Letzte Aktualisierung: 14.03.2013 *Abgabe: 14.03.2013*

VSDB Protokoll

Aufgabe 9

Replikation

Gruppenmitglieder:
Aleksandar Doknic
Gabriel Pawlowsky

5BHITT

Inhaltsverzeichnis

1 Zeitaufteilung

1.1 Geschätzter Aufwand

Arbeitspaket	Pawlowksy	Doknic
Dokumentation	2 h	2 h
Recherche	0.5 h	0.5 h
Konzeptdesign	0.5 h	1 h
Konkretes Programmdesign	1 h	0.5 h
Umgebung aufsetzen	1 h	1 h
DB Replikation Implementierung	8 h	2 h
DB Replikation Testing	1 h	1 h
Filereplikation Implementierung	2 h	8 h
Filereplikation Testing	1 h	1 h
CRON Job + Script erstellen	0.5 h	0.5 h
Summe	17.5 h	17.5 h
Gesamt	35 h	

1.2 Gemessener Aufwand

Arbeitspaket	Pawlowksy	Doknic
Dokumentation	3 h	3 h
Recherche	2 h	1 h
Konzeptdesign	2 h	2 h
Konkretes Programmdesign	1.5 h	2 h
Umgebung aufsetzen	1.5 h	3 h
DB Replikation Implementierung	8 h	1 h
DB Replikation Testing	2 h	0 h
Filereplikation Implementierung	1.5 h	9 h
Filereplikation Testing	0 h	1 h
CRON Job + Script erstellen	0.5 h	0 h
Summe	22	22.5
Gesamt	44.5 h	

2 Cron Job

Um nun die gesamte Replikation in automatisch in regelmäßigen Abständen ausführen lassen zu können, müssen diese in einen Cron-Job verpackt werden. Dazu muss ein Rechner ausgewählt werden, der diesen Cron-Job immer startet. Ein Cron Job kann allerdings auschließlich Shell-Scrips ausführen, also muss als erstes ein solches erzeugt werden. Dieses nennen wir bspw. replication.sh und schreiben folgenden Text hinein:

```
#!/bin/bash
java -jar PathToJarFile
```

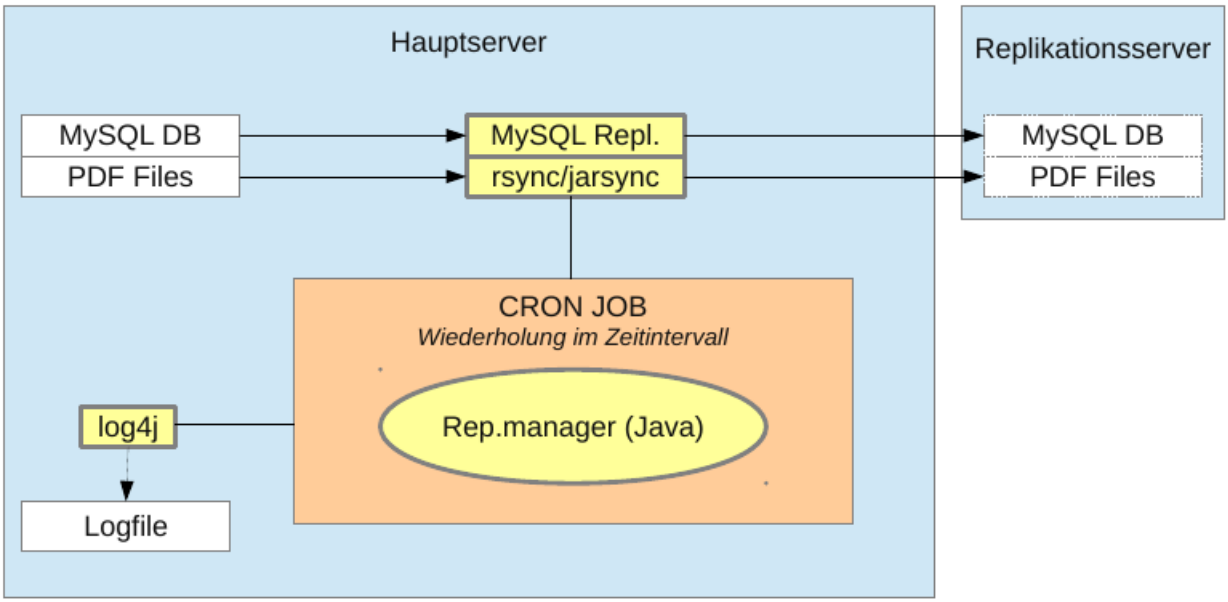
Die Config-Files sollten sich dazu allerdings ebenfalls im selben Ordner befinden. Nun kann man mit einem Texteditor seiner Wahl `/etc/crontab` editieren. In dieser Datei sollte man nun folgende Zeile einfügen:

```
0 9,15 * * * root sh PathToShFile
```

