Ergebnisbericht Übungsblatt 2 Verteilte Systeme (SoSe 2025)

Simon Haebenbrock, s4sihaeb@uni-trier.de, dc: SimonH (simon_7672)

1. Entwicklungsumgebung und Projektstruktur

Für die Implementierung und Simulation der drei DSM-Varianten wurde IntelliJ IDEA als Entwicklungsumgebung eingesetzt. Das Projekt wurde in Java 17 entwickelt. Als Simulator kam das sim4da-Framework zum Einsatz. Für Logging wurde SLF4J (Version 1.7.36) mit der Simple-Binding-Implementierung verwendet. Die Projektstruktur umfasst drei Hauptkomponenten: die DSM-Varianten, die prototypische Counter-Anwendung und einen zentralen Simulation-Controller.

2. Implementierte Klassen und Architektur

Projektstruktur (Übersicht)

dsm/

- **AbstractDSM.java**
  Basisklasse, erbt von Node, stellt lokalen Speicher und Messaging bereit.
- **APDSM.java**
  AP-Variante mit asynchroner Gossip-Replikation und Eventual Consistency.
- **CPDSM.java**
  CP-Variante mit Quorum-basierten Reads/Writes und Konsistenzgarantie.
- **CADSM.java**
  CA-Variante mit synchroner Replikation, keine Partitionstoleranz.
- **DSMMessage.java**
  Nachrichtenklasse mit Typen, Zeitstempeln und Schlüssel/Wert.
- **DSMLogger.java**
  Logging- und Ergebnisprotokollierung.
- **DSMSimulation.java**
  Orchestriert alle Varianten und Experimente.
- **DistributedSharedMemory.java**
  Interface für Key-Value-Operationen (write, read) als gemeinsame Basis aller DSM-Varianten.

app/

- **CounterApp.java**
  Testanwendung, die Zähler verwaltet und Inkonsistenzen sichtbar macht.

Die zentrale Basisklasse AbstractDSM erbt von der Node-Klasse des sim4da-Frameworks und implementiert das DistributedSharedMemory-Interface. Sie stellt die grundlegenden Funktionalitäten bereit, wie einen thread-sicheren lokalen Speicher auf Basis von ConcurrentHashMap, die Verwaltung von Knotenidentitäten über nodeName() und das asynchrone Broadcasting von Nachrichten.

Zur Kommunikation zwischen den Knoten wurde die Klasse DSMMessage entwickelt. Sie unterstützt unterschiedliche Nachrichtentypen wie WRITE, READ_REQUEST, READ_RESPONSE, SYNC_REQUEST und SYNC_RESPONSE. Jede Nachricht ist mit einem Zeitstempel versehen, um bei konkurrierenden Änderungen Konflikte nach dem Last-Write-Wins-Prinzip aufzulösen.

Als Demonstrationsanwendung wurde die CounterApp umgesetzt. Jeder Knoten verwaltet darin einen eigenen Zähler und liest in regelmäßigen Abständen die Zähler aller anderen Knoten aus dem DSM. Die Anwendung erkennt automatisch Zähler-Rücksprünge, unerwartete Wertesprünge und Divergenzen und protokolliert diese zur späteren Auswertung.

DSMLogger bietet ein duales Logging-System mit ausführlichen Debug-Logs. Für die Durchführung der Simulation wurde die Klasse DSMSimulation entwickelt, die alle drei DSM-Varianten sequenziell ausführt, Metriken sammelt und vor jedem Test einen Simulator-Reset durchführt, um saubere Ausgangsbedingungen zu gewährleisten.

3. DSM-Implementierungen im Detail

In allen Varianten wird das gleiche Interface mit write(String key, String value) und read(String key) implementiert. Die drei Varianten unterscheiden sich hinsichtlich ihrer Trade-offs zwischen Konsistenz, Verfügbarkeit und Partitionstoleranz.

