10. OKTOBER 2014

ENTWURF

VERTEILTE SYSTEME PRAKTIKUM 1

STEFFEN GIERSCH & MARIA LÜDEMANN
BAI 5 VERTEILTE SYSTEME BEI HERRN PROFESSOR KLAUCK
HAW Hamburg

1 INHALTSVERZEICHNIS

2 FUNKTIONALITÄT

Es soll eine Client/Server-Anwendung geschrieben werden bei der ein Server die Nachrichten verwaltet die ihm von unterschiedlichen Clienten (im weiteren Verlauf Redakteur genannt) zugesendet werden. Die Nachrichten die dabei getauscht werden sind eindeutig nummeriert und werden in regelmäßigen Abständen von unterschiedlichen Clienten (im weiteren Verlauf Leser genannt) abgefragt. Dabei soll Sorge getragen werden, dass ein Leser nicht alle Nachrichten immer wieder erhält sondern, sich der Server an den Leser erinnert und ihm nur die Nachrichten zuschickt die er noch nicht bekommen hat. Wenn sich ein Client einige Zeit nicht meldet vergisst der Server ihn wieder und behandelt ihn, beim erneuten Anfragen wie einen neuen Client. Der Server achtet außerdem darauf, dass die Nachrichten in der chronologisch korrekten Reihenfolge beim Clienten eingehen und nicht durcheinander wie sie beim Server ankommen könnten, im selben Zug schließt er Lücken die durch Fehler im Netzwerk oder andere Probleme entstehen können.

3 SERVER

3.1 ALLGEMEINES

Der Server nimmt die Nachrichten der Redakteure entgegen und schiebt sie erstmal in die Holdbackque und sortiert sie dort chronologisch. Wenn ein Leser nach neuen Nachrichten fragt schaut der Server nach welche Nachricht die letzte ist die dieser Client bekam und schickt ihm die nächste. Wenn in der Deliveryqueue Platz ist und in der Holdbackqueue keine akute Lücke herrscht wird die Deliveryqueue aufgefüllt. Gibt es eine akute Lücke und es wurde solange darauf gewartet dass sie sich von alleine schließt, dass die Holdbackqueue die Hälfte der möglichen Länge der Deliveryqueue erreicht hat, wird die Lücke mit einer Fehlermeldung geschlossen und die Deliveryqueue aufgefüllt.

Configparameter sind in einer server.cfg hinterlegt dort stehen:

Timeout -> Zeit des Servers die er wartet bevor er sich herunterfährt wenn keine Anfragen von Clienten mehr kommen

Deliveryqueuesize -> Die Länge die die Deliveryqueue haben darf

ClientTimeOut -> Die Zeit die sich der Server einen Clienten merkt bevor er ihn löscht wenn er sich nicht meldet.

Der Server besteht nur aus zwei Prozessen:

- Die beiden Prozesse sind einmal ein Timer, der zählt wie lange der Server nicht mehr angefragt wird und nach Ablauf der Zeit einen Timeout triggert.
- Und der Dispatcher der die Nachrichtenübermittlung und die ADTs managed.

3.2 Abblaufplan

Siehe Anhang 1

3.3 Grundlegende Datentypen

Nachricht = {Text,ClientOut,HBQIn,DLQIn,ClientIn}

Text -> String in dem die eigentliche Nachricht enthalten ist

ClientOut -> Timestamp von dem Moment wenn die Nachricht den Clienten(Redakteur) verlässt, bei Initialisierung 0

HBQIn -> Timestamp von dem Moment wenn die Nachricht die Holdbackqueue erreicht, bei Initialisierung 0

DLQInt -> Timestamp von dem Moment wenn die Nachricht die Deliveryqueue erreicht, bei Initialisierung 0

ClientIn -> Timestamp von dem Moment wenn die Nachricht den Clienten(Reader) erreicht, bei Initialisierung 0

ClientID -> Pid des jeweiligen Clients.

Nummer -> Integer

3.4 ADTs

3.4.1 Deliveryqueue

Die Deliveryqueue enthält die Nachrichten die an die Clients ausgeliefert werden können. Die Nachrichten sind immer nach Nummer aufsteigend geordnet.
Dabei ist die Deliveryqueue eine Liste von zweistelligen Tupeln, wobei das erste Element die Nachricht, und das zweite Element die Nummer der Nachricht enthält.

DLQ = [{Nachricht, Nr}]

3.4.1.1 Methoden

createNew() :: void -> DLQ

Ein neues DLQ Objekt wird erstellt und zurückgegeben. Agiert wie ein Konstruktor.

add (Msg, Nr, Queue) :: Nachricht x Nummer x DLQ -> DLQ

Die Nachricht wird der Queue hinten zugefügt, sodass sie sortiert bleibt. Wenn die Queue voll ist und eine weitere Nachricht kommt rein, wird die älteste Nachricht gelöscht, damit die Deliveryqueuesize nie überschritten wird.

get (Nr, Queue) :: Nummer x DLQ -> (Nachricht, Nummer , flag)

Gibt die Nachricht mit der angeforderten Nummer Nr zurück. Die Nummer ist die der erfolgreich angeforderten Nachricht. Das Atom „flag“ gibt an ob noch weitere Nachrichten existieren.

