Verteilte Systeme

Entwurf zur Aufgabe 1

Florian Weiß, Ditmar Lange

18. Oktober 2018

Inhaltsverzeichnis

1 Organisatorisches **3**
1.1 Aufgabenaufteilung . . . 3
1.2 Bearbeitungszeitraum . . . 3
1.3 Aktueller Stand . . . 3
1.4 Änderung des Entwurfs . . . 3

2 Struktur, Vorgaben und Ablauf **4**
2.1 Server . . . 4
2.2 Client . . . 5
2.2.1 Redaktions-Client . . . 6
2.2.2 Leser-Client . . . 6
2.3 HBQ . . . 7
2.4 DLQ . . . 7
2.5 CMEM . . . 7
2.6 Weitere Vorgaben . . . 7

3 Methodendokumentation **8**
3.1 Server . . . 8
3.2 HBQ . . . 8
3.3 DLQ . . . 9
3.4 CMEM . . . 10

4 Test-Dokumentation **12**
4.1 Testkonzept . . . 12
4.1.1 client.cfg . . . 12
4.1.2 server.cfg . . . 12

1 Organisatorisches

Team: Ditmar Lange, Florian Weiß

1.1 Aufgabenaufteilung

Ditmar Lange: Pair Programming, mehr Anteil am Entwurf und HBQ
Florian Weiß: Pair Programming, mehr Anteil an Server, Client, DLQ, CMEM, Listen- und Helperfunktionen

1.2 Bearbeitungszeitraum

Ditmar Lange:

- 23.09.: 3 Std. Entwurf
- 25.09.: 8 Std. Programmieren
- 10.10.: 4 Std. Programmieren
- 17.10.: 6 Std. Programmieren
- 18.10.: 1 Std. Testen

Florian Weiß:

- 23.09.: 1 Std. Entwurf
- 23.09.: 6 Std. Programmieren
- 25.09.: 8 Std. Programmieren
- 10.10.: 7 Std. Programmieren
- 17.10.: 6 Std. Programmieren
- 18.10.: 3 Std. Testen

1.3 Aktueller Stand

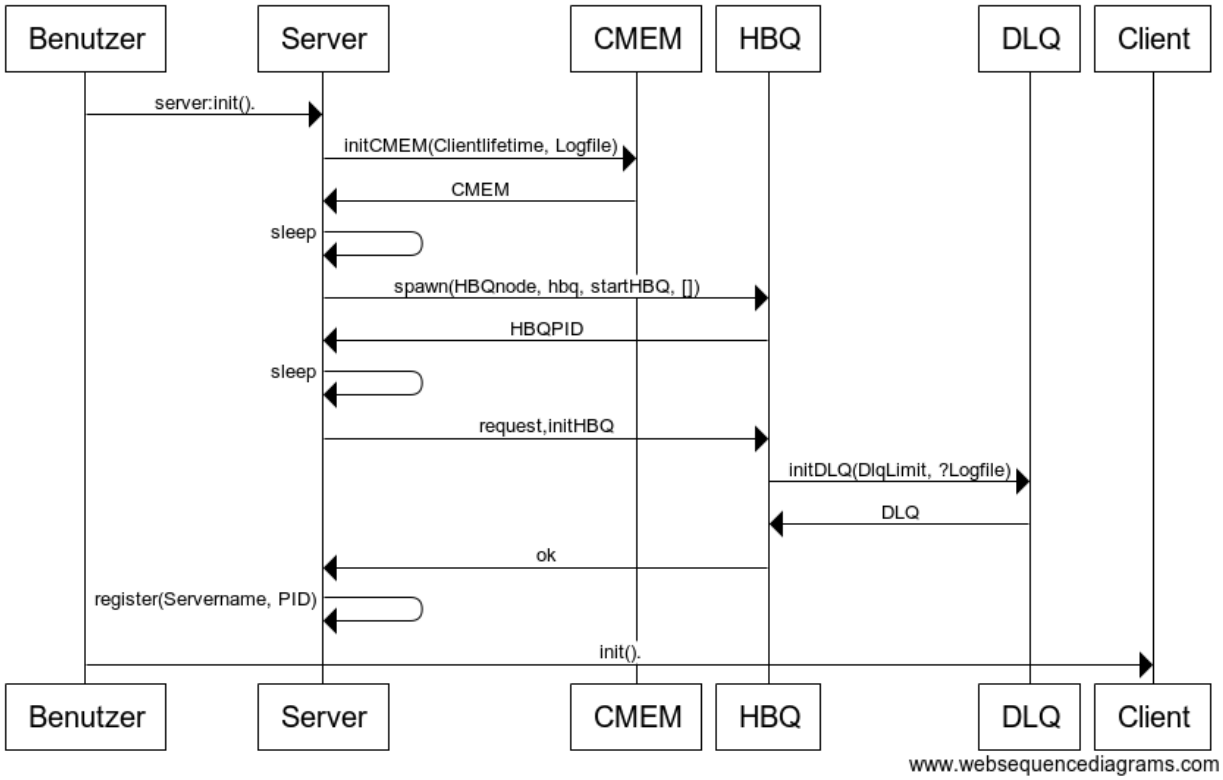
- Fertig programmiert. Einige Tests müssen noch ausgeführt werden.

