

MySQL Cluster – Evaluation and Tests

Michael Raith (B.Sc.), *Master-Student*

◆

Abstract

Websites or web applications, whether they represent shopping systems, on demand services or a social networks, have something in common: data must be stored somewhere and somehow. This job can be achieved by various solutions with very different performance characteristics, e.g. based on simple data files, databases or high performance RAM storage solutions.

For today's popular web applications it is important to handle database operations in a minimum amount of time, because they are struggling with a vast increase in visitors and user generated data. Therefore, a major requirement for modern database application is to handle huge data (also called "big data") in a short amount of time and to provide high availability for that data.

A very popular database application in the open source community is *MySQL*, which was originally developed by a swedish company called *MySQL AB* and is now maintained by *Oracle*. MySQL is shipped in a bundle with the *Apache* web server and therefore has a large distribution. This database is easily installed, maintained and administrated. By default MySQL is shipped with the *MyISAM* storage engine, which has good performance on read requests, but a poor one on massive parallel write requests. With appropriate tuning of various database settings, special architecture setups (replication, partitioning, etc.) or other storage engines, MySQL can be turned into a fast database application. For example Wikipedia uses MySQL for their backend data storage.

In the lecture "Ultra Large Scale Systems" and "System Engineering" taught by Walter Kriha at Media University Stuttgart, the question "Can a MySQL database application handle more then 3000 database requests per second?" came up some time. Inspired by this issue, I got myself going to find out, if MySQL is able to handle such a amount of requests per second. At that time I also read something about the high availability and scalability solution *MySQL Cluster* and it was the right time to test the performance of that solution.

In this paper I describe how to set up a MySQL database server with the additional MySQL Cluster storage engine "ndbcluster" and how to configure a database cluster. In addition I execute some database tests on that cluster to proof that it's possible the get a throughput of ≥ 3000 read requests per second with a MySQL database.

Index Terms

MySQL, MySQL Cluster, JMeter, high availability, high performance, auto sharding, database test

1 INTRODUCING MYSQL CLUSTER

MYSQL-CLUSTER is a distributed database system with *data nodes* storing data and several *application nodes* for query execution, e.g. a MySQL Daemon (mysqld). It is managed by special *management nodes* (see Figure 1). Clients are not able to connect directly to the data nodes to fetch data, all commands must be run and executed over the application nodes.

Compared to the normal MySQL version, MySQL-Cluster is designed for applications dealing big data and applications requiring high availability (99,999 %, see [2]). MySQL-Cluster offers high performance by holding the data in the server's RAM. Tables are automatically partitioned by the database system across multiple nodes (*auto-sharding*) to ensure easy scale-out¹ on cheap standard hardware. A further increase of high availability and reliability can be achieved by (geographical) replication over different locations. MySQL-Cluster allows planned maintenance of nodes, system scale-out and database schema upgrades while the cluster is still running and executing queries. The database allows concurrent SQL and NoSQL access to the database over the *NDB API*.

michael.raith@hdm-stuttgart.de, Computer Science and Media (Master), Faculty Print and Media, Media University (Hochschule der Medien), Nobelstraße 10, D-70569 Stuttgart

1. *scale out*: scale a system horizontally

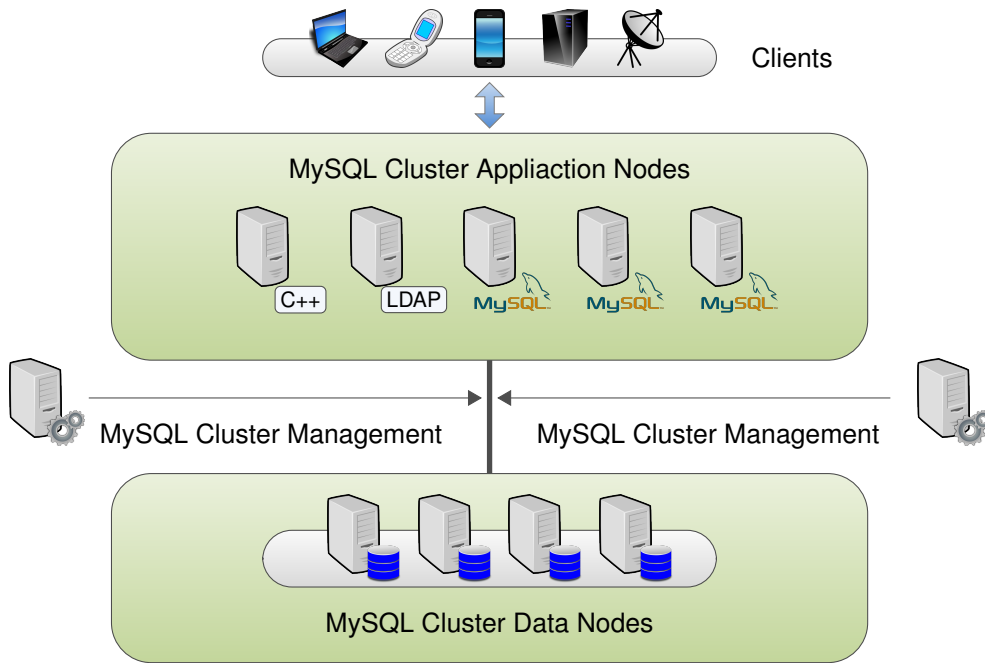


Figure 1: architecture of a MySQL-Cluster (cf. [1])

Another precondition when using MySQL-Cluster is that the number of data nodes must be a multiple of two (2, 4, 6, 8, ...), as data nodes are grouped into so-called “node groups”, i.e. pairs of nodes. This is important to use the access patterns of MySQL-Cluster properly: the database engine splits table-rows automatically into partition fragments and spreads them over the node groups (see Figure 2). Each node group holds two primary partitions and mirrors of each other in the node group. By default the partitioning process is achieved by using a hash over the table’s primary key.

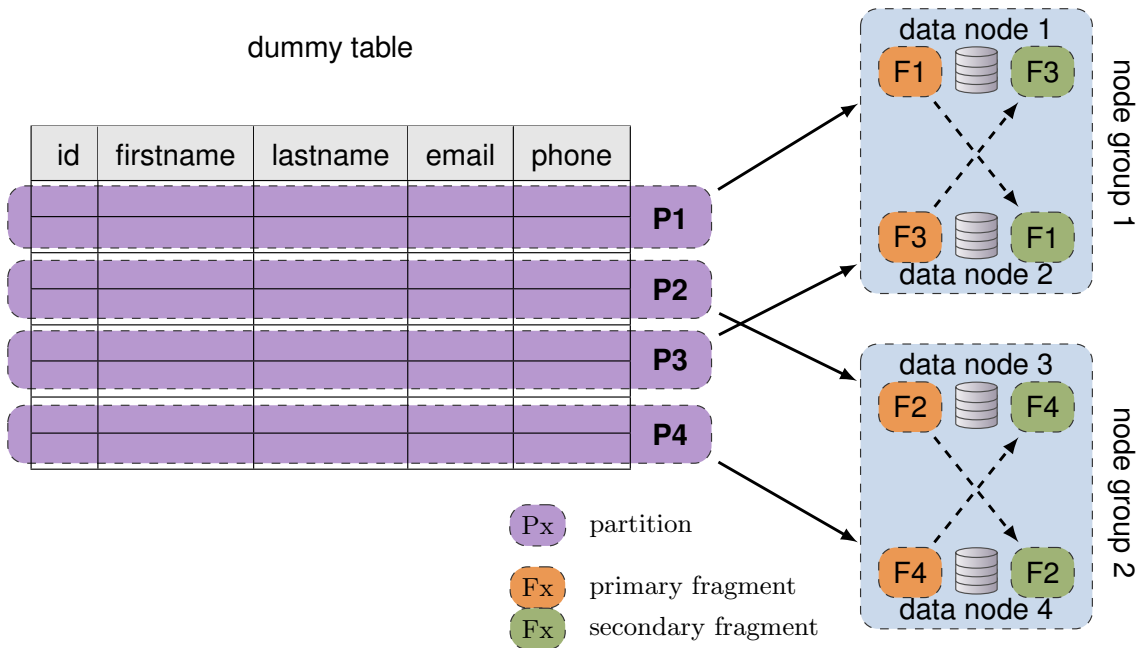


Figure 2: partitioning of MySQL-Cluster data (cf. [1])

2 SETUP

2.1 Preparing the test environment

Prior to start any test, I had to built a test environment. Therefore I set up some virtual machines at my universitie’s datacenter with the configurations listed in Table 1. As operating system I used Debian 6.0. The systems run on two different virtual machine hosts connected via a 1 GBit/s Ethernet. The *Data Nodes* run on one virtual machine host, the *MySQL Server*, *Proxy* and *Management Node* on the other host. Overall the test environment has 19 cpu cores and 21,5 GByte RAM.

Table 1: MySQL Cluster system hardware setup

Hostname	Node-Name	Core(s)	RAM (MByte)
mysc-daemon-1	MySQL Server	4	2048
mysc-daemon-2	MySQL Server	4	2048
mysc-proxy	Proxy	2	1024
mysc-mgmt	Management Node	1	512
mysc-node-1	Data Node	2	4096
mysc-node-2	Data Node	2	4096
mysc-node-3	Data Node	2	4096
mysc-node-4	Data Node	2	4096

The virtual machine hosts have the following characteristics:

- dual Opteron server with two AMD Opteron 6136 processors (8 cores each) @ 2.4 GHz
- 64 GB of RAM per server
- boot device: two mirrored 500 GB hard disks
- virtual machine’s data server: 1 TB hard disks with SAN interface running on RAID 6 plus a mirror backup system running also on RAID 6

The test environment’s architecture for the machines listed in Table 1 is shown in Figure 3. The link between the two proxy machines (“hot standby*”) was not implemented in this test architecture. The hot standby solution should be implemented in a real world scenario with critical data to achieve a more reliable system and to avoid bottlenecks.

