4 Übungsaufgabe #4: Replikation

Die Umsetzung der Fernaufrufsemantiken Last-Of-Many und At-Most-Once in Übungsaufgabe 3 ermöglichen es, dem in der Übung realisierten Fernaufrufsystem bisher vor allem durch das Netzwerk bedingte Fehler, wie zum Beispiel die Verzögerung oder den Verlust von Nachrichten, zu tolerieren. Beide Semantiken sind jedoch nicht dazu geeignet mit schwerwiegenderen Fehlersituationen, wie beispielsweise Rechnerabstürzen oder dauerhaften Verbindungsabbrüchen, umzugehen. Im Rahmen dieser Übungsaufgabe werden unter Zuhilfenahme von Replikation nun auch diese Fehlerbereiche abgedeckt. Dem Nutzer des Fernaufrufsystems soll dabei so weit wie möglich verborgen bleiben, dass er auf einen entfernten, replizierten Dienst zugreift (→ Replikationstransparenz).

4.1 Replikation der Server-Seite (für alle)

Die Replikation eines Diensts beinhaltet in erster Linie die Bereitstellung mehrerer Instanzen der Dienstimplementierung (Replikate) auf verschiedenen Rechnern. Wie in der Tafelübung erläutert, ist es für zustandsbehaftete Anwendungen darüber hinaus erforderlich, die Konsistenz der Replikatzustände sicherzustellen. Im Rahmen dieser Übungsaufgabe soll hierfür das Konzept der *aktiven Replikation* umgesetzt werden, das vorsieht, dass alle Replikate alle Anfragen in derselben Reihenfolge bearbeiten. Zur Herstellung einer über die Replikatgrenzen hinweg einheitlichen Anfragenreihenfolge kommt dabei die Gruppenkommunikation *JGroups* zum Einsatz.

Die Implementierung eines Replikats soll in einer Klasse `VSReplicaServer` (analog zu `VSServer`) erfolgen, deren Aufgabe es ist, Verbindungen und Anfragen entgegenzunehmen sowie die Antworten bearbeiteter Anfragen zur Client-Seite zurückzusenden. Im Unterschied zu `VSServer` darf `VSReplicaServer` eine Anfrage jedoch erst ausführen, wenn sie von der Gruppenkommunikation als nächste Anfrage bestimmt wurde. Des Weiteren ist darauf zu achten, dass nur das Replikat eine Antwortnachricht sendet, das auch über eine Verbindung zum Client verfügt; alle anderen Replikate führen die Anfrage nur aus, um ihren internen Zustand zu aktualisieren.

Aufgaben:

→ Implementierung der Klasse `VSReplicaServer`

→ Testen der „Schwarzes Brett“-Anwendung mit drei Replikaten auf verschiedenen Rechnern

Hinweise:

- Bei der Implementierung von `VSReplicaServer` soll so weit wie möglich auf die von `VSServer` bereits angebotene Funktionalität zurückgegriffen werden. Dies lässt sich zum Beispiel erreichen, indem `VSReplicaServer` als Unterklasse von `VSServer` realisiert wird.
- Die einzusetzende JGroups-Bibliothek ist im Pub-Verzeichnis unter `/proj/i4vs/pub/aufgabe4` bereitgestellt. Für die JGroups-Initialisierung ist die in der Tafelübung vorgestellte Konfiguration zu verwenden; diese nutzt UDP und erwartet, dass sich alle Replikate im selben Subnetz (z. B. alle im CIP-Pool) befinden.
- Es darf angenommen werden, dass die Adressen, unter denen die einzelnen Replikate Fernaufrufe entgegen nehmen, für alle Replikate konstant und auf Server-Seite (jedoch nicht den Clients!) vorab bekannt sind.
- Um Konflikte mit den Implementierungen anderer Übungsgruppen zu vermeiden, ist als JGroups-Gruppenname der Name der eigenen Übungsgruppe (`gruppe<Nummer>`) zu verwenden.
- Für den Fall, dass sich ein Client per `listen()` am schwarzen Brett registriert hat, erfolgt im replizierten Fall pro Replikat ein eigener Rückruf. Um auch in dieser Situation die Replikationstransparenz zu garantieren, wäre in einem realen System dafür zu sorgen, dass nur ein einzelnes Replikat den Rückruf tatsächlich ausführt und die Ergebnisse dieses Aufrufs gegebenenfalls an die anderen Replikate weitergibt. Die Realisierung eines solchen Mechanismus ist nicht Bestandteil dieser Übungsaufgabe.
- Im Optimalfall funktionieren die RPC-Semantiken aus Übungsaufgabe 3 weiterhin. Sollte dies nicht gelingen, darf statt der `VSBuggyConnection` wieder auf die fehlerfreie `VSObjectConnection` zurückgegriffen werden.