1.4 Änderung des Entwurfs

Noch keine

2 Struktur, Vorgaben und Ablauf

Abbildung 1: Sequenzdiagramm: Initialisierung



2.1 Server

- Terminiert wenn die letzte Anfrage vom Client größer als die Wartezeit ist (angegeben in server.cfg).
- Außerdem in server.cfg angegeben sind die Latenz, der Servername, der HBQ-Name, die HBQ-Node und die Größe der DLQ.
- Besteht aus HBQ (hbq.erl), DLQ (dlq.erl), und CMEM (cmem.erl), sowie einer Server-Schnittstelle (server.erl) die den Nummerndienst und die Nachrichtenvermittlung enthält.
- Clients werden im CMEM gespeichert, bis sie wegen Inaktivität gelöscht werden.
- Der Server ist unter einem Namen <name> im lokalen Namensdienst von Erlang zu registrieren (register(<name>,ServerPid))

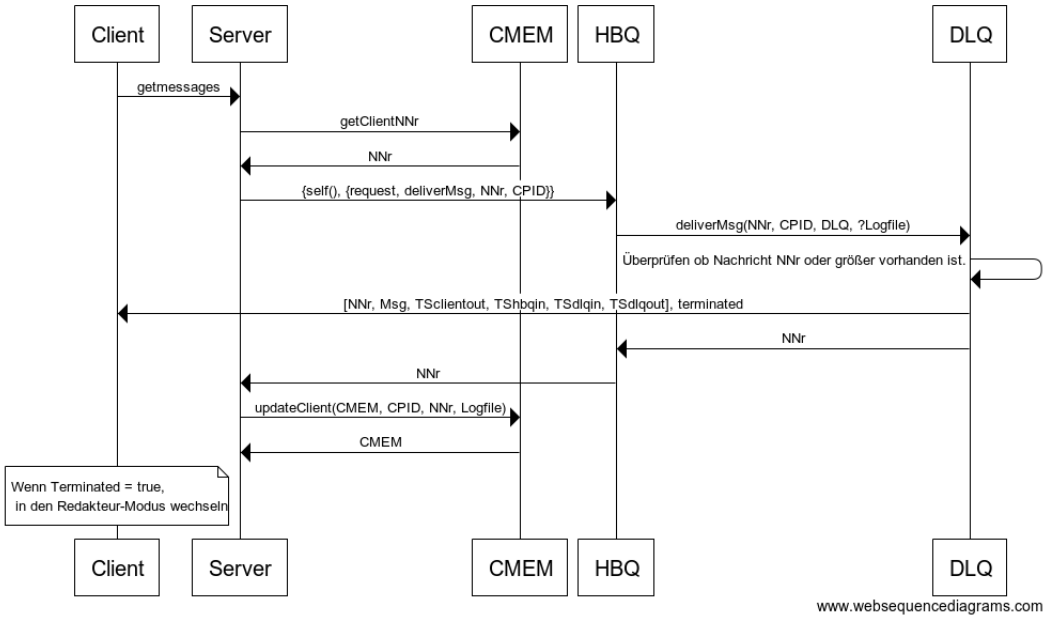
Abbildung 2: Sequenzdiagramm: Client im Redakteur-Modus

2.2 Client

Der Client besteht aus einem Leser-Teil, und einem Redakteur-Teil, die beide in einer Datei ausgeführt werden.

