Verteilte Systeme SS 2020
Übungsblatt 1

Aaron Winziers - 1176638

22. Mai 2020

Aufgabe 1

Der Quellcode zu Aufgabe 1 befindet sich in der folgenden Repository: `https://github.com/AaronWinziers/Uni/tree/master/verteilteSysteme/%C3%BCbung01/Lamport`

Aufgabe 2

Ein erster und leicht lösbarer Fehlerfall ist das Verwenden von identischen Timestamps in Requests von verschiedenen Knoten. Vor allem bei einer hohen Anzahl von Knoten mit einer hohen Frequenz von Schreibvorgängen im kritischen Abschnitt, steigt die Wahrscheinlichkeit dass dieser Fall eintritt. Dieses Problem ist jedoch leicht zu umgehen, in dem einem Knoten Priorität über den anderen gegeben wird. Dies kann entweder durch vorab zugeteilte Prioritätswerte oder lediglich durch einem Vergleich der IDs der Knoten geschehen. Alternativ wäre es möglich beim Auftreten von diesem Fall die Vorgänge abzubrechen und die anfragende Knoten den Vorgang neu anfangen zu lassen.

Zwei weitere Fehlerfälle wären Nachrichtenverlüste und Knotenausfälle. Diese müssen im ersten Moment gleich behandelt werden, da es aus der Perspektive des sendenden Knotens nicht direkt klar ist welcher Fall eingetreten ist. In den folgenden Lösungsansätze wird erst davon ausgegangen, dass eine Nachricht lediglich verloren wurde, und nicht dass ein Knoten ausgefallen ist.

Die Verlüste der Request und Acknowledge Nachrichten können durch ein Timeout gelöst werden. Ein Knoten der ein Request schickt muss notieren von welchen Knoten er schon Antworten bekommen hat. Wenn es einen oder mehrere Knoten gibt die bis zum Timeout noch nicht geantwortet haben, muss der Request erneut an diese geschickt werden.

Der Verlust einer Release Nachricht ist etwas komplexer. Angenommen der nächste Knoten in der Warteschlange die Nachricht tatsächlich bekommen hat, ist dieses Problem

zunächst weniger schwierig zu lösen. Sollte ein Knoten eine Release Nachricht bekommen von einem weiteren Knoten der an einer Stelle in der Warteschlange die nicht an der Spitze ist steht, müsste der empfangende Knoten alle Einträge bis zu dieser Stelle Löschen unter der Annahme, dass alle davor stehende Requests schon abgearbeitet wurden und diese Nachrichten verloren gegangen sind.

Ein größeres Problem tritt jedoch auf, wenn der Knoten der als nächstes in der Warteschlange steht die Nachricht nicht bekommt. Hier wäre es zwar möglich ein Ack auf Release Nachrichten zu verlangen, jedoch würde diese Lösung erstens die Nachrichtenkomplexität des Algorithmus deutlich erhöhen und zweitens in das Two Generals Problem führen.

Ein besserer Ansatz wäre es, einen weiteren Timeout zu verwenden. Falls ein Knoten an der vorletzten Stelle der Warteschlange angekommen ist, beginnt der Timeout. Wenn dieser abgelaufen ist, sendet er eine Nachricht an den Knoten der sich zur Zeit im kritischen Abschnitt befinden sollte, ob er noch im Abschnitt ist. Hierbei müssten zusätzliche Nachrichtentypen definiert werden um Fehlkommunikationen auszuschließen (zum Beispiel beim Verlust von anderen Nachrichtenypen wie).

Sollten die bisher gennannten Timeouts mehrmals ohne Antwort abgelaufen sein, muss davon ausgegangen werden, dass dieser nicht-antwortender Knoten ausgefallen ist. Ein möglicher Ansatz wäre es, dem Knoten der an der ersten Stelle der Warteschlange steht, es zu erlauben, nach einer bestimmten Anzahl von Timeouts, seinen Schreibvorgang im kritischen Abschnitt durchzuführen, ohne alle notwendigen Acks bekommen zu haben. Falls der ausgefallene Knoten bereits an der ersten Stelle steht, beziehungsweise im kritischen Abschnitt sein sollte, bekommt der Knoten an zweiter Stelle das Erlaubnis seinen Schreibvorgang durchzuführen. Im Anschluss sollten dann, wie oben in der Lösung zu verlorenen Release Nachrichten erwähnt, eine entsprechende Menge an Einträgen in der Warteschlange gelöscht werden.

Dieser Ansatz ist natürlich nicht perfekt. Es besteht keine Garantie, dass die Knoten tatsächlich ausgefallen sind, sondern wird durch mehrere Kommunikationsversuche diese Wahrscheinlichkeit erhöht. Daher könnten weiterhin Probleme im kritischen Abschnitt entstehen (vor allem wenn der ausgefallene Knoten an der ersten Stelle in der Warteschlange steht). Zudem werden alle Prozesse dadurch deutlich verlangsamt, indem jeder Schreibvorgang mehrere Timeouts abwarten muss um in den kritischen Abschnitt kommen zu dürfen, zumindest bis der ausgefallene Knoten wieder zum laufen gebracht wird.

Eine bessere Lösung wäre es die Anzahl der benötigten Acks zu reduzieren bis der verlorene Knoten wieder online ist. Eine solche Lösung wäre aber mit einem deutlich höheren algorithmischen Aufwand verbunden.