Project

Introduction

Imagine you are a company with customers using one of your products, a client-server application. When looking at the performance of such applications, one is typically interested in two problems:

- What response time do my customers experience? If the response time is high, what is the reason for this? Is the CPU or the disk in the server too slow? Or maybe the network connection?
- Assuming I get new customers in the future, how many customers can my application handle before the response time becomes too high?

In this project, you will analyze the performance (e.g. response time) of a client-server application. In order to keep things simple, we have chosen here an application that is relatively easy to setup: The server is a computer hosting a small MariaDB server, and the client is a remote computer that sends random SQL queries to the server. Please see the document "preparation.pdf" for detailed instructions how to install the server. Note that we have chosen the database *just as an example* that you should study. This project (nor the course) is *not* about database optimizations.

Background: How MariaDB (and MySQL) work

From a client-server point of view, database management systems like MariaDB are quite similar to web servers. A client sends a query to the server, the query is processed by the server, and a response is returned to the client. In SQL databases, many kinds of queries are possible. In the Java file on Moodle, we give three examples: (i) a query that asks the server to calculate an average over some table data, (ii) a query that returns table data, and (iii) a query that writes to a table (adding a new row). The performance of the system will depend on many factors, for example:

- the network bandwidth and latency between the client and the server,
- the CPU speed and the number of CPUs resp. CPU cores,
- the speed and amount of main memory (RAM),
- the speed of the backing storage (hard disk or SSD),
- the type of queries and the queried data.

By default, MariaDB creates a new thread for every client connection. However, you can also limit the maximum number of threads. In that way, an incoming query must wait if all threads are busy. To set the maximum number of threads you have to stop the server first, either with

```
sudo kill <processId>
```

or, if the server has been started automatically, with

```
service mysql stop
```

(the exact command depends on your Linux distribution). Then you can manually restart the server and set the thread number, for example to 3:

```
sudo mysqld --user mysql --innodb-thread-concurrency=3
```

Task 1: Measurements

You should perform measurements to answer these questions:

a) What is the average query response time seen by customers and how does it depend on (a) the type of queries and (b) the number of queries sent per second?
b) What are the factors that influence the response time? What are possible bottlenecks? (again, depending on the type of queries)

A typical challenge when doing measurements is to choose a realistic workload. We suggest that you modify the provided Java client code such that it sends queries to the server as a Poisson process, i.e. with negative-exponentially distributed, independent inter-arrival times. A JDBC connection can only handle one query at the same time, therefore you should use a new thread and connection[1] for each query. Do your measurements with different query types and query "difficulties". The Java code on Moodle shows some examples how to vary queries and make queries harder for the server (e.g. with LIMIT and random values). These examples are meant as an inspiration for you. You can design your own queries.

(Note that inserting a single row is extremely fast (faster than opening a connection) and it is probably better to insert several rows per connection.)

To answer the second question, you should not only measure the response times but also measure

- Network utilization. You can use tools like nethogs (in "trace mode") or ifpps.
- CPU utilization of the server. There are several tools for that on Linux, for example pidstat and top. The perf tool can also sample CPU usage.
- Disk utilization of the server. Examples for tools: pidstat, iotop (Note that you have to run pidstat with sudo, otherwise it cannot not show all statistics.)
- Cache faults on the server.

Show plots and discuss the results. What is the most important factor for the response time, depending on the chosen parameters (utilization, type of queries etc.) and why?

Task 2: Modeling

In this task you should

a) calculate the mean response time using a queueing station model with different arrival rates;
b) compare your results with the measured average response time from the experiments in task 1. Also see what happens if you limit the number of threads in the server.

You have to choose a queueing station model with parameters (arrival rate, service rate, etc.) that correspond to the behavior of the real system. How can you determine these parameters? The arrival rate is obvious, but how do you know the service time? Remember that the response time is identical to the service time if a customer arrives at a completely empty queueing station. Therefore, if you send a query to the database and you wait until it has finished before sending the next query, you can measure the service time of your real system. Then can calculate the characteristics of the service time distribution from your measurements (similar to question 1 in exercise 10).

Show plots comparing the mean response time from the model and the average response time from the experiments for different arrival rates and number of threads.

Discuss the result: Does the model give correct results? If not, what could be possible reasons? Maybe some assumptions of the models seen in the course are not correct in reality?

General remarks and requirements

- Show results and discuss them. Don't just dump plots into your report!
- Remember that we have queues and caches everywhere. It is probably better to ignore measurement results from the first 20 (or more?) queries until the system has "warmed up". Observe your data and see after how many queries the average starts to stabilize.
- The danger when showing average values is that the reader does not know what the variance of the data behind the average is. Therefore, do not only show averages but also minimum and maximum values for each point in your plots. Let's say you have measured an average

[1] There are better ways (like thread pools and connection pools), but this is the easiest way.

response time of 100ms (minimum 80ms, maximum 150ms) for an arrival rate of 5 requests/s. If you use gnuplot, you can write in a file

```
5       100     80      150
```

and plot errorbars with:

```
plot "file.txt" with errorbars
```

- Be careful about interferences with other programs running on your system or in your network. Repeat your measurements several times to see whether the results change.

Expected output

We expect from you a zip file with the source code of your client and a report. The report should be maximum 5 pages (A4, 11pt, pdf format) and contain

- Name and student number of the two group members
- Task 1:
  - Short description of your measurement setup (OS, network, etc.) and how you did your measurements (repetitions, warm up etc.)
  - Short description of your client implementation and the work load that you generate (type of queries etc.).
  - The measurement results (plots) from the experiments.
  - Discussion of the results.
- Task 2:
  - Description of the used queueing station models and their parameters, especially how you determined the service time.
  - The results (plots) comparing model and reality.
  - Discussion of the results.

Evaluation criteria

- Presentation quality (language, readability, quality of plots)
- Completeness of the descriptions
- Technical quality of the experiments (correct measurements, correct choice of models and parameters)
- Quality of the discussion