

## Aufgabenblatt 8

### Musterlösung

## Vorlesung Verteilte Systeme

### Wintersemester 2025/26

#### Aufgabe 1: Uhrensynchronisation

- a) Der Client sollte die Anfrage mit der minimalen Round-Trip-Zeit von  $20ms$  wählen  $= 0.02s$ . Falls er seine Uhr unmittelbar nach Ankunft der Antwort setzen kann, würde er seine Uhr auf  $t_{Server} + T_{round}/2$ , also auf  $11:51:25.232 + 0.02/2 = 11:51:25.242$  Uhr setzen. Da der Client in der Regel aber erst später (nach einigen Anfragen) diejenige mit der minimalen Round-Trip-Zeit bestimmen kann, muß er wie folgt vorgehen: Angenommen, die lokale Zeit des Clients ist  $t$ . Er setzt dann die Uhr auf  $t + (t_{Server} + T_{round}/2 - t_{Ankunft}) = t + (t_{Server} + T_{round}/2 - (t_{Absenden} + T_{round})) = t + 25.242 - 24.734 = t + 0.508$ , d.h., die Uhr wird um  $508ms$  vorgestellt.
- Die Genauigkeit ist  $\pm 10ms$ .
- Wenn die minimale Übertragungszeit  $8ms$  beträgt, dann bleibt die Einstellung die selbe, aber die Genauigkeit verbessert sich auf  $\pm 2ms$ .
- b) Um die Uhr innerhalb von  $1ms$  zu synchronisieren, wenn eine Übertragungszeit von mindestens  $8ms$  gegeben sei, ist die maximale Round-Trip-Zeit  $T_{round} = 2 \cdot 1ms + 2 \cdot 8ms = 18ms$ .

#### Aufgabe 2: Happened-Before-Relation, Lamport- und Vektorzeit

- a) Aus der Tatsache, dass  $L(a_3) < L(c_2)$ , kann **nicht** gefolgert werden, dass  $a_3 \rightarrow c_2$  gilt. Aus  $L(a_3) < L(c_2)$  folgt aber, dass  $L(c_2) \not\prec L(a_3)$  und damit  $c_2 \not\rightarrow a_3$ , also war entweder  $a_3$  (im Sinne der Lamport'schen *happened before* Relation) vor  $c_2$  oder die beiden Ereignisse sind nebenläufig. Eine Aussage über die Reihenfolge in Realzeit ist nicht möglich.
- Aus  $L(a_1) = L(b_1)$  lässt sich folgern (da  $L(a_1) \not\prec L(b_1)$ ), dass  $a_1 \not\rightarrow b_1$ , also war entweder  $b_1$  vor  $a_1$  oder die beiden Ereignisse sind nebenläufig. Eine Aussage über die Reihenfolge in Realzeit ist nicht möglich.
- Aus  $L(b_2) < L(c_1)$  kann **nicht** gefolgert werden, dass  $b_2 \rightarrow c_1$  gilt. Da  $L(c_1) \not\prec L(b_2)$ , folgt  $c_1 \not\rightarrow b_2$ , also war entweder  $b_2$  vor  $c_1$  oder die beiden Ereignisse sind nebenläufig. Wieder ist keine Aussage über die Reihenfolge in Realzeit möglich.
- b)  $V(a_3) = (3, 1, 0) < (4, 1, 2) = V(c_2)$ , d.h.  $a_3 \rightarrow c_2$ .  $a_3$  muß demnach auch in realer Zeit vor  $c_2$  stattgefunden haben.
- $V(a_1) = (1, 0, 0) \not\prec (0, 1, 0) = V(b_1)$  und  $V(b_1) = (0, 1, 0) \not\prec (1, 0, 0) = V(a_1)$ , d.h.  $a_1 \not\rightarrow b_1$  und  $b_1 \not\rightarrow a_1$ , d.h.  $a_1$  und  $b_1$  sind nebenläufig. Ihre Reihenfolge in realer Zeit kann beliebig sein.
- $V(c_1) = (0, 0, 1) < (0, 2, 1) = V(b_2)$ , d.h.  $c_1 \rightarrow b_2$ .  $c_1$  muß demnach auch in realer Zeit vor  $b_2$  stattgefunden haben.
- c) Für die Vektorzeit gilt:  $V(a) < V(b) \Leftrightarrow a \rightarrow b$ .
- Für die Lamport-Zeit gilt nur  $a \rightarrow b \Rightarrow L(a) < L(b)$ , aber nicht umgekehrt. Die Implikation  $a \rightarrow b \Rightarrow L(a) < L(b)$  ist aber gleichbedeutend mit  $L(a) \not\prec L(b) \Rightarrow a \not\rightarrow b$ , so dass doch gewisse Rückschlüsse auf die *happened before* Relation möglich sind (siehe Teilaufgabe a).