4.2 Ausfallsicherung der Client-Seite (für alle)

Nachdem in der vorherigen Teilaufgabe die Server-Seite repliziert zur Verfügung gestellt wurde, ist nun dafür zu sorgen, dass die Client-Seite im Fehlerfall auch auf verschiedene Replikate zugreifen kann. Hierzu müssen die einzelnen Replikate dem Stub auf Client-Seite bekannt gemacht werden. Dies erfolgt mittels einer Klasse `VSRemoteGroupReference`, die eine *Gruppenreferenz* bereitstellt und die bisherige Remote-Referenz ersetzt:

```
public class VSRemoteGroupReference {
    private VSRemoteReference[] references;
}
```

Ein Fernaufruf soll, solange keine Fehler auftreten, auch im replizierten Fall weiterhin über jeweils eine einzelne Verbindung abgewickelt werden. An welches der in der Gruppenreferenz enthaltenen Replikate ein Stub hierzu die Anfrage sendet, ist freigestellt.

Ist das gewählte Replikat jedoch nicht erreichbar bzw. bricht eine bereits bestehende Verbindung ab, bevor eine Antwortnachricht empfangen wurde, muss der Stub ein anderes Replikat kontaktieren und die Anfrage erneut senden („Failover“). Für den Fall, dass die gesamte Replikatgruppe unerreichbar ist, soll der Stub den Fernaufruf abbrechen und dem Aufrufer dies mittels einer `RemoteException` signalisieren.

Aufgaben:

→ Anpassung der Klasse `VSRemoteObjectManager`, so dass beim Exportieren von Objekten in den dynamischen Proxies Gruppenreferenzen statt einfacher Referenzen zum Einsatz kommen
→ Erweiterung der Klasse `VSInvocationHandler` um die Nutzung von Gruppenreferenzen
→ Testen der Implementierung durch manuelles Beenden einzelner bzw. mehrerer Replikate

Hinweis:

- Auf Client-Seite darf davon ausgegangen werden, dass sich die in der Gruppenreferenz enthaltenen Remote-Referenzen zu keiner Zeit ändern. Sollte ein Replikat nach einem Fehler neu gestartet werden (siehe Teilaufgabe 4.3), so ist es wieder unter derselben Adresse wie vor seinem Ausfall zu erreichen.

4.3 Neustart nach Replikatausfall (optional für 5,0 ECTS)

Der interne Zustand der Testanwendung „Schwarzes Brett“ (→ Botschaften, registrierte Clients) wird in der aktuellen Implementierung mittels `VSBoardImpl` vollständig im Hauptspeicher verwaltet. Dies hat zur Folge, dass durch den Absturz eines Replikats sämtliche dieser Daten verloren gehen. Um den Datenverlust zu kompensieren, wird in dieser Teilaufgabe dafür gesorgt, dass sich ein nach einem Ausfall neu gestartetes Replikat beim Wiedereintritt in die Replikatgruppe den aktuellen Anwendungszustand von einem der anderen Replikate holt. Die Umsetzung dieses Mechanismus erfolgt dabei mittels der von JGroups angebotenen Unterstützung von Zustandstransfers, die nicht nur für die Übermittlung der Daten sorgt, sondern auch sicherstellt, dass der Zustand konsistent ins neue Replikat übernommen wird.

Das Auslesen bzw. Setzen des Zustands ist über eine einheitliche Schnittstelle `VSRemoteObjectStateHandler` abzuwickeln, von der angenommen werden kann, dass jedes auf Server-Seite im `VSRemoteObjectManager` exportierte Remote-Objekt (im konkreten Fall also `VSBoardImpl`) sie implementiert:

```
public interface VSRemoteObjectStateHandler {
    public byte[] getState();
    public void setState(byte[] state);
}

public class VSBoardImpl implements VSBoard, VSRemoteObjectStateHandler {
    [...]
}
```

Ein Aufruf von `getState()` liefert den kompletten Anwendungszustand des Remote-Objekts zum Zeitpunkt des Aufrufs in serialisierter Form zurück. Mit Hilfe der Methode `setState()` lässt sich der Anwendungszustand auf Grundlage dieser Daten einem Remote-Objekt desselben Typs auf einem anderen Replikat zuweisen.

```
public class VSRemoteObjectManager {
    public byte[] getRemoteObjectStates();
    public void setRemoteObjectStates(byte[] states);
}
```

Analog zu `getState()` und `setState()` soll die Klasse `VSRemoteObjectManager` für die Zustandsübertragung zwei Methoden `getRemoteObjectStates()` und `setRemoteObjectStates()` zur Verfügung stellen, die es ermöglichen, die Zustände aller exportierten Remote-Objekte gebündelt auszulesen bzw. zu setzen.

