LeanXcale Code Example

Table of Contents

1. Lx-DB: A simple example 1
2. Example description 1
    2.1. Implementation details 1
3. Download 2
4. Install 2
5. Running the example 3
6. Just a Few Advanced Features 6
7. Use a client 7
8. Troubleshooting 7
9. Contact us 8
    9.1. Advanced trial 8
    9.2. Support 8

This example is for Linux, and the included libraries currently only work in Linux.

1. Lx-DB: A simple example

This example provides a quick walkthrough of a very simple example to give a general understanding of the capabilities of Lx-DB.

The example code aims to provide a starting point you can adapt to your own needs to see how Lx-DB can work for you.

The implementation consists of a set of Groovy files. Feel free to modify the code in any way you feel appropriate.

2. Example description

The example is simplified but tries to resemble a common scenario. It is built up from two sources of information, and a processor:

- A primary source containing a continuous set of events (Apache HTTP server log files).
- A second source that adding information for some information in the events log (a simulation of a component able to get postal codes from hosts accessing the HTTP server.)
- A process aggregating information about requests to the system and how they are evolving.

Running this trial on your local machine is unlikely to display any significant latency to the trial cloud instance.

If you want to experience the performance and scalability benefits of Lx-DB, please contact us at info@leanxcale.com.

We are happy to prepare a customized on-premise or cloud trial addressing your particular needs.

2.1. Implementation details

2.1.1. Batch inserts

The process that loads Apache log events is batching inserts every 1000 records.

2.1.2. Piped processes

The Groovy loading processes are piped, because we are simulating that the output from the Apache log event loader (the host names) is the source of the process simulating the enrichment with the postal code.

2.1.3. Log events simulation

The process that loads Apache log events simulates an “infinite” continuous source. Really it gets the newest timestamp inserted in the database and start the next round of loading the same file from this timestamp on.

This is, the first load will start on the 1st of July of 1995 as the file’s timestamps are, but the next time you start loading the file it will start after the 31st of July, and so on …

2.1.4. Postal code simulation

The process that loads the host and postal code only inserts new hosts. For this reason, commits are not only done by size, but also by time.

Inserts are done if more than 100 new hosts are ready to be inserted or there has been more than 200ms from the last COMMIT (whichever happens first).

3. Download

To go through the example you need to download the example source code files that we will be talking about.

The files can be downloaded as a tarball from https://s3-eu-west-1.amazonaws.com/lxdownload/client/latest/Lx-example-linux.tar.gz

4. Install

When you unpack the example package:

```
tar xvzf Lx-example-linux.tar.gz
```

You will get the following files:

create_testdb.sh, create_testdb.sql	Those are the scripts to create the database SCHEMA for the example
configexample.sh	This is a SCRIPT to reconfigure the rest of the SCRIPTs with the IP of your Lx-DB trial host
loadApachelogs.groovy	This is the process to upload data from Apache HTTP log events into Lx-DB database
loadHostPCode.groovy	This is the process that simulates enriching the HOST with the Postal Code and loading the data into Lx-DB
sqlLoadLogs.groovy	This is the same process as loadApachelogs but using JDBC rather than direct KiVi library
sqlLoadLogsBatch.groovy	This is the third version of loadApachelogs process. In this case, It uses JDBC Batches which are more efficient than direct JDBC, but less efficien than direct KiVi API
sqlPCode.groovy	Same process as loadHostPCode, but using JDBC
lxClient	This is a simple SQL command console used to create the database
NASA_access_log_Jul95.gz	This is a sample Apache HTTP log taken from NASA and used as the primary source of information (https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html)
run.sh	This is a SCRIPT that is used to start the Groovy loading processes
runLoad.sh	Is a SCRIPT similar to the previous, but to use the JDBC insert processes
runSql.sh	This is a SCRIPT to run the Groovy querying process
sqlLauncher.groovy	This is the Groovy process to run all the queries to show user behavior

5. Running the example

To run the example, you will need the **trial IP address** to a Lx-DB cloud trial instance.