Wenn keine Nachricht mit der angeforderten Nummer in der DLQ existiert, dann wird nur das Atom „false“ zurückgegeben.

getLastMsgNr(DLQueue) :: DlQueue -> Nr

Gibt die Nummer der letzten Nachricht zurück die in der Deliveryqueue liegen.

3.4.2 HoldBackQueue

Die Holbackqueue enthält die Nachrichten die von den Redakteur-Clients eingeschickt wurden und reicht diese in geordneter Reihenfolge und mit bestmöglich geschlossenen Lücken (bei Paketverlusten) an die Deliveryqueue weiter.
Dabei ist die Holdbackqueue eine Liste von zweistelligen Tupeln, wobei das erste Element die Nachricht, und das zweite Element die Nummer der Nachricht enthält.

HBQ = [{Nachricht, Nr}]

3.4.2.1 Methoden

createNew() :: void -> HBQ

Ein neues HBQ Objekt wird erstellt und zurückgegeben. Agiert wie ein Konstruktor.

add(Msg, Nr, HBQueue, DLQueue) :: Nachricht x Nummer x HBQ x DLQ -> HBQ x DLQ

Die Nachricht der Queue hinzugefügt und die Queue nach Nachrichtennummer aufsteigend sortiert. Wenn die Queue eine Größe erreicht die der Hälfte der Größe der DLQ entspricht werden Lücken zwangsweise geschlossen und die DLQ nachgefüllt.

Msg ist hier die Nachricht um die es geht, Nr die Nummer der besagten Nachricht, HBQueue ist die Queue zu der es zugefügt wird, DLQueue ist die Deliveryqueue die unter Umständen verändert wird.

3.4.3 ClientList

Die ClientList enthält alle Clients die sich innerhalb einem gewissen Zeitraum bis zu diesem Zeitpunkt beim Server gemeldet haben.
Dabei ist die ClientList eine Liste von dreistelligen Tupeln, wobei das erste Element die ClientID (Pid) des Clients ist, das zweite Element die Nummer der letzten erhaltenen Nachricht und das dritte Element der Zeitpunkt ist, zu dem sich der Client das letzte mal beim Server gemeldet hat.

ClientList = [{ClientID, LastNumber, TimeStamp}]

3.4.3.1 Methoden

createNew() :: void -> ClientList

Ein neues ClientList Objekt wird erstellt und zurückgegeben. Agiert wie ein Konstruktor.

add(ID, CurrentTime, Queue) :: ClientID x TimeStamp x ClientList -> ClientList

Fügt einen neuen Clienten mit seiner ID, der Nummer der Nachricht die er als letztes angefragt hat und der Zeit zu der er dies tat hinzu.

exists(ID, Queue) :: ClientID x ClientList -> Boolean

Fragt die Liste ob es einen bestimmten Clienten schon gibt

update(CurrentTime, Queue) :: TimeStamp x ClientList -> ClientList

Läuft über die Liste der Clients und prüft ob es Clienten gibt die sich länger als einen gewisser Intervall nicht gemeldet haben und löscht sie aus der Liste.

setTime(ID, CurrentTime, Queue) :: ClientID x TimeStamp x ClientList -> ClientList

Erneuert die Zeit die zu einem Clienten gespeichtert wird wenn er sich meldet.

lastMessageID(ID, Queue) :: ClientID x ClientList -> Number

Ermittelt die Nummer der letzten Nachricht die der Client erhalten hat.

4 CLIENT

4.1 ALLGEMEINES

Aufgeteilt in Leser und Redakteur und darf nur aus einem Prozess bestehen.

Beide Clienten Teile liegen im selben Prozess sind aber autonom und laufen Sequentiell ab

In einer Config Datei client.cfg sind, Startparameter definiert und können dort für verschiedenen Testläufe verändert werden. Dort zu finden sind:

Minimale Zeit des Wartens -> 2 Sec

Start Wartezeit -> 5 Sec Initiale Zeit die zwischen den Nachrichten gewartet wird

Nachrichten Periodendauer -> Anzahl der Nachrichtennummern die angefragt werden bevor eine Nachricht ausgelassen wird und der Redakteur an den del Leser abgibt. Initial 6

4.1.1 RedakteurClient

Fragt Nachrichtennummer vom Server ab und schreibt ihm mit der Nummer eine Antwort dies wird fünf mal wiederholt, dann holt sich der Client eine Nummer antwortet darauf aber nicht nochmal.

Danach wird der warte Intervall geändert und zum LeseClient gewechselt und ihm eine Liste der fünf Nachrichtennummern die er mindestens zu erwarten hat geschickt.

