INF6953G : Lab 3
Scaling Databases and Implementing Cloud Patterns

Foutse Khomh
S. Amirhossein Abtahizadeh
Département Génie Informatique et Génie Logiciel
École Polytechnique de Montréal, Québec, Canada
foutse.khomh[at]polymtl.ca
a.abtahizadeh[at]polymtl.ca

Le An, Alexandre Courouble, Mahdis Zolfagharinia

22th March, 2016

1 What were the results of comparing MySQL to MySQL Cluster. Were any of the results surprising, why or why not?

2 When and why should we use The Gatekeeper and Competing Consumers Patterns?

3 Describe your solutions to implement the two Cloud patterns strategies.

3.1 Competing consumers pattern

3.1.1 Overview

We use 4 AWS t2.micro machines to build the server side of the *Competing consumers pattern*. We install and configure MySQL in the 4 machines (referred as to data node in the rest of this section). We configure the 4 data nodes as a cluster, where one machine (the *master node*) receives requests from clients and distributes them either to itself or to the other three data nodes (*slave nodes*). The Competing consumers pattern only deals with `INSERT` requests. After inserting a client request into the specific data node, the master node will

reply to the client with a message (*e.g,* "Data inserted into Slave1"). Figure ??? illustrates the workflow of server which implements the Competing consumers pattern.???
In the rest of this section, we elaborate three key points of the implementation of this pattern in terms of data nodes' configuration, master node's implementation, and socket connection. Our source code in Java can be found at: `???the-link???`.

3.1.2 Configuration of data nodes

We load *Sakila* database into the 4 data nodes. The master node can send queries to the slave nodes through the port **3306**.

3.1.3 Implementation of the master node

In each client request, the master node will receive a string through the socket connection. The client request string contains the client identifier and her SQL request statement, an example of which is shown as follow:

```
200a3b9b-17a1-4808-b1ba-54d159ea4108+2006||INSERT INTO film (title,
description, release_year, language_id) VALUES ('sample_movie', 'This
is just a test', 2016, 1)
```

In the request, the client identifier and the SQL statement are separated by `||`. The master node parse the all letters and numbers (*i.e.,* `[0-9a-z]`) in the client identifier to convert each of these characters to an ASCII decimal number and sum all ASCII decimal numbers. The result is noted as $DigitSum$. As there are 4 data node in the cluster, to balance the workload in these node, we divide $DigitSum$ by 4 and calculate the remainder, noted as Rem (*i.e.,* $DigitSum\%4 = Rem$). Rem is an integer ranged from 0 to 3. The master node will assign the request (by sending the SQL statement), where $Rem = 0$, to itself; the request, where $Rem = 1$, to slave node #1; the request, where $Rem = 2$, to slave node #2; and the request, where $Rem = 3$, to slave node #3.

3.1.4 Socket connection

3.2 Gatekeeper pattern

4 Describe the implementation of multi-threaded application scenarios.

5 What were the results of comparing the patterns? Were any of the results surprising, why or why not?

6 What were the results of measuring energy consumption in scenarios? Provide a comparison between Cloud patterns in terms of both performance (application response time) and energy consumption.