Verteilte Systeme - Übung 3

Marlin Zapp

28. September 2024

In diesem Bericht wird beschrieben, wie das Agreement-Protokolls Paxos für verteilte Systeme implementiert wurde.

Inhaltsverzeichnis

1 Paxos

Für diese Implementierung wurde das Agreement-Protokoll Paxos gewählt. Das Protokoll sorgt dafür, dass Vorschläge aus einem verteilten System entweder akzeptiert oder abgelehnt werden.

Ein Knoten, der als Proposer agiert, sendet seinen Vorschlag an die Mehrheit der anderen Knoten, um eine Einigung zu erzielen. Die Vorschläge werden durchnummeriert.

Ein Vorschlag wird in zwei Phasen (Prepare, Accept) akzeptiert. Sobald der Vorbereitungs-Vorschlag (Prepare) bei einem Knoten, der somit als Acceptor fungiert, ankommt, wird dieser versprechen, keinen älteren Vorschlag (mit einer niedrigeren Nummer) mehr zu akzeptieren. Das Versprechen bedeutet, dass der Vorschlag des Proposers nur nicht akzeptiert wird, falls vor der Accept-Anfrage dieses Vorschlags eine neuere Prepare-Nachricht eines anderen Proposers ankommt. Im anderen normalen Fall wird die Accept-Anfrage akzeptiert.

Wenn die Mehrheit der Acceptors (also im Normalfall aller anderen Knoten) den Vorschlag akzeptiert hat, ist eine Einigung gefunden worden. Es können nicht mehrere Vorschläge angenommen werden, da keine zwei Mehrheiten existieren können und jeder Knoten nur einen Vorschlag annehmen kann.

Sobald sich auf einen Vorschlag geeinigt wurde, gibt es eine Teilmenge der Knoten, die als Learner agieren und den Wert des akzeptierten Vorschlags an alle Knoten senden. Dadurch dass auch hier mehrere Knoten diese Aufgabe übernehmen, ist das Protokoll auch in der Learning-Phase nicht anfällig für Ausfälle einzelner Knoten.

2 Implementierung

Für die Implementierung des Paxos-Protokoll wurde zunächst ein Anwendungsbeispiel aus dem Bereich der verteilten Systemen konstruiert. Es soll eine stark vereinfachte Simulation der Fortbewegung eines Fischschwarms darstellen.

In der Simulation kann ein Fisch sich entscheiden, eine andere Richtung einzuschlagen, zum Beispiel, weil in der Richtung Futter wartet oder weil in der anderen Richtung ein Fressfeind auftaucht. Um es einfach zu halten können die Fische in der Simulation nur in vier Richtungen schwimmen (siehe Abbildung 1).

Sobald sich ein Fisch entschieden hat, in eine neue Richtung schwimmen zu wollen, unterbreitet er den Vorschlag den anderen Fischen. Mithilfe des in Abschnitt 1 beschriebenen Paxos-Protokolls wird dieser Vorschlag ggf. gewählt. Falls dem so ist, dann schwimmen alle Fische in die gewählte Richtung, bis ein neuer Vorschlag unterbreitet wird.

2.1 Technisches

Als Grundlage für die Simulation haben die Programmiersprache Java und der gegebene Simulator für verteilte Systeme gedient.

Instruktionen zum Starten der Anwendung sind nicht nur hier, sondern auch in der Datei README.md des Projekt-Repositories zu finden. Der Simulator benötigt Java 21 zum Laufen. Mit den zwei Instruktionen aus Abbildung 2 kann die Simulation gestartet werden.

2.2 Struktur

2.2.1 Simulation

Die Hauptklasse mit der main-Methode findet sich in der Datei Simulation.java. Hier werden die Anzahl der Fische des Schwarms, die Gefahren- und Nahrungswahrscheinlichkeiten festgelegt. Man könnte diese Attribute noch als Laufzeit-Parameter auslagern, aber das würde die Ausführung unnötig verkomplizieren. Passen Sie also gerne die folgenden Standard-Parameter nach Ihrem Belieben an:

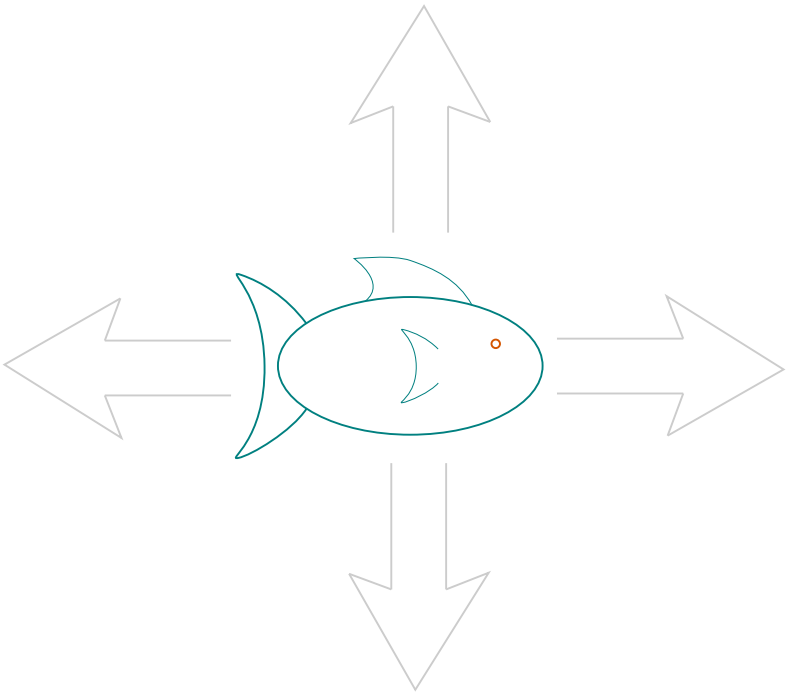

Abbildung 1: Das gewählte Beispiel für die Implementierung handelt von einer vereinfachten Richtungsentscheidung von Fischen.

Abbildung 2: Befehle zum Starten der Simulation. Ausführung im Projekt-Root-Verzeichnis mit Java 21.

```
javac -cp lib/sim4da-v2.jar -d bin $(find src -name "*.java")
java -cp bin:lib/sim4da-v2.jar paxos.Simulation
```

```
// After this time, the simulation will shutdown
public static int SIMULATION_TIME_MILLIS = 1000;
// What the swarm size will be
public static int NUMBER_OF_FISHES = 20;

// a fish will swim away from this direction
public static double DANGER_CHANCE = 0.01;
// a fish will swim in this direction
public static double FOOD_OCCURENCE = 0.05;
// each fish wants a new decision after a random amount of
// milliseconds between these two values
public static int NEW_DECISION_TIMER_MILLIS_MIN = 0;
public static int NEW_DECISION_TIMER_MILLIS_MAX = 30;
```

Abgesehen vom Starten des Simulators wird in der Simulation-Klasse noch der Fischschwarm initialisiert.

```
Swarm swarm = new Swarm(NUMBER_OF_FISHES);
```

2.2.2 Swarm

In der Swarm-Klasse werden die Fische mit Namen instanziiert. Ein Zehntel plus einem der Fische werden als Learner deklariert, die am Ende des Protokolls das Ergebnis der Entscheidung an alle Fische weitergeben.

Des Weiteren beinhaltet die Klasse eine Zählvariable für die IDs der Vorschläge (siehe Abschnitt 1). Auf diese wird von allen Instanzen des verteilten Systems, also von allen Fischen, zugegriffen.

```
public AtomicInteger proposal = new AtomicInteger(0);
```

2.2.3 Fish

Die Fish-Klasse ist diejenige, die die Netzwerkknoten des verteilten Systems darstellt. Daher erbt sie auch von der Node-Klasse des Simulators.

Alle Fische spielen sowohl zuweilen die Rolle des Proposers, als auch dauerhaft die des Acceptors. Um die Verhaltensweisen deutlich zu trennen, wurden sie in zwei private Unterklassen, die von Threads erben, ausgelagert.

1. Die Thread-Klasse für das Acceptor-Verhalten heißt Communication. Hier wird in einer Dauerschleife auf eingehende Nachrichten anderer Fische gewartet und bei Eintreffen werden die Informationen extrahiert und entsprechende Funktionen aufgerufen. Auf die verschiedenen Nachrichtentypen wird in Unterunterabschnitt 2.2.4 eingegangen.