Dies würde die Replikation jeden Tag zwei mal, einmal um 9 Uhr und einmal um 15 Uhr ausführen. In der selben Datei in der diese Zeile eingefügt werden muss, wird allerdings auch beschrieben wie man hier den Cron Job zu anderen Zeiten ausführen lassen könnte. Nun muss man nur noch /etc/init.d/cron restart ausführen und schon funktioniert der Cron Job.

3 Grundkonzept

Grundsätzlich sollte die gesamte Replikation in etwa so verwirklicht werden.



Eine Erklärung dazu kann in den folgenden Punkten gefunden werden.

4 Database Replication

4.1 Replikationsprotokoll

Bei der Replikation der Datenbank mussten wir uns grundsätzlich zwischen Remote-Write und Master-Master entscheiden.
Remote-Write hat den Vorteil, dass jeder Client direkt auf den Server schreibt und dieser die Daten dann auf alle Clients repliziert. Durch diese Zwischeninstanz des Servers kann es unmöglich zu Inkonsistenzen kommen, da alles sofort in die Master-Datenbank eingefügt wird. Allerdings benötigt man für Remote-Write eine Client- und eine Server-Applikation und Queries können auf der Master-Datenbank nur über die Zwischeninstanz der Clientapplikation abgesetzt werden. Dazu muss zusätzlich zwischen der Client- und der Serveapplikation eine Netzwerkverbindung über z.B. RMI aufgebaut werden. Dies stellt also nicht nur einen weitaus größeren Codingaufwand dar, sondern auch die Kosten für Wartung und Support steigen dadurch stark an.
Daher habe ich mich für das Master-Master-Protokoll entschieden, bei dem nur eine einzelne Applikation auf dem Server läuft. Diese bekommt über ein Config-File alle benötigten Informationen über die teilnehmenden Clients (Host-Adresse, MySQL-Username, MySQL-Passwort, Datenbankname). Dadurch müssen Clients nicht kompliziert eine neue Software installieren, um Teil der Replikation sein zu können, sie müssen sich nur in eben diesem Config-File eintragen und schon wird die Tabelle des Webshops auf sie repliziert. Außerdem wird dabei das Netzwerk nur zum Zeitpunkt der Replikation belastet und nicht andauern, wie es bei Remote-Write der Fall wäre.

