Budapest University of Technology and Economics
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

Service Integration course

Cassandra

Oszkár Semeráth Gábor Szárnyas

August 11, 2014

Contents

Chapter 1

Installation

Apache Cassandra (http://cassandra.apache.org/) is an open source NoSQL database management system based on the column family data model.

Figure 1.1: The logo of Apache Cassandra

1.1 Linux

```
sudo mkdir /var/lib/cassandra
sudo mkdir /var/log/cassandra
sudo chown -R $USER:$USER /var/lib/cassandra
sudo chown -R $USER:$USER /var/log/cassandra
```

1.2 Windows

Set the `JAVA_HOME` environment variable to to the JDK's installation directory (Cassandra will work with the JRE as well, but some other Java-based applications – such as Maven – will not).

Install **Cygwin** with **Python 2.7.x** (if you want to add the package to an existing Cygwin installation, you have to re-run the installer). Navigate to `/cygdrive/c/apache-cassandra-2.0.5`. Run the shell with the following command:

```
$ bin/cqlsh
Connected to Test Cluster at localhost:9160.
[cqlsh 4.1.1 | Cassandra 2.0.5 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
Use HELP for help.
```

Note that Cassandra itself is not supported under Cygwin (http://stackoverflow.com/questions/12355507/error-when-starting-cassandra-with-bin-cassandra-f).

1.3 Documentation pointers

- Official Apache wiki (http://wiki.apache.org/cassandra/). It is not very informative, use the DataStax documentation (http://www.datastax.com/docs) instead.
- CQL documentation: http://www.datastax.com/documentation/cql/3.1/index.html

Chapter 2

Laboratory exercises

2.1 Database design

1. You may always reset the database with the following command:

```
DROP KEYSPACE bicycle_database;
```

2. Create the **keyspace**:

```
CREATE KEYSPACE bicycle_database
WITH replication = {
  'class': 'SimpleStrategy',
  'replication_factor': 1
};
```

3. Create a **column family**:

```
USE bicycle_database;

CREATE TABLE bicycles (
  manufacturer text,
  model text,
  PRIMARY KEY (manufacturer, model)
);

INSERT INTO bicycles (manufacturer, model) VALUES ('KTM', 'Strada 1000');
```

4. Run a simple query:

```
SELECT * FROM bicycles;
```

5. Add some columns.

```
ALTER TABLE bicycles ADD year int;
ALTER TABLE bicycles ADD size int;
```

6. Insert a few lines.

```
INSERT INTO bicycles (manufacturer, model, year, size) VALUES ('KTM', 'Strada 1000', 2014, 57);
INSERT INTO bicycles (manufacturer, model, year, size) VALUES ('Trek', '1.2', 2013, 54);
```

7. Run a simple query again:

```
cqlsh:bicycle_database> SELECT * FROM bicycles;

 manufacturer | model       | size | year
--------------+-------------+------+------
          KTM | Strada 1000 |   57 | 2014
         Trek |         1.2 |   54 | 2013

(2 rows)
```

8. We would like to collect the bicycles with the frame size of 54:

```
SELECT * FROM bicycles WHERE size = 54;
```

This throws the following error: `Bad Request: No indexed columns present in by-columns clause with Equal operator.`

9. Create indices.

```
CREATE INDEX bicycles_size ON bicycles (size);
CREATE INDEX bicycles_year ON bicycles (year);
```

10. Re-run the query. It may not return any results as the rows were added *before* the index was created. Add the rows again and the query will work fine.

11. Try a query which filters for inequality:

```
SELECT * FROM bicycles WHERE year > 2013;
```

12. This also throws an error:

```
Bad Request: No indexed columns present in by-columns clause with Equal operator
```

13. Add an equality filter as well:

```
SELECT * FROM bicycles WHERE year > 2013 AND size = 57;
```

This time, the error message is the following: `Bad Request: Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING`

14. Allow filtering:

```
SELECT * FROM bicycles WHERE year > 2013 AND size = 57 ALLOW FILTERING;
```

This query works just fine. For details, see http://wiki.apache.org/cassandra/SecondaryIndexes.

