Advanced Databases Spring 2012

Project Milestone 2

Note: Also consider the alternative Milestone 2 specification (involving Twitter's Storm system rather than MySQL). Your team can choose one of the two assignments.

THE COMPUTING ENVIRONMENT

We work with a cluster that runs a virtualized Solaris-based environment with 96 hardware threads (~=cores). Four nodes are designated gateways (called "global zones on Solaris") that are visible globally and can be connected to via ssh. These nodes are called icdatasrv[1|2|3|4].epfl.ch.
Each global zone manages a blade and shares memory and I/O with 22 "local zones". These are virtual machines that each have a hardware thread exclusively assigned to them -- so work can run on each of these local zones in parallel.

There are eleven team accounts, ten for students and one for the course staff. Each of these teams have 8 local zones exclusively assigned to them -- two per blade. The team accounts are called team1 to team11. The members of a team share a team account.

Each team is assigned zones i and i+11 (where i is the team number) on each blade. Local zones are only accessible on the private network that you access by logging into one of the gateways. There, the local zones are called icdatasrvj-k, where j is the index of the blade/global zone and k is either i or i+11. So, for example, team3 owns the local zones icdatasrv1-3, icdatasrv1-14, …, icdatasrv4-3, icdatasrv4-14.

You can use any of the gateways/global zones to connect, but we suggest to always connect to the same gateway because the home directories are not shared between blades (however, home directories are shared among the global zone and all local zones of a blade).
Moreover, we suggest that you use icdatasrv1 or 2 rather than 3 or 4 because the latter two run map-reduce control jobs and may be somewhat busy later in the term.

MySQL is set up on the local zones -- each one runs its own MySQL server. You can connect to the MySQL servers remotely inside the private network in the standard way -- either by JDBC or MySQL. The MySQL servers listen at

the standard port, which is 3306. At each MySQL instance of your local zones there is an empty MySQL database called dbcourse1.

Try this. Assume you are team3. Connect to one of the gateways, say the second one:

```
ssh team3@icdatasrv2
```

Enter your password. Now you are on the private network and can reach all eight of your worker machines, in the case of team3 these are icdatasrv[1-3, 1-14, 2-3, 2-14, 3-3, 3-14, 4-3, 4-14]. Connect to one of them using ssh as follows:

```
ssh team3@icdatasrv3-14
```

Same password.

Alternatively, connect to MySQL, still from icdatasrv3 using

```
mysql --host=icdatasrv3-14 -p --database=dbcourse1
```

Using JDBC, you create a connection (again using the example of team3 and local zone 3-14) by

```
DriverManager.getConnection(
    "jdbc:mysql://icdatasrv3-14:3306/dbcourse1",
    "team3", <team password>);
```

We suggest to log in as little as possible into the local zones using ssh but to use MySQL and JDBC to connect to them. The reason for this is as follows: only two of your eight local zones with share the home directory with the gateway you are using, so you will have to scp files, which will cause additional version management issues to worry about for you.

Also, please avoid having lots of copies of large files lying around. The cluster has 8 TB of disk space, but most is allocated to MySQL and hadoop and the home directories can run out of space. You have a 30 GB quota on your home directories; the quota does not apply to MySQL databases, but please do not use more space than needed, otherwise we may run out of it.

We have randomly generated strong passwords and suggest not to change these passwords. You would have to make the same change at least 17 times (UNIX + MySQL on each or your 8 workers, plus at least one gateway).

PROJECT MILESTONE 2: REQUIREMENTS

Deadline Wednesday May 2, 2012 1:15pm (~four weeks + 1 day, since May 1 is a holiday).

NOTE: We are giving you the chance to change your team membership. Do not start working on this assignment before the new teams have been set up (there will be new team passwords). Details will be discussed in the April 3 lecture.

Your task is to write a Java/JDBC application that processes a workload of TPC-H queries efficiently given your computational resources:

(1) Install the TPC-H data generator DBGEN. This contains the sql code for schema generation (in dbgen/dss.ddl) plus a C program "dbgen" for generating test databases of that schema with different sizes, or "scaling factor"s. You can build dbgen yourself (by downloading it from http://www.tpc.org/tpch/), but for convenience, we will provide you with a binary via moodle. The compiled binary we will provide you with also includes a script that deletes the final vertical bar ('|') from each line (which MySQL does not like). Details of the benchmark and its queries are found in the TPC-H documentation which you have already been given, and which may also be found on the TPC-H webpage.

(2) Your workload will consist of two TPC-H queries, Q7 and one further query that you are free to choose among those TPC-H queries that satisfy the following constraints: the query must perform aggregation and access at least three different relations. For queries that take external parameters, replace the parameters by the standard values given in the TPC-H spec.

(3) Think about how to partition the data intelligently specifically for your workload (the two queries) -- you do not need to worry about other queries but your only choice is to store subsets of the full relations: that is, you may employ horizontal partitioning but no vertical partitioning or materialized views (precomputed joins). Also, in sum, your database must store the full data even if some of it is not needed for your query workload. There may not be any redundancy/replication across your eight databases. That is, your eight database servers have to store the database without overlap, and you should distribute the data to allow for fast parallelized processing. This choice requires an understanding of how you will process your workload. Very small relations (specifically, nation), do not have to be partitioned but may reside in entirety on a single node.

