5 Übungsaufgabe #5: Replikation

Im Rahmen dieser Übungsaufgabe soll ein aktiv replizierter Dienst zur Speicherung von Schlüssel-Wert-Paaren realisiert und getestet werden. Als erster Schritt dient hierfür in Teilaufgabe 5.1 zunächst die Umsetzung als Client-Server-Anwendung. Die eigentliche Replikation der Server-Seite erfolgt anschließend in Teilaufgabe 5.2.

5.1 Implementierung des Diensts (für alle)

Zugriffe auf den Dienst erfolgen über eine Klasse `VSKeyValueClient`, die hierfür folgende Methoden bereitstellt:

```
public class VSKeyValueClient {
    public void put(String key, String value) throws RemoteException;
    public String get(String key) throws VSKeyValueException, RemoteException;
    public void delete(String key) throws RemoteException;
    public long exists(String key) throws RemoteException;
}
```

Mittels `put()` lässt sich unter dem Schlüssel `key` die Zeichenkette `value` als Wert speichern; ein eventuell bereits existierender Datensatz mit demselben Schlüssel wird dabei ersetzt. Nachdem ein Wert im Dienst hinterlegt wurde, kann er anschließend über die Methode `get()` wieder gelesen werden. Ein Aufruf von `get()` für einen nicht existierenden Schlüssel führt zu einer `VSKeyValueException`. Per `delete()` wird ein Datensatz gelöscht. Die Methode `exists()` ermöglicht es einem Nutzer darüber hinaus zu prüfen, ob für einen Schlüssel `key` ein Datensatz vorliegt und wann dieser hinzugefügt wurde: Existiert ein solcher Datensatz, gibt `exists()` den Zeitstempel der Erstellung bzw. letzten Aktualisierung zurück, anderenfalls `-1`. Scheitert die Interaktion mit dem Dienst, wird dies dem Nutzer bei sämtlichen Methoden durch Werfen einer `RemoteException` signalisiert.

```
public interface VSKeyValueRequestHandler extends Remote {
    public void handleRequest(VSKeyValueRequest request) throws RemoteException;
}

public interface VSKeyValueReplyHandler extends Remote {
    public void handleReply(VSKeyValueReply reply) throws RemoteException;
}
```

Die Kommunikation zwischen Client und Dienst soll über anwendungsspezifische Anfrage- (`VSKeyValueRequest`) und Antwortnachrichten (`VSKeyValueReply`) abgewickelt werden. Der Austausch dieser Nachrichten erfolgt hierbei über RMI-Fernaufrufe, vergleichbar der Zustellung von Ereignisbenachrichtigungen im Kontext des Auktionsdiensts in den vorherigen Übungsaufgaben: Um von der Anwendung Antworten empfangen zu können, stellt der Client eine per Fernaufruf zugreifbare Methode `handleReply()` bereit. Zum Senden von Anfragen nutzt der Client per Fernaufruf die Methode `handleRequest()` auf Server-Seite, die im Rahmen dieser Teilaufgabe zunächst nur ein einzelnes `VSKeyValueReplica` umfassen soll. Den zum Austausch von Anfragen erforderlichen Stub erhält der Client über eine replikateigene Registry, deren Adresse (in späteren Teilaufgaben: Adressen) dem Client beim Start über eine Konfigurationsdatei bekanntgegeben wird (siehe Tafelübung).

Aufgaben:

→ Implementierung der Nachrichtenklassen `VSKeyValueRequest` und `VSKeyValueReply`
→ Implementierung der Klassen `VSKeyValueClient` und `VSKeyValueReplica`

Hinweise:

- Die unter `/proj/i4vs/pub/aufgabe5` bereitgestellten Klassen dürfen bei Bedarf beliebig erweitert werden.
- Die von `exists()` zurückgegebenen Zeitstempel müssen auf allen Replikaten konsistent sein.

5.2 Replikation des Diensts (für alle)

Zur Sicherstellung konsistenter Replikatzustände soll für den implementierten Dienst das Konzept der *aktiven Replikation* umgesetzt werden, das vorsieht, dass alle Replikate alle Anfragen in derselben Reihenfolge bearbeiten. Hierfür ist die bestehende Implementierung der Klasse `VSKeyValueReplica` so anzupassen, dass sie zur Herstellung einer über die Replikatgrenzen hinweg einheitlichen Anfragenreihenfolge auf die Gruppenkommunikation *JGroups* zurückgreift. Um starke Konsistenz gewährleisten zu können, darf ein Replikat, das von einem Client eine Anfrage erhalten hat, diese also nicht wie bisher unmittelbar ausführen, sondern muss damit warten, bis sie von der Gruppenkommunikation als nächste Anfrage bestimmt wurde.