2.2 Installing MySQL Cluster

Installing MySQL-Cluster is not toodifficult. The basic steps are:

- download the MySQL-Cluster package
- unpack the package
- create a “mysql” user and group
- add a basic configuration file for the “mysqld” process (MySQL-Server)
→ I installed them on each node to get direct access from each node to the database e.g. for debugging purposes
- create directories for storing the MySQL data
- copy startup script to “/etc/init.d/mysql”

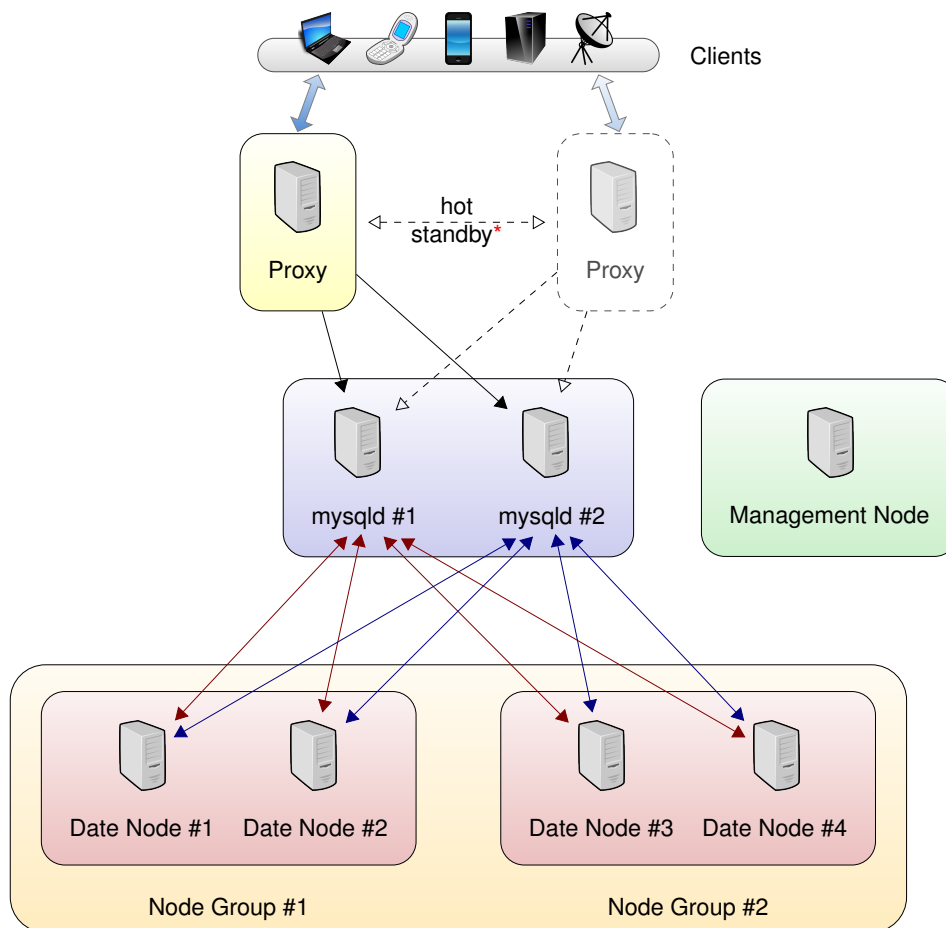


Figure 3: MySQL Cluster test setup

On the internet you are able to find different “howtos” to install and setup a MySQL-Cluster. In this example I needed a basic installation and setup of the MySQL-Cluster on every virtual machine (except the proxy machine). Due to this requirement I wrote a simple bash-script to automatically execute the above listed steps. The script is shown in Listing 5.

2.3 Setting up the MySQL-Servers

The first step after the installation process is to start up the MySQL Servers via `service mysql start` (here `mysc-daemon-1` or `mysc-daemon-2`) and to check if the MySQL Cluster engine is properly installed. Therefore you open MySQL on the shell by typing `mysql` (you may have to add a optional user plus its credentials). If you are logged in, type “`SHOW ENGINES;`” to see if the `ndbcluster` engine is installed. The field `support` must have the value `DEFAULT` or `YES`.

Now you should check if IP addresses or hostnames are working in your configuration files. In my example I had to alter the hostnames to their ip-addresses, because I experienced some errors with the hostname setup. You should also check if the data nodes (`mysc-node-1` to `mysc-node-4`) are accessible by the MySQL Servers and that the firewalls (if any exists) are configured properly.

As a next step the default database `mysql` and its tables must be converted to the `ndbcluster` storage engine, so that all changes are globally stored (this avoids duplicated or not synchronised data on the MySQL servers). You can use the sql script in Listing 1 to do that job.

Listing 1: convert the standard *mysql* database and its tables to the *ndbcluster* engine

```

1 use mysql;
2 ALTER TABLE mysql.user ENGINE=NDBCLUSTER;
3 ALTER TABLE mysql.db ENGINE=NDBCLUSTER;
4 ALTER TABLE mysql.host ENGINE=NDBCLUSTER;
5 ALTER TABLE mysql.tables_priv ENGINE=NDBCLUSTER;
6 ALTER TABLE mysql.columns_priv ENGINE=NDBCLUSTER;
7 ALTER TABLE mysql.func ENGINE=NDBCLUSTER;
8 ALTER TABLE mysql.proc ENGINE=NDBCLUSTER;
9 ALTER TABLE mysql.procs_priv ENGINE=NDBCLUSTER;
10 SET GLOBAL event_scheduler = 1;
11 CREATE EVENT 'mysql'.flush_priv_tables ON SCHEDULE EVERY 30 second ON COMPLETION
    PRESERVE DO FLUSH PRIVILEGES;

```

After successfully running that script, shut down the MySQL server. In order to save configuration time, copy `/opt/mysql_cluster/data/mysql` to the other MySQL server machine. Make sure both servers are shut down during copying.

2.4 Setting up the MySQL-Management-Node

The installation process (described in subsection 2.2) also installs a management console. You can use the management console on every machine in the cluster, but it is better to separate data nodes, MySQL server nodes and management nodes on different machines. In case of a system failure on one of the servers or data nodes the management machine will still be available.

The setup is very easy, you only have to create the configuration file `/var/lib/mysql-cluster/config.ini` with content listed in Listing 6. The file contains all necessary configuration data for the whole cluster e.g. which machine is the management node, which machines are server and data nodes and how much memory should be reserved for the data nodes (max. database size).

After creating the config file, navigate to the directory `/var/lib/mysql-cluster` to start the management console initially with this command:

```

ndb_mgmd --initial /
    -f /var/lib/mysql-cluster/config.ini /
    --config-dir=/var/lib/mysql-cluster/

```

The meaning of this command and its parameters is:

- `ndb_mgmd`: this command starts the management console daemon
- `--initial`: do a initial startup of the management node
- `-f <FILE>`: load the following config file to the server and to the data nodes
- `--config-dir=<DIR>`: store all config and logging data to this directory (on management and data nodes)

For “hard” restart on sever errors or failures, you can use the script in Listing 2.

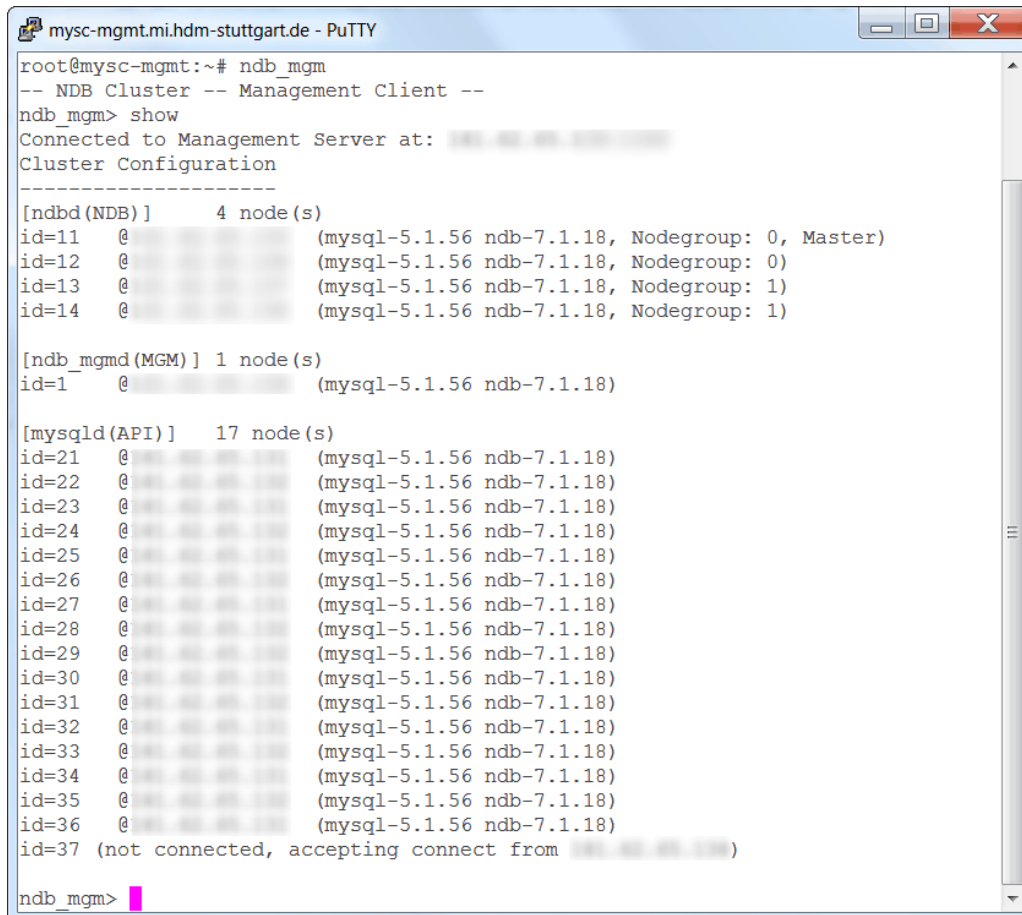
Listing 2: do a hard reset of the MySQL-Cluster

```

1 #!/bin/bash
2 MY_PWD=pwd;
3 echo "remove all old files";
4 cd /var/lib/mysql-cluster;
5 rm -rf ndb_*;
6 echo "restart ndbd service";
7 ndb_mgmd --initial -f /var/lib/mysql-cluster/config.ini --config-dir=/var/lib/mysql-cluster;
8 cd $MY_PWD;