- Die Einstellungen für den Client (Anzahl der Clients, Lebenszeit, Servername, Servernode und das Sende-Intervall) sind in der Datei client.cfg angegeben.
- Sobald die Lebenszeit abgelaufen ist, wird der Client nach Beendigung der aktuellen Aufgabe sofort terminiert.
- Die Client-Datei besitzt eine NNr-Liste. Der Redakteur übergibt bei Beendigung seiner Aufgabe die Nachrichtnummern, die er versendet hat, an den Leser-Client. Dies ist damit er die Nachrichten des eigenen Redakteurs erkennen kann.

Abbildung 3: Sequenzdiagramm: Client im Leser-Modus



2.2.1 Redaktions-Client

- Alle n Sekunden sendet der Client eine Nachricht an den Server. Die Sekundenzahl wird in der client.cfg Datei angegeben und variiert nach jeder 5. Nachricht um bis zu 50% (min. 2 Sekunden).
- Die Nachricht, die versendet wird beinhaltet folgendes: seinen Rechnernamen, die Praktikumsgruppe, Teamnummer, ":", NNr, "te_Nachricht", "C OUT:", und den aktuellen Timestamp.
- Jede 6. Nachricht ist eine simulierte vergessene Nachricht. Daraufhin wird dann auch in den Leser-Modus gewechselt.

2.2.2 Leser-Client

- Der Leser-Client ruft die Nachrichten vom Server ab (einzelnd nacheinander).
- Der Client schreibt die Nachricht und die Timestamps in die Log-Datei.
- Sollte die NNr einer der Nachrichten in der NNr-Liste des Clients stehen, muss die Zeichenfolge "*******" an die Log-Nachricht angefügt werden.
- Sollte die empfangene Nachricht einen Timestamp aus der Zukunft haben, muss "Nachricht aus der Zukunft" in die Log-Datei geschrieben werden, und die Zeitdifferenz.

- Wenn die Nachricht den ‘Terminated’-Wert ‘true’ hat, so wechselt der Client wieder in den Redaktionsmodus.

2.3 HBQ

- Die HBQ enthält eine sortierte Liste, in die die empfangenen Nachrichten geschrieben werden.
- Wenn in der HBQ mehr als 2/3-tel an Nachrichten enthalten sind, als durch die vorgegebene maximale Anzahl an Nachrichten in der DLQ stehen können, dann wird, sofern eine Lücke besteht, diese Lücke zwischen DLQ und HBQ mit genau einer Fehlernachricht geschlossen, etwa: "***Fehlernachricht fuer Nachrichtennummern 11 bis 17 um 16.05 18:01:30,580”, indem diese Fehlernachricht in die DLQ eingetragen wird und als Nachrichten-ID die größte fehlende ID der Lücke erhält.

2.4 DLQ

- Die DLQ enthält eine sortierte Liste, in die die empfangenen Nachrichten geschrieben werden.

2.5 CMEM

- CMEM besteht aus einer Liste von Tupeln, bestehend aus Clients, deren NNr, und dem Timestamp der letzten Anfrage dieses Clients.
- Nach jeder Anfrage beim CMEM eines beliebigen Clients werden alle Clients auf den vordefinierten Timeout geprüft und gegebenenfalls entfernt.

2.6 Weitere Vorgaben

- Server und Client sind in Erlang zu implementieren.
- Als Hilfsstrukturen dürfen Tupel ( ) und Listen eingesetzt werden.
- "Übergaben” von komplexeren Datenstrukturen an Erlang OTP sind untersagt.
- Das CMEM darf nur vom Server aus angesprochen werden.
- Die DLQ darf nur von der HBQ aus angesprochen werde.
- Alle Dateien müssen austauschbar sein und es dürfen keine zusätzlichen Dateien erstellt werden.

3 Methodendokumentation

3.1 Server

init Initialisiert den Server, die HBQ und CMEM. Parameter werden aus der Config-Datei eingelesen. Die HBQ wird angepingt, registriert, und initialisiert. Es wird eine Config-Liste erzeugt, die für die interne Speicherung der Parameter verwendet wird.

getmessages Sendet der HBQ den Befehl, dem Client die Nachrichten, die er noch nicht gelesen hat, zu senden.