15. Try using an `OR` logical operation in the query:

```
SELECT * FROM bicycles WHERE size = 54 OR size = 57;
```

The parser does not even recognize the operation: `Bad Request: line 1:39 missing EOF at 'OR'.`

16. CQL3 supports collections – let's use them for storing the number of cogs:

```
ALTER TABLE bicycles ADD front_cogs list<int>;
ALTER TABLE bicycles ADD rear_cogs list<int>;
```

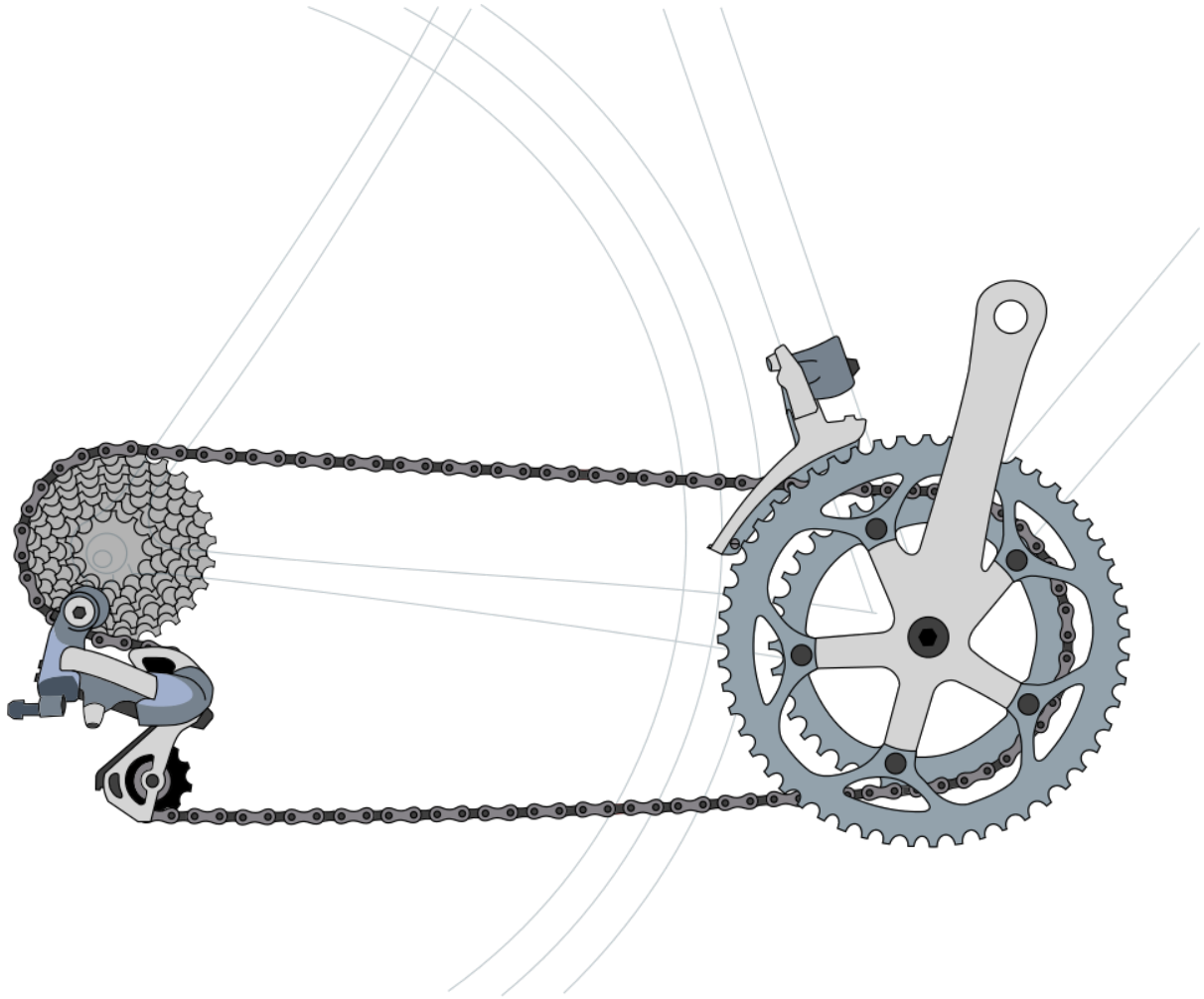

Figure 2.1: Drivetrain. Source: http://commons.wikimedia.org/wiki/File:Derailleur_Bicycle_Drivetrain.svg

17. Update the KTM bicycle with an `INSERT` operation (source: http://www.ktm-bikes.at/bikes/road.html?action=bike_details&bike_id=160&cHash=dbfc043b8dd2263b610187cf9d093fc7).

```
INSERT INTO bicycles (manufacturer, model, front_cogs, rear_cogs)
VALUES ('KTM', 'Strada 1000', [50, 34], [12, 13, 14, 15, 17, 19, 21, 23, 25, 28]);
```

