HANSER

Leseprobe

Stefan Edlich, Achim Friedland, Jens Hampe, Benjamin Brauer, Markus Brückner

NoSQL

Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken

ISBN: 978-3-446-42753-2

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-42753-2

sowie im Buchhandel.

3 Wide Column Stores

Die Idee der Wide Column Stores stammt schon aus den 80er Jahren [Kosh87]. In einem Wide Column Store wird jedes Attribut in einer eigenen Tabelle hintereinander (spaltenorientiert) und nicht wie bei relationalen Modellen in einer Tabelle untereinander (reihenorientiert) gespeichert.

Dabei ist die physische Datenorganisation anders organisiert als bei der relationalen Speichertechnik. Ein relationales Datenbankschema mit den Attributen ID, Name und Alter würde reihenorientiert so gespeichert:

```
1, Tom, 42, 2, Mary, 18, 3, Paul, 36
```

In einer spaltenorientierten Datenbank wären die Daten nach ihrer Spalte gruppiert:

```
1, 2, 3, Tom, Mary, Paul, 42, 18, 36
```

Dieses Verfahren hat viele Vorteile bei der Analyse der Daten, der Datenkompression und beispielsweise auch für das Caching. Die Aggregation von Daten ist hier aber natürlich einer der wichtigsten Vorteile, weswegen OLAP- und Data-Warehouse-Umgebungen darauf zurückgreifen.

Aber natürlich gibt es auch Nachteile. Die Suche und das Einfügen von Daten kann eventuell aufwendiger sein. Und natürlich ist das Schreiben und Lesen von Objektstrukturen, d.h. von zusammengehörigen Spaltendaten aufwendiger, da man viel springen und suchen muss (z.B. alle Daten von Tom).

Spaltenorientierte Datenbanken gibt es zwar schon seit den 90er Jahren, aber einen echten Aufschwung erleben sie seit 2000 in Umgebungen, wo ihre Vorteile entscheidend sind. Zu den leistungsfähigen Datenbanken dieser Kategorie zählen Sybase IQ, FluidDB, C-Store und MonetDB.

Partiell spaltenorientiert

Die in diesem Kapitel beschriebenen Datenbanken HBase, Cassandra und Amazon SimpleDB weichen aber von der oben beschriebenen Idee der Wide Column Stores stark ab, und zwar in der Frage des angebotenen Dimensionsformates. Hier hat das Design von Google BigTable eine Architektur vorgegeben, die als „sparse, distributed multidimensional sorted map“ beschrieben wurde. Dies sind in der Regel mehrdimensionale Tabellen im folgenden Format:

```
n*[Domain / Keyspace] x [Item / Column Family] x [Key x] n*[Key+Value]
```

Hier gibt es also meistens mehrere oder viele Dimensionen aus *Domains* oder *Keyspaces*, deren kleinstes Feld dann ein `Item` oder eine Column Family bildet. In diesem Item gibt es dann beliebig viele Key/Value-Maps, die wiederum eventuell über einen übergeordneten Schlüssel angesprochen werden können. Dies bedeutet, dass die unterste Ebene eine Menge von Key/Value-Maps ist und so eine Einheit bildet. Auf höherer Ebene findet aber eine Gruppierung ähnlich der spaltenorientierten Datenbanken statt, wo ähnliche Eigenschaften (hier zusammengehörige *Items/Column Families*) zusammengefasst werden.

Derartige Datenbanken sind sehr gut skalierbar und typischerweise für sehr große Datenmengen geeignet. Neben den hier beschriebenen Hadoop/HBase, Cassandra und SimpleDB gibt es noch das Hadoop-Derivat Cloudera, welches in Kapitel 7 kurz beschrieben wird. Weiterhin gibt es durchaus interessante Mischformen wie SciDB (http://www.scidb.org), die mehrdimensionale Arrays anbieten.

Literatur

[Kosh87] S. Khoshafian, G. Copeland, T. Jagodis, H. Boral, P. Valduriez: A query processing strategy for the decomposed storage model, 1987, ICDE.

3.1 HBase

Steckbrief

Webadresse:	http://hadoop.apache.org/hbase/
Kategorie:	Wide Column Store
API:	Java, REST, Thrift, Scala, Groovy, Jython
Protokoll:	RPC, HTTP
Geschrieben in:	Java
Concurrency:	Atomic Locking, Optimistic Concurrency Control
Replikation:	über Hadoop HDFS
Skalierung:	nativ unterstützt durch Hinzufügen neuer RegionServer
Lizenz:	Apache License, Version 2.0

3.1.1 Überblick

HBase ist vergleichbar mit dem freien und quelloffenen OpenOffice. Wie dieses ist HBase ein quelloffener Klon eines proprietären Systems. Man sollte aber nicht unterschätzen, was mit HBase quelloffen zur Verfügung gestellt wird, handelt es sich dabei doch schließlich um eine Nachmodellierung von Googles BigTable. Dieses ist bekannt dafür, dass es für eine Skalierung jenseits der Zahl von 1000 Knoten konzipiert ist.

Ende 2006 veröffentlichte Google einen wissenschaftlichen Artikel zur Architektur des intern entwickelten verteilten Datenbanksystems BigTable. Etwa zur gleichen Zeit begann

die Entwicklung von HBase. BigTable wurde von Google so gestaltet, dass es die speziellen Anforderungen der von Google über das Web bereitgestellten Dienste erfüllt.

Trotz der Veröffentlichung der strukturellen Merkmale von BigTable ist das System eine proprietäre Technologie von Google und wird Dritten nicht zur Nutzung bereitgestellt. Diesen Umstand versucht das quelloffene HBase-Projekt zu ändern.

3.1.2 Allgemeines

HBase ist Open Source und ein Teilprojekt von Apache Hadoop, einem Projekt der Apache Software Foundation. Ziel des Hadoop-Projekts ist es, eine freie Implementierung der von Google entwickelten proprietären Infrastrukturtechnologien bereitzustellen. HBase bildet dabei das proprietäre Datenbanksystem BigTable von Google nach.

Die Entwicklung von HBase begann Ende 2006 durch Chad Walters und Jim Kellerman, beide Mitarbeiter der Firma Powerset. Dort benötigte man ein Datenbanksystem wie BigTable für ein System zur Verarbeitung natürlicher Sprache im Web. Ausgangsbasis dazu war eine nahezu lauffähige Code-Basis eines BigTable-Nachbaus von Mike Cafarella. Walters und Kellerman erweiterten diese. Die Firma Powerset wurde 2008 von Microsoft übernommen. Dies hat jedoch nichts daran geändert, dass Powerset weiterhin den harten Kern der Entwickler für HBase stellt. Der Weiterentwicklung von HBase hat dies jedoch nicht geschadet. Schnell hat das System weitere Unterstützer in der IT-Industrie gefunden, darunter auch Facebook und den Sicherheitsspezialisten Trend Micro.

HBase ist ein spaltenorientiertes Datenbanksystem zur verteilten Speicherung großer Mengen semistrukturierter Daten. Das System ist in Java implementiert. Wie BigTable mit dem Google File System sieht HBase zur Speicherung der Daten und Log-Dateien die Nutzung eines verteilten Dateisystems wie Hadoops HDFS vor. HBase verzichtet wie alle hier vorgestellten Datenbanksysteme auf die Mehrzahl der Features von relationalen Datenbanken mit dem primären Ziel, möglichst einfach auf preisgünstiger Standardhardware zu skalieren. Produktiv im Einsatz ist HBase bei Firmen wie Yahoo, Adobe und StumbleUpon. Mit der Firma Cloudera existiert auch ein Anbieter von kommerziellen Support für Apache Hadoop und HBase.

3.1.3 Datenmodell

In HBase werden Daten in Tabellen gespeichert. Eine Tabelle besteht aus Zeilen und Spalten. Eine Zeile steht für einen Datensatz, während eine Spalte ein Attribut repräsentiert. Jede Zeile wird über einen eindeutigen Schlüssel identifiziert. Der Schlüssel einer Zeile in HBase ist vergleichbar mit einem Primärschlüssel in relationalen Datenbanken. Jeder Zugriff auf eine Tabelle erfolgt über diesen Schlüssel. HBase verwendet für die Schlüssel ein Byte-Array. Dies ermöglicht beim Aufbau geeigneter Schlüssel viel Freiraum, da von Zeichenketten bis zu rohen Binärdaten vieles denkbar ist. Die Zeilen einer Tabelle werden nach dem Schlüssel sortiert. Standardmäßig wird dabei die Binärsortierung verwendet.

Bis hierhin ähnelt das Datenmodell von HBase dem klassischen Modell relationaler Datenbanken. Abweichend davon haben die Spalten in HBase keinen definierten Datentyp fester Länge. Alle Zelleninhalte sind simple Byte-Arrays. Die Umwandlung in den korrekten Datentyp muss auf Ebene der Applikationslogik geschehen, wie in den noch folgenden Code-Beispielen zu sehen sein wird. Spalten können nach Bedarf zur Laufzeit hinzugefügt werden. Leere Zellen (Null), wie man sie aus relationalen Datenbanken kennt, existieren in HBase nicht. Zellen existieren nur dann, wenn sie auch einen Wert haben. Die Zellen werden versioniert. Zu diesem Zweck fügt HBase bei jeder Einfügeoperation automatisch einen Zeitstempel des Ausführungszeitpunkts an.

Ein Datenschema in HBase legt Tabellen, Spaltenfamilien und deren Eigenschaften fest. Die Spaltenfamilie ist ein von Googles BigTable übernommenes Konzept und dient der Gruppierung von datentechnisch ähnlichen Spalten. Physisch werden die Daten einer Spaltenfamilie zusammenhängend gespeichert, was das gemeinsame Unterscheidungsmerkmal einer Wide Column Datenbank wie HBase im Vergleich zu relationalen Datenbanken ist. Bezogen auf die Schemafreiheit geht HBase mit dem Konzept der Spaltenfamilie einen Mittelweg, der es erlaubt, ähnliche Datentypen in einer Spaltenfamilie zusammenzufassen. Eine Optimierung der Performance von HBase erfolgt daher auf Ebene der Spaltenfamilien. Zum aktuellen Entwicklungsstand ist es zu empfehlen, die Zahl der Spaltenfamilien auf maximal 3 zu begrenzen, da sonst die Performance von HBase stark abnimmt.

Der Zugriff auf eine konkrete Spalte erfolgt über eine Kombination aus einem Präfix für die Spaltenfamilie und dem Bezeichner der Spalte:

```
[Spaltenfamilie]:[Spaltenbezeichner]
Beispiel: map:image
```

Das Präfix für die Spaltenfamilie muss aus druckbaren Zeichen bestehen, darf aber nicht das Trennzeichen enthalten. Dieses ist per Konvention als „:“ festgelegt. Der Bezeichner für eine Spalte kann jedoch ein beliebiges Byte-Array sein. Eine Spaltenfamilie muss vorab als Teil des Schemas einer Tabelle definiert werden. Spalten können später nach Bedarf zu einer Spaltenfamilie hinzugefügt werden. Clients können in einer Update-Operation beliebige Spalten hinzufügen, solange die zugehörige Spaltenfamilie bereits definiert wurde.

Trotz der Verwendung von Bezeichnungen aus der relationalen Welt für das Datenmodell von HBase hebt sich dieses jedoch deutlich vom relationalen Modell ab. Das Datenmodell von HBase ist im Grunde ein assoziatives Array, auch als Map oder Dictionary bekannt. Dieses ist nach Schlüsseln sortiert, von denen jeder auf eine Zeile zeigt. Die Zeilen können wiederum als ein assoziatives Array interpretiert werden. In diesem Fall sind die Schlüssel die Bezeichner der Spaltenfamilien. Auch die Werte der Spaltenfamilien sind ein assoziatives Array mit den Spaltenbezeichnern als Schlüssel. Letztlich handelt es sich daher beim Datenmodell von HBase um ein mehrdimensionales assoziatives Array, wie es folgendes Listing verdeutlichen soll:

Listing 3.1.1 HBase-Datenmodell als mehrdimensionales assoziatives Array

```
KEY_1 => COLUMN_FAMILY_ONE => A => Value
                           => B => Value
      => COLUMN_FAMILY_TWO => C => Value
```

```
KEY_2 => COLUMN_FAMILY_ONE => A => Value
      => COLUMN_FAMILY_TWO => D => Value
                           => E => Value

KEY_3 => COLUMN_FAMILY_ONE => A => Value
                           => E => Value
                           => F => Value
      => COLUMN_FAMILY_TWO => C => Value
                           => D => Value
```

3.1.4 Installation

Die Installation von HBase ist ein nicht ganz triviales Unterfangen und setzt grundlegende Kenntnisse von Linux oder anderen POSIX-kompatiblen Betriebssystemen voraus. Damit wurde schon vorweggenommen, dass bisher nur eine Implementierung für Linux bereitsteht. Denn insbesondere im Bereich der Steuerung verwendet HBase ausschließlich Linux Shell-Skripte. Daher ist HBase auf Windows-Systemen nur mit der Kompatibilitätsschicht Cygwin sinnvoll nutzbar. Zur Installation lädt man das aktuellste stabile Release von einem der aufgelisteten Mirrors unter der URL *http://www.apache.org/dyn/closer.cgi/hbase/*. Dann entpackt man das Archiv und wechselt in das Verzeichnis:

```
$ tar -xzf hbase-x.y.z.tar.gz
$ cd ./hbase-x.z.z
```

HBase benötigt zur Ausführung ein installiertes Java-Paket in der Version 6. Damit HBase die zu verwendende Java-Installation finden kann, ist es wichtig, dass man entweder die Umgebungsvariable JAVA_HOME oder in der Konfigurationsdatei *conf/hbase-env.sh* die JAVA_HOME-Variable auf das lokale Installationsverzeichnis von Java setzt.

Die Programme zur Steuerung und Konfiguration von HBase befinden sich im Unterverzeichnis *bin*. Folgende Eingabe liefert eine Übersicht der verfügbaren Optionen des zentralen Shell-Skripts zur Steuerung von HBase:

Listing 3.1.2 HBase: Verfügbare Kommandozeilenoptionen des Shell-Skripts von HBase

```
$ ./bin/hbase
Usage: hbase <command>
Where <command> is one of:
    shell      run the HBase shell
    zkcli      run the ZooKeeper shell
    master     run an HBase HMaster node
    regionserver       run an HBase HRegionServer node
    zookeeper  run a ZooKeeper server
    rest       run an HBase REST server
    thrift     run an HBase Thrift server
    migrate    upgrade an hbase.rootdir
    hbck       run the hbase 'fsck' tool
    classpath  dump hbase CLASSPATH
or
    CLASSNAME  run the class named CLASSNAME
Most commands print help when invoked w/o parameters.
```

HBase ist im Auslieferungszustand als lokaler *Single-Node-Cluster* vorkonfiguriert. In diesem Modus speichert HBase seine persistenten Daten im Ordner *tmp* im Stammverzeichnis des lokalen Dateisystems. Dadurch ist es für erste Gehversuche möglich, HBase sofort nach der Installation auf dem eigenen Rechner ohne verteiltes Dateisystem zu starten. Zum Starten benutzt man einfach das mitgelieferte Shell-Skript:

```
$ bin/start-hbase.sh
```

Das Shell-Skript startet zwei Prozesse: eine lokale HBase-Instanz, die Daten persistent auf die lokale Platte speichert, und eine Instanz von ZooKeeper, dem Koordinationsdienst für HBase-Cluster.

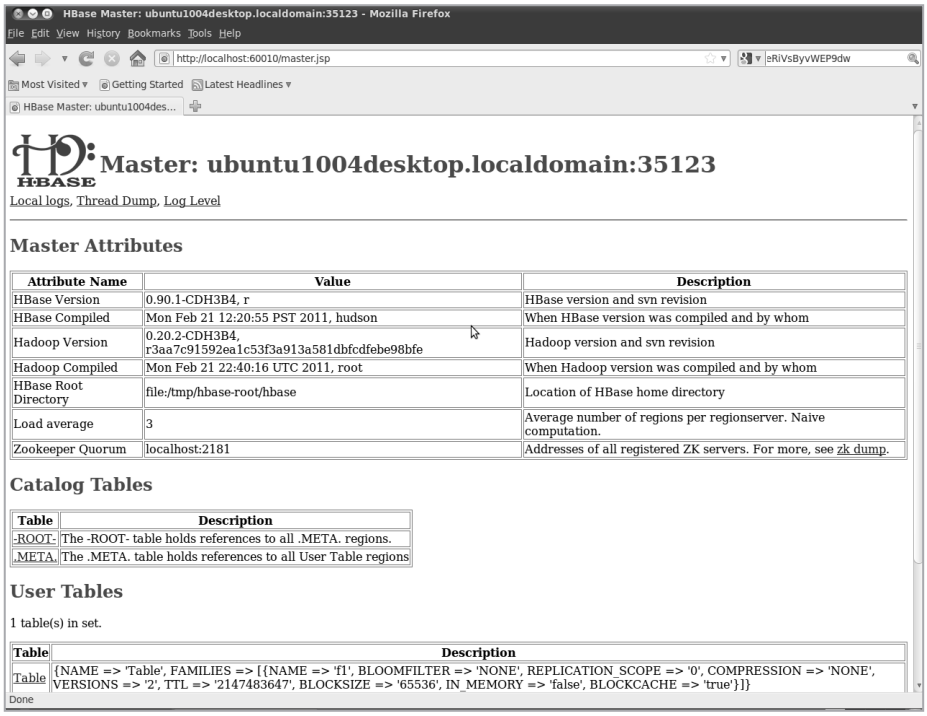

Abbildung 3.1.1 HBase: Screenshot des Web-Interfaces

Um zu prüfen, ob die lokale HBase-Instanz ordnungsgemäß läuft, ruft man im Browser das Web-Interface von HBase auf dem Port 60010 auf (*http://localhost:60010*, siehe Abbildung 3.1.1).

Zum Stoppen des HBase-Clusters steht ein weiteres Shell-Skript bereit:

```
$ bin/stop-hbase.sh
```

Der beschriebene *Single-Node-Cluster* ist vor allem für Testzwecke geeignet. In einem Produktivsystem will man jedoch meist die Vorteile der horizontalen Verteilung von HBase nutzen, indem man einen *Multi-Node-Cluster* aufsetzt. Näheres dazu wird im Abschnitt 3.1.7 zur Skalierung erläutert.

3.1.5 CRUD-Operationen

Erstellung eines Schemas

Ehe man Daten in HBase speichern kann, ist es wie bereits beschrieben notwendig, eine Tabelle mit Schema anzulegen. Dies kann über die integrierte JRuby-Shell oder die verfügbare Java-API geschehen. An dieser Stelle werden beide Wege demonstriert.