```

After successfully setting up and configuring the management node, you can open the management console via the `ndb_mgm` command. The `show` command lists all available nodes in the cluster (see Figure 4) and the command `all report` shows some statistics about the data nodes.



```

mysc-mgmt.mi.hdm-stuttgart.de - PuTTY
root@mysc-mgmt:~# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: [REDACTED]
Cluster Configuration
-----
[ndbd (NDB)]      4 node(s)
id=11 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18, Nodegroup: 0, Master)
id=12 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18, Nodegroup: 0)
id=13 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18, Nodegroup: 1)
id=14 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18, Nodegroup: 1)

[ndb_mgmd (MGM)] 1 node(s)
id=1  @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)

[mysqld (API)]   17 node(s)
id=21 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=22 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=23 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=24 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=25 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=26 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=27 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=28 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=29 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=30 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=31 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=32 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=33 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=34 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=35 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=36 @ [REDACTED] (mysql-5.1.56 ndb-7.1.18)
id=37 (not connected, accepting connect from [REDACTED])

ndb_mgm>

```

Figure 4: management node: list status of all nodes with the `show` command

Notice

You can set up several management nodes on different machines to get fault tolerant management consoles

2.5 Setting up the MySQL-Data-Nodes

After setting up the management node(s), the data nodes can be started up to fetch the configuration files from the management node(s). To achieve this, you have change to the folder `/var/lib/mysql-cluser/` and execute the following command:

```

ndbd --initial /
      --connect-string='host=<HOST OR IP>:<PORT>'

```

Parts of the command have the following meaning:

- `ndbd`: this command starts the data node
- `--initial`: do a initial startup of the data node
- `--connect-string='host=<HOST OR IP>:<PORT>'`: this connection string defines the hostname or IP address plus port number of the management node to connect to

To do a “hard” restart on sever errors or failures, you can use the script in Listing 3.

Listing 3: do a hard reset of the MySQL-Cluser

```

1 #!/bin/bash
2 MY_PWD=pwd;
3 echo "remove all old files";
4 cd /var/lib/mysql-cluster;
5 rm -rf ndb_*;
6 echo "restart ndbd service";
7 ndbd --initial --connect-string=HOST:PORT;
8 cd $MY_PWD;

```

2.6 Setting up the MySQL-Proxy

Installing the *MySQL-Proxy* package can be achieved by executing the command `apt-get install mysql-proxy` under Debian Linux. Once the installation process has finished, create the folder `/etc/mysql-proxy` and switch to that folder. The next step is to create the config file `/etc/mysql-proxy/mysql-proxy.conf` with configuration settings depending on your setup. In Listing 8 you can see some example settings for this MySQL Cluster setup.

MySQL-Proxy also has support for admin and reporter scripts respectively interfaces. You can find some examples in the *MySQL Proxy Guide* [3].

Starting up the proxy is easy by firing the following command:

```
mysql-proxy --defaults-file=/etc/mysql-proxy/mysql-proxy.conf
```

Argument `--defaults-file` defines the default configuration file with which the proxy should be started – in this example our previously created configuration file.

After starting up the proxy, the MySQL server access settings must be adjusted. Accessing a MySQL server is only allowed from *localhost* by default. In this example setup we don’t want to allow access from various clients directly to the MySQL servers, but we want to allow access over the load balancer (MySQL Proxy) to the servers. Therefore the MySQL user settings have to be adjusted.

This can be achieved by logging into one of the MySQL server via the `mysql` console. By executing the sql-command `SELECT host,user,password FROM mysql.user;` you can see the current users and their access host. To allow the access over the proxy to the servers, a new user has to be created e.g. “mysql-proxy”.

Listing 4: create a new user and allow only the access over the proxy

```

1 "cluster-user"@"ADDRESS-OR-IP-OF-PROXY" IDENTIFIED BY "PASSWORD";
2
3 GRANT ALL PRIVILEGES ON *.*
4 TO "cluster-user"@"ADDRESS-OR-IP-OF-PROXY" IDENTIFIED BY "PASSWORD";

```

In Listing 4 a new user called “mysql-proxy” is created. The statement `"ADDRESS-OR-IP-OF-PROXY"` defines the access of the proxy, so “mysql-proxy”-users are only allowed to access the MySQL servers over this host respectively proxy. Lines 3 and 4 set some rights, here the full access to all databases and their tables. Notice: Be careful granting all rights to a user, especially if third-party users have access to your systems! See [4] to get more information about this topic.

You should now be able to access the MySQL Cluster server over the proxy e.g. by this JDBC resource address `jdbc:mysql://PROXY-ADDRESS:PORT/DATABASE` and the corresponding credentials.

3 TESTS

The objective of a database (load) test is to get some numbers of maximum throughput. These numbers are important to calculate how much concurrent users a database system can handle until the maximum

throughput is reached and the response time of each request increases or breakdown. You are able to calculate your required system and hardware resources better with the knowledge of this maximum turning point (see Figure 5).

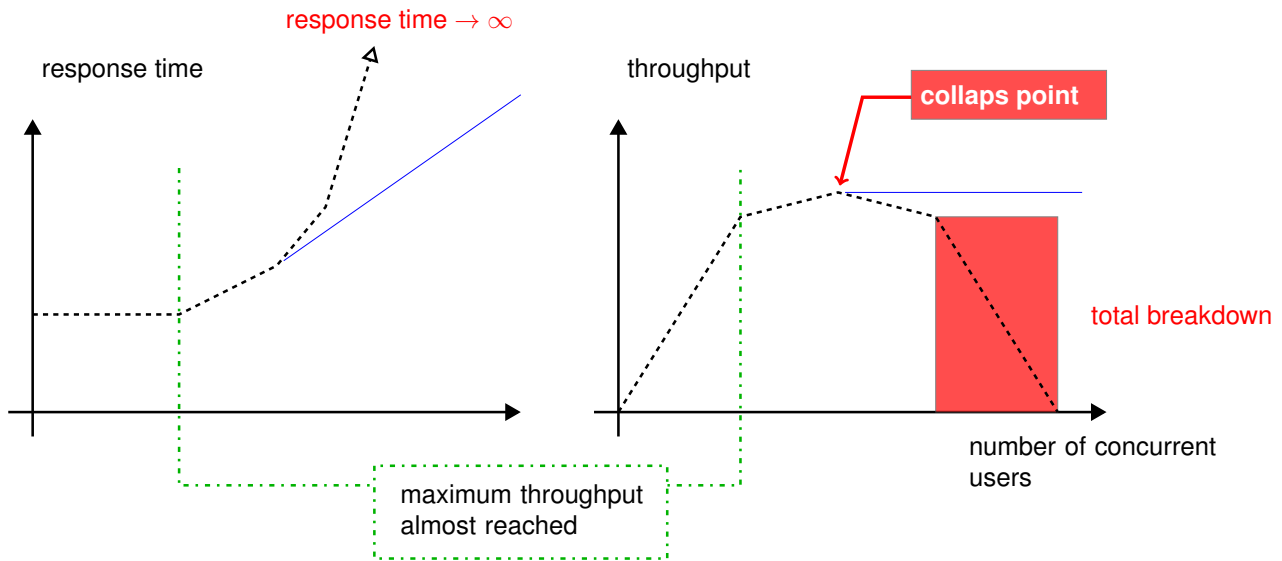


Figure 5: decay curve of a load test

In this test I want to determine how much concurrent requests on simple tables and with joins between maximum two tables a MySQL-Cluster setup can handle. Therefore I set up a system as described in subsection 2.1. This test contains only a simple database load test without a application server, because this may tamper the results.

As a dummy test scenario I created a simple state diagram for user and the resulting database interactions in a social community (see Figure 6). As you can see the user is able to log into the application, add and display friends, read and write messages or show latest status updates with the corresponding comments. The sql database layout is shown in Listing 9.