Query it, the result should look like this:

```
 manufacturer | model       | front_cogs | rear_cogs                                  | size | year
--------------+-------------+------------+--------------------------------------------+------+------
          KTM | Strada 1000 |   [50, 34] | [12, 13, 14, 15, 17, 19, 21, 23, 25, 28] |   57 | 2014
         Trek |         1.2 |       null |                                       null |   54 | 2013
```

2.2 Java client

1. Create a new **Maven project**. Choose the `org.apache.maven.archetypes:maven-archetype-quickstart` archetype. Set the **Group Id** to `hu.bme.mit.sysint` and the **Artifact Id** to `client`.
2. Go to the `App` class and run it.
3. Go to the `pom.xml` file. The `Ctrl+F11` hotkey does not run the application if we are on the POM file. Solution: go to **Window | Preferences | Run/Debug | Launching**, in the **Launch Operation** groupbox choose **Always launch the previously launched application**.
4. Add the DataStax driver dependency (https://github.com/datastax/java-driver):

```
<dependency>
  <groupId>com.datastax.cassandra</groupId>
  <artifactId>cassandra-driver-core</artifactId>
  <version>2.0.0</version>
</dependency>
```

5. Create the `CassandraClient` class.

```
private static final String KEYSPACE_NAME = "bicycle_database";
protected Session session;
protected Cluster cluster;

public CassandraClient(String host) {
  cluster = Cluster.builder().addContactPoints(host).build();
  session = cluster.connect("system");
}
```

6. Instantiate the `CassandraClient` class from the `main` method:

```
public class App {
    private static final String CASSANDRA_HOST = "127.0.0.1";

    public static void main(String[] args) {
        client = new CassandraClient(CASSANDRA_HOST);
        client.load();
    }
}
```

7. The following warning is shown.

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

8. Pick a logger implementation, e.g. **log4j** and search for its Maven dependency:

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.7.6</version>
</dependency>
```

9. Run the project.

10. The following warning is displayed:

```
log4j:WARN No appenders could be found for logger (com.datastax.driver.core.FrameCompressor).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

11. Create `log4j.properties` file in the `src/main/java` folder. The content should look something like this:

```
# Root logger option
log4j.rootLogger=INFO, stdout

# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

    Source: http://www.mkyong.com/logging/log4j-log4j-properties-examples/

12. If you'd like to hide the warning messages (e.g. for a production system), change the value of the `log4j.rootLogger` property from `INFO` to `ERROR`.

13. Alternatively, we may add the missing libraries:

```
<dependency>
  <groupId>org.xerial.snappy</groupId>
  <artifactId>snappy-java</artifactId>
  <version>1.1.0.1</version>
</dependency>

<dependency>
  <groupId>net.jpountz.lz4</groupId>
  <artifactId>lz4</artifactId>
  <version>1.2.0</version>
</dependency>
```

14. Use the following commands in the CQL shell:

```
DESCRIBE KEYSPACE bicycle_database;
DESCRIBE TABLE bicycles;
```

15. The application does not stop after it finishes its operations. This is because is has open connections. To solve this, implement the `Closeable` interface in the `CassandraClient` class:

```
public class CassandraClient implements Closeable {

    [...]

    public void close() {
      session.close();
      cluster.close();
    }
}
```

Java 7 has a feature similar to the `using` block in C#.

```
public static void main(String[] args) {
    try (CassandraClient client = new CassandraClient(CASSANDRA_HOST)) {
        client.load();
    } finally {
    }
```

16. We start the load with dropping the keyspace.

```
session.execute("DROP KEYSPACE bicycle_database");
```

This works fine for the first time, however, it throws an exception if the keyspace does not exist:

```
Exception in thread "main" com.datastax.driver.core.exceptions.InvalidQueryException:
Cannot drop non existing keyspace 'bicycle_database'.
```

17. While the Javadoc of the `Session` interface's `execute()` method shows that it may throw an exception, the application is not required to catch this exception as it is a `RuntimeException`. For details, see the Javadoc of the `RuntimeException` class.