In der AP-Variante (Availability und Partition Tolerance) liegt der Fokus auf maximaler Verfügbarkeit. Alle Operationen erfolgen lokal, Inkonsistenzen werden bewusst akzeptiert. Die Nachrichtenpropagation erfolgt asynchron mit einer konfigurierten Message-Drop-Rate von 70 Prozent und einer Verzögerungsrate von bis zu fünf Sekunden. Partitionen werden zufällig simuliert. Bei Konflikten erfolgt eine Auflösung nach dem Last-Write-Wins-Prinzip. Zusätzlich besteht eine Wahrscheinlichkeit von 30 Prozent, dass veraltete Werte gelesen werden, und eine 15-prozentige Chance, dass Werte gar nicht propagiert werden. Ein separater Thread verarbeitet eingehende Nachrichten mit zufälliger Reihenfolge, um Eventual Consistency realistisch zu simulieren.

Die CP-Variante (Consistency und Partition Tolerance) setzt auf Quorum-basierte Operationen. Schreibzugriffe werden erst dann bestätigt, wenn eine Quorum-Anzahl von Knoten (mindestens 30 Prozent, mindestens jedoch die Mehrheit plus eins) die Operation bestätigt hat. Leseoperationen aggregieren Werte aus dem Quorum, um Konsistenz zu gewährleisten. Bei unzureichendem Quorum blockieren die Operationen. Ein Timeout von 200 ms sorgt dafür, dass Operationen bei Nichterreichen des Quorums automatisch abgebrochen werden. Die Synchronisation erfolgt über CountDownLatch.

In der CA-Variante (Consistency und Availability) wird keine Partitionstoleranz vorgesehen. Hier erfolgen alle Operationen synchron, wobei Schreiboperationen erst abgeschlossen sind, wenn alle Knoten Bestätigungen gesendet haben. Das System erkennt Partitionen anhand ausbleibender Bestätigungen und stoppt alle Operationen, um Konsistenz zu

gewährleisten. Für fehlerhafte Acknowledgments wurde ein Retry-Mechanismus mit bis zu drei Wiederholungen und exponentiellem Backoff implementiert. Zusätzlich lösen 80 Prozent der Leseoperationen proaktive Konsistenzprüfungen aus.

4. Experimentelle Ergebnisse und Analyse

Die Tests wurden mit fünf Knoten und jeweils fünf Iterationen pro Knoten durchgeführt. Jede DSM-Variante wurde in einer separaten Simulationsphase mit einem vollständigen Simulator-Reset gestartet, um konsistente Ausgangsbedingungen zu gewährleisten.

Für die AP-Variante wurden insgesamt 154 Operationen (137 Lese- und 17 Schreiboperationen) durchgeführt. Es traten drei Inkonsistenzen auf, die vor allem in Situationen mit hoher Verzögerung und simuliertem Nachrichtenverlust auftraten. Typisch waren Zähler-Rücksprünge, etwa wenn counter_node0 von 3 auf 2 fiel oder counter_node1 von 1 auf 0. Diese Effekte traten in Konstellationen auf, in denen Nachrichten nicht oder verspätet propagiert wurden, was den Eventual Consistency-Charakter bestätigt. Unter stabilen Netzbedingungen ohne Verzögerung wurde hingegen keine Inkonsistenz festgestellt. Die Analyse zeigt damit, dass Inkonsistenzen vor allem unter Netzwerkpartitionen oder hoher Verzögerung auftreten, während bei synchronen Phasen der Datenstand korrekt angeglichen wird.

Im CP-Modus wurden insgesamt 160 Operationen durchgeführt, ohne dass Inkonsistenzen festgestellt wurden. Alle Schreib- und Leseoperationen konnten durch erfolgreiche Quorum-Bildung bestätigt werden. Die Quorum-Erfolgsrate lag bei 100 Prozent. Selbst bei simulierten Verzögerungen führte das adaptive Timeout von 200 ms in diesem Testlauf zu keinen Blockierungen. Das System verhielt sich daher konsistent und verfügbar, solange das Netzwerk stabil war. Unter realen Bedingungen mit stärkerer Partitionierung wäre jedoch zu erwarten, dass Operationen blockieren, bis ein Quorum wieder erreichbar ist.