This scenario results in different read requests to the database with different ratios:

- login → get user data (1%)
- messages → load a messages / conversation between two users (25%)
- friends → load the user's friends (4%)
- status updates → get latest 10 status messages of a user (70%)

These resulted in four different tests, which were executed with a total number of 1000 test users. To accomplish this, a JMeter cluster with one JMeter management node and three load generating clients was also set up. The different request ratios were modelled by using connection pools in JMeter with a different number of maximum connections.

The test time was set to a fixed value of 20 minutes and the rampup period was set to 2 minutes to constantly increase the number of users to “warm up” the database.

JMeter was the reason why the test only contains read and no write requests (see section 6).

4 CREATING THE TEST DATABASE DATA

For this test “big data” is needed, because MySQL-Cluster is supposed to handle a lot of data efficiently. The creation of diverse and realistic test data is challenging!

I set myself a goal to create 100.000 users and every user should also have 100 friends. There should also be 1 to 50 randomly created status update text messages per user. In addition there should also be 1 to 10

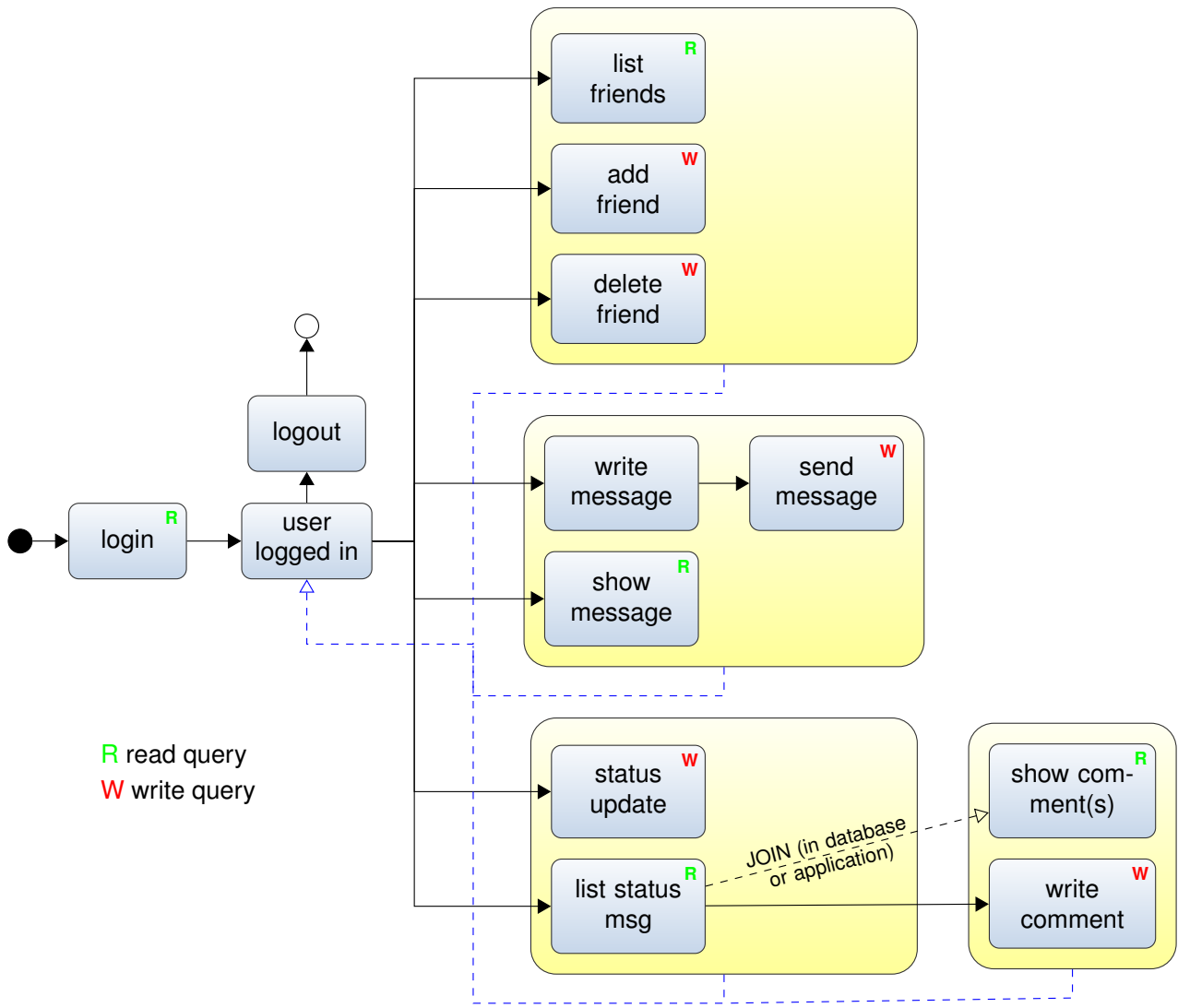


Figure 6: dummy state diagram for the test cases

randomly created messages between a user and a randomly chosen second user. These preconditions may result in over 10 million records divided over several tables.

Creating such a huge amount of data in a short time is a challenge. You can insert record after record one by one into the database, but this will probably result in a very long inserting process. A better solution would be to insert data into bulks e.g. by using the sql command `INSERT INTO [...] VALUES ([...]), ([...]), ([...]), ...`. This solution may work if you are creating test data from scratch. In case you are using old user data and you want to randomly select users to create the friend relation between them, this solution may be to slow.

The bottleneck creating huge data is the *data access layer* (short *DAL*) and the network between the application and database server. Sending, transporting and receiving data over the network comes with significant latencies. After that, the data must be analysed by the application and if necessary sent back to the database. This process slows down the insertion process (see Figure 7).

Therefore I created a stored procedure in the database (see Listing 9 line 50 – 100), which creates 100 friend relations with randomly selected users from a user table. The algorithm to select users randomly is extremely efficient and it also detects holes² in the table. The stored procedure is more complex than

2. A *hole* in database table occurs if data gets inserted and deleted.

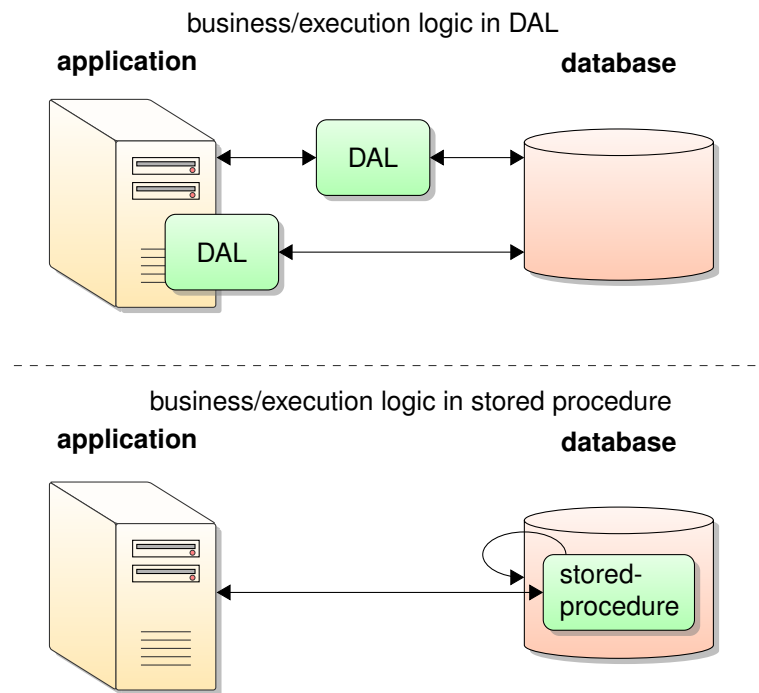


Figure 7: business/execution logic in DAL or stored procedures directly in the database

using `INSERT INTO . . .` and not easily maintainable, but it is fast and there is no time lost by a round trip through the DAL. To speed up the insertion process even more, the stored procedure is not called via an application on a remote client (this approach was still too slow); instead it was fired directly from the database server in parallel.

Overall, the data creation process for creating user and friend relation data was done in about 2 to 6 hours, depending on the workload of the underlying virtual machine.

The other data for this test (messages, status updates) have to be created on a remote client, because they should contain random text data. Therefore I used a simple script, which inserted some *lorem ipsum* data via bulk inserts into the database. This process took about 1 to 2 hours, depending on how many messages should be created and how high the virtual machine was utilized.

5 RESULTS

Overall I ran two different tests: test 1) with default JMeter settings and test 2) with reduced JMeter logging information to conserve network bandwidth.

In test 1) I ran all four previously defined queries (see section 3) at their specified ratios. The results with the average execution time, response latency and requests throughput are displayed in Figure 10 to 13 (see appendix “test results”). In Figure 8 you can see the summed up average throughput of all queries in test 1).

After the ramp up period of 2 minutes plus and some commute time (20 to 30 seconds), the test ran with an average throughput of about 3000 requests per second. There are also some deviations at about 650, 720, 790, 860, 930 and 1080 seconds, which were caused by some interrupts on the JMeter server or clients (system or user interactions).

In test 1) I noticed that the MySQL application server ran at a CPU utilization of 125 – 143 percent³ and the data nodes server at 49 – 54 percent. The application server is running on a quad core machine and each

3. One CPU can be utilized up to a maximum of 100%. In a multi-core system the CPU utilization often gets summed up, so it is possible that the utilization is $\geq 100\%$.