1. Überprüfen, welche Nachrichten der Leser schon bekommen hat durch getClient-NNr(CMEM, ClientID).
2. Sendet dem Leser die Nachricht mit der niedrigsten NNr aus der DLQ, die eine größere NNr hat als die letzte die der Leser bekommen hatte, indem deliverMSG von der HBQ ausgeführt wird, mit der Nachrichtnr. und ClientID als Parameter.

dropmessage Empfängt eine Nachricht, und schreibt diese in die HBQ.
Format der zu empfangenden Nachricht: [INNr,Msg,TSclientout]

1. Empfängt die Nachricht und schreibt sie mittels pushHBQ in die HBQ.

getmsgid Gibt die nächste unvergebene NNr zurück.

1. Der Server hat einen Zähler, der bei jedem Aufruf von getmsgid um 1 inkrementiert wird. Bei Initialisierung des Servers, wird diese Variable mit 1 initialisiert.

listDLQ Loggt eine Liste aller Nachrichtnummern der in der DLQ enthaltenen Nachrichten. Dafür wird listdlq bei der HBQ aufgerufen.

listhbq Loggt eine Liste aller Nachrichtnummern der in der HBQ enthaltenen Nachrichten an den Server aus. Dafür wird listhbq bei der HBQ aufgerufen.

listCMEM Loggt die Liste der dem CMEM aktuell bekannten Clients, mit dessen zuletzt empfangenen Nachrichtnummer. Dafür wird listCMEM im CMEM aufgerufen.

3.2 HBQ

startHBQ Liest die Config-Datei aus. Dann wird auf den initHBQ-Request gewartet.

initHBQ Initialisiert die HBQ und die DLQ

1. Erstellt eine Queue HBQ und eine Queue DLQ durch initDLQ
2. Gibt ok zurück
3. Außerdem wird berechnet und gespeichert, wie viel 2/3 von dem DLQ-Limit sind.

pushHBQ Fügt der HBQ eine Nachricht ein

1. Aktuellen Timestamp speichern (Eingangszeitstempel für die HBQ)
2. Bei der DLQ die als nächstes erwartete Nachrichtnr. anfragen.
3. Die HBQ sortieren.
4. Falls die empfangene NNr kleiner als die von der DLQ erwartete ist, wird die Nachricht verworfen.
5. Sollte die NNr der erwarteten NNr entsprechen, so wird die Nachricht direkt an die DLQ weitergeleitet (dlq:push2DLQ). Daraufhin wird ein ok zurückgegeben, und überprüft, ob die auf die NNr folgenden Nachrichten schon in der HBQ sind. In dem Fall werden die auch an die DLQ weitergeleitet und aus der HBQ entfernt.
6. Ist die empfangene NNr größer als die erwartete, so wird die Nachricht an die HBQ angefügt. Sollte damit das 2/3 Limit erreichtworden sein, so wird eine Fehlernachricht erzeugt und die HBQ-Nachricht mit der niedrigsten NNr wie in Schritt 5 an die DLQ weitergeleitet.

deliverMSG Beauftragt die DLQ (durch dlq:deliverMSG) damit, dem Client seine ungelesenen Nachrichten zu senden. Bekommt von der DLQ die tatsächlich gesendete NNr zurück, und sendet diese als Rückgabe an den Server.

listDLQ Sendet der DLQ den listDLQ Befehl, damit dessen Nachrichtennummmern geloggt werden. Bei Erfolg bekommt der Server ein ok.

listHBQ Loggt die Länge der HBQ, und die vorhandenen Nachrichtnummern. Der Server bekommt ein ok als Rückgabe.

dellHBQ Terminiert die HBQ.

1. Löscht die DLQ mittels dlq:delDLQ.
2. Gibt ok als Rückgabewert zurück.

3.3 DLQ

initDLQ(Size,Datei) Initialisiert die DLQ

1. Erstellt eine Queue DLQ mit Grösse size.

delDLQ(Queue) Löscht die DLQ und gibt ok zurück.

expectedNr(Queue) Liefert die NNr. der nächsten Nachricht, die in die DLQ geschrieben werden darf.

