ./process -p 1236 -c 1234 1235
Listening. Press any key to connect...
[2020-04-12 15:26:51.674] [info] Received Message: Bye from 1234 / Lamport Time: 4
```

Die Ausgabe der Lamport Time ist dabei die eigene Lamport Time, nachdem das Event eingetreten ist. Wir können sicher sagen, dass "Sent a message" mit den Lamport Zeiten 3 und 5 bei Prozess 1 vor "Received Message" mit den Lamport Zeiten 6 (Prozess 2) und 4 (Prozess 3) passiert ist. Wir beobachten dabei ebenfalls, dass die Lamport Zeit von Prozess 2 und 3 anhand der gesendeten von Prozess 1 geupdated wurde. Ebenfalls erkennen wir die Reihenfolge der Events in Prozess 1 klar.

Würden wir jetzt in Prozess 2 und dann in Prozess 3 Enter drücken, würde sich das wiederholen - das wär doch fad. Versuchen wir also, dass Prozess 2 und Prozess 3 gleichzeitig Events haben! Wir drücken also so gleichzeitig es geht für Prozess 2 und Prozess 3 Enter. Auch hier sehen wir, dass die Lamport Zeit immer steigt. Zwar wahrscheinlich nicht in Einerschritten, wie es nacheinander der Fall wäre, aber darum geht es uns ja auch nicht.

2.5 Anwendungsfälle

Nach dem Lesen der Kapitel sind einem selber vielleicht schon der eine oder andere Anwendungsfall in den Sinn gekommen. Noch weitere Beispiele sind zum Beispiel: Jede Software, die Dinge koordiniert. Zum Beispiel: Es ist ein Platz bei einer Vorführung frei und wir haben zwei Anfragen bekommen. Wer war zuerst, also wer bekommt den Platz? Oder beispielsweise kann man bei Multiversion Concurrency Control einer Datenbank logische Uhren einsetzen, um jeder Version eine eindeutige Nummer zuzuweisen und bestimmen zu können, welche älter ist.