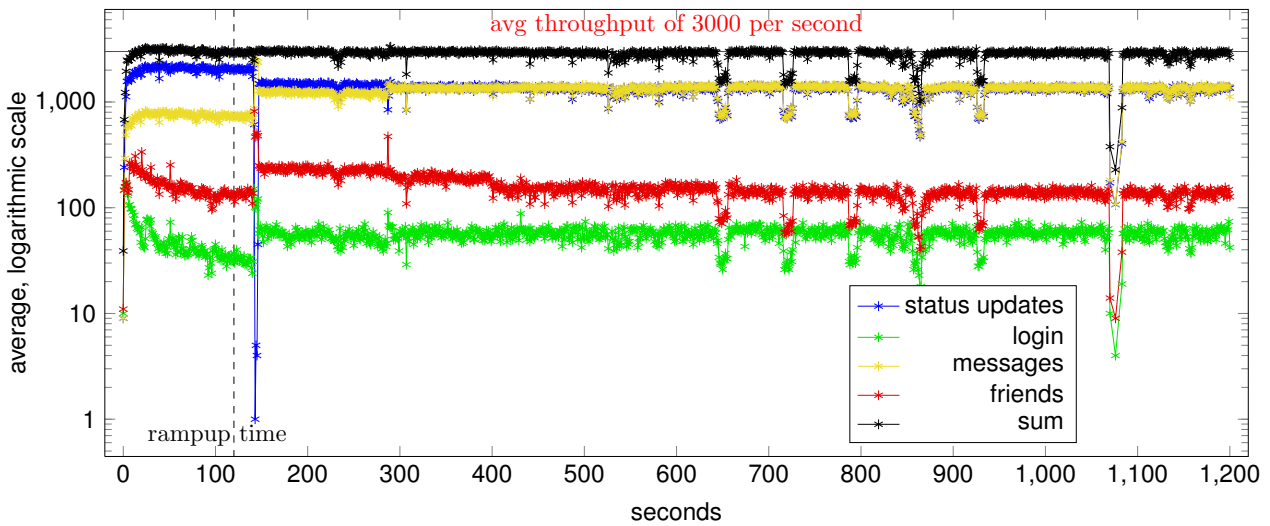


Figure 8: test 1: overall test results, throughput

data node on a dual core machine, so the servers are not efficiently used and there is still enough room for performance improvements.

A quick analysis revealed that the network bandwidth was used up to 100% and therefore was throttling the requests' throughput. As a consequence I reduced the JMeter logging information in test 2) to a minimum and restarted the test. Figure 14 to 17 (see appendix "test results") shows an overview of the average throughput of all requests summed up.

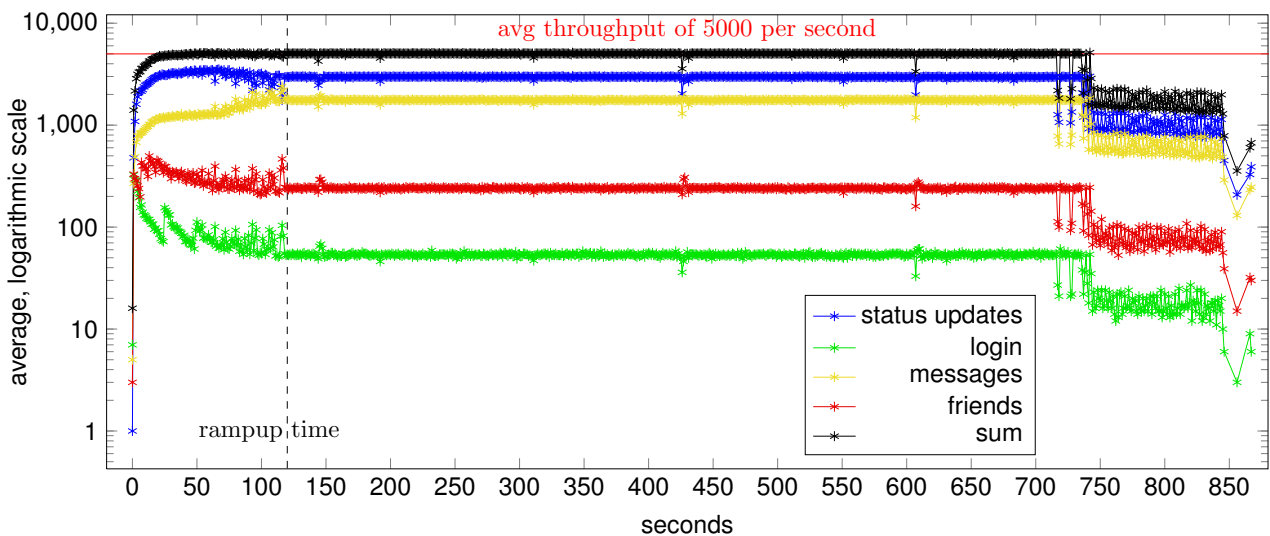


Figure 9: test 2: overall test results, throughput

Again, after a ramp up period of 2 minutes the test ran with a maximum average throughput of about 5000 requests per second. As you can see, there are less peaks caused by interrupts in test 2). After 720 seconds the requests throughput decreased, which was caused by a stuck MySQL-Proxy (see section 6).

Analysing the data showed that the application server was still not running efficiently at a load of 191 – 213 percent. The data nodes had a load between 69 – 72 percent. This time I had run a network monitor along, which showed that the network was again the throttling factor. Reducing JMeter's logging data wouldn't gain any further improvement, because the logging data was already reduced to its minimum.

6 PROBLEMS

The first problem I ran into was JMeter: the test application was not able to execute update and delete statements with prepared statement data. The result statements of the database server and JMeter were not helping either to solve that problem. A test with raw update and delete sql-queries showed that these queries were working fine. Therefore I had to discard all write tests with JMeter.

Another problem occurred randomly in the *MySQL proxy* after some time. The proxy had a CPU utilization of 100 percent and got stuck. Curiously, there were no entries in the log files and now error statements. A search for this strange behaviour was without any result. The only solution that worked was to “kill” the proxy application, so it was difficult to execute all tests successfully. In a live environment, I would probably not rely on *MySQL Proxy* and rather use another proxy application or a hardware load balancer.

During the tests I ran into another problem: the network capacity was at its limit. Reducing the transferred logging data produced by JMeter lowered that issue only a little bit. In a real world scenario a single 1 GBit/s interface would not be enough to handle all the resulting requests generated by JMeter.

7 CONCLUSION

The test showed that it is possible to read more than 3000 requests concurrently from a specialized MySQL database. MySQL-Cluster was designed to handle such high number of requests.

The results also showed that the database in this test scenario was not fully utilized during tests and probably had been able to handle a more requests. I tried to improve the throughput by reducing the logging and meta data of JMeter, but the network bandwidth was still the limiting factor in this test setup. In a real world scenario I would probably configure the system to use more than one network interface in parallel or install a 10 GBit/s network interface. I would also use real physical machines rather than virtual machines, because the virtual IO mapping in this test scenario is a limiting factor too.

Setting up a MySQL-Cluster is not easy at the beginning, but you get used to the behaviour of the database and know how to handle it soon. The management console is a really handy tool to manage the whole cluster and to do planned system updates or maintenance. If you know how to design database tables for a distrusted application (less table columns and joins), you get a really powerful, high available and easily scalable in memory database which can be used in addition to existing MySQL database servers.

APPENDIX

REFERENCES

- [1] Sun Microsystems Inc., *Leistungsoptimierung für das Datenbanksystem MySQL Cluster*, 2010.
- [2] “MySQL Cluster CGE,” [Online Available 2012-10-01]. [Online]. Available: <http://www.mysql.com/products/cluster/>
- [3] Oracle, “MySQL Proxy Guide,” December 2011, [Online; accessed 18-August-2012]. [Online]. Available: <http://downloads.mysql.com/docs/mysql-proxy-en.pdf>
- [4] —, “MySQL GRANT Syntax,” [Online; accessed 18-August-2012]. [Online]. Available: <http://dev.mysql.com/doc/refman/5.1/en/grant.html>

SCRIPTS

Listing 5: `install-ndb-cluster-package.sh` – script to automatic download, install and pre-set up mysql cluster

```

1 #!/bin/bash
2 echo ">> get the ndb-cluster tar package from mysql.com ...";
3 cd /usr/local;
4 wget http://www.mysql.com/get/Downloads/MySQL-Cluster-7.1/mysql-cluster-gpl-7.1.18-linux-x86_64-glibc23.tar.gz/from/http://sunsite.informatik.rwth-aachen.de/mysql/;
```

```

5 mv index.html mysql-cluster-gpl-7.1.18-linux-x86_64-glibc23.tar.gz;
6
7 echo ">> un-tar package ...";
8 tar -zxvf mysql-cluster-gpl-7.1.18-linux-x86_64-glibc23.tar.gz;
9 rm -f mysql-cluster-gpl-7.1.18-linux-x86_64-glibc23.tar.gz;
10 ln -s mysql-cluster-gpl-7.1.18-linux-x86_64-glibc23 mysql;
11 cd mysql;
12
13 echo ">> adding mysql group and user ...";
14 groupadd mysql;
15 useradd -g mysql mysql;
16
17 echo ">> create the mysql servers config file";
18 cat > /etc/my.cnf <<DELIM
19 [mysqld]
20 datadir = /opt/mysql_cluster/data
21 socket = /opt/mysql_cluster/mysql.sock
22 basedir = /usr/local/mysql
23
24 event_scheduler = on
25 default-storage-engine = ndbcluster
26 ndbcluster
27 # IP/host of the NDB_MGMT-node(s)
28 ndb-connectstring = mysc-mgmt.mi.hdm-stuttgart.de
29
30 [mysqld_safe]
31 err-log = /var/log/mysqld.log
32 pid-file= /var/run/mysqld/mysqld.pid
33 DELIM
34 mkdir -p /var/run/mysqld;
35
36 echo ">> create a symbolic link to the mysql.sock file in '/tmp' to avoid startup problems";
37 ln -s /opt/mysql_cluster/mysql.sock /tmp/mysql.sock;
38
39 echo ">> create mysql dirs and install mysql server";
40 mkdir -p /var/lib/mysql-cluster/;
41 mkdir -p /opt/mysql_cluster/data/;
42 cd /usr/local/mysql/;
43 /usr/local/mysql/scripts/mysql_install_db;
44 chown -R root:mysql /usr/local/mysql/;
45 chown -R mysql:mysql /opt/mysql_cluster/;
46
47 echo ">> copy mysql startup script to init.d ...";
48 cp /usr/local/mysql/support-files/mysql.server /etc/init.d/mysql;
49
50 echo ">> ready to start the MySQL server or the ndb-cluster ...";