It would require a lot of work for you to modify the generator to create partitions, so instead you may load all the data into your eight databases and then use SQL DELETE statements to remove what you don't want in there.

(4) Implement the Bloom Join operator in Java.

- Try to push as much computation into SQL to be executed by your MySQL servers as reasonably possible. (Usally, MySQL executes query operators more efficiently than your Java code.)

- Can you find a good way to push the computation of bitmaps to the server so that only these bitmaps are returned to your Java program via JDBC (and from there forwarded to other DB server nodes according to the Bloom Join algorithm). “No” may be the right answer, but if so, explain your thinking and what approaches to performing this push you considered. By DB server, we mean the MySQL server processes running on your eight worker machines, not your machines themselves. In particular, you are not expected to build separate Java programs that run on your workers, communicate with your central Java program and the local MySQL database, and do the bitmap computations there. That might be a good solution but is too much work.

(5) Your final java program will run on the gateway machine that you use (icdatatsrvX) and will use eight JDBC connections to connect to your eight MySQL databases.

Your java code implements the operators you need for your query workload in a way that makes best use of your parallel resources for efficiency (in particular, it uses the bloomjoin operator you have implemented for distributed joining). Since you data is managed by MySQL, you should push as much work as you can into MySQL. You do this by running SQL statements against the MySQL instances, in parallel. This requires your java program to be multithreaded. Do not connect to the different databases and execute your queries one after another, that would not exploit parallelization and would be very bad!

Try to minimize the amount of data that you send over the network or put into the memory of your java program. Creating temporary tables from queries is permissible if you feel you need to do this.

- Your java code may hardwire the two query plans by providing either a way to choose which query to run or by providing two main programs, one for each query. The Java program writes the final query result to System.out.
- Implement general query operators such as s Bloom join and sum/group-by aggregation operators, which would also work for other queries, not just the two of your workload.
- Note that, in general, a single bloom join between two nodes will not be sufficient, but you will have to join data from multiple pairs of nodes.
- Do not implement a query parser to read in the SQL of your query workload. Hardwire the queries, but structure and document the code well, to keep the query plans you intended identifiable.

(6) Perform experiments to find out what the largest scaling factor among 0.001, 0.01, 0.1, 1, 10 (up to the maximum scaling factor you can experiment with given your quota) is at which your query runs in no more than ten minutes wall-clock time. Report that scaling factor and the running time of your query. The relevant timing numbers are for a "hot cache" -- that is, run the query twice in short sequence and report the second running time. It would be better to get the cold cache times, but it is not realistic for you to obtain those because the main memory of the cluster is very large (288 GB) and you cannot reboot the machines.

DELIVERABLES

As your solution, submit a .zip file in moodle by the deadline (Wednesday two weeks from now, time see above/moodle) containing the following files:

- Your Java program file(s), including all your source code, for executing your two TPC-H queries.

- A file entitled README describing the precise sequence of “installation instructions” for your implementation, including the commands necessary to run each of your queries. This will allow us to get your code to work and reproduce your findings. The README file should also include instructions for changing the database node names, the username and password, and the location of all input data files.

- Either SQL scripts or Java programs for deleting the data from a database generated by DBGEN to obtain the partitioned databases you want to use. (But do not submit the output of dbgen itself!)

- A text file reporting your experimental findings, i.e. at least the scaling factor you have chosen and the time it takes to execute your query at that scaling factor, not counting the creation, population, and purging (to create the partitions) of your databases. Does the system perform better or worse on Q7 than the Hadoop-based implementation of Milestone 1. Why?

- Try to submit clean source code. Document your source code and/or give an overview of architecture and design decisions that went into your implementation in the text document.

- In case you decide there is no good way for you to push the bloom join

bitmap computation to the database servers, provide your discussion of this in the text document.

PRACTICAL ADVICE

Developing parallel programs can be very time-consuming if you do it in a sloppy way without a plan. Try to develop and debug it as much as you can on a single node (maybe your private computer, or a machine in one of the labs) before you deploy it on the cluster. Test and debug your code first with small data, otherwise you will be spending lots of time waiting for the system to terminate. When going to parallel mode first try it with two rather than all eight local zones. Start working with very small databases so that you don't have to wait long while debugging.

If you do things in a reasonable way, this is task requires only a moderate amount of work. You are part of a large team, so share out the work intelligently and as early as possible. It is hard to work in a large team under time pressure.

Remember that if you start a query or update, there are ways to interrupt your session without closing the connection and stopping the database job. In that case the job will continue on the database server and will consume resources. An update operation will even keep locks and you may not be able to do further updates. For that reason, it is a good idea to test as much as you can on a local installation of MySQL, with little data. Note that you can see what MySQL jobs are running on your local machine with

```
mysqladmin proc –v
```

and you can kill a job with

```
mysqladmin kill <pid>
```

You can load data into MySQL with a statement like

```
LOAD DATA LOCAL INFILE '<full file path>' REPLACE
INTO TABLE `<table name>` FIELDS TERMINATED BY '|';
```

This requires that you call MySQL with --local-infile=1. The full file path is local to the machine that you are running MySQL on. Google it.