# Logische Uhren

Peter Maximilian Kain

10. 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 Ereignissen 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 Ereignissen

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 Ereignisse in eine Reihenfolge zu bringen, müssen wir sagen können, dass das Ereignis a zum Beispiel vor dem Ereignis b eingetreten ist. Lamport hat dafür die Schreibweise "a $\rightarrow$ b"definiert - also quasi "Auf a folgt (irgendwann) b". Diese Schreibweise gibt an, dass Ereignis a vor Ereignis 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 Ereignisse 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 Ereignis a um 12:00:00 und Ereignis b um 12:00:01 ein, könnte man sagen, dass Ereignis a vor Ereignis 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 "globalenSZeit 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. Im Fall des Lamport Algorithmus, der später noch genauer erklärt wird, wird die Synchronisierung von Prozessen dadurch erreicht, dass beim Senden und Empfangen von Nachrichten eine Zählervariable inkrementiert wird, welche zwischen den Prozessen synchronisiert wird.