4.1.2 LeseClient

Der LeseClient fragt den Server an ob es Nachrichten für ihn gibt und zeigt diese dann in seiner GUI an. Solange ihm der Server signalisiert, dass es noch neue Nachrichten gibt fragt der Client weiter nach. Erst wenn es keine mehr gibt wird zum Redakteur gewechselt. Der Leser überprüft ob er all die Nachrichten bekommt deren Nummer in der Liste stehen die er vom Redakteur bekommen hat und loggt ob sie alle da sind oder ob welche fehlen.

5 SCHNITTSTELLEN

Abfragen einer neuen Nachrichten-ID. Wird benutzt damit jeder Redakteur einzigartige IDs nutzen kann.

Server ! {getmsgid, Pid}
receive{nid, Number}

Senden einer neuen Nachricht an den Server. Dabei muss die Nr eine Nummer sein, die der Server vorher an diesen Client verteilt hat um Dopplungen und Inkonsistenz zu vermeiden.

Server ! {dropmessage, {Nachricht, Nr}}

Empfangen einer Nachricht die der nachfragende Client noch nicht erhalten hat. Welche Nachricht der Client benötigt wird vom Server gespeichert.

Server ! {getmessages, Pid}
receive ! {reply, Number, Nachricht, Terminated}

Der Server sendet eine Nachricht an seinen Timer damit dieser weiß, dass der Server noch aktiv ist.

Server ! {ping}

Der Timer sendet eine Nachricht an den Server, damit dieser herunterfährt wenn kein Client mehr aktiv ist.

Server ! {shutdown}

6 LOGGING

Die hier beschrieben Logs sind der Aufgabenstellung entnommen.

6.1 SERVER

Der Server nutzt einen Logging Prozess der in NServer.log schreibt

Serverstart:

Server Startzeit: 30.04 17:37:12,375| mit PID <0.870.0>

Nachrichtennummer an Clienten verschickt:

Server: Nachrichtennummer 5 an <9595.773.0> gesendet

Client fragt Nachricht an:

2-client@Brummpa-<0.771.0>-KLC: 45te_Nachricht. C Out: 30.04 17:37:32,874|(45); HBQ In: 30.04 17:37:32,875| DLQ In:30.04 17:37:38,969|.(45)-getmessages von <9595.772.0>-false

Server fährt herunter:

Downtime: 30.04 17:38:40,719| vom Nachrichtenserver <0.870.0>; Anzahl Restnachrichten in der HBQ:5

6.1.1 ADTs

HBQ loggt wenn sie Nachrichten transferiert

QVerwaltung1>>> Nachrichten [10,9,8,7,6,5,4,3,2,1] von HBQ in DLQ transferiert.

HBQ nimmt neue Nachricht auf

4-client@Brummpa-<0.773.0>-KLC: 5te_Nachricht. C Out: 30.04 17:37:16,515|(5); HBQ In: 30.04 17:37:16,516|-dropmessage

HLQ schließt Lücken: BSP.

***Fehlernachricht fuer Nachrichten 61 bis 70 um 30.04 17:37:54,328|.

DLQ löscht Nachrichten

QVerwaltung1>>> Nachrichten [6,5,4,3,2,1] von DLQ geloescht.

ClientList vergisst Clienten

Client <9595.772.0> wird vergessen! **********

6.2 Client

Der Client nutzt einen Logging Prozess der in client_<Nummer><Clienthost>.log schreibt.

Client startet:

2-client@Brummpa-<0.771.0>-KLC Start: 30.04 17:37:13,483|.

6.2.1 Redakteur Client

Nachricht senden:

2-client@Brummpa-<0.771.0>-KLC: 3te_Nachricht. C Out: 30.04 17:37:16,515| gesendet

Nachricht vergessen zu senden

28te_Nachricht um 30.04 17:37:28,577| vergessen zu senden ******

Nachrichtensende Intervall ändert sich:

Neues Sendeintervall: 2 Sekunden (3).

6.2.2 Lese Client

Fremde Nachricht vom Server empfangen:

0-client@Brummpa-<0.769.0>-KLC: 1te_Nachricht. C Out: 30.04 17:37:16,515|(1); HBQ In: 30.04 17:37:16,516| DLQ In:30.04 17:37:19,531|. ; C In: 30.04 17:37:28,593|

Eigene Nachricht vom Server empfangen:

2-client@Brummpa-<0.771.0>-KLC: 3te_Nachricht. C Out: 30.04 17:37:16,515|(3); HBQ In: 30.04 17:37:16,516| DLQ In:30.04 17:37:19,531|.own Message; C In: 30.04 17:37:28,608|

Nachrichten abfragen beenden:

..getmessages..Done...

Fehlernachrichten bei fehlenden Nachrichtennummern:

***Fehlernachricht fuer Nachrichten 26 bis 35 um 30.04 17:37:38,969|. ; C In: 30.04 17:37:39,077|