```

Listing 6: `config.ini` – the config file for the MySQL Cluster management node

```

1 [TCP DEFAULT]
2 #PortNumber = 2202
3 SendBufferMemory = 2M
4 ReceiveBufferMemory = 2M
5
6
7 [NDB_MGMD DEFAULT]
8 PortNumber = 1186
9
10 [NDB_MGMD]
11 HostName = mysc-mgmt.mi.hdm-stuttgart.de
12 NodeId = 1
13 LogDestination = FILE:filename=ndb_1_cluster.log,maxsize=10000000,maxfiles=6

```

```
14 ArbitrationRank = 1
15
16 # 2 Managment Server (not configurated at the moment)
17 #[NDB_MGMD]
18 #HostName =
19 #NodeId = 2
20
21
22 [NDBD DEFAULT]
23 NoOfReplicas = 2
24 DataDir = /var/lib/mysql-cluster
25 DataMemory = 2560M
26 IndexMemory = 768M
27
28 MaxNoOfExecutionThreads = 4
29 MaxNoOfConcurrentTransactions = 8192
30 MaxNoOfConcurrentOperations = 65536
31
32 [NDBD]
33 HostName = mysc-node-1.mi.hdm-stuttgart.de
34 NodeId = 11
35 [NDBD]
36 HostName = mysc-node-2.mi.hdm-stuttgart.de
37 NodeId = 12
38 [NDBD]
39 HostName = mysc-node-3.mi.hdm-stuttgart.de
40 NodeId = 13
41 [NDBD]
42 HostName = mysc-node-4.mi.hdm-stuttgart.de
43 NodeId = 14
44
45
46 [MYSQLD DEFAULT]
47 DefaultOperationRedoProblemAction = QUEUE
48 BatchSize = 512
49 #BatchByteSize = 2048K
50 #MaxScanBatchSize = 2048K
51
52 # 2 MySQL Nodes
53 [MYSQLD]
54 HostName = mysc-daemon-1.mi.hdm-stuttgart.de
55 NodeId = 21
56 [MYSQLD]
57 HostName = mysc-daemon-1.mi.hdm-stuttgart.de
58 NodeId = 23
59 [MYSQLD]
60 HostName = mysc-daemon-1.mi.hdm-stuttgart.de
61 NodeId = 25
62 [MYSQLD]
63 HostName = mysc-daemon-1.mi.hdm-stuttgart.de
64 NodeId = 27
65 [MYSQLD]
66 HostName = mysc-daemon-2.mi.hdm-stuttgart.de
67 NodeId = 22
68 [MYSQLD]
69 HostName = mysc-daemon-2.mi.hdm-stuttgart.de
70 NodeId = 24
71 [MYSQLD]
72 HostName = mysc-daemon-2.mi.hdm-stuttgart.de
73 NodeId = 26
74 [MYSQLD]
```

```

75 HostName = mysc-daemon-2.mi.hdm-stuttgart.de
76 NodeId = 28
77
78 # additional slots ...
79 [MYSQLD]
80 [MYSQLD]
81 [MYSQLD]
82 [MYSQLD]
83 [MYSQLD]
84 [MYSQLD]
85 [MYSQLD]
86 [MYSQLD]
87 ### SLOTS (one for each ndb_mgmd) FOR HELPER APPLICATIONS SUCH AS ndb_show_tables etc
88 [MYSQLD]
89 Hostname = mysc-mgmt.mi.hdm-stuttgart.de

```

Listing 7: my.cnf – the config file for the MySQL Server nodes

```

1 [MYSQLD]
2 datadir = /opt/mysql_cluster/data
3 socket = /opt/mysql_cluster/mysql.sock
4 basedir = /usr/local/mysql
5
6 event_scheduler = on
7
8 sort_buffer_size = 512K
9 key_buffer_size = 16M
10 max_allowed_packet = 16M
11
12 # query cache -> may be inefficient on ndb!
13 query_cache_type = 2
14 query_cache_limit = 2M
15 query_cache_size = 64M
16 query_cache_min_res_unit= 4K
17
18 thread_cache_size = 200
19 # thread_concurrency = 2 * (no. of CPU)
20 thread_concurrency = 8
21 # The number of threads that have taken more than slow_launch_time seconds to create
22 set=low_launch_threads = 1
23
24 #max_connections = 500
25 #max_user_connections = 150
26
27 table_cache = 1024
28
29 # log slow queries
30 #log-slow-queries = slow.log
31 #long_query_time = 2
32 #log-queries-not-using-indexes
33
34 # cluster settings
35 default-storage-engine = ndbcluster
36 ndbcluster
37 # IP/host of the NDB_MGMT-node(s)
38 ndb-connectstring = mysc-mgmt.mi.hdm-stuttgart.de
39 ndb-cluster-connection-pool = 8
40
41 # InnoDB
42 skip-innodb
43
44 [MYSQL]

```

```

45 socket = /opt/mysql_cluster/mysql.sock
46
47 [MYSQL_CLUSTER]
48 ndb-connectstring = mysc-mgmt.mi.hdm-stuttgart.de
49
50 [MYSQLD_SAFE]
51 err-log = /var/log/mysqld.log
52 pid-file= /var/run/mysqld/mysqld.pid

```

Listing 8: `mysql-config.conf` – the config file for the MySQL Proxy

```

1 [mysql-proxy]
2 daemon = true
3 keepalive = true
4 event-threads = 2
5 proxy-skip-profiling = true
6 proxy-address = mysc-proxy.mi.hdm-stuttgart.de:4040
7
8 # do not change allow the "CHANGE USER" command
9 proxy-pool-no-change-user = true
10
11 ## backend servers ##
12 proxy-backend-addresses = mysc-daemon-1.mi.hdm-stuttgart.de:3306,mysc-daemon-2.mi.hdm-stuttgart.de:3306
13
14 ## log stuff ##
15 log-file = /var/log/mysql-proxy.log
16 log-level = warning
17 #log-backtrace-on-crash

```

Listing 9: `install.sql` – installation script for the test database

```

1 CREATE DATABASE IF NOT EXISTS 'fakebook' CHARACTER SET utf8 COLLATE utf8_general_ci;
2 USE 'fakebook';
3
4 CREATE TABLE IF NOT EXISTS 'fakebook'.users (
5   'id' bigint(20) unsigned NOT NULL AUTO_INCREMENT
6   , 'name' varchar(50) NOT NULL
7   , 'name_first' varchar(50) NOT NULL
8   , 'pass' varchar(40) NOT NULL
9   , 'mail' varchar(100) NOT NULL
10  , UNIQUE KEY 'id' ('id')
11  , UNIQUE KEY 'mail' ('mail')
12 );
13
14 CREATE TABLE IF NOT EXISTS 'fakebook'.relations (
15   'id_1' bigint(20) unsigned NOT NULL
16   , 'id_2' bigint(20) unsigned NOT NULL
17   , CONSTRAINT 'relations_ibfk_1' FOREIGN KEY ('id_1') REFERENCES 'fakebook'.users ('id') ON DELETE CASCADE
18   , CONSTRAINT 'relations_ibfk_2' FOREIGN KEY ('id_2') REFERENCES 'fakebook'.users ('id') ON DELETE CASCADE
19 );
20
21 CREATE TABLE IF NOT EXISTS 'fakebook'.relations_2 (
22   'uid' bigint(20) unsigned NOT NULL,
23   'friends' TEXT NOT NULL,
24   CONSTRAINT 'relations_2_ibfk_1' FOREIGN KEY ('uid') REFERENCES 'fakebook'.users ('id') ON DELETE
25   CASCADE
26 );
27
28 --
29 -- list all user's friends in a view