Für den Weg über JRuby startet man zunächst die Shell durch folgende Eingabe:

```
$ bin/hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Version: 0.90.1-CDH3B4, r, Mon Feb 21 12:20:55 PST 2011
hbase(main):001:0>
```

Nun kann ein Schema erstellt werden. Ein Schema für eine Tabelle besteht aus deren Namen und der Deklaration von zugehörigen Spaltenfamilien. Für jede Spaltenfamilie können verschiedene Attribute spezifiziert werden wie beispielsweise die Zahl der zu speichernden Versionen einer Zelle. Die Deklarationen der Spaltenfamilien folgen in der JRuby-Shell als *Dictionaries* nach dem Tabellennamen.

```
hbase(main):002:0> create 'Table', {NAME => 'F1', VERSIONS => 2}
```

Der Befehl *create* erzeugt eine neue Tabelle mit dem Namen „Table“ und eine Spaltenfamilie mit dem Namen „F1“. Alle Zellen einer Spaltenfamilie speichern die letzten zwei Versionen (durch `VERSIONS => 2`). Mit dem Befehl *list* lassen sich alle Tabellen der verbundenen HBase-Instanz anzeigen.

```
hbase(main):002:0> list
Table

1 row(s) in 0.1030 seconds
```

Möchte man dasselbe Ergebnis mit der Java-API erreichen, bindet man die von HBase mitgelieferte hbase-x.x.x.jar und die im HBase-Verzeichnis im Ordner *lib* abgelegten Bibliotheken commons-logging-x.y.z.jar, hadoop-core.jar, log4j-x.y.z.jar und zookeeper.jar über den CLASSPATH ein und verwendet folgenden Code:

Listing 3.1.3 HBase: Erstellen eines Schemas mit der Java-API

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;

public class FirstHBaseClient {
 public static void main(String[] args) throws IOException
 {
    HTableDescriptor table = new HTableDescriptor("Table");
    HColumnDescriptor family = new HColumnDescriptor("F1");
    family.setMaxVersions(2);
    table.addFamily(family);

    Configuration config = HBaseConfiguration.create();
    HBaseAdmin admin = new HBaseAdmin(config);
    admin.createTable(table);
```

```
    HTableDescriptor tables = admin.listTables();
    for (int i=0; i<tables.length; i++)
      System.out.println( tables[i].getNameAsString() );
 }
}
```

Einfügeoperationen mit Java

Zum Einfügen einer neuen Zeile in eine Tabelle erzeugt man ein Objekt der Klasse *Put* aus dem Paket *org.apache.hadoop.hbase.client.* Dem Konstruktor des *Put*-Objekts wird dabei der Schlüssel für die neue Zeile übergeben. Dieser muss zwingend ein Byte-Array sein. Für die Umwandlung einer Vielzahl von Datentypen in ein Byte-Array stellt die Java-API von HBase die statische Utility-Klasse *Bytes* aus dem Paket *org.apache.hadoop.hbase.util* bereit. Diese wird über alle folgenden Beispiele hinweg zur Umwandlung von Strings in Byte-Arrays verwendet. Mit der *add*-Methode werden Zellen in dieser neuen Zeile mit Werten belegt. Dabei sind neben dem zu speichernden Wert der Name der zu verwendenden Spaltenfamilie und der Spaltenname als Parameter anzugeben. Dies muss wiederum als Byte-Arrays erfolgen. Wie oben beschrieben, muss nur die Spaltenfamilie bereits bei der Schemadefinition angelegt worden sein. Neue Spalten können von einem Client jederzeit zu einer bestehenden Spaltenfamilie hinzugefügt werden. Übergibt man die *Put*-Objektinstanz der *add*-Methode einer Objektinstanz der *HTable*-Klasse, wird die neue Zeile in der von dem Objekt repräsentierten Tabelle gespeichert.

Listing 3.1.4 HBase: Einfügeoperation mit der Java-API

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

public class FirstHBaseClient {
 public static void main(String[] args) throws IOException
 {
    Configuration config = HBaseConfiguration.create();

    HTable table = new HTable(config, "Table");

    Put p = new Put(Bytes.toBytes("FirstRowKey"));

    p.add(Bytes.toBytes("F1"), Bytes.toBytes("FirstColumn"),
    Bytes.toBytes("First Value"));

    table.put(p);
 }
}
```

Leseoperationen mit Java

Für den lesenden Zugriff auf eine Zeile erzeugt man ein Objekt der Klass *Get*. Dem Konstruktor übergibt man den Schlüssel der zu lesenden Zeile als Parameter. Für eine Ein-

schränkung des Bereichs wie beispielsweise der Spalten oder der Spaltenfamilie, die man lesen möchte, stehen Methoden wie beispielsweise *addColumn* oder *addFamily* bereit. Um die Ergebnisse aus der Tabelle abzurufen, übergibt man sein erzeugtes *Get*-Objekt der *get*-Methode einer Objektinstanz der Klasse *HTable*. Diese liefert ein Objekt der Klasse *Result* als Ergebnis, welches die abgefragten Daten enthält. Die Klasse *Result* bietet verschiedene Zugriffsmethoden, um auf die Zellen der Spalten einer Zeile zuzugreifen. Die *getValue*-Methode liefert beispielsweise den Inhalt einer Zelle als Byte-Array.

Listing 3.1.5 HBase: Leseoperation mit der Java-API

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

public class FirstHBaseClient {
 public static void main(String[] args) throws IOException
 {
    Configuration config = HBaseConfiguration.create();

    HTable table = new HTable(config, "Table");

    Get get = new Get(Bytes.toBytes("FirstRowKey"));

    Result result = table.get(get);

    byte[] value = result.getValue(
      Bytes.toBytes("F1"), Bytes.toBytes("FirstColumn"));

    System.out.println(Bytes.toString(value));
 }
}
```

Aktualisierungsoperationen mit Java

Die Aktualisierung einer Zeile ist eine unspektakuläre Operation. Die Java-API von HBase sieht dafür dieselbe Vorgehensweise wie beim Einfügen vor. So erzeugt man erst ein *Put*-Objekt und fügt dann per *add*-Methode die gewünschten Änderungen hinzu. Für ein Code-Beispiel sei deshalb auf die Einfügeoperation verwiesen.

Löschoperationen mit Java

Analog zu den vorangegangenen Vorgehensweisen bietet die Java-API von HBase auch für die Löschoperationen eine eigene Klasse, nämlich die *Delete*-Klasse an. Wie bei den anderen Klassen bezieht sich eine Objektinstanz immer auf eine Zeile und benötigt im Konstruktor deren Schlüssel. Es ist aber auch möglich, nur bestimmte Spaltenwerte einer Zeile zu löschen. Durch Aufruf der *delete*-Methode und Übergabe der *Delete*-Objektinstanz als Parameter auf einer Objektinstanz der Klasse *HTable* wird die Löschoperation ausgeführt.

Listing 3.1.6 HBase: Löschoperationen mit der Java-API

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;


public class FirstHBaseClient {
 public static void main(String[] args) throws IOException
 {
    Configuration config = HBaseConfiguration.create();

    HTable table = new HTable(config, "Table");

    Delete del = new Delete(Bytes.toBytes("FirstRowKey"));

    table.delete(del);
 }
}
```

Zugriff über REST

Der Zugriff auf HBase ist auch über eine REST-API möglich. Dazu dient ein im heruntergeladenen Release enthaltenes Java-Servlet namens Stargate. Das Servlet wird in einem integrierten Jetty-Servlet-Container ausgeführt. Um diesen im Vordergrund zu starten, gibt man folgendes Kommando auf der Konsole ein:

```
$ ./bin/hbase rest start -p <port>
```

Möchte man den Jetty-Container als Daemon im Hintergrund ausführen, so verwendet man folgendes Kommando:

```
$ ./bin/hbase-daemon.sh start rest -p <port>
```

Dabei ist bei beiden Kommandos der Schalter `-p` mit der Angabe der Ports <*port*> optional (der Standardport ist 8080). Zum Stoppen des Daemons verwendet man folgendes Kommando:

```
$ ./bin/hbase-daemon.sh stop rest
```

Nun ist Stargate bereit, auf HTTP-Anfragen zu antworten. Zum Test, ob der Daemon korrekt funktioniert, kann man z.B. mit dem Tool curl über die URL *http://localhost:<Port>/<Path>/<Tabellenname>/schema* eine Beschreibung des Schemas der zuvor angelegten Tabelle „Table“ abrufen:

```
$ curl http://localhost:8080/Table/schema
{ NAME=> 'Table', IS_META => 'false', IS_ROOT => 'false', COLUMNS => [ {
    NAME => 'F1', BLOCKSIZE => '65536', BLOOMFILTER => 'NONE', BLOCKCACHE =>
    'true', COMPRESSION => 'NONE', VERSIONS => '2', REPLICATION_SCOPE =>
    '0', TTL => '2147483647', IN_MEMORY => 'false' } ] }
```

Lässt man wie oben die Angabe des *Encodings* im *Http-Header* weg, liefert Stargate die Darstellung des Schemas standardmäßig im JSON-Format (*application/json*). Weitere un-

terstützte Formate sind binär (*application/binary*), XML (*application/xml*) und Googles protobuf (*application/x-protobuf* siehe auch unter *http://code.google.com/p/protobuf/).*

Für lesende und schreibende Zugriffe auf eine HBase-Datenbank mittels Stargate stehen die für REST üblichen HTTP-Operationen GET, POST, PUT und DELETE bereit. Wie man diese für die üblichen CRUD-Operationen verwendet, fasst die folgende Tabelle zusammen:

Tabelle 3.1.1 CRUD-Operationen über die REST-Schnittstelle von HBase

Operation	URL-Schema	HTTP-Request
Einfügen/ Aktualisieren einer Zelle	/<Table>/<RowKey>/ <ColumnFamily>:<Column>	PUT/POST mit dem zu speichernden Wert als Datenpaket im angegebenen Content-Type
Abrufen einer Zelle	/<Table>/<RowKey>/ <ColumnFamily>:<Column>	GET mit dem gewünschten Content-Type (xml, json, protbuf, binary)
Abrufen einer Zeile	/<Table>/<RowKey>	GET mit dem gewünschten Content-Type (xml, json, protbuf)
Löschen einer Zelle	/<Table>/<RowKey>/ <ColumnFamily>:<Column>	DELETE
Löschen einer Zeile	/<Table>/<RowKey>	DELETE

Im folgenden Beispielaufruf wird mit dem Tool curl eine HTTP GET-Operation ausgeführt, um eine über den Schlüssel identifizierte Zeile mit einer URL der Form <*Table*>/<*RowKey*> aus HBase im XML-Format abzurufen:

```
$ curl -H "Accept: text/xml" http://localhost:8080/Table/FirstRowKey
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CellSet>
 <Row key="Rmlyc3RSb3dLZXk=">
    <Cell timestamp="1268772237126"
      Column="RjE6Rmlyc3RDb2x1bW4=">VmFsdWU=</Cell>
 </Row>
</CellSet>
```

Die Antwort repräsentiert die unter dem in der URL angegebenen Schlüssel gespeicherte Zeile. Das Wurzelelement einer solchen XML-Datei ist ein *CellSet*, das ein *Row*-Element mit der abgefragten Zeile enthält. Jede Zelle einer Zeile wird durch ein *Cell*-Element repräsentiert. Das *Column*-Attribut des *Cell*-Elements bezeichnet den Spaltennamen der Zelle und das *timestamp*-Attribut den Zeitstempel der Version der Zelle. Die Werte sind alle Byte-Arrays, die BASE64-enkodiert sind, und müssen beim Parsen des XML in einer Anwendung in den zu verwendenden Datentyp umgewandelt werden.

Für die Nutzung der REST-API in einem Java-Programm stellt HBase die Klasse *Client* im Paket *org.apache.hadoop.hbase.rest.client* bereit. Die Nutzung der REST-API sowie der Aufbau der URLs für alle weiteren Operationen sind im Projekt-Wiki von HBase unter *http://wiki.apache.org/hadoop/Hbase/Stargate* beschrieben.

Zugriff über Thrift

Zu guter Letzt bietet HBase auch einen Zugriff über Thrift, welches ebenfalls ein Open Source-Projekt aus dem Kosmos der Apache Foundation (*http://incubator.apache.org/thrift/*) ist. Thrift ist ein Framework für die Entwicklung von Services, die programmiersprachenübergreifend genutzt werden können. Thrift Services werden in der Meta-Sprache Thrift IDL definiert, und mit einem Generator werden daraus Code-Vorlagen zur Implementierung des Services auf Server- und Client-Seite erzeugt. Aktuell bietet Thrift die Erstellung von Code-Vorlagen in den Sprachen C++, C#, Erlang, Haskell, Java, Objective C/Cocoa, OCaml, Perl, PHP, Python, Ruby und Smalltalk an.

Die aktuelle HBase-Version [0.90.01] enthält eine Implementierung des Servers in Java. Der Thrift-Server zur Bereitstellung der Services kann mit folgendem Kommando bequem über die Shell gestartet werden:

```
$ bin/hbase-daemon.sh start thrift
```

Zum Beenden des Servers verwendet man folgendes Kommando:

```
$ bin/hbase-daemon.sh stop thrift
```

Die Servicebeschreibung *HBase.thrift* zur Generierung eines Clients findet man im Repository von HBase unter der URL *http://svn.apache.org/viewvc/hbase/trunk/src/main/resources/org/apache/hadoop/hbase/thrift/Hbase.thrift*. Ist Thrift auf dem verwendeten System installiert, kann man mit folgendem Kommando aus dieser Servicebeschreibung einen Client erzeugen, wobei [lang] durch das Kürzel für die gewünschte Zielsprache ersetzt wird (java, cpp, rb, py oder perl für Java, C++, Ruby, Python oder Perl):

```
$ thrift-gen [lang] ./Hbase.thrift
```

Die Dokumentation zur Nutzung der Thrift-API von HBase ist im Wiki des Thrift-Projekts unter der URL *http://wiki.apache.org/thrift/FrontPage* zu finden.

3.1.6 Fortgeschrittene Abfragen mit Map/Reduce

Die Möglichkeit, fortgeschrittene Abfragen in einer eigens dafür vorgesehenen Abfragesprache wie SQL zu formulieren, ist in HBase derzeit nicht gegeben. HBase ist ein Datenbanksystem, das von Anfang an auf eine möglichst simple Skalierbarkeit hin entwickelt wurde. Aus diesem Grund sind Funktionen wie eine Abfragesprache und dazugehörige Abfrageoptimierer nicht von oberster Priorität für das Projekt.

Für den Anwender bedeutet dies, dass er für Abfragen, die über das hinausgehen, was sich auf Client-Seite mit der geschickten Komposition von CRUD-Operationen performant lösen lässt, nicht um die Nutzung des integrierten Map/Reduce-Frameworks von HBase herumkommt. Für die steile Lernkurve bei der ersten Verwendung von Map/Reduce wird man mit einer deutlich gesteigerten Performance im Vergleich zu klassischen relationalen Datenbanksystemen belohnt, wenn man Datenmengen im Terabyte- oder Petabyte-Bereich verarbeitet.

Die Basis für das Map/Reduce-Framework von HBase bildet die vom Mutterprojekt Hadoop bereitgestellte Implementierung in der Version 0.20. Die Nutzung dieses Map/Reduce-Frameworks mit HBase erfolgt über eine Reihe von für HBase angepassten Java-Klassen, die im Paket *org.apache.hadoop.hbase.mapreduce* zu finden sind. Damit ist es möglich, HBase als Datenquelle und/oder -senke für Map/Reduce-Jobs zu verwenden.

Die Nutzung dieses Frameworks soll an einem Beispiel in Java demonstriert werden. Im Beispiel soll aus einer Spalte mit Integer-Werten der maximale Wert ermittelt werden. Dazu wird eine eigene Klasse *MaxFinder* implementiert, die die zwei inneren Klassen *MaxFinderMapper* und *MaxFinderReducer* beinhaltet.

MaxFinderMapper erweitert die abstrakte Klasse *TableMapper* und überschreibt die *map*-Methode, die das eigentliche Mapping vornimmt. Die abstrakte Klasse *TableMapper* ist eine generische Klasse, die bei der Erweiterung die Angabe der Klassentypen für den Schlüssel und den Wert der von der Map-Prozedur zurückgelieferten Key-Value-Paare erfordert. Zulässige Klassentypen sind für HBase im Paket *org.apache.hadoop.hbase.io* und allgemein für Hadoop im Paket *org.apache.hadoop.io* zu finden. Im Beispiel wird für den Schlüssel die Klasse *ImmutableBytesWritable* und für den Wert die Klasse *IntWritable* verwendet. Die *map*-Methode implementiert im Beispiel das simple Sammeln der Werte aus der interessierenden Spalte und die Konvertierung des Byte-Arrays in einen Integer. Für den Zugriff auf das Framework steht ein context-Objekt zur Verfügung. Über die write-Methode werden die Key-Value-Paare in der *map*-Methode an das Map/Reduce-Framework weitergereicht, das diese für die Reduce-Sequenz sammelt.

Die Reduce-Sequenz wird über eine Erweiterung der abstrakten Klasse *TableReducer* in der *MaxFinderReducer*-Klasse implementiert. Diese ebenso generische Klasse erfordert bei der Erweiterung die Angabe der Klassentypen für den Schlüssel und den Wert der Key-Value-Paare, die von der Map-Sequenz geliefert werden (wie man im Beispiel sieht, müssen diese mit den für die Map-Sequenz angegebenen Typen übereinstimmen), sowie für den Schlüssel der Ausgabe der Reduce-Sequenz. Die konkreten Schritte der Reduce-Sequenz sind in einer Überschreibung der *reduce*-Methode zu implementieren. Über die Konfigurationsparameter *OutputFormat* und *InputFormat* des Map/Reduce-Jobs lassen sich die Datenquelle und -senke des Jobs festlegen. Ist das *OutputFormat* wie im Beispiel auf die TableOutputFormat-Klasse festgelegt, dann kann dem context-Objekt in der reduce-Methode als Wert ein *Put*- oder *Delete*-Objekt übergeben werden, die dem *Map/Reduce*-Job zugrunde liegende Tabelle ändern. Der zugehörige Schlüssel wird in diesem Fall ignoriert. So wird im Beispiel eine neue Zeile mit dem Schlüssel „max“ angelegt und der Maximalwert in der dem Job übergebenen Spalte gespeichert.

Zur Ausführung des Map/Reduce-Jobs erzeugt man eine Instanz der *Job*-Klasse. Im Beispiel geschieht dies in der Methode *createSubmittableJob*. Dem Konstruktor der *Job*-Klasse übergibt man einen Verweis auf ein *Configuration*-Objekt. Im Beispiel ist dies ein Objekt der Klasse *HBaseConfiguration*, die die allgemeine *Configuration*-Klasse beerbt. Als zweiten Parameter übergibt man dem Konstruktor der *Job*-Klasse einen Namen für den Job. Die Klasse, die den auszuführenden Map/Reduce-Job implementiert, teilt man dem *Job*-Objekt mit der Methode *setJarByClass* mit. Im Beispiel ist dies die Klasse *MapFinder*. Der Beispielcode zeigt anschließend die Erzeugung eines *Scan*-Objekts, einem Zeiger vergleichbar, das

die Werte aus der HBase-Datenbank liefert, die der Map-Sequenz vom Framework übergeben werden. Das *Scan*-Objekt im Beispiel wird mit der *addColumns*-Methode auf die relevante Spalte eingegrenzt. Zur Konfiguration eines Map/Reduce-Jobs für die Verwendung mit HBase als Datenquelle oder -senke steht die Utility-Klasse *TableMapReduceUtil* bereit. Im Beispiel wird diese verwendet, um dem *Job*-Objekt mit der Methode *initTableMapperJob* die *Mapper*-Klasse, den Tabellennamen, das *Scan*-Objekt und die Typen von Schlüssel und Wert der Key-Value-Paare, die die Map-Sequenz ausgibt, mitzuteilen. Zur Einstellung der Reduce-Sequenz wird die Methode *initTableReduceJob* benutzt, die wiederum den Tabellennamen, die Klasse für die Reduce-Sequenz und die Klasse für die Partitionierung der Reduce-Jobs auf die Regionen des HBase-Clusters einstellt. Im Beispiel wird für die Partitionierung, die von HBase als Standard gelieferte Klasse *HRegionPartitioner* verwendet. Zur Ausführung des Jobs ruft man die *waitForCompletion*-Methode auf, die bei Erfolg als Rückgabewert *true* liefert.