1. Open a terminal
2. Navigate to the directory where you unpacked the `.tgz` example file
3. Configure your scripts to run with your instance

```
bash configexample.sh {trial IP address}
```

4. Create the database SCHEMA

```
./create_testdb.sh
```

5. Run the loading processes as a continuous source of information

```
while [ 1 ]; do gunzip -c NASA_access_log_Jul95.gz | ./run.sh ;
sleep
60; done
```

The terminal should display something like this:

```
Apache Logs Load Progressing ... - COMMIT: Throughput(Tuples
written/second) 11614
Host-Postal Code feed Progressing ... COMMIT: (Throughput) 93
Rows: 58
Apache Logs Load Progressing ... - COMMIT: Throughput(Tuples
written/second) 12601
Host-Postal Code feed Progressing ... COMMIT: (Throughput) 78
Rows: 49
Apache Logs Load Progressing ... - COMMIT: Throughput(Tuples
written/second) 11583
Host-Postal Code feed Progressing ... COMMIT: (Throughput) 80
Rows: 51
Apache Logs Load Progressing ... - COMMIT: Throughput(Tuples
written/second) 10643
Host-Postal Code feed Progressing ... COMMIT: (Throughput) 82
Rows: 50
Apache Logs Load Progressing ... - COMMIT: Throughput(Tuples
written/second) 11525
This shows some messages each commit so you can see the system is
progressing.
```

6. Open a separate terminal
7. In the new terminal run the `./runSql.sh` file as a periodical view with the `watch` command:

```
watch -d -n 5 ./runSql.sh
```

The terminal should display something like this:

```
QUERY 1: TOP 10 URLS in the latest 1 HOUR
=====================================
URL: GET /images/KSC-logosmall.gif HTTP/1.0 --> COUNT:136
URL: GET /images/NASA-logosmall.gif HTTP/1.0 --> COUNT:103
URL: GET /shuttle/countdown/ HTTP/1.0 --> COUNT:98
URL: GET /shuttle/countdown/count.gif HTTP/1.0 --> COUNT:89
URL: GET /shuttle/missions/sts-71/sts-71-patch-small.gif HTTP/1.0
--> COUNT:76
URL: GET /images/launch-logo.gif HTTP/1.0 --> COUNT:67
URL: GET /images/WORLD-logosmall.gif HTTP/1.0 --> COUNT:63
URL: GET /images/ksclogo-medium.gif HTTP/1.0 --> COUNT:61
URL: GET /shuttle/missions/sts-71/mission-sts-71.html HTTP/1.0 -->
COUNT:60
URL: GET /shuttle/countdown/video/livevideo.gif HTTP/1.0 -->
COUNT:58

QUERY 2: TOP 5 POSTAL_CODE in the latest 1 HOUR
=====================================
POSTAL_CODE: 05489 --> COUNT:227
POSTAL_CODE: 01254 --> COUNT:149
POSTAL_CODE: 12489 --> COUNT:135
POSTAL_CODE: 19489 --> COUNT:125
POSTAL_CODE: 00489 --> COUNT:116

QUERY 3: FOR EACH POSTAL_CODE, TOP 2 URLS in the latest 1 HOUR
=====================================
POSTAL_CODE:05489
URL: GET /images/KSC-logosmall.gif HTTP/1.0 --> COUNT:7
URL: GET /images/ksclogo-medium.gif HTTP/1.0 --> COUNT:6
`----
POSTAL_CODE:01254
URL: GET /shuttle/technology/sts-newsref/sts_mes.html HTTP/1.0 -->
COUNT:5
URL: GET /shuttle/countdown/count.gif HTTP/1.0 --> COUNT:4
`----
POSTAL_CODE:12489
URL: GET /shuttle/missions/sts-71/sts-71-patch-small.gif HTTP/1.0
-->
COUNT:11
URL: GET /images/KSC-logosmall.gif HTTP/1.0 --> COUNT:9
```

8. You could also run the ingestion through JDBC (though It is significantly slower). You can do it using JDBC batches or not (the "*batch*" attribute is optional):

```
while [ 1 ]; do gunzip -c NASA_access_log_Jul95.gz |
./runLoadJDBC.sh [batch]; sleep
60; done
```

6. Just a Few Advanced Features

OK, you have just seen a simple working example with no advanced functionality. In fact, you created the tables with no partitioning so all data was using the same datastore which means there was no distribution.