```



```

30 --
31 DROP VIEW IF EXISTS 'fakebook'. 'users_friends';
32 CREATE VIEW 'fakebook'. 'users_friends' AS
33     SELECT 'id'
34           , 'name'
35           , 'name_first'
36           , 'mail'
37           , COUNT( r.id_1 ) AS friends
38           , GROUP_CONCAT( r.id_2 ORDER BY r.id_2 ASC SEPARATOR " ") AS friend_ids
39 FROM 'users' AS u
40 LEFT JOIN relations AS r ON u.id = r.id_1
41 GROUP BY u.id;
42
43 --
44 -- stored procedure to insert random friend data
45 -- param-1: min. userID to start from
46 -- param-2: max. userID to go to --> min. to max. is th range
47 -- param-3: insert this number of friends at once
48 --
49 DROP PROCEDURE IF EXISTS 'fakebook'. 'addRandFriends';
50 DELIMITER $$
51 CREATE PROCEDURE 'fakebook'. 'addRandFriends'(IN iMin INTEGER unsigned, IN iMax INTEGER unsigned, IN
        numFriends INTEGER unsigned)
52 BEGIN
53     DECLARE a,b,v,rand_num INT;
54     SET v = iMin;
55     SET @stmt_text = "INSERT IGNORE INTO relations (id_1,id_2) VALUES ";
56
57     -- iterate over each user in the range
58     WHILE v <= iMax DO
59         SET b = numFriends;
60
61         WHILE b >= 1 DO
62             -- generate some random user-friend-id's ...
63             -- see: http://jan.kneschke.de/projects/mysql/order-by-rand/
64             SET rand_num = (SELECT r1.id FROM users as r1 JOIN (SELECT (RAND())*(SELECT MAX(id) FROM users)) AS
                id) AS r2 WHERE r1.id >= r2.id ORDER BY r1.id ASC LIMIT 1);
65             -- ... and insert values
66             SET @stmt_text = CONCAT(@stmt_text, "(, v, ,, rand_num, )");
67
68             IF v = iMax AND b = 1 THEN
69                 SET @stmt_text = CONCAT(@stmt_text, ",");
70             ELSE
71                 SET @stmt_text = CONCAT(@stmt_text, ",");
72             END IF;
73
74             SET b = b - 1;
75         END WHILE;
76
77         -- next user id ...
78         SET v = v + 1;
79     END WHILE;
80
81     -- disable unique constraints checks
82     SET unique_checks=0;
83     SET foreign_key_checks=0;
84
85     -- start a transaction to speed up the insert process
86     START TRANSACTION;
87
88     -- fire prepared insert statement

```

```

89 PREPARE stmt FROM @stmt_text;
90 EXECUTE stmt;
91 DEALLOCATE PREPARE stmt;
92
93 -- fire transaction
94 COMMIT;
95
96 -- re-enable checks ...
97 SET foreign_key_checks=1;
98 SET unique_checks=1;
99 END; $$
100 DELIMITER ;
101
102 --
103 -- stored procedure to insert random friend data, but into another way as before. we do not use foreign key constraints for
the friends, we'll store them all at once in a big TEXT field
104 -- param-1: min. userId to start from
105 -- param-2: max. userId to go to --> min. to max. is th range
106 -- param-3: insert this number of friends at once
107 --
108 DROP PROCEDURE IF EXISTS 'fakebook'. 'addRandFriendsV2';
109 DELIMITER $$
110 CREATE PROCEDURE 'fakebook'. 'addRandFriendsV2'(IN iMin INTEGER unsigned, IN iMax INTEGER unsigned, IN
numFriends INTEGER unsigned)
111 BEGIN
112 DECLARE a,b,v,rand_num INT;
113 SET v = iMin;
114 SET @stmt_text = "INSERT IGNORE INTO relations_2 (uid, friends) VALUES ";
115
116 -- iterate over each user in the range
117 WHILE v <= iMax DO
118 SET b = numFriends;
119 SET @uids = "";
120
121 WHILE b >= 1 DO
122 -- generate some random user-friend-id's ...
123 -- see: http://jan.kneschke.de/projects/mysql/order-by-rand/
124 SET rand_num = (SELECT r1.id FROM users as r1 JOIN (SELECT (RAND())*(SELECT MAX(id) FROM users)) AS
id) AS r2 WHERE r1.id >= r2.id ORDER BY r1.id ASC LIMIT 1);
125 -- ... and add them to a string
126 SET @uids = CONCAT(@uids, rand_num, " ");
127
128 SET b = b - 1;
129 END WHILE;
130
131 IF v = iMax THEN
132 SET @stmt_text = CONCAT(@stmt_text, "(" , v , " , " , @uids , " ,");
133 ELSE
134 SET @stmt_text = CONCAT(@stmt_text, "(" , v , " , " , @uids , " ,");
135 END IF;
136
137 -- next user id ...
138 SET v = v + 1;
139 END WHILE;
140
141 -- disable unique constraints checks
142 SET unique_checks=0;
143 SET foreign_key_checks=0;
144
145 -- start a transaction to speed up the insert process
146 START TRANSACTION;

```

```

147
148 -- fire prepared insert statement
149 PREPARE stmt FROM @stmt_text;
150 EXECUTE stmt;
151 DEALLOCATE PREPARE stmt;
152
153 -- fire transaction
154 COMMIT;
155
156 -- re-enable checks ...
157 SET foreign_key_checks=1;
158 SET unique_checks=1;
159 END; $$
160 DELIMITER ;
161
162 --
163 -- create a table to exchange messages between users
164 --
165 CREATE TABLE IF NOT EXISTS 'fakebook'. 'messages' (
166 'msg_id' bigint(20) unsigned NOT NULL AUTO_INCREMENT,
167 'sender' bigint(20) unsigned NOT NULL,
168 'receiver' bigint(20) unsigned NOT NULL,
169 'header' varchar(100) NOT NULL,
170 'date' timestamp NOT NULL,
171 'content' TEXT NOT NULL,
172 PRIMARY KEY ('msg_id'),
173 CONSTRAINT 'messages_ibfk_1' FOREIGN KEY ('sender') REFERENCES 'fakebook'. 'users' ('id') ON DELETE
CASCADE,
174 CONSTRAINT 'messages_ibfk_2' FOREIGN KEY ('receiver') REFERENCES 'fakebook'. 'users' ('id') ON DELETE
CASCADE
175 );
176
177 --
178 -- create a table for user notifications
179 --
180 CREATE TABLE IF NOT EXISTS 'fakebook'. 'notification' (
181 'notification_id' bigint(20) unsigned NOT NULL AUTO_INCREMENT,
182 'uid' bigint(20) unsigned NOT NULL,
183 'date' timestamp NOT NULL,
184 'content' TEXT NOT NULL,
185 PRIMARY KEY 'notification_id' ('notification_id'),
186 CONSTRAINT 'notification_ibfk_1' FOREIGN KEY ('uid') REFERENCES 'fakebook'. 'users' ('id') ON DELETE CASCADE
187 );
188
189 --
190 -- create a table for comments on user notifications
191 --
192 CREATE TABLE IF NOT EXISTS 'fakebook'. 'comments' (
193 'comment_id' bigint(20) unsigned NOT NULL AUTO_INCREMENT,
194 'notification_id' bigint(20) unsigned NOT NULL,
195 'uid' bigint(20) unsigned NOT NULL,
196 'date' timestamp NOT NULL,
197 'content' TEXT NOT NULL,
198 PRIMARY KEY 'comment_id' ('comment_id'),
199 CONSTRAINT 'comment_ibfk_1' FOREIGN KEY ('uid') REFERENCES 'fakebook'. 'users' ('id') ON DELETE CASCADE,
200 CONSTRAINT 'comment_ibfk_2' FOREIGN KEY ('notification_id') REFERENCES 'fakebook'. 'notification' ('notification_id')
ON DELETE CASCADE
201 );