Dieses simple Beispiel macht deutlich, welchen Aufwand die Entwicklung selbst einfacher Aufgaben mit dem Map/Reduce-Framework von HBase erfordert. In einem konkreten Anwendungsszenario ist es daher sinnvoll, eine eigene Bibliothek mit Wrapper-Klassen für wiederkehrende Patterns zu implementieren.

Listing 3.1.7 HBase: Beispielimplementierung eines Map/Reduce-Jobs in Java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.filter.FirstKeyOnlyFilter;
import org.apache.hadoop.hbase.io.*;
import org.apache.hadoop.hbase.mapreduce.*;
import org.apache.hadoop.hbase.util.*;
import org.apache.hadoop.hbase.HBaseConfiguration;

public class MaxFinder{

 private static String Spaltenfamilie;
 private static String column;

 public static void main(String[] args) throws Exception {
    Configuration conf = HBaseConfiguration.create();
    String[] otherArgs = new
      GenericOptionsParser(args).getRemainingArgs();
    if (otherArgs.length < 3) {
      System.err.println("ERROR: Wrong number of parameters: " +
        args.length);
      System.err.println("Usage: MaxFinder <tablename> <Spaltenfamilie>
        <column>");
      System.exit(-1);
    }
 Job job = createSubmittableJob(conf, otherArgs);
 System.exit(job.waitForCompletion(true) ? 0 : 1);
 }

 public static Job createSubmittableJob(Configuration conf,
    String[] args) throws IOException {
```

```
      String tableName = args[0];
      Spaltenfamilie = args[1];
      column = args[2];
      Job job = new Job(conf, "maxfinder_" + tableName);
      job.setJarByClass(MaxFinder.class);
      Scan scan = new Scan();
      scan.addColumns(Spaltenfamilie + ":" + column);
      TableMapReduceUtil.initTableMapperJob(tableName, scan,
        MaxFinderMapper.class, ImmutableBytesWritable.class,
        IntWritable.class, job);
      TableMapReduceUtil.initTableReducerJob(tableName,
      MaxFinderReducer.class , job, HRegionPartitioner.class);
      return job;
  }

  static class MaxFinderMapper
     extends TableMapper<ImmutableBytesWritable, IntWritable> {

     @Override
     protected void map(ImmutableBytesWritable row, Result values,
       Context context)
       throws IOException, InterruptedException {
         int value = Bytes.toInt(values.value());
         String key = "max";
         context.write(new ImmutableBytesWritable(Bytes.toBytes(key)),
          new IntWritable(value));
     }
  }

  static class MaxFinderReducer
     extends TableReducer<ImmutableBytesWritable, IntWritable ,
       ImmutableBytesWritable>
     {
       @Override
       protected void reduce(ImmutableBytesWritable key,
         Iterable<IntWritable> values, Context context)
           throws IOException ,InterruptedException {
         int max = 0;
         for(IntWritable value : values ) {
           if(value.get() > max)
           max = value.get();
         }
         Put put = new Put(Bytes.toBytes("max"));
          put.add(Bytes.toBytes(Spaltenfamilie),
         Bytes.toBytes(column),Bytes.toBytes(max));
          context.write(key, put);
       }
     }
}
```

3.1.7 Skalierung und Konfiguration

Der Kern von HBase ist so implementiert, dass das System unkompliziert skaliert. Dazu ist ein HBase-Cluster nach dem 1-Master-/N-Slaves-Prinzip aufgebaut. Ein zentraler Master, der auch in der Terminologie von HBase so genannt wird, überwacht den Cluster mehrerer

Slaves, die als RegionServer bezeichnet werden. Die RegionServer stellen den Zugriff auf die Daten bereit und übernehmen die persistente Speicherung. Bei Bedarf braucht man zum Skalieren für HBase nur einen zusätzlichen RegionServer einzurichten. Die Verteilung einer in Regions aufgeteilten Tabelle erfolgt dann automatisch durch den Master, was Entwickler und Administratoren entlastet.

Im Ausgangszustand umfasst eine Tabelle nur eine Region. Überschreitet die Datenmenge einer Region einen festgelegten Grenzwert, wird diese automatisch in zwei neue Regions etwa gleicher Größe aufgeteilt. Diese werden auf die im Cluster verfügbaren RegionServer verteilt. Die Verwaltung der Zuteilung von Regions zu den RegionServern übernimmt der Master. Ein RegionServer hält alle Datensätze von einem Start- bis zu einem bestimmten Endschlüssel. Die Datensätze werden nach dem Schlüssel sortiert gespeichert. So kann der RegionServer, der eine Zeile speichert, über deren Schlüssel bestimmt werden. Für die allgemeine Synchronisation, Konfiguration und Verfügbarkeitskontrolle des Clusters greift HBase auf einen ZooKeeper-Cluster zurück (siehe *http://hadoop.apache.org/zookeeper/*).

Den dargestellten Aufbau eines HBase-Clusters verdeutlicht die Abbildung 3.1.2

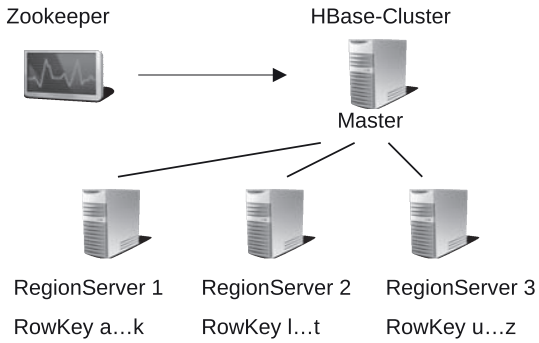

Abbildung 3.1.2
HBase: Schematischer Aufbau eines Clusters

Zur Konfiguration des HBase-Clusters dienen die Dateien *regionservers* und *hbase-site.xml* im *conf*-Verzeichnis der HBase-Installation auf dem Master. Die Datei *regionservers* enthält pro Zeile den Name eines RegionServers. Für die Erweiterung des Clusters um einen neuen RegionServer muss dessen IP-Adresse nur in dieser Datei eingetragen werden. In der XML-Datei *hbase-site.xml* werden allgemeine Einstellungen für den Cluster festgelegt. Beispiele wichtiger Einstellungsoptionen sind in Tabelle 3.1.2 beschrieben. Die Default-Einstellungen von HBase sind dabei jeweils mit angegeben.

Tabelle 3.2.2 Wichtige Konfigurationsoptionen für einen HBase-Cluster

Konfiguration	Beschreibung
hbase.rootdir Default: file:///tmp/hbase-${user.name}/hbase	Verzeichnis für die persistente Datenspeicherung der RegionServer, z.B. hdfs://HDFS_NAMENODE:PORT/HBASE_ROOTDIR
hbase.master.port Default: 60000	Port für den Master Node

Konfiguration	Beschreibung
hbase.regionserver.port Default: 60030	Port für den lokalen RegionServer
hbase.cluster.distributed Default: false	Schalter für den verteilten Modus (true) und pseudo-verteilten bzw. Einzelmodus (false)
hbase.client.write.buffer Default: 2097152 (2MB)	Größe des Schreibpuffers in Bytes. Je größer der Puffer, desto höher der Hauptspeicherbedarf auf Client- und Serverseite. Um den Verbrauch auf Serverseite abzuschätzen, muss die Größe des hbase.client.write.buffer mit der Anzahl hbase.regionserver.handler.count multipliziert werden.
hbase.regionserver.handler.count Default: 10	Zahl der auf einem RegionServer laufenden RPC-Server-Instanzen. Diese bestimmt die Zahl der gleichzeitig bearbeitbaren Requests.
hbase.client.retries.number Default:10	Maximale Zahl der Versuche, um eine Operation wie beispielsweise das Auslesen eines Zellenwertes oder der Beginn des Updates einer Zeile auszuführen.
hbase.hregion.max.filesize Default: 268435456 (256MB)	Maximale Größe einer HStore-Datei für eine Spaltenfamilie. Nach dem Überschreiten dieser Grenze wird die Region aufgeteilt.
hbase.client.keyvalue.maxsize Default: 10485760 (10MB)	Maximal erlaubte Größe eines Key-Value-Paares (Summe aus Key und Value). Diese sollte so gewählt werden, dass die Größe der Region durch diese teilbar ist.

3.1.8 Replikation

In der aktuellen Version 0.90.1 von HBase ist der erste experimentelle Entwurf eines integrierten Replikationsmechanismus enthalten. Dieser arbeitet auf der Ebene des gesamten HBase-Clusters. Die Replikation soll es in Zukunft ermöglichen, für einen HBase-Cluster einen oder mehrere replizierende Cluster aufzubauen. Damit ist beispielsweise ein Szenario denkbar, in dem der produktive HBase-Cluster einer Web-Applikation in einen zweiten HBase-Cluster repliziert, welcher zur Analyse der Daten mittels Map/Reduce verwendet wird. Da die integrierte Replikation sich noch in einem experimentellen Entwicklungsstadium befindet, ist der Einsatz in einer produktiven Umgebung nur mit erheblichen Einschränkungen möglich.

Die Replikation arbeitet nach einer Master-Push-Architektur. Das bedeutet, es gibt einen Master-Cluster, der seine Daten in den oder die Slave-Cluster repliziert (Push). Dabei ist es nicht notwendig, dass der Slave-Cluster dieselbe Knotenstruktur, d.h. dieselbe Zahl an RegionServer, aufweist. Zum aktuellen Entwicklungsstand von HBase (0.90.1) ist die Replikation auf einen Slave-Cluster pro Master-Cluster beschränkt. Diese Einschränkung soll aber in einem zukünftigen Release aufgehoben werden.

Technisch sind es die RegionServer des Master-Clusters selbst, die die Replikation durchführen. Dazu wird für jeden registrierten Slave-Cluster ein eigenes Log geführt, in dem alle für die Replikation bestimmten Datenänderungen des primären Logs des RegionSer-

vers gepuffert werden. Bei einem festgelegten Schwellenwert der Größe des Logs für die Replikation versucht der RegionServer, nach einem zufallsbasierten Verfahren einen RegionServer des zugehörigen Slave-Clusters zu erreichen und diesem die zu replizierenden Daten zu übermitteln. Die Verteilung der Daten im Slave-Cluster übernimmt der kontaktierte RegionServer.

Möchte man die noch experimentelle Replikation testen, sind zwei HBase-Cluster aufzubauen. Beide Cluster müssen dieselben Releases der HBase- und Hadoop-Bibliotheken verwenden. Alle Rechner der beiden Cluster müssen sich über das Netzwerk erreichen können. Alle Tabellen und Spaltenfamilien, die repliziert werden sollen, sollten in beiden Clustern unter dem exakt selben Namen existieren. Darüber hinaus ist es notwendig, für beide Cluster manuell einen ZooKeeper-Cluster einzurichten und den integrierten ZooKeeper-Cluster von HBase zu deaktivieren. Repliziert werden dann nur die Spaltenfamilien, die dafür konfiguriert sind. Dazu setzt man zum aktuellen Entwicklungsstand die Property *Replication_Scope* auf den Wert 1, was die Replikation für diese Spaltenfamilie aktiviert. Der Standardwert ist 0 für keine Replikation. Die möglichen Werte dieser Property sollen sich aber in zukünftigen Releases noch ändern. Informationen zur Replikation findet man in den Javadocs von HBase in der Beschreibung des Packages *org.apache.hadoop.hbase.replication* unter der URL *http://hbase.apache.org/apidocs/index.html?overview-summary.html.*

Möchte man die integrierte Replikation auf Ebene des Clusters noch nicht einsetzen, so bleibt noch die Möglichkeit der Replikation auf Dateiebene über das vom Mutterprojekt Apache Hadoop bereitgestellte verteilte Dateisystem HDFS. Die RegionServer eines HBase-Clusters speichern die Daten persistent in einem HDFS-Dateisystem auf einem Hadoop-Cluster. Dieser Cluster erledigt dann die Replikation der im Dateisystem gespeicherten HBase-Datenbankdateien. Dadurch entfällt für einen HBase-Cluster der Overhead für die Verwaltung der Replikation. Dies wird durch den Hadoop-Cluster erledigt. Die Beschreibung der Replikation eines Hadoop-Clusters und des verteilten Dateisystems HDFS ist in der Dokumentation des Hadoop-Projekts unter der URL *http://hadoop.apache.org/hdfs/* zu finden.

3.1.9 Bewertung

HBase mit seinem Stack an Hadoop-Komponenten (HDFS, Map/Reduce, ZooKeeper) ist ein Schwergewicht unter den NoSQL-Datenbanken. Dieses Schwergewicht verlangt im praktischen Einsatz auch nach einer ihm ebenbürtigen Aufgabe. Dies sind Einsatzbereiche, in denen wirklich große Datenmengen im oberen Tera- oder Petabyte-Bereich mit Zeilenzahlen im hohen Millionenbereich oder darüber, ständigem Wachstum der Datenmenge und einem wahlfreien Zugriff auf diese Daten verarbeitet werden sollen. Wenn darüber hinaus die Anforderung des zeitkritischen Ausführens von Abfragen oder Analysen der Daten hinzukommt, sollte HBase mit integriertem Map/Reduce das System der Wahl sein.

Vorteile

- HBase skaliert nativ durch simples Hinzufügen eines RegionServers. Dadurch lässt sich auf handelsüblicher Standardhardware kostengünstig ein leistungsstarkes verteiltes Datenbanksystem aufsetzen.
- HBase nutzt mit dem integrierten Map/Reduce-Framework die volle Leistung eines verteilten Systems durch verteiltes Rechnen von erweiterten Abfragen aus.
- Das HBase-Projekt und das Hadoop-Mutterprojekt haben eine starke und aktive Community, die von renommierten Organisationen gesponsert wird.
- HBase bietet mit einer Java-API, REST, Thrift und Jython vielfältige Schnittstellen zur Anbindung an eigene Anwendungen an. So kann HBase in unterschiedlichsten Anwendungskontexten als Datenbanksystem eingesetzt werden.

Nachteile

- Das Aufsetzen, Optimieren und Warten eines HBase-Clusters mit dazugehörigem ZooKeeper und Hadoop-Cluster ist sehr komplex und aufwendig.
- Die Erstellung erweiterter Abfragen, die in SQL leicht von der Hand gehen, ist mit Map/Reduce etwas aufwendiger.
- Die integrierte Replikation auf Ebene des Datenbank-Clusters ist derzeit noch in einem experimentellen Entwicklungsstadium.

Da HBase Googles BigTable nachbildet, ist der offensichtlich typische Anwendungsfall eine Suchmaschine, die das Internet oder Teile davon indiziert. Aber auch andere datenintensive Webanwendungen wie Googles GMail sind typische Beispiele für ein ideales Einsatzgebiet von HBase.

Schlechter ist HBase geeignet, wenn keine wirklich großen Datenmengen verarbeitet und gespeichert werden müssen. Für richtig große Daten gibt es aber derzeit kein besser in der Praxis erprobtes System als HBase.

Links & Literatur:

Homepage: *http://hbase.apache.org/*

Tom White: Hadoop: The Definitive Guide, O'Reilly, 2009

Blogpost von Jim Wilson zu HBase:
http://jimbojw.com/wiki/index.php?title=Understanding_Hbase_and_BigTable

Blogpost von Lars George zu HBase-Map/Reduce:
http://www.larsgeorge.com/2009/05/hbase-mapreduce-101-part-i.html

Blogpost von Lars George zu HBase-Architektur:
http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

Blogpost von Cosmin Lehene zu HBase:
http://hstack.org/why-were-using-hbase-part-1/

3.2 Cassandra

Cassandra wurde von Facebook entwickelt und 2008 als Open Source freigegeben. Es wurde von der Apache Software Foundation Anfang 2010 als Top-Level-Projekt aufgenommen. Wie auch bei einigen anderen NoSQL-Ansätzen (z.B. HBase) war Googles BigTable hier Vorbild. Cassandra hebt sich aber von diesen Ansätzen durch seinen hybriden Ansatz ab: Cassandra enthält sowohl Key/Value-Eigenschaften als auch eine relativ flexible Schemaunterstützung. Damit will Cassandra einerseits eine größtmögliche Flexibilität und Skalierbarkeit und andererseits auch eine durch SQL-Datenbanken vertraute Schemasicherheit bieten.

Ein weiteres Schlüsselmerkmal von Cassandra ist die verteilte Architektur, die eine absolut notwendige Anforderung im Kontext von Facebook und anderen sozialen Netzwerken darstellt, die mit Millionen von Nutzern hohe Datenaufkommen verwalten. Vor der Entwicklung von Cassandra stand Facebook vor der Aufgabe, die Nachrichten der Nutzer untereinander zu speichern. Der Versuch, diese Anforderung mit dem Ansatz der vertikalen Skalierung – die Performance eines Rechenknotens immer weiter zu verbessern – zu lösen, wurde bei der Entwicklung von Cassandra zugunsten der horizontalen Skalierung mit vielen Knoten im Cluster aufgegeben. Der Preis für die hiermit gewonnene Verfügbarkeit ist die absolute Konsistenz der Daten. Cassandra verwendet die bei vielen NoSQL-Systemen propagierte Eventual Consistency, d.h. die Daten werden nur zu einem bestimmten Zeitpunkt konsistent sein. Wann dieser Zeitpunkt sein wird, kann durch später vorgestellte Parameter konfiguriert werden.

Steckbrief

Webadresse:	http://cassandra.apache.org
Kategorie:	Wide Column Store / Extensible Record Stores
API:	Thrift (Support für viele Sprachen, darunter Ruby, Java, Python, C#, PHP, C++, Erlang etc.)
Protokoll:	RPC
Geschrieben in:	vollständig in Java
Concurrency:	Atomic Locking , Optimistic Concurrency Control
Replikation:	über Hadoop HDFS
Skalierung:	unterstützt durch Hinzufügen neuer Knoten
Lizenz:	Apache License, Version 2.0

3.2.1 Allgemeines

Cassandra wurde 2007 von den Facebook-Mitarbeitern Prashant Malik und Avinash Lakshman entwickelt. Lakshman war auch einer der Autoren von Amazon Dynamo (siehe Abschnitt 8.3.1). Die Herausforderung bestand darin, Indizes (genauer Reverse Indices) von Nutzernachrichten effizient zu speichern und eine performante Suche innerhalb be-

stimmter Service Level Agreements und Kostenparameter zu ermöglichen. Hohe Verfügbarkeit und Skalierbarkeit wurden höher bewertet als sofortige Konsistenz, daher bot sich eine verteilte Datenbanklösung mit relativ dynamischem Datenschema zur Realisierung an. Eine Verteilung der Daten auf mehrere Datenbankknoten verhindert einen *Single Point of Failure* und erfüllt die Anforderung einer „Zero-Downtime", also die möglichst ständige Erreichbarkeit der Anwendung bei ständiger Weiterentwicklung. Letzteres ist ohne ein dynamisches, flexibles Datenschema deutlich schwieriger zu erreichen. Man stelle sich z.B. Migrationsszenarien vor wie „Füge eine Spalte in einer relationalen Tabelle mit Millionen von Einträgen hinzu, bei konstantem Antwortverhalten des DBMS".

Die Flexibilität des Cassandra-Schemas ist im Vergleich zu anderen NoSQL-Vertretern wie z.B. den Document Stores eingeschränkt, was die grobe, anwendungsdefinierte Datenstruktur betrifft. Als aus Anwendungssicht oberste Struktureinheit dient der *Keyspace* (vergleichbar zur Datenbank in RDBMS), der meist einer Anwendung zugeordnet ist. Darunter können *Column Families* definiert werden, ähnlich den Tabellen einer relationalen Datenbank. Diese wiederum werden mit *Columns* und *SuperColumns* (dazu später mehr) gefüllt, in denen sich die eigentlichen Daten befinden. Bei der Definition dieser Datenstrukturen hat Cassandra seit dem Erscheinen der ersten Auflage dieses Buches einen großen Schritt nach vorn gemacht: Seit Version 0.7 können (eigentlich: müssen) alle Elemente der Datenstruktur dynamisch angelegt werden. Das früher notwendige Ändern von Konfigurationsdateien und der daraus resultierende Neustart des Clusters entfallen.

3.2.2 Installation

Bevor wir im Folgenden auf Cassandras Datenmodell eingehen, soll hier kurz ein Abschnitt zur Installation der Datenbank eingeschoben werden. Auf diese Art und Weise können interessierte Leser dann die Beispiele im nächsten Abschnitt direkt nachvollziehen. Wer nur an einem kurzen Überblick über die Hintergründe von Cassandra interessiert ist, ohne eigene Experimente unternehmen zu wollen, kann den folgenden Abschnitt getrost überspringen.

Cassandra ist in Java implementiert und setzt eine Java Virtual Machine Version 1.6 voraus. Ist die Umgebungsvariable JAVA_HOME gesetzt und die JVM im PATH vorhanden, beschränkt sich die Installation von Cassandra auf das Herunterladen des aktuellen Releases (aktuell 0.7.5, *http://cassandra.apache.org/download/)* und Entpacken in ein Verzeichnis:

```
$ tar -xzf apache-cassandra-0.7.5-bin.tar.gz
```

Die vor Version 0.7 wichtige *storage-conf.xml* ist nun bedeutungslos (und wird folgerichtig auch ignoriert). Wie bereits erwähnt, werden die Teile des Datenmodells dynamisch angelegt. Die statische Konfiguration in der Datei *conf/cassandra.yaml* ist daher nur noch für Dinge wie Datenbankverzeichnisse zuständig. Hier lauert auch die erste (und meist einzige) größere Stolperfalle bei der Installation: Nach dem Entpacken zeigen die Variablen `data_file_directories`, `commitlog_directory` und `saved_caches_directory` nach `/var/lib/cassandra/*`, wo man im Normalfall als Nutzer keine Schreibrechte

hat. Will man also die Software nicht gleich so im System verankern, dann müssen diese Konfigurationseinträge entsprechend angepasst werden. Danach kann das System einfach mittels des vorgesehenen Skriptes gestartet werden (der Parameter `-f` führt dazu, dass Cassandra im Vordergrund bleibt und Logmeldungen auf die Konsole ausgibt):

```
$ cd apache-cassandra-0.7.5
```

Nach einer ganzen Reihe von Meldungen wird sich Cassandra bereit zur Arbeit melden:

```
INFO 23:43:57,685 Listening for thrift clients...
```

Beim Starten beschwert sich das System unter Umständen darüber, dass es die Datei */var/log/cassandra/system.log* nicht findet. Dieses Problem spielt zwar für erste Experimente mit Logging auf der Konsole keine Rolle, kann aber trotzdem leicht in conf/log4j-server.properties beseitigt werden. Dort werden – wie der Name schon andeutet – die Logausgaben des Servers konfiguriert.

Cassandra nimmt Anfragen via Apache Thrift entgegen. Dieses programmiersprachenunabhängige Interface bietet die Möglichkeit, Services und die dabei zwischen Client und Server ausgetauschten Daten zu definieren und mit einem Codegenerator und Compiler die für die Kommunikation notwendigen Serialisierungs- und Transportmechanismen zu erzeugen. Thrift stellt eine Anbindung für Java, C#, C++, Python und viele andere zur Verfügung. Für weitere Information in Bezug auf das System sei auf die Thrift-Homepage von Apache verwiesen (*http://incubator.apache.org/thrift*).

Über Thrift greift auch das Cassandra Command Line Interface auf den Server zu. Dieses unter *bin/cassandra-cli* zu findende Programm eignet sich zum Management des Datenmodells eines Clusters sowie für erste Experimente mit Cassandra.

3.2.3 Datenmodell

Das Datenmodell lässt sich im Detail am besten anhand eines Beispiels erläutern. Die hypothetische Anforderung lautet: Es ist die Datenstruktur einer Facebook-Anwendung zu modellieren, genauer gesagt eines „Social Games“. Der Begriff „Social Games“ spiegelt die Einbettung des Spieles innerhalb eines sozialen Netzwerks wie Facebook wider und hat – wie später zu sehen ist – direkte Auswirkungen auf die Datenmodellierung. Bei diesem Spiel soll es sich um eine Art „Wild West“-Spiel handeln, bei dem die Nutzer des sozialen Netzwerks Rollen wie „Outlaw“, „Sheriff“, „Cowboy“ etc. annehmen und miteinander agieren können.

Wie schon erwähnt, ist die oberste Struktureinheit des Datenmodells – der *Keyspace* – in etwa vergleichbar mit der Datenbank in relationalen Systemen. In unserem Beispiel sei der Keyspace „WildWest“ derjenige, der alle anwendungsrelevanten Daten kapselt. Die darin enthaltenden *Column Families* „Users“, „Horses“ und „Weapons“ erfassen die statische Klassifikation der im Modell verwendeten Daten. Da es sich hierbei um ein Spiel handelt, das auf einer Plattform mit ausgeprägter Nutzerinteraktion spielbar sein soll, benötigt man zusätzlich Datenstrukturen, die die Interaktion der Spieler protokollieren, beispielsweise Duelle untereinander und die Benachrichtigungen der Mitspieler darüber.

Die notwendigen Strukturen werden mithilfe von *cassandra-cli* dynamisch angelegt[1]. Das Command Line Interface stellt ein eigenes Prompt zur Verfügung, in das die verschiedenen Befehle eingegeben werden können. Außerdem können vergleichbar zu vielen SQL-Systemen die Befehle in einer Textdatei niedergelegt und dann als Batch an *cassandra-cli* übergeben werden. Eine erste Version der Konfiguration sieht wie folgt aus:

Listing 3.2.1 Die erste Version der datamodel.txt

```
create keyspace WildWest;
use WildWest;
create column family Users;
create column family Horses;
create column family Weapons;
create column family Duels;
create column family Messages;
```

Die so gefüllte Datei kann nun an *cassandra-cli* übergeben werden:

```
    $ bin/cassandra-cli -h localhost -f datamodel.txt
```

Mit diesem Kommando wird *cassandra-cli* angewiesen, sich auf den auf *localhost* erreichbaren Cluster-Node zu verbinden und die in *datamodel.txt* abgelegten Befehle abzuarbeiten. Die Datei an sich ist verhältnismäßig einfach aufgebaut: Zuerst wird der Keyspace definiert, und dann wird sich an ihm angemeldet (`use WildWest`). Alle darauf folgenden Befehle werden im Kontext von WildWest ausgeführt; die einzelnen Column Families landen alle darin.

Eine Column Family enthält eine Reihe von Rows, vergleichbar mit einer Tabelle in einem relationalen Datenbanksystem. Jede Row wird über einen eindeutigen Schlüssel identifiziert und besteht wiederum aus einer Sammlung von Columns. Im einfachsten Fall sind diese 3-Tupel aus Name, Wert und Zeitstempel. Der Name dient Cassandra hierbei zum schnellen Auffinden einer Column über einen automatisch erzeugten Index. Listing 3.2.2 zeigt einen möglichen Zustand des „WildWest“-Keyspaces nach dem Einfügen einiger Daten.

Bisher haben an „WildWest“ die beiden Spieler Jim und Joe teilgenommen, Letzterer auch „Little Joe“ genannt. Während Jim ein Pferd namens Kelly besitzt, trägt Joe einen Colt. Dadurch konnte er ein Duell (`high_noon`) gegen Jim gewinnen, was er diesem per Message mitteilte (`high_noon_1`).

Listing 3.2.2 Cassandra-Zustand nach ersten Inserts

```
Keyspace: WildWest
 Users:
   Jim = { reactivity : "2"}
   Joe = { reactivity : "9", nick_name : "Little Joe" }
 Horses:
   Kelly = { color : "white", owner : "Jim" }
```

[1] Grundsätzlich stellt Cassandra Funktionalität bereit, um ein Datenmodell aus der Datei *cassandra.yaml* im *conf*-Verzeichnis einzulesen. Da dies hauptsächlich für das Upgrade existierender Cluster gedacht ist, soll hier nicht weiter darauf eingegangen werden. Interessierte Leser seien auf die FAQ unter http://wiki.apache.org/cassandra/FAQ#no_keyspaces verwiesen.

```
Weapons:
  Colt = { ammo : "15", owner : "Joe" }
Duels:
  high_noon = { player_1 : "Jim", player_2 : "Joe", winner : "Joe" }
Messages:
  high_noon_1 = { from : "Joe", to : "Jim", text : "Looooser!" }
```

Möchte man in einer CF mehrere zueinander gehörende Daten unterbringen, bietet es sich an, diese stärker zu strukturieren, als es der einfache Tupel-Ansatz ermöglicht. Cassandra stellt zu diesem Zweck sogenannte SuperColumns zur Verfügung. Eine SuperColumn ist ein 2-Tupel aus Name und Wert[2]. Der Wert ist hier statt eines einfachen Basistyps wie z.B. String eine Row, also eine Sammlung von Columns. SuperColumns fügen so eine weitere Strukturierungebene in das Cassandra-Datenmodell ein.

So könnten beispielsweise alle Informationen zu einem Duell (Teilnehmer, Austragungsort ...) in einer SuperColumn erfasst werden. Dazu muss Duels als SuperColumn definiert werden[3]:

```
create column family Duels with column_type = 'Super';
```

Column Families lassen sich über die hinter „with“ angegebenen Eigenschaften umfassend konfigurieren[4]. Die hier gesetzte Eigenschaft `column_type` kennzeichnet Duels als eine Sammlung von SuperColumns. Welche Unterspalten „Duels“ enthält, wird nicht konfiguriert, sondern zur Laufzeit entschieden. Das obige Duell zwischen Jim und Joe kann nun durch den Austragungsort erweitert werden: In diesem Spiel wird nun jeder Austragungsort per x- und y-Koordinate auf einer Karte und einer Beschreibung identifiziert. Wie der Zustand nach dem neuen Einfügen dieses Duells in Cassandra aussieht, zeigt Listing 3.2.3.

Listing 3.2.3 Duels als SuperColumn

```
Keyspace: WildWest
 …
 Duels:
    high_noon = {
        result : { player_1 : "Jim", player_2 : "Joe", winner : "Joe" }
        position : { x : "13", "y" : "21", desc : "Main Street" }
    }
 …
```

Was die Modellierung der Daten für die Anwendung betrifft, ist das Beispiel komplett. Allerdings fehlt eine für Cassandra notwendige Information: Wie werden die Einträge einer Column Family bei der Sortierung miteinander verglichen? Generell werden alle Columns beim Einfügen anhand ihres Namens sortiert. Die Daten sind somit stets sortiert und zur Laufzeit schnell aufzufinden. Um Cassandra mitzuteilen, welche Art der Sortierung anzu-

2 Cassandra verzichtet in SuperColumns auf das Führen eines Zeitstempels.

3 Ist Duels bereits definiert, so muss sie vorher mit `drop column family Duels;` entfernt werden.

4 Die möglichen Konfigurationsparameter können der guten integrierten Hilfe von *cassandra-cli* entnommen werden. Einfach den betreffenden Befehl (in unserem Fall `create column family`) mit `help` davor eingeben.

wenden ist, existiert die Eigenschaft `comparator` einer Column, die folgende Werte annehmen kann:

- **AsciiType**: Sortierung nach dem Byte-Wert des Namens; dabei findet eine Prüfung statt, ob dieser als US-ASCII interpretiert werden kann.
- **BytesType**: Sortierung nach Byte-Wert des Namens
- **UTF8Type**: Sortierung nach dem als UTF-8-Zeichenkette interpretierten Namen
- **LexicalUUIDType**: 128bit UUID, der Byte-Wert wird lexikalisch verglichen.
- **TimeUUIDType**: 128bit UUID, der Zeitstempel des Namens wird verglichen.
- **LongType**: Werte des Namens werden als 64bit Long-Werte interpretiert.

Eine Sonderstellung nimmt hierbei die Option TimeUUIDType ein, da hierbei nicht der Wert des Namens zum Vergleich der Einträge herangezogen wird, sondern der Zeitstempel des Eintrags. Dieser Zeitstempel wird von Cassandra automatisch jedem Eintrag hinzugefügt. Weiterhin werden beim Einfügen der Daten deren Namen validiert, sofern es sich beim Vergleichstyp nicht um BytesType handelt. Der Versuch, Daten zu einem Namen „Joe“ einzufügen, wenn die Column Family den Vergleichstyp LongType besitzt, würde somit zu einem Fehler führen.

Die bisherige Konfiguration wird nun pro Column Family um eine Vergleichsoption erweitert:

```
create column family Users with comparator = 'UTF8Type';
create column family Horses with comparator = 'UTF8Type';
create column family Weapons with comparator = 'UTF8Type';
create column family Duels with column_type = 'Super'
      and comparator = 'UTF8Type' and subcomparator = 'BytesType';
create column family Messages with comparator = 'UTF8Type';
```

Bei einer SuperColumn muss durch die Eigenschaft `subcomparator` zusätzlich angegeben werden, nach welchem Prinzip deren Unterspalten sortiert werden sollen. Im obigen Beispiel werden die SuperColumns anhand ihres Namens interpretiert und als UTF8-Zeichenkette sortiert, während der Name der enthaltenen Columns einfach als beliebiger Bytewert aufgefasst wird.

Aufgrund seines Aufbaus kann Cassandra nicht mit beliebigen Anfragen ausgewertet werden, wie dies bei RDBMS der Fall ist. Es ist also zwingend notwendig, das Datenmodell bereits so anzulegen, dass die benötigten Abfragen durch einfaches Auslesen von Bereichen geschehen können. In unserem Wild-West-Beispiel bietet sich die Abfrage aller Duelle nach Zeit sortiert an, um dem Nutzer einen Verlauf seiner Spiele anzuzeigen. Das Datenmodell so, wie es oben zu sehen ist, gibt das nicht her, da die Rows der CF Duels dazu anhand ihres Keys sortiert werden müssten. Eine Sortierung nach Key ist allerdings in Cassandra nicht vorgesehen, sodass wir uns hier einen entsprechend sortierten Index erst erschaffen müssen:

```
create column family DuelHistory with comparator = 'TimeUUIDType';
```

Die Column Family DuelHistory speichert eine zeitlich sortierte Folge der einzelnen Duelle. Sie kann von der Spieleapplikation abgefragt werden, um dem Nutzer seine vergangenen Spiele aufzulisten. Da die Informationen zu den Duellen selbst bereits in der CF Duels

abgelegt sind, werden hier nur noch die Keys der entsprechenden Einträge aus Duels gespeichert:

```
Keyspace: WildWest
 DuelHistory:
    Joe = { timeuuid1 : "high_noon", timeuuid2 : "dawn" }
    Jim = { timeuuid1 : "high_noon" }
```

Die History speichert zu jedem Spieler die Duelle, geordnet in zeitlicher Reihenfolge. Mit einer einfachen Abfrage dieser Column Family kann nun also anhand des Spielernamens der komplette, zeitlich sortierte Verlauf der Duelle dieses Spielers abgefragt werden. Werden einzelne Daten zu einem bestimmten Duell benötigt, so kann dieses wiederum über den in der History vermerkten Key referenziert werden.

Das Beispiel zeigt, dass die Skalierbarkeit von Cassandra mit einer gewissen Unflexibilität hinsichtlich der Anfragemöglichkeiten erkauft wird. Entwickler sollten also schon bei der Entwicklung des Datenmodells die späteren Abfragen im Auge behalten, um entsprechende Vorkehrungen treffen zu können.

Die Comparator-Types dienen noch einem weiteren Zweck in Cassandra. Auch wenn die Datenbank im Grundsatz schemalos daherkommt, bietet das System doch einige Möglichkeiten zur Validierung der eingegebenen Daten. Auf diese Art und Weise lassen sich zumindest einige Dateninkonsistenzen vermeiden. Zu jeder Column Family kann ein Metadatensatz hinterlegt werden, der sowohl die Art der einzelnen Columns als auch eventuelle zusätzliche Indexe beschreibt (dazu später mehr). Soll beispielsweise festgehalten werden, dass die `reactivity` eines Spielers immer ein ganzzahliger Wert zu sein hat, so würde das wie folgt formuliert:

```
create column family Users with comparator = 'UTF8Type' and
column_metadata = [{column_name:reactivity,
                    validation_class:IntegerType}];
```

Der Metadatensatz der Column Family Users definiert nun also, dass wenn es eine Spalte *reactivity* gibt, diese nur Integer-Werte enthalten darf. Der Versuch, beispielsweise den String „ABC“ dort einzufügen, führt zu einer Fehlermeldung. Das erlaubt zumindest eine einfache Validierung der eingegebenen Daten. Darüber hinausgehende Möglichkeiten sind allerdings nicht vorgesehen. Schon die Existenz einer bestimmten Spalte kann nicht sichergestellt werden.

3.2.4 CRUD-Operationen

Für erste Experimente mit Cassandra kann wieder *cassandra-cli* zum Einsatz kommen. Hier können mittels `set`, `get` und `delete` Werte in der Datenbank manipuliert und wieder ausgelesen werden. Zusätzlich soll hier noch der Zugriff aus der Programmiersprache Python mittels pycassa[5] dargestellt werden, um ein Gefühl für die Einbindung von Cassandra in eigene Programme zu vermitteln. Auf die Installation von pycassa soll hier nicht

[5] http://pycassa.github.com/pycassa/ Achtung: Entgegen der Aussagen in der Installationsanleitung kann es vorkommen, dass Thrift als Abhängigkeit nicht automatisch mit installiert wird. In dem Fall muss einfach manuell wie ebenfalls in der Anleitung beschrieben das Packet thrift05 hinzugefügt werden.

näher eingegangen werden. Soweit verfügbar kann diese einfach über easy_install erfolgen. Als dritte Möglichkeit soll hier die noch in der Entwicklung befindliche Cassandra Query Language (CQL) vorgestellt werden. Die SQL-ähnliche Abfragesprache wird ab Cassandra 0.8[6] speziell Umsteigern aus der SQL-Welt eine vertraute Umgebung bieten. Für Experimente mit CQL findet sich in den Cassandra-Quellen im Verzeichnis *drivers/py/cqlsh* eine Shell ähnlich zu *cassandra-cli*. Eine recht umfassende Dokumentation wird im Verzeichnis *doc/cql* mitgeliefert. Das entschädigt für die leider bei weitem nicht mit *cassandra-cli* vergleichbare Online-Hilfe von cqlsh.

Um den Schreibzugriff auf eine normale Column Family zu demonstrieren, sollen zuerst unsere beiden Nutzer aus Listing 3.2.1 angelegt werden. Cassandra unterscheidet hier nicht zwischen Create und Update. Beides wird mittels des Kommandos `set` im CLI ausgeführt:

```
use WildWest;
set Users['Jim']['reactivity'] = 1;
set Users['Joe']['reactivity'] = 9:
set Users['Joe']['nick_name'] = 'Little Joe';
```

Die gleiche Operation sieht in Python etwas übersichtlicher aus[7]:

```
import pycassa
pool = pycassa.connect('WildWest')
users = pycassa.ColumnFamily(pool, 'Users')
users.insert('Jim', { 'reactivity' : '1' })
users.insert('Joe', { 'reactivity' : '9', 'nick_name' : 'Little Joe' })
```

Nutzer mit SQL-Kenntnissen werden eine Unterscheidung zwischen INSERT und UPDATE vermissen: CQL unterscheidet hier nicht, Schreiboperationen werden immer mit UPDATE angestoßen[8]:

```
use WildWest;
update Users set reactivity=9 where key=Jim;
update Users set nick_name='Little Joe', reactivity=9 where key=Joe;
```

Columns werden im CLI grundsätzlich einzeln eingefügt, auch wenn sie am Ende zu einem Datensatz gehören (beispielsweise unter dem Schlüssel „Joe“ im obigen Beispiel), während Schnittstellen wie pycassa Abstraktionen bieten, um ganze Hashmaps in eine Row zu befördern. CQL bietet gegenüber dem CLI hier eine deutliche Verbesserung bezüglich der Übersichtlichkeit.

Wir haben nun zwei Einträge in der Column Family Users, jeweils referenziert durch die Keys „Jim“ und „Joe“. „Jim“ enthält den Wert „reactivity“ = 1, während „Joe“ mit „reactivity“ = 9 und „nick_name“ = „Little Joe“ zwei Werte enthält.

---

6 Version 0.8 befindet sich zum aktuellen Zeitpunkt noch in Entwicklung und kann nur direkt aus dem SVN-Repository gebaut werden. Es könnten sich also bis zur Veröffentlichung noch Änderungen an den hier vorgestellten Informationen ergeben.

7 Das Beispiel ist zwar länger, aber die ersten beiden Zeilen sind Initialisierungscode, der in einem größeren Programm nur einmal ausgeführt werden würde. Selbst der Zugriff auf die Column Family muss nur einmal erfolgen. Das resultierende Objekt kann dann für beliebig viele Operationen verwendet werden.

8 Cassandra 0.8 nimmt die Angabe von Datentypen für einzelne Felder wesentlich genauer. Die Standardklasse BytesType für den Schlüssel wird so beispielsweise nicht mehr den Wert ‚Joe‘ oder ‚Jim‘ akzeptieren. Hier muss durch das Setzen der entsprechenden Metadaten der Typ auf UTF8Type umgestellt werden.

SuperColumns fügen eine weitere Strukturierungsebene ein, die sich im CLI durch einen weiteren Key äußert[9]:

```
set Duels['dawn']['result']['player_1'] = 'Jim';
set Duels['dawn']['result']['player_2'] = 'Joe';
set Duels['dawn']['result']['winner'] = 'Joe';
set Duels['dawn']['position']['x'] = '13';
set Duels['dawn']['position']['y'] = '21';
set Duels['dawn']['position']['desc'] = 'Main Street';
```

Hier zeigt pycassa deutlich den Unterschied zwischen Columns und SuperColumns: Letztere haben als Wert Columns.