4.2 Replikation

Die Applikation zur Synchronisierung der Datenbanken wird in einem Cron-Job in regelmäßigen Abständen ausgeführt. Sie verbindet sich daraufhin mit allen Clients aus der Config-Datei und selected all deren Daten. Danach wird eine Methode aufgerufen, die aus allen Daten eine virtuelle „perfekte“ Tabelle erzeugt, die die Daten aller Tabellen zusammengemerged repräsentiert.
Das einzige Problem bei einer Master-Master-Replikation in regelmäßigen Abständen ist, das man bei zwei Tabellen von denen eine einen Datensatz enthält nicht wissen kann ob dieser auf der einen Tabelle gelöscht, oder auf der anderen eingefügt wurde. Deshalb wurde im Create-Script der Datenbank eine neue Spalte eingefügt, in der gespeichert wird, ob ein Datensatz gelöscht werden soll. Wenn man nun also in einer der Tabellen einen Datensatz entfernen möchte, muss man in Wirklichkeit eine UPDATE-Query absetzen, die den Datensatz als zu löschend markiert. Bei der nächsten Ausführung der Software werden diese Datensätzen dann wirklich gelöscht. Außerdem gibt es in der Datenbank eine Versionsnummer, die nach jedem UPDATE eines Datensatz erhöht werden sollte, damit die Software weiß, welcher der neueste Datensatz ist. Sollten zwei Datensätze unterschiedliche Daten, aber die selbe Versionsnummer besitzen, so wird eine zufällige der beiden Tabellen übernommen.

Das Create-Script der Datenbank sieht so aus:

```
DROP DATABASE IF EXISTS vsdb_webshop;
CREATE DATABASE vsdb_webshop;
USE vsdb_webshop;

DROP TABLE IF EXISTS artikel;
```

```
CREATE TABLE artikel (
id INT,
version INT,
kategorie VARCHAR(255),
abez VARCHAR(255),
abesch VARCHAR(255),
preis DECIMAL(10,2),
deleted BOOLEAN,
PRIMARY KEY (id)
)ENGINE = INNODB;

INSERT INTO artikel VALUES (1,1,"Getraenk","Red
Bull","Erfrischungsgetraenk fuer Idioten",1.29,false);
INSERT INTO artikel VALUES (2,1,"Nahrungsmittel","Felix
Ketchup","Gut!!!",0.99,false);
INSERT INTO artikel VALUES (3,1,"Nahrungsmittel","Brot","Brot ist
gesund.",0.39,false);
INSERT INTO artikel VALUES (4,1,"Nahrungsmittel","Kuchen","Wer
kein Brot hat, soll Kuchen essen.",0.79,false);
INSERT INTO artikel VALUES
(5,1,"Hygieneartikel","Zahnpasta","Macht Zaehne
sauber.",1.99,false);
INSERT INTO artikel VALUES (6,1,"Hygieneartikel","Shampoo","Macht
Haare sauber.",0.99,false);
INSERT INTO artikel VALUES (7,1,"Freizeitartikel","Fahrrad","Kann
man besteigen.",999.99,false);
INSERT INTO artikel VALUES (8,1,"Bueromaterial","Papier
500Stk.","Zum Drucken von Ausarbeitungen.",5.99,false);
INSERT INTO artikel VALUES
(9,1,"Getraenk","Milch","Pferdemilch?",1.99,false);
INSERT INTO artikel VALUES (10,1,"Hygieneartikel","Parfum","Bitte
nicht.",0.99,false);
```