Aufgaben:

→ Erweiterung der Klasse `VSBoardImpl` zum Auslesen und Setzen des Objektzustands
→ Erweiterung der Klasse `VSRemoteObjectManager` um den Zustandstransfer für mehrere Remote-Objekte
→ Testen der Implementierung durch manuelles Beenden und Neustarten einzelner bzw. mehrerer Replikate

Hinweise:

- Es darf angenommen werden, dass zu jeder Zeit mindestens ein Replikat aktiv ist und korrekt funktioniert.
- Der zu sichernde Zustand eines `VSBoardImpl`-Objekts umfasst nicht nur alle seit Anwendungsstart verfassten Botschaften, sondern auch alle am schwarzen Brett registrierten `VSBoardListener`.

4.4 Effizienzsteigerung durch deterministische parallele Ausführung (für alle)

Die gegenwärtige Implementierung des Fernaufrufsystems garantiert die Konsistenz von Replikatzuständen auf die denkbar einfachste Art: Jedes Replikat führt alle Anfragen sequentiell und in derselben Reihenfolge wie alle anderen Replikate aus. Ein eklatanter Nachteil dieses Ansatzes besteht darin, dass aufgrund der erzwungenen sequentiellen Ausführung der in der bisherigen Implementierung durch eine parallele Bearbeitung von Anfragen erzielte Effizienzgewinn verloren geht. Insbesondere für aufwendige Anfragen mit langer Bearbeitungszeit (z. B. aufgrund von häufigen oder lang andauernden Ein-/Ausgabeoperationen) ist somit ein deutlich geringerer Durchsatz im Vergleich zum nicht replizierten System aus Übungsaufgabe 3 zu erwarten.

Um die Lücke zwischen effizienten (aber nicht fehlertoleranten) und fehlertoleranten (aber ineffizienten) Systemen zu schließen, wurde in den letzten Jahren die Forschung auf diesem Gebiet intensiviert, mit dem Ziel, parallele und deterministische Ausführung in Einklang zu bringen. Einer der in diesem Zuge vorgeschlagenen Ansätze zur Lösung des Problems stellt die DTHREADS-Bibliothek dar:

> Tongping Liu, Charlie Curtsinger, and Emery D. Berger
> Dthreads: Efficient Deterministic Multithreading
> *Proceedings of the 23rd Symposium on Operating Systems Principles (SOSP '11)*, pages 327-336, 2011.

Im Rahmen dieser Teilaufgabe soll das DTHREADS-Papier sorgfältig gelesen und anschließend ein Review dazu verfasst werden. Wie in der Tafelübung näher erläutert, ist das Review dabei wie folgt zu strukturieren:

- *Gesamturteil*: Eine Gesamtnote (*Strong Reject*, *Reject*, *Weak Reject*, *Weak Accept*, *Accept*, oder *Strong Accept*), die die Empfehlung des Review-Autors widerspiegelt, das Papier anzunehmen bzw. abzulehnen.
- *Zusammenfassung*: Eine kurze (höchstens 5 Sätze), objektive Übersicht über den Inhalt und die wichtigsten Punkte des Papiers, die erkennen lässt, dass der Review-Autor das Papier verstanden hat.
- *Kurzbegründung*: Eine stichpunktartige Auflistung der Stärken und Schwächen des Papiers, die bei der Festlegung der Gesamtnote eine Rolle gespielt haben.
- *Detaillierte Kommentare*: Nähere Erläuterungen zu den in der Kurzbegründung aufgeführten Stichpunkten mit dem Ziel, die Empfehlung zur Akzeptanz bzw. Ablehnung des Papiers nachvollziehbar zu begründen. Darüber hinaus: Hinweise auf eventuelle handwerkliche Fehler (z. B. Rechtschreibfehler) sowie, soweit vorhanden, Präsentation von Verbesserungsvorschlägen.

Aufgabe:

→ Verfassen eines Review zum DTHREADS-Papier (5,0 ECTS: $\geq$ 400 Wörter; 7,5 ECTS: $\geq$ 600 Wörter)

Hinweise:

- Eine Kopie des DTHREADS-Papiers befindet sich im Pub-Verzeichnis unter `/proj/i4vs/pub/aufgabe4`.
- Das Review kann in deutscher oder englischer Sprache verfasst werden.

Abgabe: Teilaufgaben 4.1 bis 4.3 am Mi., 17.6.2015 in der Rechnerübung, Teilaufgabe 4.4 bis spätestens Mo., 22.6.2015 per E-Mail

Die für die Teilaufgaben 4.1, 4.2 und 4.3 erstellten Klassen sind in einem Subpackage `vsue.replica` zu bündeln. Der für Teilaufgabe 4.4 zu verfassende Text ist bis zum Abgabetermin unter Angabe des Gruppennamens per E-Mail an `vs@i4.informatik.uni-erlangen.de` zu senden.