```
try {
  session.execute("DROP KEYSPACE bicycle_database");
} catch (InvalidQueryException e) {
  System.out.println("Cannot drop keyspace.");
}
```

A more elegant solution is to use:

```
DROP KEYSPACE IF EXISTS bicycle_database;
```

18. Create the keyspace:

```
String createKeyspace = "CREATE KEYSPACE " + KEYSPACE_NAME
    + " WITH replication = {\r\n"
    + "  'class': 'SimpleStrategy',\r\n"
    + "  'replication_factor': '1'\r\n" + "};";
session.execute(createKeyspace);
session.execute("USE " + KEYSPACE_NAME);
```

19. Create the table. Autoformatting may be messy here. To correct this, Go to **Window** | **Preferences** | **Java** | **Code Style** | **Formatter**. Create a new profile (e.g. `Eclipse FTSRG`) and edit it. Go to the **Off/On Tags** tab and check **Enable Off/On** tags.

```
// @formatter:off
String createTable = "CREATE TABLE bicycles (\r\n" +
    "  manufacturer text,\r\n" +
    "  model text,\r\n" +
    "  front_cogs list<int>,\r\n" +
    "  rear_cogs list<int>,\r\n" +
    "  size int,\r\n" +
    "  year int,\r\n" +
    "  PRIMARY KEY (manufacturer, model)\r\n" +
    ")";
session.execute(createTable);
// @formatter:on
```

20. Implement a method for creating indices:

```
protected void createIndex(String columnFamily, String columnName) {
    String createIndex = String.format("CREATE INDEX %s_%s ON %s (%s)",
    columnFamily, columnName, columnFamily, columnName);
    session.execute(createIndex);
}
```

Invoke the method:

```
createIndex("bicycles", "size");
createIndex("bicycles", "year");
```

21. The insertion uses the builder pattern (http://en.wikipedia.org/wiki/Builder_pattern):

```
Insert insert = QueryBuilder.insertInto(KEYSPACE_NAME, "bicycles");
insert.value("manufacturer", "KTM");
insert.value("model", "Strada 1000");
insert.value("size", 57);
insert.value("year", 2014);
insert.value("front_cogs", Arrays.asList(53, 39));
insert.value("rear_cogs", Arrays.asList(12, 13, 14, 15, 17, 19, 21, 23, 25, 28));
session.execute(insert);
```

22. Run some queries:

```
System.out.println("Query 1");
for (Row row : resultSet1) {
    System.out.println(row);
    String manufacturer = row.getString("manufacturer");
    String model = row.getString("model");
    List<Integer> frontCogs = row.getList("front_cogs", Integer.class);
    List<Integer> rearCogs = row.getList("rear_cogs", Integer.class);
    System.out.println(manufacturer + " " + model + ", front cogs: "
            + frontCogs + ", rear cogs: " + rearCogs);
}

System.out.println("Query 2");
String select2 = QueryBuilder.select("manufacturer").from("bicycles").
    where(QueryBuilder.eq("size", 57)).getQueryString();
ResultSet resultSet2 = session.execute(select2);
for (Row row : resultSet2) {
    System.out.println(row);
}
```

23. To improve the readability of the code, add a static import. **Tip:** use the `Ctrl+Shift+M`, while the cursor is in the `eq` part of `QueryBuilder.eq`.

```
import static com.datastax.driver.core.querybuilder.QueryBuilder.eq;

[...]

String select2 = QueryBuilder.select("manufacturer").from("bicycles").
    where(eq("size", 57)).getQueryString();
```

2.3 Integration

1. Install Maven: set the `M2_HOME` environment variable and add `M2_HOME/bin` to the `PATH` environment variable.
2. Check if Maven is working properly:

   ```
   mvn --version
   ```

3. The standard Maven command for initiating the *build* process is:

   ```
   mvn clean install
   ```

4. The build may throw the folloing error: `Fatal error compiling: tools.jar not found:`
   To fix this, set the `JAVA_HOME` environment variable from the JRE's directory to the JDK's directory.
5. The `CassandraClient` class may cause an error related to the Java version, e.g. `diamond operator is not supported in -source 1.5`. Add the following lines to the POM file:

   ```
   <build>
       <plugins>
           <plugin>
               <artifactId>maven-compiler-plugin</artifactId>
               <configuration>
                   <source>1.7</source>
                   <target>1.7</target>
               </configuration>
           </plugin>
       </plugins>
   </build>
   ```

   Source: http://stackoverflow.com/questions/14043042/compiling-java-7-code-via-maven
6. Once the build succeeds, try to run the application.

   ```
   java -jar target\hu.bme.mit.sysint-0.0.1-SNAPSHOT.jar
   ```

7. It throws the following error:

   ```
   no main manifest attribute, in target\hu.bme.mit.sysint-0.0.1-SNAPSHOT.jar
   ```

   Define the main class in the POM file:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <configuration>
        <archive>
            <manifest>
                <mainClass>hu.bme.mit.sysint.cassandraclient.App</mainClass>
            </manifest>
            <manifestEntries>
                <mode>development</mode>
                <url>${pom.url}</url>
            </manifestEntries>
        </archive>
    </configuration>
</plugin>
```