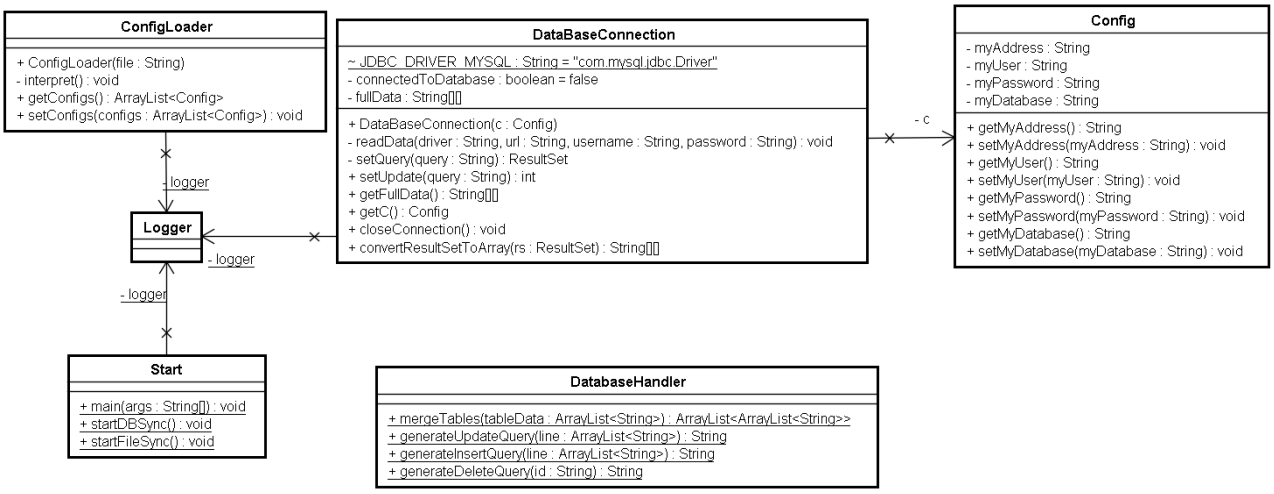
4.3 Probleme und Lösungen

- **Delete-Flag:** Das erste Problem, das sich ergeben hat war, dass man beim Vergleich zweier Datenbanken unmöglich wissen kann, ob ein neuer Eintrag in einer der Datenbanken inserted, oder in der anderen deleted wurde. Daher können auch nicht einfach Schlüsse für die Replikation auf die jeweils andere Datenbank gezogen werden.
  Lösung: Es wurde in der Datenbank in der Tabelle artikel eine Spalte delete hinzugefügt. Will ein User nun in einer der Datenbanken eine Zeile löschen, so muss er einfach nur die betroffene Zeile updaten und den delete-Flag auf true setzen. Bei der nächsten Durchführung der Software wird dieser Datensatz dann in allen Datenbanken geloescht.
- **Versionsnummer:** Wurde ein Datensatz in einer Datenbank verändert und diese Datenbank soll dann mit einer anderen Datenbank verglichen werden, kann man nicht wissen, in welcher Datenbank der Datensatz nun neuer ist, also welcher Datensatz nun auf alle Datenbanken verteilt werden soll.
  Lösung: Dazu wurde ebenfalls eine weitere Spalte in der Tabelle artikel ergänzt, nämlich eine Versionsnummer. Diese muss nun ebenfalls erhöht werden, wenn ein Datensatz aktualisiert wird. Dadurch kann die Software immer erkennen, welcher Datensatz der neuere

ist. Werden zwei Datensätze mit gleichen Versionsnummern und Ids, aber unterschiedlichen Daten gefunden, so wird einer der beiden als Master angenommen und auf alle anderen Datenbanken repliziert.

- **Entfernter Datenbankzugriff:** Weiters stellte sich die Frage, wie man auf die MySQL Datenbanken im Netzwerk zugreifen soll. Eine Möglichkeit wäre es gewesen eine Clientapplikation zu schreiben und diese über RMI mit der Serverappliaktion reden zu lassen. Diese könnten dann alle mit ihren lokalen MySQL-Servern reden. Da wir allerdings auf dem Prinziop des Cron-Job aufbauen wollten, würde dies rechtz wenig Sinn ergeben. Daraufhin haben wir uns für Remote-Zugriffe auf die MySQL-Server entschieden. Diese sind allerdings mit einem normalen öffentlichen User nicht durchführbar.
  Lösung: Wir mussten einen neuen User mit Remot-Zugriffs Priviligien (ALL PRIVILENGES ist auch möglich) erstellen, der dann im Config File festgehalten werden konnte. Über diesen kann sich unsere Software mit den entfernten MySQL-Datenbanken verbinden.
- **Trennung von Server- und Clientapplikation:** Ich fand es ein wenig unschön, dass bei einer Remote-Write Implementierung jeder Client eine Kopie der Client-Applikation benoetigt und dann über das Netzwerk mittels eines zusätzlichen Frameworks mit einer Serverapplikation kommuniziert.
  Lösung: Dies ist der Grund, warum wir uns für die Entfernten Datenbankzugriffe und nur eine zentrale Applikation auf einem Rechner entschieden haben. Diese Applikation ist außerdem nicht an einen bestimmten Ort oder eine bestimmte Maschine gebunden, sie muss nur im gleichen Netzwerk wie die anderen Datenbanken sein und auf diese zugreifen zu können.