2. Die Thread-Klasse für das Proposer-Verhalten heißt Behaviour. Hier wird bestimmt, dass ein Fisch normalerweise einfach schwimmt. Nach einer zufälligen Zeit „guckt sich der Fisch um“und entdeckt zu einer bestimmten Wahrscheinlichkeit einen Fressfeind oder Essen. Daraufhin schlägt er vor, das Schwimmverhalten des Schwarms entsprechend anzupassen.

2.2.4 MessageType

Um das Paxos-Protokoll umzusetzen, müssen die Fische verschiedene Nachrichten verschicken können. Entsprechend der Phasen des Protokolls (siehe Abschnitt 1) wurde das folgende enum mit Nachrichtentypen implementiert.

```
public enum MessageType {
        PREPARE_REQUEST,
        PREPARE_ANSWER,
        ACCEPT_REQUEST,
        ACCEPT_ACK,
        LEARN
}
```

2.2.5 Learner

Auf dieselbe Weise, wie in der Fish-Klasse (siehe Unterunterabschnitt 2.2.3) das Proposer- und Acceptor-Verhalten mittels Unterklassen aufgetrennt worden ist, hätte man auch das Learner-Verhalten implementieren können. Dies hätte jedoch die sowieso schon große Klasse sehr unübersichtlich gemacht. Die in dem Fall bessere Lösung war es, eine Klasse Learner zu implementieren, die von der Klasse Fish erbt.

Das gesamte Acceptor- und Proposer-Verhalten wird somit übernommen, aber zusätzlich werden Nachrichten vom Typ ACCEPT_ACK verarbeitet und Nachrichten vom Typ LEARN versendet. Die Klasse erweitert also nur die Methode handleMessage und implementiert die neue Methode:

```
private void handleAcceptAcknowledgement(Proposal proposal)
```

3 Ergebnis

Die Entscheidungsfindung lässt sich mithilfe der Konsolennachrichten nachvollziehen. Alternativ leiten Sie den Konsolen-Output gerne in eine Datei um, damit die vielen Nachrichten übersichtlicher werden.

Die Nachricht, mit der ein Entscheidungsprozess anfängt, kann wie folgt aussehen:

```
Corkelbelvinor wants to swim LEFT (proposal number is 0).
```

Falls bevor eine Entscheidung gefallen ist, weitere Entscheidungsprozesse gestartet werden, wird es interessant:

```
Falanos wants to swim BACKWARD (proposal number is 1).
Bemidorkellanis wants to swim RIGHT (proposal number is 2).
Giis wants to swim RIGHT (proposal number is 3).
```

In diesem Moment sind viele Nachrichten vom Typ PREPARE_REQUEST unterwegs. Sobald eine dieser Nachrichten angekommen ist, antwortet der Fisch,

der nun als Acceptor fungiert, mit einer PREPARE_ANSWER. In der Konsole erscheint dann eine Nachricht der folgenden Form:

```
Inel promises Giis to accept no proposals with numbers lower than 3.
```

Die Proposer erhalten eine Menge dieser Nachrichten. Sobald sie Nachrichten von mehr als der Hälfte der Fische erhalten haben, können sie einen ACCEPT_REQUEST versenden, der einer Konsolen-Nachricht dieser Form begleitet wird:

```
Bemidorkellanis sends accept request for swimming RIGHT.
```

Sobald ein Fisch eine solche Nachricht erhält, schickt er allen Learnern einen ACCEPT_ACK und hat somit den Vorschlag akzeptiert und wird seine Meinung auch nicht zurücknehmen, bis die Entscheidung gefallen ist:

```
Faus accepted proposal 8.
```

Falls ein Fisch danach nochmals einen PREPARE_REQUEST erhält, wird er dem Proposer mit seinem akzeptierten Vorschlag antworten. Daraufhin übernimmt der Proposer den Wert des höchstnummerierten akzeptierten Vorschlags, von dem er erfährt. In der Konsole erscheint also ggf. eine Nachricht der folgenden Art:

```
Habelmoius may support proposal 8 in swimming LEFT.
```

Sobald ein Learner von der Mehrheit der Fische einen ACCEPT_ACK erhalten hat, teilt er allen Fischen mit, dass die Entscheidung gefallen ist. Sobald diese LEARN-Nachricht bei einem Fisch angekommen ist, erscheint in der Konsole eine Nachricht der folgenden Art:

```
Decision 0: Faia is now swimming LEFT.
```