## Aufgabe 4: Schnappschuß-Algorithmus nach Chandy/Lampport

1.  $P$  sendet  $m$ .
2.  $P$  initiiert den Schnappschuß, speichert den Zustand 101, und sendet einen Marker an  $Q$ .
3.  $Q$  empfängt  $m$ , sein Zustand ist damit 102.
4.  $Q$  empfängt den Marker und speichert seinen Zustand 102. Damit hat  $Q$  bereits alle Marker empfangen, der aufgekennzeichnete Zustand des Nachrichten-Kanals von  $P$  nach  $Q$  ist damit leer.
5.  $Q$  sendet einen Marker an  $P$
6.  $Q$  sendet  $m$  wieder zurück an  $P$
7.  $P$  empfängt den Marker
8.  $P$  speichert den den Zustand des Nachrichten-Kanals von  $Q$  nach  $P$ . Dieser ist leer, da  $P$  vor dem Marker keine Nachricht empfangen hat.

Ermittelter Zustand:  $P : 101, Q : 102$ , beide Kanäle sind leer.

Anmerkung:  $Q$  könnte Schritt 6 auch unmittelbar nach Schritt 3 ausführen. In diesem Fall wäre der Zustand:  $P : 101, Q : 102$ , Kanal  $Q \mapsto P$  enthält Nachricht  $m$ .

## Aufgabe 5: Bully-Algorithmus

- a) Alle höher nummerierten Prozesse erhalten zwei ELECTION-Nachrichten. Sie beantworten auch beide mit OK, so dass beide Initiatoren die Wahl abbrechen. Wenn ein Prozess eine ELECTION-Nachricht erhält, aber bereits eine Wahl abhält, so ignoriert er diese (redundante) Nachricht. Die höher nummerierten Prozesse ignorieren also jeweils die zweite ELECTION-Nachricht und führen Ihre Wahl wie üblich durch.
- b) Zunächst ist zu beachten, dass dies ein unerwünschtes Verhalten ist, wenn es keinen direkten Vorteil bei der Verwendung eines höher nummerierten Prozesses gibt, denn eine Wiederwahl ist meistens verschwenderisch. Jedoch kann die Nummerierung von Prozessen ihren relativen Nutzen widerspiegeln (beispielsweise höher nummerierte Prozesse, die auf schnelleren Maschinen ausgeführt werden). In diesem Fall kann der Nutzen, der durch eine Wiederwahl entsteht, die Kosten dieser wert sein. Die Kosten für eine Wiederwahl umfassen auch die erneute Kommunikation, also die Runden, in denen die ELECTION-Nachrichten versendet werden, sowie die Übermittlung von Statusnachrichten von Koordinator zu Koordinator.

Um eine Wiederwahl zu vermeiden, könnte ein sich erholender Prozess lediglich eine requestStatus-Nachricht an die nächsten aufeinanderfolgenden unteren nummerierten Prozesse senden. Damit könnte er herausfinden, ob ein anderer Prozess bereits gewählt wurde und wählt sich selber nur dann, wenn er eine negative Antwort erhält. Danach kann der Algorithmus wie zuvor arbeiten: Wenn der gerade erholte Prozess entdeckt, dass der Koordinator versagt oder er eine ELECTION-Nachricht empfängt, sendet er eine Koordinator-Nachricht an die verbleibenden Prozesse.

- c) Mit denen in der Frage genannten Voraussetzungen können wir nicht garantieren, dass jederzeit ein eindeutiger Prozess gewählt werden kann.

Stattdessen können wir es als ausreichend akzeptieren, Untergruppen von Prozessen zu bilden, die ihrem Koordinator als solchem zustimmen und zu erlauben, dass mehrere solcher Untergruppen auf einmal existieren.

Wenn zum Beispiel ein Netzwerk in zwei geteilt wird, könnten wir zwei grösstmögliche Untergruppen bilden, von denen jede den Prozess mit der höchsten Kennung unter seinen Mitgliedern zum Koordinator wählt. Allerdings sollte, für den Fall, dass die Partitionen wieder zusammengeführt werden, sichergestellt werden, dass aus den beiden Gruppen eine Einzige mit nur einem Koordinator entsteht.

Der sog. Invitation-Algorithmus von Hector Garcia-Molina erreicht dies. Er wählt einen einzigen Koordinator innerhalb einer Untergruppe deren Mitglieder miteinander kommunizieren können. In regelmässigen Abständen fragt der Koordinator andere Mitglieder der gesamten Reihe von Prozessen an, um zu versuchen sich mit anderen Gruppen zu verschmelzen. Wenn ein anderer Koordinator gefunden wird, sendet ein Koordinator eine Einladungsnachricht, um ihn dazu einzuladen, eine zusammengeführte Gruppe zu bilden. Wenn ein Prozess eine Nichterreichbarkeit oder einen Misserfolg seines Koordinators vermutet, führt er, wie im Bully-Algorithmus, eine Wahl durch.