4.4 Klassendesign



Man kann erkennen, dass grundsätzlich eine Startklasse implementiert wurde, die die gesamte Datenbankreplikation in Gang setzt. Zuerst wird das im nächsten Punkt beschriebene Config-File ausgelsen. Danach wird eine Verbidnung (DataBaseConnection) zu jeden Datenbank aufgebaut und es werden alle deren Daten selected. Die kommen dann zum DatabaseHandler und werden zu einer Datenbank zusammengemergt. Dieser gibt eine ArrayList aus Queries zurück, die dann in die jeweiligen Datenbanken eingefügt werden können (wieder mit DataBaseConnection). Danach sollten alle Datenbanken auf dem selben Stand sein.

4.5 Config-File

Ein Beispiel-Config-File könnte in etwa so aussehen:

```
hostAddr1 MySQLUser1 MySQLPW1 DBName1
hostAddr2 MySQLUser2 MySQLPW2 DBName2
```

und muss als mysql.config Datei im gleichen Ordner wie die .jar-Datei, die aausgeführt wird, gespeichert werden. Dabei können unendlich viele User mit eingebunden werden und sind somit alle Teil der Replikation. Da es sich um ein Master-Master-System handelt können somit alle Clients auch auf die Datenbank schreiben und es wird an alle anderen repliziert.

4.6 Logging

Für das Logging haben wir das Framework Log4J benutzt. Dieses Framework, sowie das benutzte properties-File sind anbei in dieser Abgabe.

5 File Replication

5.1 Besonderheiten dieses Konzepts

- Die *Consumer* benötigen keine speziellen Managementprogramme.
  - Vorteile
    - **Geringer Aufwand:** Auf den *Consumern* des Replikationssystem muss nur ein entsprechender *MySQL* und *SSH Server* muss installiert und konfiguriert werden.
    - **Weniger Speicherverbrauch:** Die Konsumenten benötigenkeine *JVM*.
    - **Weniger CPU Auslastung:** Die Konsumenten müssen sich nicht um die Replikation kümmern.
    - **Einfache Wartung:** Die Konfiguration des Replikationmanagers muss nur auf dem Provider geändert werden.
  - Nachteile
    - **Netzwerkauslastung:** Es müssen regelmäßig Daten übertragen werden, um zu prüfen, ob eine Änderung stattgefunden hat. Die Datenmenge ist jedoch relativ gering, da es sich nur um Dateinamen, Dateigröße und Änderungsdatum handelt.
    - **Versionsvergleich:** Es ist kein Hashvergleich möglich, da eine entsprechende Software auf jedem Consumer Gerät installiert sein müsste. Die Dateigröße muss für den Versionsvergleich herhalten.
    - **Fremdverzeichnispfade:** Der zentrale Server muss Betriebssystem und Zielverzeichnis der Konsumenten kennen.
- MySQL und DB Replikation sind grundsätzlich voneinander unabhängig
  - Vorteile
    - **Modularität:** Das Replikationssystem kann leicht modifiziert und den Anforderungen angepasst werden. Es gibt keine Grund ein untrennbar verschachteltes System für Datenbank und Datenreplikation zu entwickeln.
  - Nachteile
    - **Mehr Programmcode:** Wenige kilobyte spielen mit den heutigen Speicherkapazitäten praktisch keine Rolle mehr.
- Benutzer bekommt keine Meldung, sollte der Provider und damit die Synchronisation ausfallen
- rsync funktioniert nicht unter Windows:

```
protocol version mismatch -- is your shell clean?
(see the rsync man page for an explanation)
rsync error: protocol incompatibility (code 2) at compat.c(174) [Receiver=3.0.9]
```

5.2 Probleme

Heterogenität kann zu Fehlern führen durch...

