Universität Trier

Fachbereich IV / Informatik
Prof. Dr. Peter Sturm
D-54286 Trier
Telefon 0651/201-3313
Fax 0651/201-3842
Email sturm@uni-trier.de

Verteilte Systeme, Übungsblatt 2, Sommer 2025

Distributed Shared Memory im CAP-Spannungsfeld

In diesen Aufgaben erweitern Sie den sim4da-Simulator um einen einfachen verteilten Speicher (DSM = Distributed Shared Memory), der allen simulierten Knoten eine einheitliche Sicht auf einen gemeinsam genutzten Schlüssel-Werte-Speicher zur Verfügung stellt. Die zentrale Idee: Die Illusion eines gemeinsamen Speichers soll erhalten bleiben – unabhängig davon, wie verteilt, fragmentiert oder partiell das zugrunde liegende System tatsächlich ist. Sie sollen drei alternative Implementierungen entwerfen, die jeweils zwei der drei CAP-Eigenschaften (Consistency, Availability, Partition Tolerance) erfüllen. Ziel ist es, die Unterschiede im Verhalten, insbesondere hinsichtlich der Konsistenz, erfahrbar und analysierbar zu machen.

Aufgabe 1 - Drei DSM-Varianten im Sinne des CAP-Theorems

Implementieren Sie jeweils eine Variante des DSM-Systems für folgende Kombinationen:

- AP (Availability & Partition Tolerance): Jeder Knoten speichert lokale Kopien der Daten. Schreibzugriffe propagieren asynchron an die anderen Knoten (z. B. per Gossip oder einfachem Broadcast). Lesende Zugriffe erfolgen lokal. Inkonsistenzen werden akzeptiert und ggf. später aufgelöst (z. B. durch "Last Write Wins" oder Vektoruhren).
  ➤ Hohe Verfügbarkeit – schwache Konsistenz.
- CP (Consistency & Partition Tolerance): Lese- und Schreibzugriffe erfordern ein Quorum von Knoten (z. B. Mehrheit). Wenn das Quorum nicht erreichbar ist (z. B. bei Partitionierung), blockieren Operationen.
  ➤ Starke Konsistenz – Verfügbarkeit eingeschränkt.
- CA (Consistency & Availability) Es wird angenommen, dass keine Partitionen auftreten (z. B. zentralisierte Koordination, keine Netzwerkfehler). Alle Zugriffe erfolgen synchron, mit sofortiger Sichtbarkeit aller Änderungen.
  ➤ Maximale Konsistenz – keine Partitionstoleranz.

Die DSM-Schnittstelle soll mindestens folgende Operationen unterstützen:

```
void write(String key, String value);
String read(String key);
```

Aufgabe 2 – Prototypische Anwendung zur Sichtbarmachung von Inkonsistenzen

Implementieren Sie auf Basis Ihrer DSM-Varianten eine einfache verteilte Anwendung, die Dateninkonsistenzen sichtbar macht. Ziel ist es, Unterschiede im Verhalten der drei Implementierungen im praktischen Einsatz zu erkennen und zu bewerten. Beispielidee: Jeder Knoten verwaltet einen lokalen Zählerstand im DSM. Alle Knoten lesen regelmäßig alle Zählerstände und prüfen auf Unstimmigkeiten (z. B. Rücksprünge, fehlende Updates, Divergenzen). Bei jeder Anomalie wird ein Logeintrag erzeugt oder farblich hervorgehoben.

Sie können die Anwendung frei gestalten – wichtig ist, dass:

- sie bei allen DSM-Varianten funktionsfähig ist,
- auftretende Inkonsistenzen klar erkennbar dokumentiert oder visualisiert werden,
- möglichst unterschiedliche Zugriffsmuster (lesend, schreibend, parallel) verwendet werden.

Aufgabe 3 – Vergleich und Bewertung

Führen Sie für alle drei DSM-Varianten systematische Experimente im sim4da-Simulator durch. Untersuchen Sie dabei insbesondere:

- Wie häufig und unter welchen Bedingungen treten Inkonsistenzen auf?
- Wie verhält sich das System bei Netzwerkverzögerungen oder simulierten Partitionen?
- Welche Varianten sind für welche Anwendungsszenarien geeignet?

Abgabe

Eine erfolgreiche Abgabe dieses zweiten Teils der Portfolioprüfung besteht aus: (1) dem Sourcecode zu den Aufgaben 1 und 2, (2) einem 3-5-seitigen Bericht und (3) einem ca. 15-minütigen Vortrag in der Übung. Den Sourcecode geben Sie mittels E-Mail an mich ab, vorzugsweise mit einem Link auf ein Github-Repository oder inklusive einer ZIP-Datei. Alternative Formen der Abgabe sollten Sie vorher mit mir absprechen. Der Bericht soll sich auf die von Ihnen gewählten Implementierungen konzentrieren und anschließend im Detail die gewonnenen Ergebnisse analysieren. Verwenden Sie zur Visualisierung komplexer Strukturen und Abläufe geeignete UML-Diagramme. Gemessene Ergebnisse profitieren in der Regel auch von einer Darstellung in einer geeigneten Diagrammform. Im Vortrag stellen Sie ebenfalls Ihre Lösungen vor und diskutieren mit den Teilnehmern Ihre Ergebnisse und Erfahrungen.

Dieses Übungsblatt bezieht sich auf eine Lösung der Aufgaben unter Verwendung des Simulators sim4da. Sollte Ihnen eine andere Programmiersprache als Java lieber sein, dürfen Sie die Lösung der Aufgaben gerne auch anders angehen (Verwendung eines anderen geeigneten Simulators, komplette Eigenimplementierung zum Beispiel in Python), solange der Charakter der Aufgabenstellung dabei nicht verloren geht. So ist es aus naheliegenden Gründen nicht zulässig, auf vorhandene DSM-Frameworks zurückzugreifen.

Die Abgabefrist ist der 6.7.2025. Sollten bis zur Abgabefrist Fragen oder Probleme aufkommen, können Sie sich gerne über die üblichen Kommunikationswege an uns wenden.