```
duels = pycassa.ColumnFamily(pool, 'Duels')
duels.insert('dusk', { 'result' : { 'player_1' : 'Joe',
                                    'player_2' : 'Hank',
                                    'winner' : 'Joe'},
                       'position' : { 'x' : '12', 'y' : '15',
                                      'desc' : 'Saloon' } })
```

CQL unterstützt leider in Version 1.0.0 keine SuperColumns, sodass ein entsprechendes Beispiel hier entfallen muss.

Das Auslesen eines einzelnen Elementes in Cassandra könnte einfacher nicht sein:

```
get Users['Joe'];
=> (column=nick_name, value=536c6f7768616e64, timestamp=1301174261419383)
=> (column=reactivity, value=30, timestamp=1301174261419383)
```

Beachtenswert ist hier, dass der Wert der Column „nickname“ hexadezimal angezeigt wird. Die bereits vorgestellten Validation Classes werden von Cassandra auch zur Anzeige der Daten herangezogen. Die Standardklasse BytesType wird hierbei hexadezimal dargestellt. Wäre in unserem Beispiel für die Column „nickname“ die Validation Class UTF8Type gesetzt worden, so würde das CLI hier als Wert eine lesbare Zeichenkette anzeigen.[10] Zur temporären Umsetzung des Anzeigetyps kann das `get`-Kommando erweitert werden:

```
 get Users['Joe'] as utf8;
=> (column=nick_name, value=Little Joe, timestamp=1301174261419383)
=> (column=reactivity, value=30, timestamp=1301174261419383)
```

Damit wird das CLI angewiesen, die Columns innerhalb von Users als UTF8-Zeichenketten zu interpretieren, soweit nichts anderes gesetzt ist.

Die entsprechende Abfrage sieht in Python fast identisch aus:

```
users = pycassa.ColumnFamily(pool, 'Users')
users.get('Joe')
OrderedDict([(u'nick_name', 'Little Joe'), (u'reactivity', '9')])
```

Die pycassa-Bibliothek verwandelt die zurückgegebenen Daten in entsprechende Python-Objekte, um eine einfache Weiterverarbeitung zu ermöglichen.

Die CQL-Syntax wird vielen Nutzern wieder vertraut vorkommen:

---

9 Initialisierungsanweisungen wie use Keyspace im CLI oder import pycassa etc. in Python werden der Übersichtlichkeit wegen im Folgenden weggelassen.

10 Ein Beispiel für die Verwendung von Schema-Informationen wie Validation Classes findet sich in der Kurzanleitung zu *cassandra-cli* unter http://wiki.apache.org/cassandra/CassandraCli.

```
select * from Users;
u'Jim' | u'reactivity',9
u'Joe' | u'nick_name',u'Little Joe' | u'reactivity',3
```

Zuletzt soll noch das Löschen von Columns aus der Datenbank demonstriert werden. Gelöscht werden können hierbei ganze Rows oder auch nur einzelne Columns. Im CLI ist die Syntax vergleichbar mit einem `get`:

```
del Users['Joe']['nick_name'];
del Users['Jim'];
```

Cassandra hat wie viele NoSQL-Datenbanken kein Konzept von referenzieller Integrität, um zu prüfen, ob die Rows noch irgendwo im Datenmodell referenziert werden. Hier ist also seitens der Anwendung eine entsprechende Prüfung angezeigt. In Python stellt das ColumnFamily-Objekt eine Methode zum Löschen von Daten bereit:

```
cf = pycassa.ColumnFamily(pool, 'Users')
cf.remove('Joe', ['nick_name'])
```

Auch hier können wieder ganze Rows oder nur einzelne Columns gelöscht werden. Als Besonderheit gegenüber dem CLI können in Python mehrere Columns auf einmal gelöscht werden. Die entsprechenden Namen werden einfach als Liste im zweiten Parameter von remove übergeben.

Der entsprechende Befehl in CQL lautet DELETE:

```
delete nick_name from Users where key=Joe;
delete from Users where key=Jim;
```

Auch hier können wieder mehrere Rows auf einmal gelöscht werden, indem ihre Schlüssel in der Form „`keys in (key1, key2, …)`“ angegeben werden.

3.2.5 Abfragen in Cassandra

Einfaches Lesen und Schreiben von Daten anhand bekannter Schlüssel ist meist nicht genug. Im Alltag stellen sich oft Probleme, die ein Auffinden von Daten anhand bestimmter Kriterien ermöglichen. So könnten beispielsweise alle Nachrichten in einem bestimmten Zeitraum gefragt sein oder alle Duelle, die ein bestimmter Nutzer gewonnen hat. Cassandra bietet daher für das Auffinden von Daten sogenannte *Range Queries* an. Mit diesen lassen sich ganze Bereiche von Datenelementen ausfiltern.

Grundregel in Cassandra: Abfragen lassen sich nur auf Indexen ausführen[11]. Das System hält die Daten grundsätzlich sortiert vor, um Abfragen schnell und skalierbar zu gestalten. Zuallererst werden die Daten nach ihrem Schlüssel sortiert. Range Queries sind also über die Schlüssel möglich. Allerdings gilt es hierbei eine Einschränkung zu beachten: Cassandra verteilt die Daten auf mehrere Knoten eines Clusters anhand eines sogenannten *Parti-*

11 Das ist nur die halbe Wahrheit. Abfragen mit mehreren Kriterien können sich auch auf Felder ohne Index beziehen, solange das erste Kriterium ein indiziertes Feld ist. Cassandra filtert dann über das indizierte Feld vor und wendet die anderen Kriterien in einer einfachen Schleife an. Je nach Datenmenge kann das natürlich zu hoher Systemlast führen und sollte mit Bedacht eingesetzt werden.

tioners. In der Standardeinstellung ist das der *RandomPartitioner*, welcher den MD5-Hash des Schlüssels bildet und diesen zur Sortierung und Verteilung verwendet. Dadurch ergibt sich unabhängig von der statistischen Verteilung der Schlüssel eine hinreichend gleichmäßige Aufteilung der Daten auf die Knoten. Damit werden die einzelnen Knoten gleichmäßig ausgelastet, und die Skalierbarkeit des Systems steigt. Die Skalierbarkeit bezahlt man allerdings mit der verlorenen Ordnung der Daten. Wie der Name schon andeutet, verteilt der RandomPartitioner die Daten (pseudo-)zufällig und zerstört damit deren Ordnung. Dank des konsistenten Hash-Verfahrens weiß das System bei einer einzelnen Anfrage sofort, auf welchem Knoten sich der Schlüssel befinden muss. Eine Auflistung einzelner Elemente nach der Ordnung ihrer Schlüssel ist so allerdings nicht mehr effizient möglich. Stattdessen müssen alle Knoten nach einer Menge an Schlüsseln gefragt werden. Als Alternative bietet Cassandra den *OrderPreservingPartitioner*. Dieser bewahrt wie der Name schon sagt, die Ordnung der Schlüssel, um effiziente Range Queries zu ermöglichen. Allerdings ist auch hier wieder ein Preis zu zahlen: Eine ungleiche Verteilung der Schlüssel führt zu einer ungleichen Verteilung der Daten (und damit Systemlast) im Cluster. Wie so oft ist hier also eine genaue Kenntnis des Anwendungsgebietes notwendig. Ein Detail gilt es bei der Auswahl des Partitioners zu beachten: Dieser kann nicht geändert werden. Vom Partitioner hängt unter anderen auch die genaue Ausgestaltung des Datenformates auf der Platte ab, sodass eine Umstellung nur durch Verwerfen und Neuanlegen des kompletten Clusters einschließlich aller Keyspaces möglich ist.

Seit Cassandra 0.7 können Indexe nicht nur über die Schlüssel eines Elementes gebildet werden, sondern über jedes beliebige Feld. Diese *Secondary Indices* erlauben wesentlich flexiblere Suchen nach Elementen (allein schon aufgrund der Tatsache, dass Columns im Gegensatz zu Keys keine eindeutigen Werte haben müssen). Ein Beispiel: Die Nutzer des WildWest-Onlinespiels sollen einer Fraktion beitreten können. Entweder sind sie Gesetzeshüter (*law*) oder Verbrecher (*outlaw*). Dazu werden dem existierenden Datenmodell einige Informationen hinzugefügt. Die Column Family Users wird durch einen Metadaten ergänzt, die für die Column *group* einen Typ (UTF8Type) und eine Indexart (KEYS[12]) angibt.

```
update column family Users with column_metadata=[{ column_name:group,
validation_class:UTF8Type, index_type:KEYS}];
```

Die Spalte *group* kann nun ebenfalls für Abfragen herangezogen werden. Im CLI kann dann mittels einer `where`-Klausel in der Anfrage nach bestimmten Werten gefiltert werden:

```
get Users where group='law';
```

Diese Anfrage liefert (wie zu erwarten) alle Nutzer der Gruppe ‚law' zurück.

12 KEYS ist momentan die einzig verfügbare Indexart. Sie ist vergleichbar mit der Sortierung der Rows nach Schlüsseln.

3.2.6 Replikation und Skalierung

Zwei der zentralen Entwurfskriterien für Cassandra waren Skalierbarkeit und Robustheit. Im Idealfall sollen die Daten in einem Cluster jederzeit verfügbar sein und problemlos unter großer Last abgefragt werden können. Um das zu erreichen, orientiert sich Cassandra an dem Prinzip der *Distributed Hash Table*. Hierbei werden die Daten über Schlüssel adressiert (wie man im Datenmodell ja bereits gesehen hat). Die Schlüssel verteilen sich über einen Schlüsselraum fester Größe (im Falle von Cassandra 127 Bit, also 2^{127} Schlüssel). Dieser Schlüsselraum wird gleichmäßig über alle Knoten eines Clusters verteilt, sodass diese einen logischen Ring bilden (der allerdings nichts mit der physikalischen Struktur des zugrundeliegenden Netzes zu tun haben muss). Durch das *Consistent Hashing* genannte Verfahren der Schlüsselvergabe kann jeder Rechner des Clusters jederzeit anhand eines gegebenen Schlüssels direkt bestimmen, welcher Knoten aktuell für diesen Schlüssel zuständig ist, und Anfragen direkt weiterleiten.

Die einfache Skalierung eines Clusters im laufenden Betrieb ist eines der Kern-Features von Cassandra. Ein neuer Node, der in den Ring einsteigen möchte, muss lediglich die Adresse eines anderen Knotens kennen (*seed node*). Ein Bootstrap-Verfahren sorgt dann dafür, dass das Wissen, welcher Knoten wofür verantwortlich ist, optimal verteilt wird. Diese Verteilphase führt beim Erweitern des Rings in der Latenzzeitverteilung allerdings zu Schwankungen in der Latenz von Anfragen, sodass es sich empfiehlt, einige Minuten zu warten, bis der Ringraum aufgeteilt ist und sich alle Knoten per Bootstrapping initialisiert haben.

Fällt ein Knoten des Rings aus, so übernehmen die Replikate seinen Platz solange, bis er wieder verfügbar ist. Kommt ein Knoten zurück, werden die mittlerweile aufgelaufenen Writes synchronisiert und wieder ein konsistenter Zustand hergestellt. Dieser Automatismus bereitet natürlich Probleme beim dauerhaften Entfernen eines Knotens aus dem Ring (beispielsweise durch Umbau der Infrastruktur). Cassandra bietet daher im Clustermanagementtool mittels `nodetool decommision` die Möglichkeit, Knoten dauerhaft aus dem Ring zu entfernen und Ihre Daten so neu zu verteilen. Diese Art der „Rückskalierung“ fehlt bei einigen anderen NoSQL-Datenbanken.

Die Datenkonsistenz kann wie bei Riak mit den Parametern `R` und `W` für Lese- und Schreiboperationen eingestellt werden. Bei vielen Operationen können auch individuelle Consistency-Level eingestellt werden, sodass hier gut zwischen Performance und Konsistenz gewählt werden kann:

- **ZERO**: Für Schreiboperationen wird nichts garantiert.
- **ANY**: Die Schreiboperation muss mindestens auf einem Node geschrieben worden sein.
- **ONE**: Stellt beim Schreiben sicher, dass mindestens ein Node in den Commit Log und in die RAM-Tabellen geschrieben hat, bevor der Client eine Bestätigung bekommt. Leseoperationen erhalten das erste verfügbare Ergebnis. Eine *repair*-Operation stellt jedoch sicher, dass nachfolgende Leseoperationen garantiert ein richtiges Ergebnis liefern.
- **QUORUM**: Beim Schreiben müssen erst `ReplikationsFaktor / 2 + 1` Replikas geantwortet haben. Beim Lesen wird der Eintrag mit dem letzten TimeStamp zurückge-

liefert (des Weiteren gibt es mit DCQUORUM eine weitere `rack-aware`-Strategie). Inkonsistenzen während der Synchronisationsphase können so vor dem Client verborgen und immer die neuesten Daten zur Verfügung gestellt werden.

- **ALL**: Eine Operation muss von allen Replika-Nodes eine positive Antwort erhalten, bevor die Client-Operation Erfolg hat. Diese teuerste Variante bleibt Daten vorbehalten, bei denen ein inkonsistenter Zustand in gar keinem Fall vorliegen darf.

Cassandra verwendet Zeitstempel zur Konflikterkennung/-behebung, setzt also eine clusterweit einheitliche Zeit voraus. Nodes, deren Daten nicht aktuell sind, werden über Neuerungen informiert und können sich mit den neuesten Daten versorgen. Mittels `nodetool repair` können die Nodes außerdem dazu gebracht werden, derartige Reparaturen selbsttätig zu finden und durchzuführen (beispielsweise nach einem größeren Problem im Cluster, bei dem weitreichende Inkonsistenzen bekannt sind und vorab behoben werden sollen). Knoten, die komplett ausgefallen sind, können über die Bootstrap-Methode durch neue Instanzen ersetzt werden.

Cassandra sollte mit viel verfügbarem RAM aufgesetzt werden, da alle Operationen initial im Cache ausgeführt werden. Schreiboperationen werden geloggt und zunächst im Hauptspeicher abgelegt. Erst später wird ein *commit-log* ausgeführt und danach die Operationen mit einer `flush`-Operation versendet. Zusammen mit einer optimistischen Konsistenzstrategie kann Cassandra so zwar Writes schnell und effizient abarbeiten, allerdings zum Preis eines erhöhten RAM-Verbrauchs. Speziell beim Nachziehen des *commit-logs* nach einem Knotenausfall kann es dazu kommen, dass aufgrund fehlenden Arbeitsspeichers ein Knoten gar nicht mehr in den Cluster integriert werden kann.

Die Daten eines Cassandra-Clusters können mit einem Snapshot-Tool gesichert werden. Weiterhin stellt Cassandra ein Werkzeug zur Verfügung (*sstable2json* und *json2sstable*), mit dem komplette Datenbankdateien oder einzelne Keys über JSON im- und exportiert werden können. Zusätzlich verfügt Cassandra über ein JMX-Interface, mit dem viele Parameter überwacht werden können. Hier sind insbesondere die Lese- und Schreiboperationen pro Zeit und die Latenzzeit für diese Operationen interessant.

3.2.7 Bewertung

Cassandra befindet sich immer noch in einem verhältnismäßig frühen Stadium der Entwicklung, sodass sich zwischen den einzelnen Versionen teilweise gravierende Änderungen ergeben. Dennoch ist es bereits bei vielen Firmen – allen voran Facebook, wo die Entwicklung ursprünglich begann – produktiv im Einsatz.

Vorteile

- Extrem einfach skalierbar. Das Hinzufügen eines weiteren Knotens ist mit dem Starten einer neuen Cassandra-Instanz erledigt. Weitere Aktivitäten sind nicht notwendig. Es gibt bei Cassandra keinen *Single Point of Failure*. Das Schreiben oder Lesen von Daten ist

nicht auf bestimmte Nodes beschränkt. Cassandra-Cluster sind selbstheilend und können auf wenig Latenz optimiert werden. Der Ring ist zudem leicht verkleinerbar.

- Erweiterungen des Datenmodells im vorgegebenen Rahmen sind leicht möglich. KeySpaces und Column Families können erweitert werden, müssen aber bekannt gemacht werden. Das kann seit Version 0.7 auch online ohne Neustart eines Clusternodes erfolgen. Innerhalb der Column Families ist man genauso frei wie bei einer Dokumentdatenbank. Neue Key/Value-Paare können zur Laufzeit beliebig eingefügt werden. Mit den SuperColumns wurde ähnlich wie bei Redis eine weitere Dimension eingefügt, mit der Columns Listen von Key/Value-Paaren sind.
- Mit CQL steht ab Cassandra 0.8 eine vertraute Abfragesprache für Nutzer aus dem SQL-Umfeld zur Verfügung.
- Über indizierte Daten kann Cassandra effiziente Bereichsabfragen durchführen. Indexe können dabei nicht nur über die Schlüssel eines Datensatzes gebildet werden, sondern in gewissen Grenzen auch über beliebige Columns (*secondary indices*).
- Cassandra bietet detailliert konfigurierbare Replikationsstrategien, die auch eine Verteilung von Daten über verschiedene Standorte erlauben.
- Konsistenz, Dauerhaftigkeit und Latenzzeit sind recht gut konfigurierbar, sodass der Anwender selbst wählen kann, in welcher Ecke des CAP-Dreiecks das System eher liegen sollte. Viele Parameter sind mit JMX gut zu überwachen. Tools für Dumping, Im- und Export stehen bereit.

Nachteile

- Cassandra verfügt über viel weniger Abfragemöglichkeiten als beispielsweise MongoDB. Map/Reduce-ähnliche Berechnungen können allerdings über die Cassandra-Hadoop-Integration nachgerüstet werden.
- Versions-Updates für Cassandra sind derzeit mit Vorsicht zu genießen, da sich das Speicherformat bisher einige Male verändert hat. Da das Speicherformat zusätzlich von gewissen konfigurierbaren Parametern abhängt (beispielsweise der Partitionierung von Daten), muss beim Entwurf des Datenmodells darauf geachtet werden, welche Konfiguration gewählt wird. Mancher Partitioner erlaubt zum Beispiel keine Bereichsabfragen mit oberen Grenzen. Muss diese Funktionalität im laufenden System hinzugefügt werden, so geht nichts an einem Neuaufsetzen des Clusters vorbei. Eine Änderung dieses Parameters für existierende Cluster ist nicht möglich.
- Ab und zu landen kritische Fehlermeldungen im Jira-System von Cassandra. Diese sollte man sorgfältig beobachten (*https://issues.apache.org/jira/browse/cassandra*).
- Cassandra benötigt synchronisierte Uhren für seine Konfliktbehandlung. Daher besteht hier eine Abhängigkeit zu einer funktionierenden Zeitsynchronisation im Cluster, beispielsweise mit NTP (Network Time Protocol).
- Die Dokumentation ist – auch im Wiki – stark verstreut. Allerdings existiert seit Dezember 2010 mit „Cassandra: The Definitive Guide“ ein Buch zur Dokumentation. Wer jedoch nur auf die frei verfügbare elektronische Dokumentation bauen will, muss einige Zeit für Recherche einplanen.

Das typische Einsatzgebiet von Cassandra sind skalierende Webanwendungen. Facebook selbst ist hier das beste Beispiel, wo Cassandra auf Hunderten von Nodes eingesetzt wird. Dort sind keine zu komplexen Abfragen nötig, aber das System muss stabil antworten und skalieren. Die größte Cassandra-Installation ist derzeit auf mehr als 150 Rechnern mit mehr als 100 TB Daten verteilt. Twitter wollte ebenfalls auf Cassandra migrieren, schreckte aber im Sommer 2010 anscheinend noch vor dem großen Migrationsaufwand zurück, da unter anderem auch Clients in der Programmiersprache Scala verwendet werden müssen. Yahoo hat Cassandra ebenfalls evaluiert. Cassandra ist bei Cisco, Mhalo, Ooyala, Digg, Rackspace, Reddit, Cloudkick und vielen weiteren Firmen aktuell im produktiven Einsatz.

Links

Cassandra Wiki: *http://wiki.apache.org/cassandra/*

WTF is a SuperColumn? An Intro to the Cassandra Data Model:
http://arin.me/blog/wtf-is-a-supercolumn-cassandra-data-model

Secondary indexes in Cassandra:
http://www.anuff.com/2010/07/secondary-indexes-in-cassandra.html

Python-Client: *http://github.com/pycassa/pycassa*

3.3 Amazon SimpleDB

Steckbrief

Webadresse:	http://aws.amazon.com/de/simpledb/
Kategorie:	Wide Column Store
API:	SOAP, REST , Java, Ruby, C#, Perl, PHP, Python und JavaScript
Protokoll:	HTTP, HTTPS
Geschrieben in:	vermutlich Erlang*
Concurrency	Eventually Consistent Read und Consistent Read
Replikation:	Automatisch über AWS geografisch verteilt
Skalierung	Automatisch über AWS, horizontale Skalierung über neue Domänen
Lizenz:	AWS Customer Agreement

* Vgl. [Müller10], Neue DBMS im Vergleich, Seite 124

Das Webportal Amazon Web Services (AWS) realisiert weltweites Cloud-Computing für Entwickler und Unternehmen und bietet eine Infrastruktur für Computing, Bereitstellung von Inhalten, E-Commerce, Messaging, Überwachung, Zahlungen und Rechnungsstellung, Support, Web-Datenverkehr und Speicherung von Daten in seinen Datenbanken. Das alles basiert auf dem Web-Service SimpleDB. Damit bietet Amazon seinen Kunden ein einfaches hochverfügbares, geografisch verteiltes, schemafreies und skalierbares Datenbankmanagementsystem, dessen Kosten nutzungsabhängig berechnet werden.

3.3.1 Allgemeines

Amazon SimpleDB ist eine hochverfügbare und skalierbare Datenbanklösung. Während die Daten wachsen, werden automatisch verstreut neue Domains erstellt und Sicherheitskopien auf den Domains verteilt. Es existieren SimpleDB-Zentralen in den USA (2x), in Irland und Singapur. Als Anwender meldet man eine dieser Regionen als Hauptdomain an. Die Kommunikation mit der Datenbank erfolgt über Web-Service-Anfragen der Form PUT, DELETE, etc., für die Amazon REST- und SOAP-Zugriffe zur Verfügung stellt. Die abgelegten Daten werden automatisch indiziert. Ein besonderer Vorteil von SimpleDB ist sicherlich, dass man nur die Ressourcen zahlt, die man tatsächlich benötigt. Ein weiterer Vorteil der Datenhaltung ist, dass man sich an kein festes Schema halten muss. Das Schema ist beliebig erweiterbar. Aufgrund der gleichen Lokalität der Services treten dann beim Abruf der Daten laut Amazon nur LAN-ähnliche Verzögerungszeiten auf. Mit den beschriebenen Features ist Amazon SimpleDB ideal für Anwendungen innerhalb von EC2 (*Amazon Elastic Compute Cloud*). Bezüglich der Datenkonsistenz unterstützt SimpleDB zwei Optionen zum Lesen von Daten:

- *Eventually Consistent Read* – Besitzt eine geringe Latenz und ermöglicht einen hohen Durchsatz für Leseoperationen. Liefert aber unter Umständen nicht die Ergebnisse einer kürzlich abgeschlossenen Schreiboperation, da die Herstellung der Konsistenz über alle Kopien der Daten in der Regel eine Sekunde dauert.
- *Consistent Read* – Besitzt eine höhere Latenz und einen geringeren Datendurchsatz bei Leseoperationen. Gewährleistet aber die Konsistenz der Daten bezüglich der vor der Leseoperation erfolgreich abgeschlossenen Schreiboperationen.

Per Standard werden die Leseoperationen als *Eventually Consistent Read* durchgeführt.

Die Preisgestaltung[13] von SimpleDB basiert auf Grundlage der tatsächlichen Nutzung. Diese wird gemessen und auf den nächstliegenden Cent gerundet. Es treten die folgenden Nettokosten auf:

- **Übertragungskosten**: Das erste GB ist kostenlos, danach fallen 8–20 US-Cent pro GB je nach Region und Volumen an.
- **Speicherkosten**: Auch hier ist das erste GB kostenlos, danach fallen ab 25–29 US-Cent je nach Region pro GB und Monat an.
- **Anfragekosten**: Jede Abfrage – insbesondere auch Query – wird auf einen 1,7 GHz Xeon 2007er Rechner normiert und kostet dann 14 Cent pro Maschinenstunde.

Nutzung und auflaufende Kosten können über den eigenen AWS-Account eingesehen und kontrolliert werden.

3.3.2 Datenmodell

SimpleDB ist eine schemafreie Datenbank, zur Strukturierung der Daten wird zwischen Domänen (*Domains)*, Elementen (*Items)*, Attributen (*Attributes)* und Werten (*Values)* unter-

13 Vgl. http://aws.amazon.com/de/simpledb/pricing/

schieden. Diese Strukturelemente sind mit denen einer Tabellenkalkulation vergleichbar und erinnern ein wenig an die Tabelle eines RDBMS, eine Schemadefinition wird zur strukturierten Ablage der Daten aber trotzdem nicht benötigt.

- *Domains* sind vergleichbar mit einem Arbeitsblatt, also Tabellen, die gleiche Daten enthalten.
- *Items* entsprechen den Zeilen der Tabelle und somit einem Objekt, das ein oder mehrere Attribute besitzen kann.
- *Attributes* entsprechen den Spalten der Tabelle und repräsentieren Attributkategorien von Daten, die zugeordnet werden können.
- *Values* sind die Werte der Zellen und repräsentieren die Instanzen der Attribute eines Elements. In einer Zelle können auch mehrere Werte enthalten sein.

Jedes Nutzerkonto hat initial 250 Domänen zur Verfügung, in denen Daten gespeichert werden können. Pro Domäne dürfen bis maximal 10 GB Daten in Form von maximal 1 Milliarde Attributen enthalten sein. In den Tabellen von SimpleDB können beliebige UTF-8-Zeichenketten gespeichert werden. Für jedes Element kann man 256 Attribute definieren. Element- und Attributname dürfen wie auch der Wert eines Attributs eine Länge von 1024 Bytes aufweisen[14]. Die Datenbank fügt selbst 45 Bytes an Metainformation je Element, Attributname und Attributpaar hinzu, die bei der Preisberechnung berücksichtigt werden müssen. Sollten 100 Domänen nicht reichen, können mehr Domänen beantragt werden.

SimpleDB unterstützt eine Leistungsverbesserung durch die Parallelisierung von Abfragen auf partitionierte Daten. Die Datenmenge wird dazu in kleinere Datensätze auf verschiedene Domänen aufgeteilt. Da die Abfragen bei SimpleDB grundsätzlich nur gegen eine Domäne erfolgen können, muss eine Aggregation der Ergebnisse dann in der Anwendung selbst erfolgen.

3.3.3 Datensicherheit und Datenschutz

Mit AWS stellt Amazon eine hoch skalierbare und hoch verfügbare Cloud-Computing-Plattform zur Verführung. Um in dieser Cloud-Computing-Plattform Sicherheit anzubieten und private Daten vor fremden Zugriffen zu schützen, setzt Amazon auf Dienste, die mit den heute üblichen Sicherungsmaßnamen übereinstimmen. Zur Gewährleistung einer sicheren Infrastruktur verfolgt Amazon die folgenden grundlegenden Ansätze[15]:

- *Zertifizierungen und Akkreditierungen* – AWS hat erfolgreich ein SAS70[16] Typ-II-Audit bestanden und wird auch weiterhin die geeigneten Sicherheitsmaßnahmen ergreifen, um Akkreditierungen zu erhalten, die die Sicherheit der Infrastruktur und Dienstleistungen demonstrieren.

14 Vgl. Amazon SimpleDB Developer Guide (API Version 2009-04-15), Seite 9

15 Vgl. Amazon Web Services – Sicherheitsprozesse im Überblick, November 2009

16 SAS – **S**tatement on **A**uditing **S**tandards, http://sas70.com/

- *Sichere Designprinzipien und Dienste* – Die Softwareentwicklung bei Amazon erfolgt nach sicheren, bewährten Methoden (*Best Practices*) und beinhaltet unter anderem: *Design Reviews* durch das interne Sicherheitsteam, eine Risikobewertung, statistische Analysen sowie Tests durch ausgesuchte Industrieexperten. Jeder Dienst innerhalb von AWS wurde so entwickelt, um Sicherheit zu bieten, und enthält eine Reihe von Funktionen, die unbefugten Zugriff oder Verwendung beschränken, ohne dabei die Flexibilität der Kunden einzuschränken.
- *Physical Security* – Amazon hat viele Jahre Erfahrung in der Planung sowie dem Bau und Betrieb großer Rechenzentren. Die AWS-Infrastruktur ist in kontrollierten Rechenzentren auf der ganzen Welt untergebracht. Nur geprüftes Personal mit einer speziellen Berechtigung kennt die tatsächliche Lage der Rechenzentren. Die Rechenzentren selbst sind mit einer Vielzahl von physischen Barrieren gesichert, um unbefugten Zugriff zu verhindern.
- *Backups* – Die Daten der AWS werden redundant an verschiedenen Standorten gespeichert.
- *Datenschutz* – Schutz gegen unberechtigten Zugriff wird durch AWS-Authentifizierungsmechanismen bereitgestellt. Dazu nutzt AWS bewährte kryptographische Verfahren.
- *Netzwerksicherheit* – Das AWS-Netzwerk bietet Schutz gegen eine Vielzahl von bekannten Angriffsmethoden und Schwachstellen im Internet wie z. B. DDoS-Angriffe (*Distributed Denial Of Service*), MITM-Angriffe (*Man In the Middle*), *IP-Spoofing*, *Port Scanning*, *Packet Sniffing*.

3.3.4 Installation

Von einer Installation im herkömmlichen Sinne kann man bei der Amazon SimpleDB nicht reden, da sie als *Software as a Service* (SaaS) innerhalb der AWS-Infrastruktur bereitgestellt wird. Grundsätzlich ist zur Nutzung aller in der AWS-Infrastruktur angebotenen Dienste die Erstellung eines AWS-Accounts notwendig. Anschließend kann man sich für einen oder mehrere Dienste anmelden. Im Falle der Nutzung der SimpleDB sind dazu folgende Schritte notwendig:

1. Die Anmeldung über den Link „Jetzt Anmelden“ auf *http://aws.amazon.com/de/*
2. Eingabe einer gültigen E-Mail-Adresse und Auswahl von „*I am a new customer*“.
3. Auf der nächsten Seite müssen Name und Passwort eingegeben werden.
4. Auf der folgenden *Account Info*“Seite sind weitere Kontaktdaten einzugeben, und das *AWS Customer Agreement* ist zu bestätigen.

Mit Erhalt einer Bestätigungsmail ist die Erstellung des AWS-Accounts abgeschlossen. Man kann sich nun zur Nutzung der SimpleDB anmelden:

5. Die Anmeldung erfolgt über den Link „Anmelden für Amazon SimpleDB“ auf der Website *http://aws.amazon.com/de/simpledb/*

6. Die Bezahlung des Dienstes erfolgt über eine Kreditkarte. Auf den folgenden Seiten sind Angaben zur Kreditkarte und Rechnungsadresse einzugeben.
7. Nach einer Überprüfung der Angaben wird die Anmeldung durch den Erhalt einer Bestätigungsmail abgeschlossen.

Für den autorisierten Zugriff auf Daten in der SimpleDB werden zwei *AWS Access Key Identifiers* benötigt. Diese werden beim Anlegen des AWS-Accounts von Amazon erstellt:

- Access Key ID (20 alphanumerische Zeichen) z.B.: `022QF06E7MXBSH9DHM02`
- Secret Access Key (40 alphanumerische Zeichen)
  z.B.: `kWcrlUX5JEDGM/LtmEENI/aVmYvHNif5zB+d9+ct`

Die nicht geheime *Access Key ID* ist mit dem Account verbunden und dient der Referenzierung des erstellten Accounts. Er kann von jedem für Anfragen an AWS genutzt werden. Der *Secret Access Key* sollte, wie der Name schon sagt, geheim gehalten werden. Er dient zur Generierung einer digitalen Unterschrift, mit der der Nutzer seine Identität beim Zugriff auf den Account nachweisen kann. Die *AWS Access Key Identifiers* werden bei der Erstellung des AWS-Accounts dargestellt und können nach der Anmeldung über die *AWS Access Key Identifiers Page* jederzeit angezeigt werden.

3.3.5 CRUD-Operationen

Der native Zugriff auf SimpleDB kann über das Architekturprinzip REST oder über das Web-Service-Protokoll SOAP erfolgen und setzt hiermit auch auf bewährte Web-Techniken. Die API von SimpleDB ist recht einfach und beschränkt sich auf Grundoperationen. Für das Domänenmanagement gibt es die Befehle `CreateDomain`, `DeleteDomain`, `ListDomain` und `DomainMetadata`. Für die Manipulation der einzelnen Attribute in der Domain gibt es die Befehle `PutAttributes`, `BatchPutAttributes`, `DeleteAttributes`, `BatchDeleteAttributes` und `GetAttributes`. Interessant ist, dass es keinen Update-Befehl für die einzelnen Attribute gibt, da `PutAttributes` sowohl für das initiale Anlegen als auch für Veränderungen zum Einsatz kommt. Für allgemeine Abfragen gibt es den `Select`-Befehl, der dem gleichen Befehl in SQL entspricht, aber ein geringeres Subset darstellt. Der generelle Aufbau der Select-Abfrage ist der folgende:

Listing 3.3.1 Amazon SimpleDB: Select-Abfrage