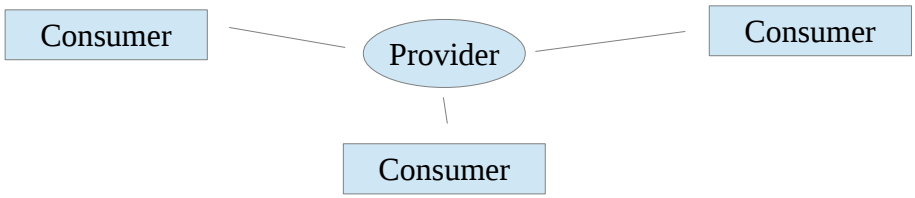
- unterschiedliche Dateisysteme
- unterschiedliche Betriebssysteme
- falsche Codierung von Sonderzeichen und Umlauten
- unerlaubte Zeichen in Dateinamen
- allgemein korrupte Daten / falsche Metadaten
- systemspezifische Verzeichnisstruktur (Line seperator, Partitionsbezeichnung...)

- unterschiedliche Zugriffsrechte
- Der Provider muss der Uhrzeit der Consumer vertrauen.

5.3 Synchronisation

Struktur

Es wird ein zentraler Provider mit beliebig vielen Consumern verwendet. Wir haben damit eine sternförmige Struktur:



Der Provider verfügt über eine lokale Kopie der Daten. In regelmäßigen Abständen prüft er die Kopien der Consumer auf Änderungen und lädt schließlich die neuen Dateien bzw. Dateien, die in einer aktuelleren Version vorliegen, als seine eigenen, runter. Dieser Prozess wird bei allen Consumern durchgeführt. Sobald alle Consumer abgearbeitet wurden, synchronisiert er deren Daten mit den eigenen.

Anmeldung eines Consumers

Hat ein Consumer keine Daten im Verzeichnis, so wird angenommen, dass dieser neu im System ist. Alle Daten werden gepusht.

Vereinigen und replizieren der Daten

Die Datenreplikation wird in Wellen abgearbeitet.
Eine Welle läuft über alle Consumer, aktualisiert den Datenbestand des Providers und pusht diesen am Schluss zurück zu den Consumern. Mit jedem Consumer werden die Providerdaten aktueller.
Die iterative Vorgehensweise ist nur möglich, da rsync den Zeitstempel beibehält. Somit können die Providerdaten, falls notwendig, mehrfach überschrieben werden und wir erhalten letzten Endes die aktuellste Version.

Add Liste	
Beschreibung	Enthält Dateien, die neu zum Provider hinzugekommen sind. Dateien aus der Delete Liste, können nicht in der Add Liste landen.
Typ	ArrayList<String Dateiname>
Zweck	Sperrt Daten, damit diese nicht sofort gelöscht werden.
Lebenszeit	Bis zum Ende der Replikation.

Delete Liste	
Beschreibung	Enthält Dateien, die bei Consumern, aber nicht mehr beim Provider vorhanden sind. Dateien aus der Add Liste, können nicht in der Delete Liste landen.

Typ	ArrayList<String Dateiname>
Zweck	Dateien, die in der Delete Liste sind, werden gelöscht.
Lebenszeit	Bis zum Ende der Replikation.

Sync Liste	
Beschreibung	Enthält neue Dateien und Dateien, deren Änderungsdatum aktueller ist, als das der Providerversion. Dateien aus der Delete Liste, können nicht in der Sync Liste landen.
Typ	ArrayList<String Dateiname>
Zweck	Nur die Dateien die sich in dieser Liste befinden, werden gepullt.
Lebenszeit	Bis zum Consumerabgleich.

