Abschlussbericht Verteilte Systeme - Übung 3 - 1540051

30. September 2024

1 Aufgabenstellung

In dieser Übung sollen wir uns in ein Agreement-Protokoll unserer Wahl (vorgeschlagen wurden Raft oder Paxos) einarbeiten und dieses anschließend in einer Beispielanwendung umsetzen. Dabei ist es uns freigestellt, ob wir für diese Beispielanwendung den Simulator sim4da nutzen oder uns für eine Programmiersprache unserer Wahl entscheiden.

Ich habe mich für das Agreement-Protokoll Raft entschieden und werde dieses mit dem Simulator sim4da umsetzen.

2 Raft - Agreement Protokoll

2.1 Konsensprotokolle

In verteilten Systemen ist es von besonderer Bedeutung, dass mehrere Knoten (Computer oder Server), die an einem gemeinsamen Prozess beteiligt sind, einen übereinstimmenden Zustand erreichen. Diese Knoten können Computer oder Server sein. Besonders in Umgebungen, in denen die Konsistenz von Daten entscheidend ist, wie bei verteilten Datenbanken, Blockchain-Netzwerken oder Cloud-Anwendungen, ist dieser übereinstimmende Zutsand und die gleiche Datenbasis notwendig. Konsensprotokolle sind darauf ausgelegt, dieses Problem zu lösen, indem sie sicherstellen, dass alle Knoten, selbst bei Ausfällen mancher Knoten, auf denselben Zustand oder dieselben Daten zugreifen.

Ein Konsensprotokoll wie Raft ermöglicht es, dass sich alle Knoten eines verteilten Systems bei Entscheidungen einigen können. Diese Fähigkeit zur Fehlertoleranz und Konsistenz ist besonders wichtig in Szenarien, bei denen hohe Verfügbarkeit und Datenintegrität erforderlich sind, wie beispielsweise bei Finanzsystemen, großen Cloud-Umgebungen oder bei Blockchain-Anwendungen. Die zunehmende Verbreitung von verteilten Systemen macht Konsensprotokolle zu einem zentralen Bestandteil moderner IT-Infrastrukturen.

2.2 Funktionsweise Raft

Raft ist ein weit verbreitetes Konsensprotokoll und soll die Verwaltung eines verteilten Systemes einfacher und verständlicher machen. Angewendet wird es oft bei verteilten Daten-

banken, da es zudem eine zuverlässige Synchronisation zwischen mehreren Knoten in einem Netzwerk sicherstellen möchte, sodass alle Knoten in diesem Netzwerk die gleiche Ansicht der Daten beibehalten können.
Um die verher genannten Punkte zu erreichen, legt Raft einen Fokus auf die sog. Leader-basierte Konsenzfindung. Dabei werden den teilnehmenden Knoten verschiedenen Rollen zugeteilt: *Leader*, *Follower* oder *Candidate*.

Zu Beginn sind alle Knoten in der Rolle Follower. Diese Rolle empfängt vom *Leader* einen *Herzschlag*. Sobald ein *Follower*-Knoten keinen Herzschlag mehr empfängt, startet er eine Wahl, welcher Knoten der nächste *Leader* sein soll. Durch die dynmaische Neuwahl eines Anführers wird das System vor Ausfällen geschützt, solange weitere Knoten vorhanden sind, die die Funktionen eines *Leaders* übernehmen können.

Der Knoten, der die Wahl gestartet hat, stellt sich auch als *Candidate* vor. Nur, wenn ein Knoten mehr als die Hälfte der Stimmen aller sich im Netzwerk befindenden Knoten bekommen hat, kann er der neue *Leader* werden. Diese Einschränkung verhindert eine Split-Brain-Situation, bei der zwei Knoten als *Leader* agieren.