```
select output_list // *, itemName(), count(*), list of attributes
from domain_name
[where expression]
[sort_instruction]
[limit limit] // default 100, max 250
```

Die Ausdrücke (Expressions) können mit vielen Vergleichsoperatoren versehen werden wie: `=`, `!=`, `>`, `>=`, `<`, `<=`, `like`, `not like`, `between`, `in`, `is null`, `is not null`, `every`.

Das strenge Sicherheitskonzept in AWS erfordert für die meisten Anfragen eine Authentifizierung des Nutzers. Damit wird sichergestellt, dass der Nutzer die nötigen Berechtigun-

gen für die entsprechende Anfrage besitzt und der Account nicht durch unerlaubte Anfragen mit Kosten belastet wird. Beim Zugriff auf SimpleDB mittels SOAP ohne Nutzung von WS-Security[17] oder REST erfolgt die Authentifizierung mittels der folgenden Komponenten:

- *Access Key ID* – zur Identifizierung des Accounts.
- *Signature* – Aus *Secret Access Key* und Inhalt berechnete HMAC-SHA[18]-Signatur.
- *Date* – Jede Anfrage benötigt je nach verwendeter API einen Zeitstempel oder eine Zeitspanne in Form eines `dateTime`[19]-Objektes, in der die Abfrage erfolgen muss. Bei Angabe eines Zeitstempels muss die Anfrage innerhalb von 15 Minuten erfolgen, hier ist auf die Zeit des AWS-Servers zu achten.

Der generelle Ablauf zur Authentifizierung ist dann der folgende:

1. Erstellung der Anfrage an SimpleDB mit Zeitstempel.
2. Berechnung der Signatur aus *Secret Access Key* und Inhalt der Anfrage.
3. *Access Key ID* und Signatur wird in die Anfrage integriert und an SimpleDB gesendet. Bei der Verwendung von REST erfolgt dieses über HTTP oder HTTPS, und bei der Verwendung von SOAP wird über HTTPS gesendet.
4. Bei einer Nutzung von SOAP mit WS-Security wird ein X.509-Zertifikat zur Authentifizierung benötigt. Auch das kann über den eigenen Amazon Account erstellt werden[20].

Wie schon im Steckbrief beschrieben, werden für die AWS-Infrastruktur APIs für die folgenden Sprachen bereitgestellt:

- Java – AWS SDK for Java: *http://aws.amazon.com/sdkforjava/*
- Ruby Developer Center: *http://aws.amazon.com/ruby/*
- AWS Toolkit for Eclipse: *http://aws.amazon.com/eclipse/*
- C# – AWS SDK for .NET: *http://aws.amazon.com/sdkfornet/*
- Perl – Perl Library for Amazon SimpleDB: *http://aws.amazon.com/code/1136*
- PHP – PHP Library for Amazon SimpleDB: *http://aws.amazon.com/sdkforphp/*
- Python Developer Center: *http://aws.amazon.com/python/*
- JavaScript/AJAX – Scratchpad for Amazon SimpleDB:*http://aws.amazon.com/code/1137*

Die letztgenannte Anwendung *Scratchpad for Amazon SimpleDB* ist ein einfaches HTML- und JavaScript-Web-Interface, das es erlaubt, die API von SimpleDB ohne zusätzlichen Code zu testen. Um SimpleDB mit Scratchpad zu erkunden, müssen lediglich die unter Apache License 2.0 stehenden gepackten Dateien der Anwendung heruntergeladen und in einem beliebigen Verzeichnis entpackt werden. Gestartet wird die Anwendung durch einfaches Aufrufen der Datei *index.html* im Dateiordner */AmazonSimpleDB-2009-04-15-scratchpad/webapp/*. Wie in Abbildung 3.3.1 zu sehen ist, kann die API über das Menü *Explore*

17 WebService Security, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

18 Keyed-Hashing for Message Authentication, http://www.ietf.org/rfc/rfc2104.txt

19 XML-dateTime, http://www.w3.org/TR/xmlschema-2/#dateTime

20 Vgl. Amazon SimpleDB Developer Guide (API Version 2009-04-15), Seite 28

API erkundet werden. Zur Authentifizierung wird eine *Access Key ID* und ein gemeinsames Geheimnis in Form des *Secret Access Key* benötigt.

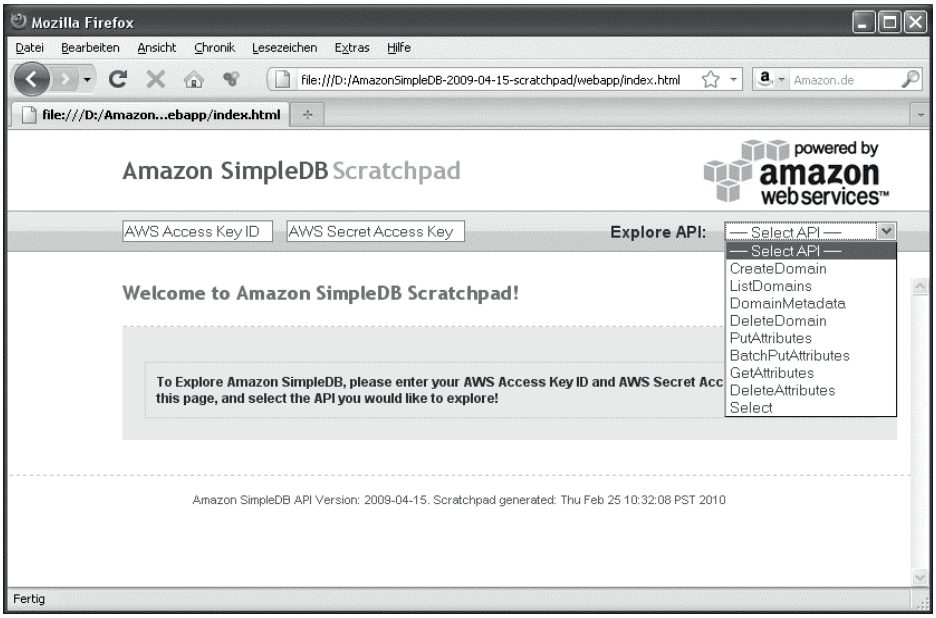

Abbildung 3.3.1 Amazon SimpleDB: Scratchpad

Create-Operation

Das Erzeugen und Befüllen einer Domäne über das Web-Interface Scratchpad ist recht einfach und erklärt sich weitgehend selbst. Das Anlegen einer Domäne erfolgt durch Auswahl von `CreateDomain` über das erwähnte Menü. In der erscheinenden Ansicht kann der gewünschte Name der zu erzeugenden Domäne, z.B. `my_first_simpledb`, direkt angegeben werden. Es stehen nun jeweils mehrere Möglichkeiten zur Erkundung der API zur Verfügung:

- *Display String to Sign* – ermöglicht die Anzeige der zur Authentifizierung zu signierenden Zeichenkette:

```
ActionCreateDomainAWSAccessKeyId< Access Key ID >DomainName
my_first_simpledbSignatureVersion1Timestamp2010-07-27T14:17:12.000Z Versi-
    on2009-04-15
```

- *Display Signed URL* – ermöglicht die Anzeige der signierten Anfrage:

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=CreateDomain
&Version=2009-04-15
&DomainName=my_first_simpledb
&Timestamp=2010-07-27T14%3A18%3A55.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Diese REST-Anfrage kann kopiert und direkt in den Browser eingefügt und an den SimpleDB-Account gesendet werden. Amazon SimpleDB wird eine entsprechende Antwort zurückliefern.

- *Invoke Request* – sendet die signierte Anfrage über Scratchpad an den SimpleDB-Account. Amazon SimpleDB liefert eine entsprechende Antwort zurück:

```
<CreateDomainResponse>
 <ResponseMetadata>
    <RequestId>b2350602-de5d-47c6-0130-4b972f9e0392</RequestId>
    <BoxUsage>0.0055590278</BoxUsage>
 </ResponseMetadata>
</CreateDomainResponse>
```

- *Reset Form* – setzt die Anfrage auf den Urzustand zurück.

Diese Möglichkeiten stehen grundsätzlich beim Erkunden aller Operationen mit dem Scratchpad bereit. Die dabei erzeugten REST-Aufrufe werden mit GET oder POST an SimpleDB übertragen. Der Action-Parameter bestimmt die Methode, die aufgerufen wird. Die Antwort auf eine Anfrage ist jeweils ein XML-Dokument. Es beinhaltet neben den Ergebnissen der jeweiligen Abfrage auch immer den *Tag* `ResponseMetadata`. Er beinhaltet eine `RequestID` zum Verfolgen der Nachricht sowie eine Bewertungszahl, die die Maschinenauslastung ohne Speicherung und Transferkosten für die jeweilige Anfrage wiedergibt. Dieser Benchmark dient der Optimierung der eigenen Abfragen. Ein hoher Box*Usage*-Wert zeigt, dass die Anfrage kostenintensiv bezüglich der Maschinenauslastung ist.

Über die Operation `ListDomain` lassen sich alle erzeugten Domänen anzeigen. Hierzu ist im Scratchpad zur Limitierung der Antwort die maximale Anzahl von Domänen anzugeben. Sollten mehrere Domänen vorhanden sein, so liefert SimpleDB ein `Token` zurück, über den die noch nicht dargestellten Domänen angezeigt werden können. Scratchpad bietet hierfür die Eingabezeile `Next Token` an. Die signierte Anfrage und zurückgelieferte Antwort auf `ListDomain` sind in Listing 3.3.2 und Listing 3.3.3 zu sehen.

Listing 3.3.2 Signed URL für ListDomains

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=ListDomains
&Version=2009-04-15
&MaxNumberOfDomains=5
&Timestamp=2010-07-27T16%3A04%3A30.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Listing 3.3.3 Antwort auf ListDomain-Operation

```
<ListDomainsResponse>
 <ListDomainsResult>
    <DomainName>my_first_simpledb</DomainName>
    <DomainName>my_second_simpledb</DomainName>
 </ListDomainsResult>
 <ResponseMetadata>
    <RequestId>cd95079c-7e31-c2eb-4a4a-774a52556318</RequestId>
```

```
    <BoxUsage>0.0000071759</BoxUsage>
  </ResponseMetadata>
</ListDomainsResponse>
```

Zum Befüllen der Domäne mit Daten stehen die Operationen `PutAttributes` und `BatchPutAttributes` im Menü zur Verfügung. `BatchPutAttributes` ermöglicht im Gegensatz zu `PutAttributes` die Übertragung von Attribut-Wert-Paaren für mehrere Elemente. Scratchpad stellt dazu die in Abbildung 3.3.2 dargestellte Eingabemaske bereit.

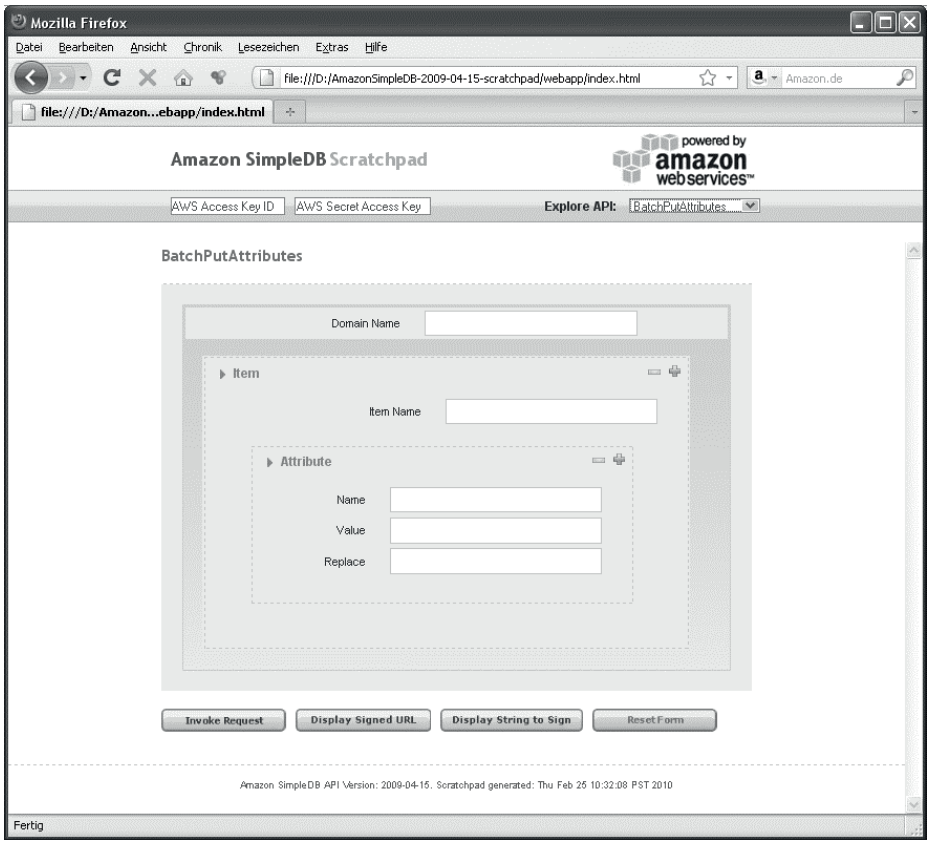

Abbildung 3.3.2: Amazon SimpleDB: Scratchpad, BatchPutAttributes

Über die Plus- und Minus-Buttons können die gewünschten Elemente und Attribut-Wert-Paare hinzugefügt werden. Im Eingabefeld `Replace` ist jeweils der Wert `false` einzugeben. Mit der in Listing 3.3.4 über Scratchpad erzeugten Anweisung wird die Tabelle 3.3.1 in der Domäne `my_first_simpledb` erzeugt.

Tabelle 3.3.1 my_first_simpledb-Inhalte

ItemName	Typ	Name	Quantity	Supplier
Element_01	Fruit	Banana	4200	
Element_02	Fruit	Apple	5300	
Element_03	Fruit	Cherry	20000 10000	Food Ltd. Amarena Ltd.

Listing 3.3.4 Signed URL für BatchPutAttributes

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=BatchPutAttributes&Version=2009-04-15
&DomainName=my_first_simpledb
&Item.1.ItemName=Element_01
&Item.1.Attribute.1.Name=Typ
&Item.1.Attribute.1.Value=Fruit
&Item.1.Attribute.1.Replace=false
&Item.1.Attribute.2.Name=Name
&Item.1.Attribute.2.Value=Banana
&Item.1.Attribute.2.Replace=false
&Item.1.Attribute.3.Name=Quantity
&Item.1.Attribute.3.Value=4200
&Item.1.Attribute.3.Replace=false
&Item.2.ItemName=Element_02
&Item.2.Attribute.1.Name=Typ
&Item.2.Attribute.1.Value=Fruit
&Item.2.Attribute.1.Replace=false
&Item.2.Attribute.2.Name=Name
&Item.2.Attribute.2.Value=Apple
&Item.2.Attribute.2.Replace=false
&Item.2.Attribute.3.Name=Quantity
&Item.2.Attribute.3.Value=5300
&Item.2.Attribute.3.Replace=false
&Item.3.ItemName=Element_03
&Item.3.Attribute.1.Name=Typ
&Item.3.Attribute.1.Value=Fruit
&Item.3.Attribute.1.Replace=false
&Item.3.Attribute.2.Name=Name
&Item.3.Attribute.2.Value=Cherry
&Item.3.Attribute.2.Replace=false
&Item.3.Attribute.3.Name=Quantity
&Item.3.Attribute.3.Value=20000
&Item.3.Attribute.3.Replace=false
&Item.3.Attribute.4.Name=Quantity
&Item.3.Attribute.4.Value=10000
&Item.3.Attribute.4.Replace=false
&Item.3.Attribute.5.Name=Supplier
&Item.3.Attribute.5.Value=Food%20Ltd.
&Item.3.Attribute.5.Replace=false
&Item.3.Attribute.6.Name=Supplier
&Item.3.Attribute.6.Value=Amarena%20Ltd.
&Item.3.Attribute.6.Replace=false
&Timestamp=2010-07-28T13%3A26%3A56.000Z
&AWSAccessKeyId=[valid Access Key ID]&Signature=[valid Signature]
```

Listing 3.3.5 Antwort auf die BatchPutAttributes-Operation

```
<BatchPutAttributesResponse>
 <ResponseMetadata>
    <RequestId>3d6d33c8-8355-4cdc-02f1-e0cf6c77b4b3</RequestId>
    <BoxUsage>0.0000462183</BoxUsage>
 </ResponseMetadata>
</BatchPutAttributesResponse>
```

Im `Element_03` wurde das zusätzliche Attribut `Supplier` hinzugefügt, ebenso wurden den Attributen `Quantity` und `Supplier` jeweils zwei Werte zugewiesen. Diese Flexibilität spiegelt einen entscheidenden Vorteil von SimpleDB wider. Wie bei einem Telefonbuch, das man in einem Texteditor führt, gibt es bei manchen Personen keinen Geburtstag, keine E-Mail, keine Handynummer usw. Manchmal aber eben doch. Entscheidend sind dabei nicht die eventuell fehlenden Werte, die auch bei relationalen Datenbanken üblich sind, sondern die selbständige Erweiterung des Schemas bei neuen Daten.

Read-Operation

Für das Lesen der Daten stehen die zwei Operationen `GetAttributes` und `Select` bereit. Mit `GetAttributes` können die Attribute eines Elementes gezielt abgefragt werden, wie in Listing 3.3.6 und Listing 3.3.7 wiedergegeben wird. Hier werden die Attributwerte für `Quantity` und `Supplier` vom `Element_03` abgefragt. Durch Setzen von `ConsistentRead=[true|false]` kann dabei zwischen *Eventually Consistent Read* und *Consistent Read* gewählt werden.

Listing 3.3.6 Signed URL für GetAttributes

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=GetAttributes
&Version=2009-04-15
&DomainName=my_first_simpledb
&ItemName=Element_03
&AttributeName.1=Quantity
&AttributeName.2=Supplier
&ConsistentRead=true
&Timestamp=2010-07-28T16%3A53%3A21.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Listing 3.3.7 Antwort auf GetAttributes

```
<GetAttributesResponse>
 <GetAttributesResult>
    <Attribute>
      <Name>Quantity</Name>
      <Value>10000</Value>
    </Attribute>
    <Attribute>
      <Name>Quantity</Name>
      <Value>20000</Value>
    </Attribute>
    <Attribute>
```

```
      <Name>Supplier</Name>
      <Value>Amarena Ltd.</Value>
    </Attribute>
    <Attribute>
      <Name>Supplier</Name>
      <Value>Food Ltd.</Value>
    </Attribute>
 </GetAttributesResult>
 <ResponseMetadata>
    <RequestId>457fd438-e4f4-b7a6-2560-72797581e0b3</RequestId>
    <BoxUsage>0.0000093282</BoxUsage>
 </ResponseMetadata>
</GetAttributesResponse>
```

Die `Select`-Operation entspricht der gleichen Operation in SQL, stellt aber ein geringeres Subset dar. Der generelle Aufbau der Select-Abfrage wurde bereits in Listing 3.3.1 gezeigt. Eine einfache Abfrage in der Form `select * from my_first_simpledb where Supplier='Amarena Ltd.'` wird in Listing 3.3.8 und Listing 3.3.9 gezeigt.

Listing 3.3.8 Signed URL für Select

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=Select&Version=2009-04-15
&SelectExpression=select%20*%20from%20my_first_simpledb%20
                  where%20Supplier%3D'Amarena%20Ltd.'
&ConsistentRead=true
&Timestamp=2010-07-28T18%3A19%3A16.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Listing 3.3.9 Antwort auf einfaches Select

```
<SelectResponse>
 <SelectResult>
    <Item>
      <Name>Element_03</Name>
      <Attribute>
        <Name>Name</Name>
        <Value>Cherry</Value>
      </Attribute>
      <Attribute>
        <Name>Supplier</Name>
        <Value>Amarena Ltd.</Value>
       </Attribute>
       <Attribute>
         <Name>Supplier</Name>
         <Value>Food Ltd.</Value>
       </Attribute>
       <Attribute>
         <Name>Quantity</Name>
         <Value>10000</Value>
       </Attribute>
       <Attribute>
         <Name>Quantity</Name>
         <Value>20000</Value>
       </Attribute>
```

```
        <Attribute>
          <Name>Typ</Name>
          <Value>Fruit</Value>
        </Attribute>
    </Item>
 </SelectResult>
 <ResponseMetadata>
    <RequestId>d943e992-18f9-fc7b-91fb-d34d65e471bc</RequestId>
    <BoxUsage>0.0000228616</BoxUsage>
 </ResponseMetadata>
</SelectResponse>
```

Update-Operation

Ein Update der Attribute erfolgt auch über die Operationen `BatchPutAttributes` und `PutAttributes`, hierzu ist lediglich der Parameter `Replace=true` zu setzen, so wie im Beispiel in Listing 3.3.10 und Listing 3.3.11 zu sehen ist. SimpleDB bietet auch die Möglichkeit, ein *Conditional PutAttributes* durchzuführen. Einfügen oder Ersetzen von Werten für ein oder mehrere Attribute eines Elements erfolgt dann nur, wenn der vorhandene Wert eines Attributs den vorgegebenen Bedingungen entspricht. Hiermit kann das bei parallelen Schreibzugriffen auftretende *Lost-Update*-Problem vermieden werden. Ein *Conditional PutAttributes* geht allerdings nur bei Attributen, die einzelne Werte enthalten. Die Bedingungen, das heißt die erwarteten Werte sind wie in Listing 3.3.12 in den Parametern für `Expected` anzugeben. Im Beispiel wird im `Element_02` der `Supplier='Malus Ltd.'` Gesetzt, wenn `Name='Apple'` existiert.