1. Dateiliste von Consumer n wird abgerufen
2. Vergleich zwischen Provider und Consumer Daten
    1. Gleiche Dateien → ignorieren
    2. Neue Dateien → Sync Liste
    3. Geänderte Dateien → Sync Liste
    4. Gelöschte Dateien → Delete Liste
3. Dateien aus der Sync Liste werden auf den Server kopiert
4. Dateien aus der Delete Liste werden vom Server gelöscht
    → Erstellung einer Datei namens <Dateiname>.ghost in einem Extra Verzeichnis
5. Der Vorgang wird mit Consumer n+1 wiederholt, bis alle durch sind
6. Die Daten vom Provider werden jetzt auf alle Consumer gepusht. Es findet eine komplette Synchronisation statt. Alle Dateien werden synchronisiert und alle Dateien, die nicht auf dem Provider zu finden sind, werden gelöscht.

Wurde eine Datei auf Consumer 1 geändert und auf Consumer 2 gelöscht, so wird sie gelöscht. Die Änderungen gehen verloren.

Beispiel

Runde 1						
Provider			**Consumer 1**			**Aktion**
Datei	**Größe**	**Datum**	**Datei**	**Größe**	**Datum**	
R1.pdf	1230	20:00	R1.pdf	4041	21:00	**Sync List**
R2.pdf	1981	20:00				**Delete List**
R3.pdf	814	20:00	R3.pdf	412	19:00	
			R4.pdf	948	19:00	**Sync List** **Add List**

Runde 2						
Provider			**Consumer 2**			**Aktion**
Datei	**Größe**	**Datum**	**Datei**	**Größe**	**Datum**	
R1.pdf	4041	21:00	R1.pdf	2025	23:00	**Sync List**

			R2.pdf	1981	23:00	(Add Liste)
R3.pdf	814	20:00	R3.pdf	814	23:00	
R4.pdf	948	19:00				(Delete Liste)

Runde 3						
Provider			Consumer 3			Aktion
Datei	Größe	Datum	Datei	Größe	Datum	
R1.pdf	2025	23:00	R1.pdf	4041	21:00	
R3.pdf	814	20:00	R3.pdf	9001	21:00	Sync List
R4.pdf	948	19:00	R4.pdf	948	23:00	Sync List

Jetzt wird der finale Datensatz vom Provider an alle Consumer geschickt. Es wird dabei zu 100% synchronisiert. Dateien, die der Provider nicht (mehr) hat, werden ebenfalls gelöscht.
An diesem Punkt sind alle Daten konsistent.

Nach dem Push...								
Consumer 1			Consumer 2			Consumer 3		
Datei	Größe	Datum	Datei	Größe	Datum	Datei	Größe	Datum
R1.pdf	2025	23:00	R1.pdf	2025	23:00	R1.pdf	2025	23:00
R3.pdf	814	21:00	R3.pdf	814	21:00	R3.pdf	814	21:00
R4.pdf	948	19:00	R4.pdf	948	19:00	R4.pdf	948	19:00

Hinweis:
Änderungen zwischen Pull und Push werden verworfen. Das Zeitfenster sollte daher klein gehalten werden. Das kann zB. dadurch erreicht werden, dass der Push in mehreren Threads stattfindet. Schneller würde es gehen, wenn der Push sofort nach dem Abgleich mit jedem einzelnen Consumer erfolgt. Somit müsste dieser nicht warten, bis alle anderen abgeglichen wurden. Das würde allerdings dazu führen, dass die Daten zu keinem Zeitpunkt zwingend konsistent sind.