1. Wenn die DLQ leer ist, 1 zurückgeben
2. Ansonsten solange durch die Nachrichten iterieren, bis man bei der letzten angekommen ist. Dessen NNr + 1 zurückgeben.

push2DLQ([NNr,Msg,TSclientout,TShbqin],Queue,Datei) Speichert eine Nachricht in der DLQ und fügt einen Eingangszeitstempel an.

deliverMSG(MSGNr,ClientPID,Queue,Datei) Sendet eine Nachricht an den Leser.

1. Überprüft ob in der DLQ eine Nachricht mit NNr gleich MSGNr vorhanden ist.
2. Falls diese Nachricht nicht vorhanden ist, wird die erste Nachricht der DLQ genommen.
3. Fügt der Nachricht einen Zeitstempel TSdlqout an.
4. Überprüft ob weitere Nachrichten in der DLQ vorhanden sind, indem die nächste von der DLQ erwartete NNr mit der NNr der Nachricht verglichen wird. Ist die erwartete NNr um 1 größer als die dieser Nachricht, so wird der Parameter 'Terminated' auf 'true' gesetzt, als Zeichen dafür, dass dies die letzte Nachricht ist. Ansonsten ist 'Terminated' 'false'.
5. Sendet die Nachricht an den Client.
6. Gibt die gesendete NNr zurück.

listDLQ(Queue) Erzeugt eine Liste der in der DLQ vorhandenen Nachrichtnummern und gibt diese zurück.

lengthDLQ(Queue) Zählt die Anzahl der Nachrichten, die in der DLQ sind, und gibt diese zurück.

3.4 CMEM

initCMEM(RemTime,Datei) Erstellt CMEM und gibt leeres CMEM zurück

delCMEM(CMEM) Löscht CMEM und gibt ok zurück.

updateClient(CMEM,ClientID,NNr,Datei) Speichert die letzte NNr, die der Client bekommen hat

1. Überprüfen ob ein Client mit dieser ClientID schon im CMEM ist.
   a) Wenn ja, seine NNr und seinen Timestamp aktualisieren
   b) Ansonsten den Client im CMEM speichern, zusammen mit der NNr und dem Timestamp.
2. Alle Clients werden auf einen Timeout überprüft. Sollte einer vorhanden sein, so wird der Client aus dem CMEM entfernt.

getClientNNr(CMEM,ClientID) Gibt die NNr der Nachricht zurück, die der Client als nächstes bekommen sollte.

1. Wenn der Client nicht im CMEM vermerkt ist, wird 1 zurückgegeben.
2. Ansonsten gebe die gespeicherte NNr des Clients + 1 zurück.

listCMEM(CMEM Iteriert durch die Clientliste des CMEMs und speichert alle ClientIDs und dessen zuletzt bekommene NNr in einer Liste solcher Tupel. Die Liste wird daraufhin zurückgegeben.

lengthCMEM(CMEM) Die dem CMEM bekannten Clients werden gezählt, und die Anzahl dann zurückgegeben.

4 Test-Dokumentation

4.1 Testkonzept

Alle Dateien miteinander ausführen und auf richtige Funktionalität überprüfen. Außerdem bei der Implementierung der einzelnen Klassen, mit den Beam-Dateien von Prof. Klauck ausführen. Testen, ob das System weiterhin funktioniert, wenn man die ADTs mit den vorgegebenen ADTs austauscht.

Den Server und den Client mit dieser Konfiguration starten:

4.1.1 client.cfg

{clients, 2}. Anzahl der Clients, die gestartet werden sollen
{lifetime, 42}. Laufzeit der Clients
{servername, wk}. Name des Servers
{servernode, '<ServerName>@<NodeName>'}. Node des Servers
{sendeintervall, 3}. Zeitabstand der einzelnen Nachrichten

4.1.2 server.cfg

{latency, 60}. Zeit in Sekunden, die der Server bei Leerlauf wartet, bevor er sich beendet
{clientlifetime,5}. Zeitspanne, in der sich an den Client erinnert wird
{servername, wk}. Name des Servers als Atom
{hbqname, hbq}. Name der HBQ als Atom
{hbqnode, '<hbqNode-Name>@<NodeName>'}. Name der Node der HBQ als Atom
{dlqlimit, 13}. Größe der DLQ

Literatur

[1] http://learnyousomeerlang.com/recursion

[2] VS-Vorlesungsfolien