Listing 3.3.10 Signed URL für PutAttributes mit Replace=true

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=PutAttributes
&Version=2009-04-15
&DomainName=my_first_simpledb
&ItemName=Element_01
&Attribute.1.Name=Quantity&Attribute.1.Value=5000
&Attribute.1.Replace=true
&Timestamp=2010-07-28T19%3A35%3A44.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Listing 3.3.11 Antwort auf PutAttributes mit Replace=true

```
<PutAttributesResponse>
 <ResponseMetadata>
    <RequestId>f4641617-73c5-55b6-48ea-6d8b9a6f27a3</RequestId>
    <BoxUsage>0.0000219909</BoxUsage>
 </ResponseMetadata>
</PutAttributesResponse>
```

Listing 3.3.12 Signed URL für Conditional PutAttributes

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=PutAttributes
&Version=2009-04-15
```

```
&DomainName=my_first_simpledb
&ItemName=Element_02
&Attribute.1.Name=Supplier
&Attribute.1.Value=Malus%20Ltd.
&Attribute.1.Replace=true
&Expected.Name=Name
&Expected.Value=Apple
&Expected.Exists=true
&Timestamp=2010-07-30T16%3A48%3A03.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Listing 3.3.13 Antwort auf Conditional PutAttributes

```
<PutAttributesResponse>
 <ResponseMetadata>
    <RequestId>9e42232c-97ea-2d20-cec1-21a385337e24</RequestId>
    <BoxUsage>0.0000219909</BoxUsage>
 </ResponseMetadata>
</PutAttributesResponse>
```

Delete-Operation

Das Löschen von Werten, Attributen und Domänen erfolgt, wie der Name der Operationen schon verrät, mit `DeleteAttributes` und `DeleteDomain`, dieses wird in Listing 3.3.14 bis Listing 3.3.19 dargestellt. Auch die Löschoperationen können als *Conditional Delete* erfolgen, aber allerdings auch nur bei Attributen mit nur einem Wert. Über die Operation `BatchPutAttributes` ist es möglich, mit nur einem Aufruf mehrere Attribute zu löschen. Diese Operation wird aktuell im Scratchpad nicht unterstützt.

Listing 3.3.14 Signed URL für DeleteAttributes Value

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=DeleteAttributes
&Version=2009-04-15
&DomainName=my_first_simpledb
&ItemName=Element_03
&Attribute.1.Name=Quantity
&Attribute.1.Value=20000
&Timestamp=2010-07-30T17%3A17%3A51.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Listing 3.3.15 Antwort auf DeleteAttributes Value

```
<DeleteAttributesResponse>
 <ResponseMetadata>
    <RequestId>f1a042d8-f0f2-972e-6576-1c49f4810a80</RequestId>
    <BoxUsage>0.0000219909</BoxUsage>
 </ResponseMetadata>
</DeleteAttributesResponse>
```

Listing 3.3.16 Signed URL für DeleteAttributes

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=DeleteAttributes
&Version=2009-04-15
&DomainName=my_first_simpledb
&ItemName=Element_03
&Attribute.1.Name=Supplier
&Timestamp=2010-07-30T17%3A22%3A44.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Listing 3.3.17 Antwort auf DeleteAttributes

```
<DeleteAttributesResponse>
 <ResponseMetadata>
    <RequestId>7cbfada9-9ecd-0b3a-666e-3f0e522905f5</RequestId>
    <BoxUsage>0.0000219909</BoxUsage>
 </ResponseMetadata>
</DeleteAttributesResponse>
```

Listing 3.3.18 Signed URL für DeleteDomain

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=DeleteDomain
&Version=2009-04-15
&DomainName=my_second_simpledb
&Timestamp=2010-07-30T17%3A33%3A19.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Listing 3.3.19 Antwort auf DeleteDomain

```
<DeleteDomainResponse>
 <ResponseMetadata>
    <RequestId>416b3ffe-a4c4-02e6-6e5b-e57a6e132d32</RequestId>
    <BoxUsage>0.0055590278</BoxUsage>
 </ResponseMetadata>
</DeleteDomainResponse>
```

Metadatenabfrage

Über die Operation `DomainMetadata` können Informationen über eine angegebene Datenbank abgerufen werden. Diese Daten beinhalten Informationen über die Anzahl der Elemente und Attribute und die Größe der Attributnamen und -werte.

Listing 3.3.20 Signed URL für DomainMetadata

```
https://sdb.amazonaws.com?SignatureVersion=1
&Action=DomainMetadata
&Version=2009-04-15
&DomainName=my_first_simpledb
&Timestamp=2010-07-30T17%3A26%3A18.000Z
&AWSAccessKeyId=[valid Access Key ID]
&Signature=[valid Signature]
```

Listing 3.3.21 Antwort auf DomainMetadata-Anfrage

```
<DomainMetadataResponse>
 <DomainMetadataResult>
    <ItemCount>3</ItemCount>
    <ItemNamesSizeBytes>30</ItemNamesSizeBytes>
    <AttributeNameCount>3</AttributeNameCount>
    <AttributeNamesSizeBytes>15</AttributeNamesSizeBytes>
    <AttributeValueCount>9</AttributeValueCount>
    <AttributeValuesSizeBytes>45</AttributeValuesSizeBytes>
    <Timestamp>1280510861</Timestamp>
 </DomainMetadataResult>
 <ResponseMetadata>
    <RequestId>ab3102ae-e1a4-cd3a-4d87-06d2142c3f0a</RequestId>
    <BoxUsage>0.0000071759</BoxUsage>
 </ResponseMetadata>
</DomainMetadataResponse>
```

3.3.6 Zugriff mit dem AWS Toolkit für Eclipse

Amazon bietet für den Zugriff auf seine Services eine Vielzahl von Sprachunterstützungen an. Ein Beispiel hierfür ist das AWS Toolkit für Eclipse. Unter anderem unterstützt es die Entwicklung von Anwendungen in Java für SimpleDB und ermöglicht darüber hinaus das Management der Datenbank. Die Installation des Toolkits erfolgt über die für Eclipse übli-

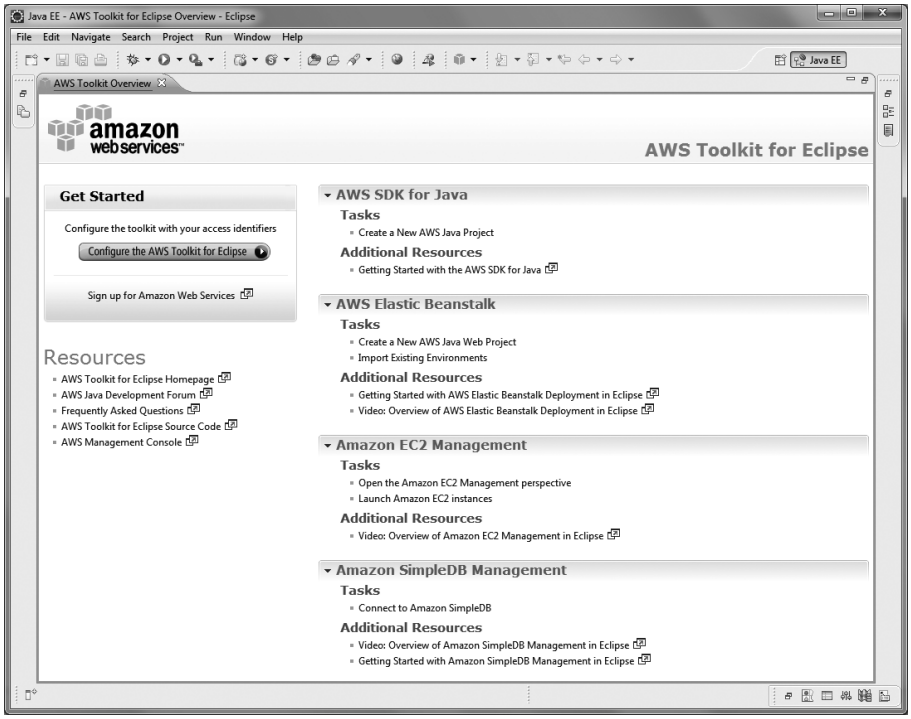

Abbildung 3.3.3 AWS Eclipse Toolkit

che Art der Plugins. Die Seite http://aws.amazon.com/eclipse/ liefert dazu das gewünschte AWS Toolkit. Vorausgesetzt wird eine Eclipse IDE ab Version 3.5 mit einer *Data Tools Platform Version 1.7* oder höher, so wie in der *Eclipse IDE for Java EE Developers* bereits enthalten. Nach erfolgter Installation des Toolkits stehen einem, wie auch in Abbildung 3.3.3 (vorige Seite) zu sehen ist, viele Hilfen und Videotutorials zur Verfügung. Die Anmeldung und der Umgang mit dem Toolkit werden dadurch ausführlich erklärt.

Amazon SimpleDB Management

Über das Amazon SimpleDB Management Plugin kann man sich mit seinem Account der SimpleDB verbinden und die Datenbank auf komfortable Weise verwalten und bearbeiten. Dazu werden die folgenden Perspektiven bereitgestellt:

- DataSource Explorer – dient der Navigation innerhalb der Datenbanken, dem Datenexport, dem Ausführen von CRUD-Operationen und zum Aufrufen des SQL Scrapbooks.
- SQLScrapbook – ermöglicht die Ausführung von SELECT Befehlen.
- SQL Results View – stellt die Ergebnisse der Anfragen dar.
- Tabulator Data Editor – ermöglicht das einfache Editieren von Daten innerhalb der Domain.

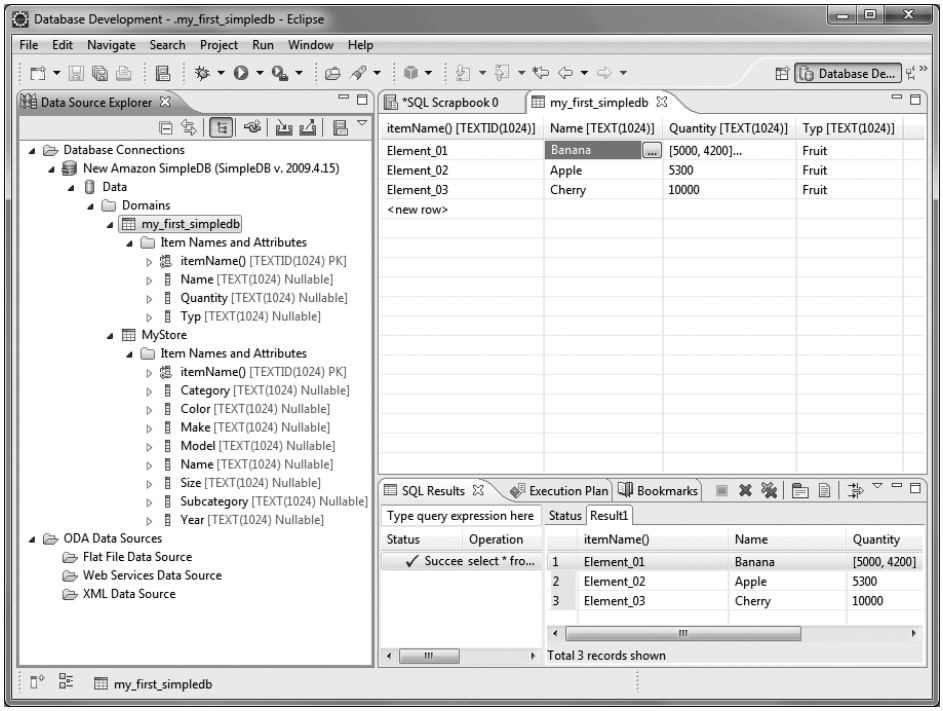

Abbildung 3.3.4 AWS Eclipse SimpleDB Management

AWS SDK für Java

Im AWS SDK für Java befindet sich unter anderem auch ein einfaches Beispiel für die SimpleDB, mit dem die grundlegenden CRUD-Operationen aufgezeigt werden. Ein AWS Java Projekt kann nun über den Menüpunkt *File* und dann weiter mit *New*, *Other* und *AWS* neu angelegt werden. Durch Auswahl von AWS Java Project gelangt man zum *Eclipse Project Wizard*, hier müssen Projektname, *Acces Key* und *Secret Key* angegeben werden. Durch Auswahl von *Amazon SimpleDB Sample* und Drücken des Buttons *Finish* wird das im Toolkit enthaltene Beispiel in Eclipse erzeugt. In der Java EE Perspektive kann dies nun bearbeitet und über den Menüpunkt Run ausgeführt werden.

CRUD-Operationen mit Java

Im Folgenden werden die wesentlichen Punkte der Java API kurz dargestellt. Der Zugriff auf eine SimpleDB Datenbank erfolgt über ein Objekt der Klasse `AmazonSimpleDB`. Die benötigten Keys sind in der Datei `AwsCredentials.properties` enthalten:

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.simpledb.AmazonSimpleDB;
import com.amazonaws.services.simpledb.AmazonSimpleDBClient;
..
AmazonSimpleDB sdb = new AmazonSimpleDBClient(new PropertiesCredentials(
 SimpleDBSample.class.getResourceAsStream("AwsCredentials.properties")));
```

- Erzeugen einer Domain:

```
import com.amazonaws.services.simpledb.model.CreateDomainRequest;
..
sdb.createDomain(new CreateDomainRequest("mySecondDomain"));
```

- Befüllen der Domain:

```
import com.amazonaws.services.simpledb.model.ReplaceableItem;
import com.amazonaws.services.simpledb.model.ReplaceableAttribute;
import com.amazonaws.services.simpledb.model.BatchPutAttributesRequest;
..
List<ReplaceableItem> sampleData = new ArrayList<ReplaceableItem>();
sampleData.add(new ReplaceableItem("Element01").withAttributes(
                new ReplaceableAttribute("Name", "Apple", true),…)));
..
sdb.batchPutAttributes(new BatchPutAttributesRequest(“mySecondDomain“,
   sampleData));
```

- Abfrage mit Select:

```
import com.amazonaws.services.simpledb.model.SelectRequest;
..
String select;
select = "select * from `" + "mySecondDomain" + "` where Name = 'Apple'";
SelectRequest selectRequest = new SelectRequest(select);
```

- Update eines Attributs:

```
import com.amazonaws.services.simpledb.model.ReplaceableAttribute;
import com.amazonaws.services.simpledb.model.PutAttributesRequest;
..
```

```
List<ReplaceableAttribute> updateAttributes;
updateAttributes = new ArrayList<ReplaceableAttribute>();
updateAttributes.add(new ReplaceableAttribute("Name", "Cherry", true));
sdb.putAttributes(new PutAttributesRequest(“mySecondDomain“, „Element01“,
    updateAttributes));
```

- Löschen eines gesamten Eintrages – gesamtes Item

```
import com.amazonaws.services.simpledb.model.DeleteAttributesRequest;
..
sdb.deleteAttributes(new DeleteAttributesRequest("mySecondDomain",
                     "Element01"));
```

- Löschen der Domain:

```
import com.amazonaws.services.simpledb.model.DeleteDomainRequest;
..
sdb.deleteDomain(new DeleteDomainRequest("mySecondDomain"));
```

3.3.7 Replikation und Skalierung

SimpleDB wird im Kontext von AWS als SaaS bereitgestellt. Die Indizierung der Daten sowie die Erstellung geografisch redundanter Replikationen erfolgt automatisch. Dadurch wird eine hohe Verfügbarkeit gewährleistet. Ebenso erfolgt eine automatische Skalierung bei einer Änderung des Aufkommens von Anfragen und der Datenbanknutzung. Eine horizontale Skalierung kann durch Erzeugung neuer Domänen erfolgen.

3.3.8 Bewertung

Mit SimpleDB bietet Amazon ein einfaches Datenbanksystem zur Nutzung innerhalb der AWS-Dienste an. Die Vor- und Nachteile werden im Folgenden zusammengefasst beschrieben.

Vorteile

- Bereitstellung als SaaS in der AWS-Umgebung: Die Bereitstellung der Infrastruktur und Software erfolgt über Amazon, daraus ergibt sich eine sofortige Verfügbarkeit und eine geringere administrative Belastung. Der geringere Verwaltungsaufwand lässt dem Entwickler mehr Zeit, sich auf die Entwicklung der Anwendung zu konzentrieren.
- Einfache, flexible, schemafreie Datenstruktur analog einer Tabellenkalkulation.
- Einfache, gut dokumentierte Schnittstellen und APIs für die gängigen Programmiersprachen.
- Automatische, geografisch verteilte Replizierung und spontane Skalierung zur Gewährleistung einer hohen Verfügbarkeit und eines hohen Durchsatzes. Horizontale Skalierung durch Erstellung neuer Domänen möglich.
- Faire, erschwingliche Preisgestaltung, die nur die tatsächliche Nutzung der Datenbank berechnet.

Nachteile

- Die Daten liegen bei einem Dritten (Datenschutz). Zum Schutz der Daten betreibt Amazon aber ein aufwendiges Sicherheitskonzept.
- Eine direkte Kontrolle über die Datenbank und die Art der Speicherung ist nicht möglich. Im Wesentlichen für den Einsatz in AWS-Kontext konzipiert.
- Daten werden als UTF-8-String gespeichert. Vergleiche von Daten erfolgen nur nach lexikographischer Ordnung. Es entsteht somit mehr Aufwand bei der Nutzung von negativen Zahlen. Hier muss mit einem Offsetwert und einer vergleichbaren Anzahl von Zeichen ein positiver vergleichbarer Wert hergestellt werden. Datum und Zeiten sollten in ISO-8601 konvertiert werden, ISO-8601 unterstützt Vergleiche nach lexikographischer Ordnung.
- Eventuell können sich Nachteile durch die technische Beschränkung der Anzahl von Domänen und Attributen ergeben.

Abschließend ein kurzer Vergleich, wie ihn auch Amazon selbst für alle Datenbanken und Speicherlösungen angibt:

- **S3**: Ist der normaler Speicherservice für unstrukturierte Rohdaten, die üblicherweise größer als SimpleDB-Einträge sind. Auch hier wird nur bei Bedarf bezahlt.
- **SimpleDB**: Skaliert von selbst, beinhaltet eine kleine Blockgröße, erweitert sein Schema von selbst und rechnet nur das verbrauchte Volumen ab.
- **RDS**: Steht für Relational Database Service und liefert eine gemanagte und automatisch skalierende Lösung auf MySQL-Basis an. Hier sind neben dem Skalieren auch der Mehrwert der Einrichtung, automatisierte Backups, Snapshots und Management-Tools interessant.
- **EC2 AIMs**: Amazon selbst stellt im Rahmen seiner EC2-Serverkapazität Images zur Verfügung, die vom Anwender geladen und verwendet werden können. Gespeichert wird dann in Amazons *Elastic Block Store* (EBS). Zur Verfügung stehen hier die Datenbanken IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase und Vertica (wobei Letztere auch als spaltenorientierte NoSQL-Datenbank angesehen werden kann). Vorteil dieser Lösung ist sicherlich die volle Kontrolle über die Datenbank.

Amazon selbst nennt als Beispiele für den Einsatz von SimpleDB Logging-Anwendungen, Online-Spiele oder Metadatenindizierung, weitere wie Failover, Datenverteilung, Replikation, Datenkonsistenz mittels Transaktionen und der Lastverwaltung. SimpleDB ist auch dafür konzipiert, mit anderen AWS-Dienstleistungen wie z.B. Amazon S3 und EC2 zusammenzuarbeiten.

Links

[Müller10] Frank Müller: Es geht auch einfach – Datenbanken ohne SQL und Relationen, Heise Zeitschriften Verlag GmbH, *http://www.heise.de/kiosk/archiv/ix/2010/2/122*

Amazon SimpleDB – *http://aws.amazon.com/de/simpledb/*

Amazon SimpleDB Getting Started Guide (API Version 2009-04-15), *http://awsdocs.s3.amazonaws.com/SDB/latest/sdb-gsg.pdf*

Amazon SimpleDB Developer Guide (API Version 2009-04-15),
http://awsdocs.s3.amazonaws.com/SDB/latest/sdb-dg.pdf

Amazon Web Services – Sicherheitsprozesse im Überblick, November 2009
http://awsmedia.s3.amazonaws.com/de/Whitepaper_AWS_Security_Whitepaper(DE).pdf

Amazon SimpleDB Management in Eclipse,
http://awsdocs.s3.amazonaws.com/Eclipse/getting-started-simpledb-eclipse.pdf