```

TEST RESULTS

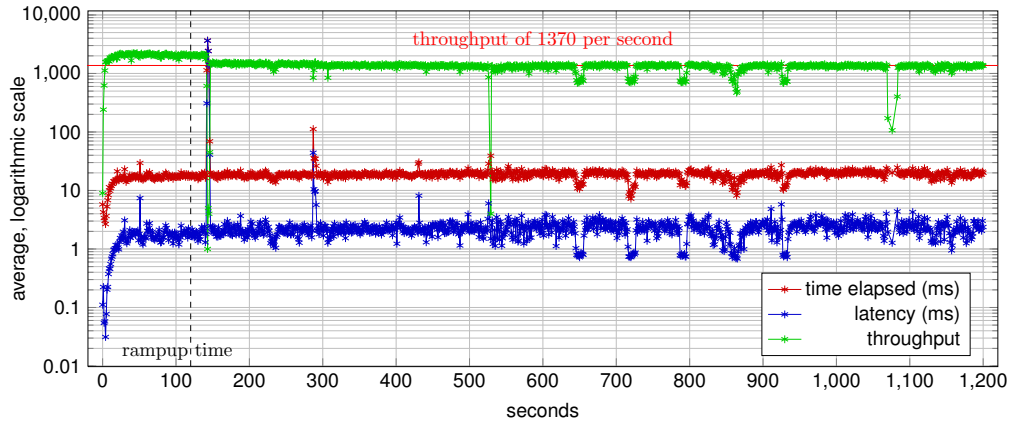


Figure 10: test 1: get 10 different status messages

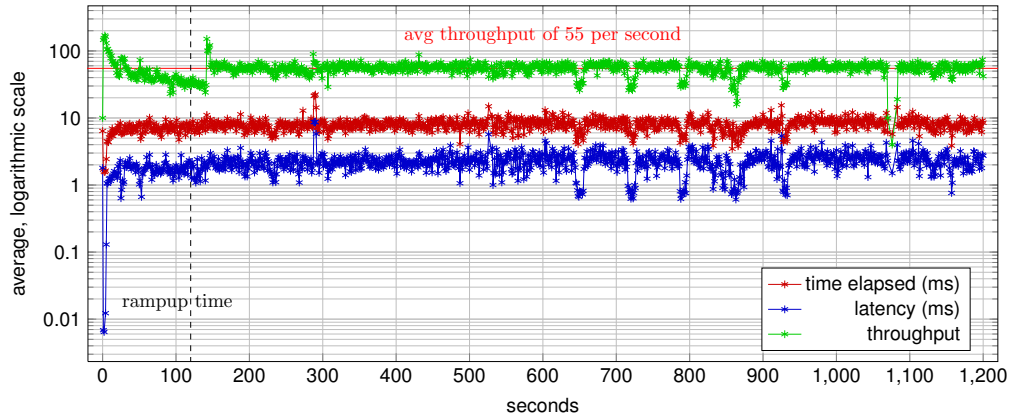


Figure 11: test 1: get a random user's login data

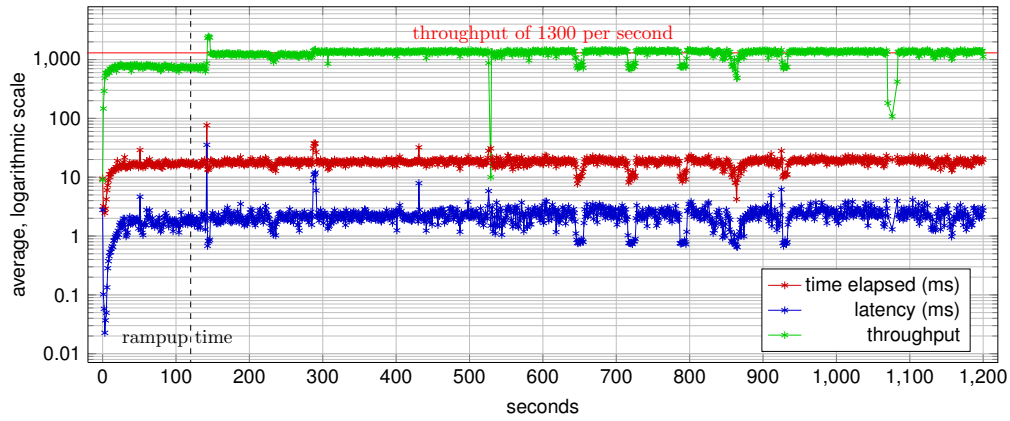


Figure 12: test 1: load a message between two users (one is chosen randomly)

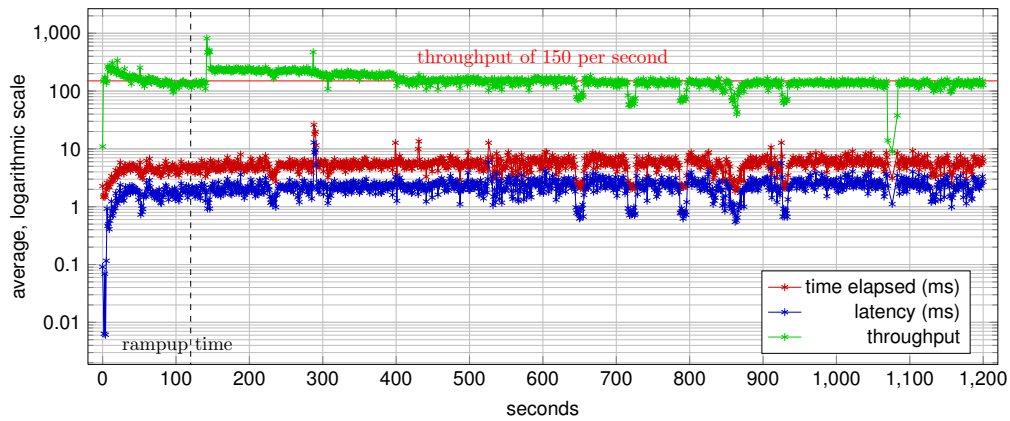


Figure 13: test 1: get all friends (uid) of a randomly chosen user

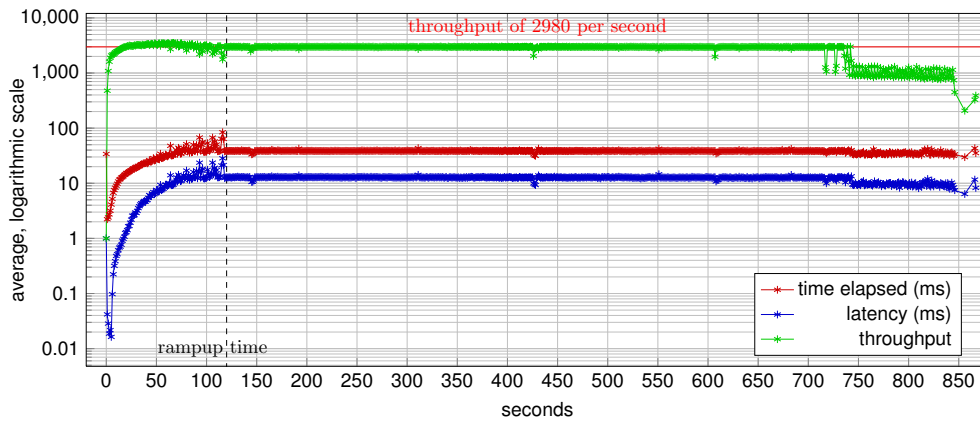


Figure 14: test 2: get 10 different status messages

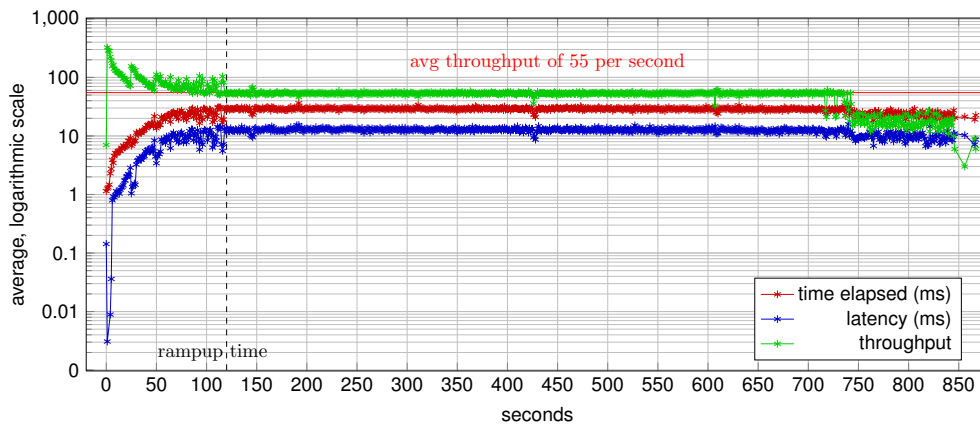


Figure 15: test 2: get a random user's login data

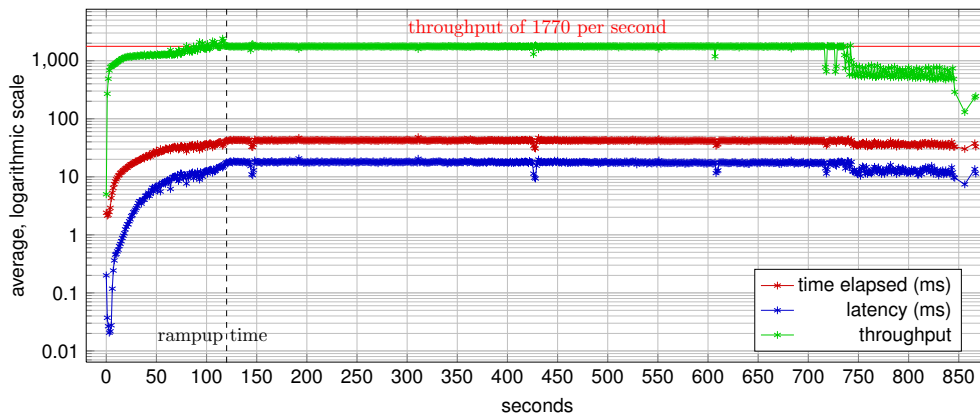


Figure 16: test 2: load a message between two users (one is chosen randomly)

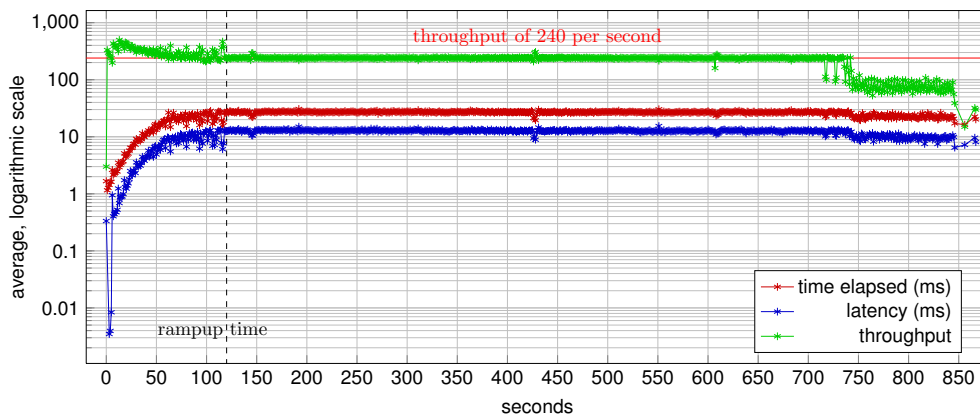


Figure 17: test 2: get all friends (uid) of a randomly chosen user