Source: http://docs.codehaus.org/pages/viewpage.action?pageId=72602

8. It throws the following exception:

```
Exception in thread "main" java.lang.NoClassDefFoundError: com/datastax/driver/core/Statement
        at hu.bme.mit.sysint.cassandraclient.App.main(App.java:13)
Caused by: java.lang.ClassNotFoundException: com.datastax.driver.core.Statement
```

The reason for this is that the dependencies are not available for the application. To solve this, we have to

(a) Copy the dependencies:

```
<plugin>
    <artifactId>maven-dependency-plugin</artifactId>
    <executions>
        <execution>
            <phase>install</phase>
            <goals>
                <goal>copy-dependencies</goal>
            </goals>
            <configuration>
                <outputDirectory>${project.build.directory}/lib</outputDirectory>
            </configuration>
        </execution>
    </executions>
</plugin>
```

The `execution` tag shows an error:

```
maven-dependency-plugin (goals "copy-dependencies", "unpack") is not supported by m2e.
```

Use the provided **Permanently mark goal copy-dependencies in pom.xml as ignored in Eclipse build.** fix. This adds a long `<pluginManagement>` tag.

(b) Add them to the classpath:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <configuration>
        <archive>
            <manifest>
```

```
                    [...]
                    <addClasspath>true</addClasspath>
                    <classpathPrefix>lib/</classpathPrefix>
                </manifest>
            </archive>
        </configuration>
    </plugin>
```

For some libraries (e.g. `emfjson`) you may get a `java.lang.NoClassDefFoundError` caused by a `java.lang.ClassNotFoundException`. The reason for this is that Maven sometimes resolves the `SNAPSHOT` string to the actual snapshot version in the generated JAR's `MANIFEST.MF` file (e.g. `org.eclipselabs.emfjson-0.7.0-SNAPSHOT.jar` becomes `org.eclipselabs.emfjson-0.7.0-20140221.135604-5`) but the filename of the dependency's JAR stays the same. To solve this, add a `<useUniqueVersions>false</useUniqueVersions>` element between the `<manifest>` tags (source: http://jira.codehaus.org/browse/MJAR-156).

Source: http://stackoverflow.com/questions/97640/force-maven2-to-copy-dependencies-into-target-lib

9. The old `slf4j` warning is back. The solution is to create a the `src/main/resources` folder and move the `log4j.properties` file to it.

10. If you would like to package your project as one "fat JAR", use the `maven-assembly-plugin`.

```
<plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <configuration>
        <archive>
            <manifest>
                <mainClass>hu.bme.mit.sysint.cassandraclient.App</mainClass>
            </manifest>
        </archive>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
    </configuration>
    <executions>
        <execution>
            <id>make-assembly</id> <!-- this is used for inheritance merges -->
            <phase>package</phase> <!-- bind to the packaging phase -->
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Source: http://stackoverflow.com/questions/574594/how-can-i-create-an-executable-jar-with-dependencies-using-maven

If you just want to get the fat JAR, build the project with:

```
mvn clean compile assembly:single
```

2.4 Version control

1. Start by cloning the repository to your local computer. To do this, go to the **Git Repositories** view (from the **Window | Show View | Other...** window). Provide the clone URI to clone the Git repository.

2. Right click the project an choose **Team | Share project...**. Choose **Git** and select the repository.
3. Add a `.gitignore` file to the project. Search for GitHub's `gitignore` repository (https://github.com/github/gitignore) repository and use the `Maven.gitignore` and the `Java.gitignore` files as a guideline.
4. **Commit** and **Push** the project.

2.5 Eclipse tips

- In the **Window | Preferences** dialog, go to **Java | Editor | Typing**.
  - In the **Automatically insert at correct position** group, check the **Semicolons** checkbox.
  - In the **In string literals** group, check **Escape text when pasting into a string literal**.