So let's start again partitioning the data. To do so:

```
#This will wipe all your example data and start the database from
scratch
#It may take one or two minutes
./sendcommand {Your trial IP address} start new
./create_testdb.sh
./sendcommand {Your trial IP address} partition APACHE_LOGEVENTS [
0.0.0.0,0 --:- zserver,0]
```

The last command will partition data assuming a uniform distribution over the full range of values defined (0.0.0.0 to zserver). Now, you have a new schema that is partitioned over all the datastores in you trial deployment (typically 2).

You can run the rest of the example and you should notice (unless the network latency is predominant) a higher performance when running the example. If we had accomplished an even partitioning schema we could be seeing a performance improvement though It will most likely not be very noticeable because of network latency, but you could also parallelize loading. Any way, It won't be exactly 2 because data distribution is not uniform in the example.

Again, If you really want to notice performance, contact us and we can extend the trial with another instance so you can run against the server with a low latency connection.

Now, let's go a step further. The data we are using has a typical time varying pattern that can be highly benefited from LeanXcale's bidimensional partitioning. Bidimensional partitioning allows for an internal management of time series variation to have the active data in memory and provide a better performance. Let's start again and split according to a bidimensional partitioning schema:

```
#This will wipe all your example data and start the database from
scratch
#It may take one or two minutes
./sendcommand {Your trial IP address} start new
./create_testdb.sh
./sendcommand {Your trial IP address} partition APACHE_LOGEVENTS [
0.0.0.0,IGN --:- zserver,IGN]
```

The important difference in this command is that IGN in the timestamp field will inform the database engine this value is a field to be considered as a second dimension.

You can now play the example again and see the behavior. You will start noticing a difference in a long running example (Well, not that long running because the size of the trial deployment is not that big).

7. Use a client

In order to run any queries, you can configure the connection in any SQL client.

If you have your own preferred client, use the following connection string:

```
jdbc:leanxcale:direct://{Your trial IP address}:1529/testdb
```

If you want to try an open source client, see one of our Lx-DB client articles.

8. Troubleshooting

Some possible problems and their causes:

I cannot seem to get the JDBC connection working. After some time I get this error message related with Zookeeper.
Note the `direct` in the connection string: `jdbc:leanxcale:direct://{Your trial IP}:1529/testdb;create=true`

I try to run the example, but I get the following error:

```
Exception in thread "main" java.lang.IllegalArgumentException:
Prefix string too short
        at java.io.File.createTempFile(File.java:2001)
        at java.io.File.createTempFile(File.java:2070)
        at kv.Conn.loadLibrary(Conn.java:79)
        at
com.leanxcale.kivi.session.ConnectionFactory.<init>(ConnectionFactor
y.java:44)
        at
com.leanxcale.kivi.session.ConnectionFactory.getInstance(ConnectionF
actory.java:100)
        at
com.leanxcale.kivi.session.ConnectionFactory.connect(ConnectionFacto
ry.java:73)
        at
com.adquiver.leanxcale.LeanxcaleTest.writeKVTest(LeanxcaleTest.java:
31)
        at com.adquiver.App.main(App.java:19)
```

This error appears when trying to run the code in Windows. The example libraries only work on Linux currently.

9. Contact us

If you reached this point maybe you would like to go a step further.

9.1. Advanced trial

If you want to set up a more advanced trial, please contact us at info@leanxcale.com.

We are happy to prepare a customized on-premise or cloud trial addressing your particular needs.

9.2. Support

If you have questions or comments on the example or your trial, don’t hesitate to contact support@leanxcale.com.