Die CA-Variante zeigte ebenfalls keine Inkonsistenzen. Hier kam es jedoch zu partiellen Blockaden, da die Partitionserkennung in mehreren Fällen den Betrieb für einzelne Knoten stoppte. Diese Fail-Stop-Mechanismen führten dazu, dass Knoten node1 und node3 deutlich weniger Operationen ausführten (nur 17 bzw. 19), während andere Knoten auf bis zu 28 Operationen kamen. Insgesamt wurden 111 Operationen gezählt. Das Verhalten bei simulierten Partitionen zeigt, dass CA-Systeme zwar perfekte Konsistenz liefern, jedoch bei Netzwerkausfällen komplett die Arbeit einstellen.

Die Ergebnisse beantworten damit konkret die Fragen nach Häufigkeit und Bedingungen von Inkonsistenzen:

- Im AP-Modus traten Inkonsistenzen regelmäßig unter künstlicher Verzögerung und Nachrichtenausfall auf.
- CP und CA zeigten in der Simulation keine Inkonsistenzen, allerdings führte CA bei Partitionen zu Operationsstopps.
- Verzögerungen hatten bei CP keinen Einfluss auf Konsistenz, bei AP jedoch signifikant.

5. Bewertung und Fazit

Die Simulationsergebnisse bestätigen die theoretischen Annahmen des CAP-Theorems. Die AP-Variante demonstrierte unter den getesteten Bedingungen, dass hohe Verfügbarkeit und Partitionstoleranz nur unter Inkaufnahme temporärer Inkonsistenzen möglich sind. Dies zeigte sich insbesondere bei den drei dokumentierten Rücksprüngen und Wertesprüngen während Perioden asynchroner Replikation und hoher Nachrichtenverluste.

Das CP-System garantierte über Quorum-Mechanismen vollständige Konsistenz aller Operationen. Im Test traten aufgrund stabiler Netzbedingungen keine Blockierungen auf, jedoch zeigt die Architektur, dass unter realen Partitionen Operationen blockiert werden müssten, um Konsistenz aufrechtzuerhalten.

Die CA-Variante zeigte die erwartete hohe Konsistenz und Verfügbarkeit in partitionfreien Abschnitten. Sobald die Partitionserkennung eintrat, beendeten die betroffenen Knoten ihre Operationen vollständig, was in einer signifikant niedrigeren Operationanzahl resultierte.

Für die Beurteilung der Anwendbarkeit in Szenarien ergeben sich folgende Empfehlungen: AP ist besonders geeignet für Anwendungen, bei denen Verfügbarkeit und Partitionstoleranz Vorrang haben und temporäre Inkonsistenzen akzeptabel sind, wie bei Caching-Systemen oder Social Media Feeds. CP eignet sich für sicherheitskritische Szenarien mit Konsistenzanforderungen, etwa Finanztransaktionen oder Inventarverwaltung, in denen bei einer Partition eine Blockierung toleriert werden kann. CA-Systeme sind optimal für Umgebungen ohne Partitionierungsrisiko, zum Beispiel hochverfügbare Cluster in Rechenzentren mit garantierter Verbindung.

Insgesamt zeigt die Implementierung, dass alle drei DSM-Varianten die theoretischen Einschränkungen des CAP-Theorems praktisch nachvollziehbar machen. Die Inkonsistenzen im AP-Modus, das konsistente, aber potenziell blockierende Verhalten des CP-Systems sowie die partitionsempfindliche CA-Implementierung liefern empirische Belege für die Unvereinbarkeit aller drei Eigenschaften.

Quellen

Vorlesungsskripte „Verteilte Systeme“ SoSe 2025 (2025S_DS01–2025S_DS07)
SLF4J API Reference (http://www.slf4j.org/)
sim4da-Framework
GitHub Copilot (Unterstützung bei Boilerplate-Code)