Auf Client-Seite läuft die Interaktion mit dem replizierten Dienst wie folgt ab: Jeder Client bestimmt aus der Replikatgruppe ein *Kontaktreplikat*, an das er seine Anfragen schickt; zur besseren Lastverteilung sollen unterschiedliche Clients nach Möglichkeit verschiedene Kontaktreplikate nutzen. Da jede Ausführung einer Anfrage auf einem Replikat zu einer eigenen Antwort führt, erhält der Client im Normalfall mehrere Antworten auf dieselbe Anfrage. Die erste bei ihm eintreffende dieser Antworten soll der Client zur Bestimmung des Ergebnisses nutzen, alle weiteren ignorieren. In dem Fall, dass nach dem Senden der Anfrage jegliche Antworten ausbleiben, muss der Client sein Kontaktreplikat wechseln und die Anfrage erneut senden. Scheitern derartige Versuche bei sämtlichen Replikaten der Gruppe, bricht der Client die Operation ab und signalisiert dem Aufrufer die Fehlersituation per `RemoteException`.

Aufgaben:

→ Aktive Replikation der Server-Seite unter Zuhilfenahme von JGroups
→ Erweiterung der Klasse `VSKeyValueClient` um die Unterstützung von Zugriffen auf eine Replikatgruppe
→ Testen der Implementierung mit drei Replikaten auf verschiedenen Rechnern

Hinweise:

- Die einzusetzende JGroups-Bibliothek ist im Pub-Verzeichnis unter `/proj/i4vs/pub/aufgabe5` bereitgestellt. Für die JGroups-Initialisierung ist die in der Tafelübung vorgestellte Konfiguration zu verwenden; diese erwartet, dass sich alle Replikate einer Gruppe im selben Subnetz (z. B. alle im CIP-Pool) befinden.
- Um Konflikte mit den Implementierungen anderer Übungsgruppen zu vermeiden, ist als JGroups-Gruppenname der Name der eigenen Übungsgruppe (`gruppe<Nummer>`) zu verwenden.
- Ein Client muss sicherstellen, dass verspätet eintreffende Antworten auf vorherige Aufrufe nicht fälschlicherweise als Ergebnis nachfolgender Aufrufe interpretiert werden.

5.3 Verifizierung von Ergebnissen (optional für 5,0 ECTS)

Abhängig vom Fehlermodell ist der in Teilaufgabe 5.2 realisierte Ansatz, auf Client-Seite stets die schnellste Antwort eines Replikats zur Ermittlung des Ergebnisses zu nutzen, nicht immer sinnvoll. Muss beispielsweise davon ausgegangen werden, dass fehlerhafte Replikate mitunter auch falsche Antworten liefern können, so besteht bei dieser Vorgehensweise etwa die Gefahr, dem Aufrufer falsche Resultate zurückzugeben.

```
public class VSKeyValueClient {
    public long reliableExists(String key, int threshold) throws RemoteException;
}
```

Die Ausführung einer Anfrage auf mehreren Replikaten bietet in aktiv replizierten Systemen Clients die Möglichkeit, dieses Problem durch einen Vergleich der Antworten verschiedener Replikate zu lösen. Exemplarisch soll diese Strategie durch eine Methode `reliableExists()` realisiert werden, bei der ein Nutzer über einen Parameter `threshold` angeben kann, wie viele Replikate dasselbe Ergebnis geliefert haben müssen, bevor der Client es als korrekt akzeptiert; abgesehen davon hat `reliableExists()` dieselbe Semantik wie `exists()`.

Aufgabe:

→ Implementierung der Methode `reliableExists()` der Klasse `VSKeyValueClient`

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

Fällt ein Replikat aus und wird anschließend wieder neu gestartet, verfügt es im Allgemeinen nicht mehr über den aktuellen Zustand des Diensts. Um dies zu kompensieren, ist es das Ziel dieser Teilaufgabe dafür zu sorgen, 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 soll dabei auf dem von JGroups angebotenen Zustandstransfermechanismus basieren.

Aufgaben:

→ Erweiterung der Klasse `VSKeyValueReplica` um die Unterstützung von Zustandstransfers bei Neustarts
→ Testen der Implementierung durch manuelles Beenden und Starten von Replikaten

Hinweis:

- Da der Zustandstransfer in JGroups asynchron zur Zustellung totalgeordneter Nachrichten erfolgt, muss sich ein Replikat selbst um die konsistente Übernahme des Zustands kümmern.
- Einige bereits im Zustand berücksichtigte Nachrichten können nach dem Einspielen des Zustands erneut zugestellt werden. Das Replikat muss sicherstellen, dass sein Zustand dadurch nicht inkonsistent wird.

Abgabe: am Mi. 10.7.2019 in der Rechnerübung

Die für diese Übungsaufgabe erstellten Klassen sind in einem Subpackage `vsue.replica` zusammenzufassen.