Sobald ein Knoten zum neuen *Leader* gewählt wurde, hat dieser besondere Aufgaben. Dazu gehören der bereits erwähnte Herzschlag, das Synchronisieren und Vereinheitlichen der Log-Dateien und die Koordination bei Entscheidungen. Er muss dafür sorgen, dass jeder Knoten die gleichen Log-Einträge gespeichert hat und dass neue Log-Einträge an alle Knoten verteilt werden. Nur so kann gewährleistet werden, dass ein beliebiger Knoten die Rolle des *Leaders* übernehmen kann.

Die Vorteile des Raft-Algorithmus sind vorallem die hohe Verfügbarkeit und die Skalierbarkeit.
Die hohe Verfügbarkeit ist durch die Log-Replikation und die Neuwahlen des *Leaders* auch gewährleistet, wenn ein Knoten ausfällt. Die Skalierbarkeit beschreibt hierbei die leichte Anbindung weiterer Knoten ohne komplexe Neukonfigurationen.

2.3 Raft vs. Paxos

Im Grunde sind Raft und Paxos zwei Konsensprotokolle und haben das gleiche Ziel: beide wollen in verteilten Systemen bewirken, dass die Knoten einen gemeinsamen Zustand behalten, auch wenn ein Knoten ausfallen sollte. Dabei arbeiten beide mit Mehrheitsabstimmungen und Replikationen von Log-Dateien.

Allerdings gibt es auch einige Unterschiede. Paxos ist beispielsweise komplex und schwieriger zu implementieren, während Raft leicht verständlich und implementierbar ist. Auch die übersichtlicheren Strukturen und die klare Dokumentation sorgen dafür, dass Raft bei Entwicklern beliebter ist. Dafür ist Paxos für seine Robustheit bekannt und geschätzt.
Auch bei dem Thema *Leader* gibt es Unterschiede: Während Raft einen klar definierten Leader besitzt, kann bei Paxos jeder Knoten einen Vorschlag einbringen und direkt mit den anderen Knoten kommunizieren.

Die Mehrheit der Unterschiede lässt sich durch die Tatsache erklären, dass Raft entwickelt wurde, um die Komplexitätsbarrieren von Paxos zu durchbrechen.

Welches Konsensprotokoll genommen werden sollte, hängt von der Art der Anwendung ab, bei der ein Konsensprotokoll benötigt wird. Raft wird beispielsweise häufig für verteilte Key-Value-Datenbanken, wie beispielsweise etcd oder TiKV, genutzt, während Paxos bei größeren und evtl. globalen Projekten, wie Google Spanner, zum Einsatz kommt, bei denen eine höhere Robustheit erforderlich ist.

3 Umsetzung

Die Beispielanwendung habe ich mit dem Simulator sim4da umgesetzt. Um die Simulation zu starten, muss man die Anzahl der zu simulierenden Knoten angeben. Jeder dieser Knoten wird nun initialisiert und direkt in eine Liste geschrieben, sodass immer eine Liste existiert mit den aktuell aktiven Knoten. Für jeden Knoten wird auch zu Beginn eine Liste angelegt, in der sich alle Knoten, außer diesem Knoten selber, befinden. Diese Liste wird verwendet, wenn Nachrichten gesendet werden sollen, wie beispielsweise bei den *Leader*-Wahlen oder dem Herzschlag.

Bei der Initialisierung eines Knotens wird auch direkt die Verbindung zum Logger der NetworkConnection gezogen, der in der Oberklasse *Node* bereits initialisiert wurde. Mit diesem Logger können die Ankündigungen der neuen *Candidates* und *Leader* übersichtlich in der Log-Datei des Simulators eingefügt werden.

Die Klasse *Knoten* erbt von der Klasse *Node*. Zusätzlich exisiteren noch einige Methoden, die ein Knoten in dem Raft-Konsensprotokoll haben muss. Dazu gehört zunächst eine Methode, um die Herzschläge zu senden. Diese Methode wird ausgeführt, sobald ein Knoten zum *Leader* gewählt wurde. Nur ein aktiver Knoten im Status *Leader* kann die Methode ausführen. Um einen Ausfall eines *Leaders* zu simulieren, wurde eine Zufallszahl eingefügt, sodass ein Knoten nur zwischen 5 und 35 Herzschläge senden kann. Danach wird er auf inaktiv gesetzt und damit komplett aus der Simulation genommen, als wäre dieser Knoten tatsächlich ausgefallen. Die Nachrichten, die der *Leader* beim Herzschlag sendet, sind vom Typ „Heartbeat“ und übergeben auch den Term, der die Wahlperiode darstellt. Der Herzschlag wird jede Sekunde gesendet.