5.4 Provider

Der Provider muss folgende Programme installiert (und ggf. im PATH eingetragen) haben:

- Java Runtime
- rsync
  - Ermöglicht eine effiziente und schnelle Datenreplikation
- sshpass
  - Ermöglicht das Ausführen von Remote Befehlen über SSH

Beispiel für das Abrufen von Verzeichnisinhalten

```
sshpass -p <passwort> ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no <user>@<host> 'ls -l'
```

Beispiel für Synchronisation:

```
rsync -v -u --delete --rsh='sshpass -p <passwort> ssh -o
UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -l user' -r
<Quellordner>/ user@10.155.188.106:<Zielordner>/
```

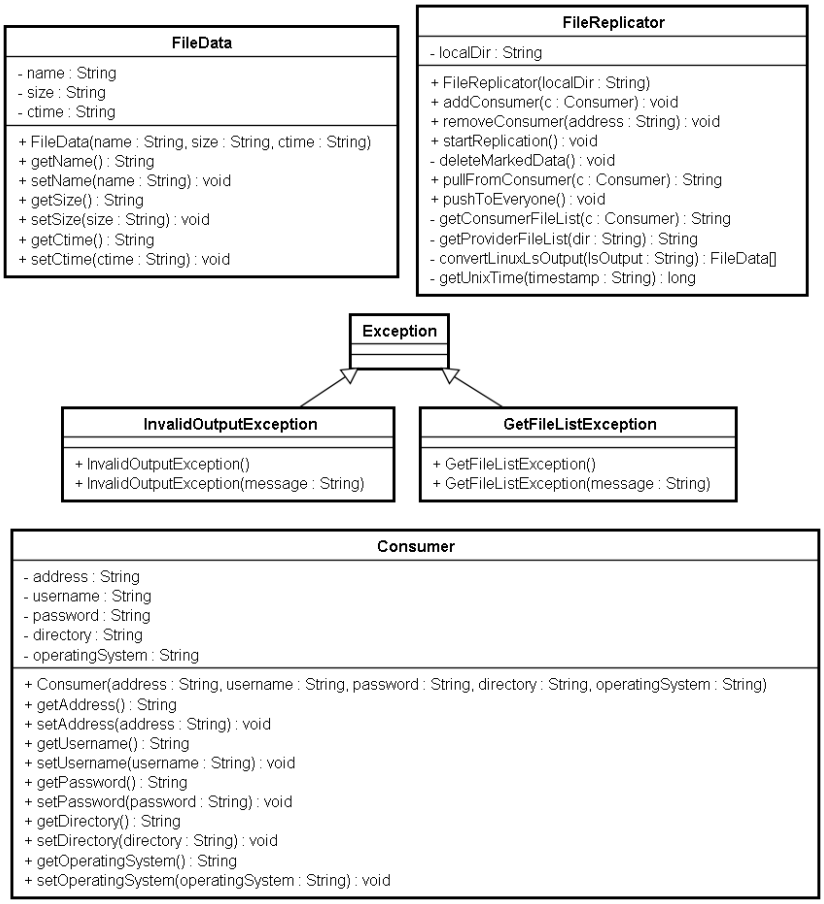
Mit der –delete Option werden Daten gelöscht, die auf dem Zielrechner nicht existieren.
Anmerkung: Mit -u werden nur Dateien geladen, die aktueller sind, als die vom Provider.
Damit wird die „Sync List“ nicht mehr benötigt!
Die Add List und die Delete List dienen nur noch dem Zweck, gelöschte Dateien zu erkennen.

Consumer

Der Client muss einen SSH Server mit Schreibrechten zur Verfügung stellen. Das genaue Zielverzeichnis wird vom Provider festgelegt.

5.5 Klassendesign

Das Klassendesign der FileReplication sieht so aus:



6 Quellen

http://ubuntuforums.org/showthread.php?t=1809158
http://stackoverflow.com/questions/8775303/read-properties-file-outside-jar-file
http://javawords.com/2008/01/08/how-to-use-log4j-logging-api/