Sobald ein Knoten keinen Herzschlag mehr feststellen kann und er nicht der letzte aktive Knoten in dem Netzwerk ist, startet er eine Wahl und wird selbst zum *Candidate*. Er sendet Nachrichten an alle anderen aktiven Knoten und fragt sie, welchem *Candidate* sie ihre Stimme geben wollen. Diese Nachrichten sind vom Typ „ID“ und übergeben den Term und die ID des Knotens, der die Wahl gestartet hat. Der *Candidate* wartet, bis genügend Stimmen eingetroffen sind. Dabei überprüft er alle 100ms, ob ausreichend Stimmen eingegangen sind. Da auch ein *Follower*-Knoten ausfallen könnte, wird nicht darauf gewartet, dass alle anderen Knoten geantwortet haben. Sobald ein Knoten mehr als die Hälfte der Stimmen von allen aktiven Knoten hat, wird dieser zum neuen *Leader*. Falls es neben dem Knoten, der

die Wahl starten möchte, keinen weiteren Knoten mehr gibt, wird die Simulation beendet, was im realen Fall einen Ausfall des gesamten Systems zur Folge hätte. Deswegen würden im realen Fall die Knoten repariert oder ersetzt werden, damit es nicht zu einem kompletten Ausfall des Systems kommt.

Der Leader hat die besondere Aufgabe die Log-Einträge der einzelnen Knoten zu sammeln und an alle aktiven Knoten zu verteilen, sodass alle Knoten die gleichen Log-Einträge erhalten und ein beliebiger Knoten die Position des *Leaders* jederzeit übernehmen kann. Der *Leader* klärt Konflikte zwischen den einzelnen Log-Einträgen der Knoten auf und sorgt dafür, dass wirklich jeder Knoten die aktuellen Einträge erhält. Die Nachrichten, mit denen der *Leader* die Einträge verteilt, sind vom Typ „AppendEntries“ und übergeben den aktuellen Term, den LogIndex und den Text des Log-Eintrages. Der LogIndex ist relevant, damit ein Knoten mitbekommt, wenn eine Log-Nachtricht des *Leaders* ihn nicht erreicht hat oder er die Log-Nachricht bereits eingetragen hat.

Wenn ein Knoten eine Nachricht erhalten hat, schaut zu zunächst, welchen Typ die Nachricht hat, um dann die entsprechenden Aktionen auszuführen.

4 Fazit

Raft hat sich als geeignetes Protokoll herausgestellt, da es eine verständliche Struktur bietet, die sich einfach implementieren ließ und leicht zu erlernen war. Es konnte gezeigt werden, wie Raft durch die Leader-basierte Konsensfindung für eine hohe Verfügbarkeit sorgt. Die durchgeführte Simulation zeigte, dass Raft auch unter schwierigeren Bedingungen, wie dem Ausfall von Knoten, zuverlässig arbeitet und sicherstellt, dass alle Knoten im Netzwerk zu einem einheitlichen Zustand kommen.

Insbesondere ist Raft für Anwendungen geeignet, bei denen hohe Verfügbarkeit und Konsistenz der Daten von großer Bedeutung sind, wie beispielsweise bei verteilten Datenbanken oder verteilten Speichersystemen. Die klare Rollenverteilung zwischen *Leader*, *Follower* und *Candidate* vereinfacht dabei das Verständnis der Zustandsänderungen und des Ablaufs der Konsensfindung innerhalb des Systems.

Die Ergebnisse der Simulation bestätigen die Leistungsfähigkeit des Raft-Protokolls in typischen Anwendungsszenarien.