File server with backend authentication

Major Amit Pathania 163054001
Nivia Jatain 15305R007

Indian Institute of Technology Bombay

November 16, 2017

Table of Contents

System Architecture

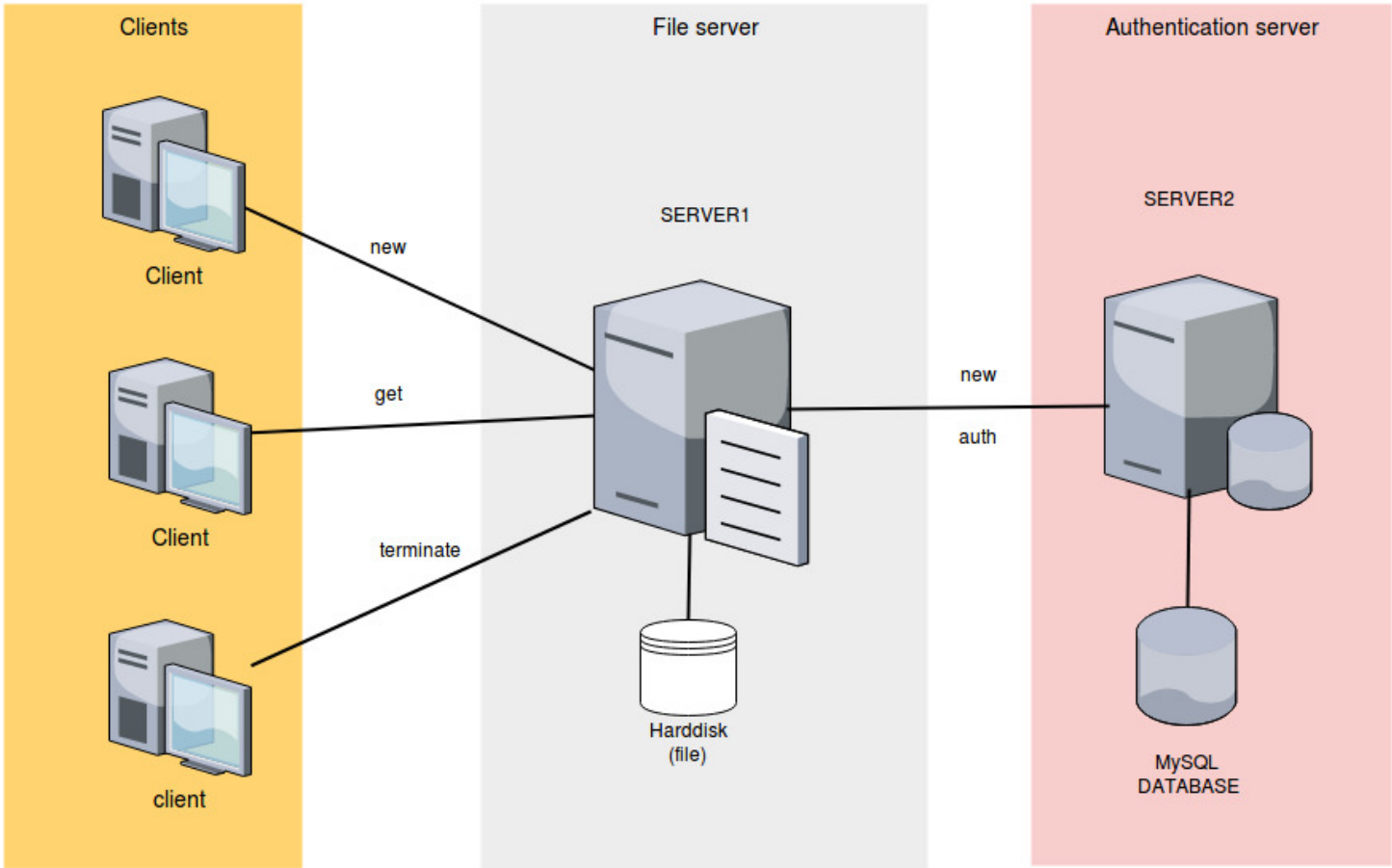
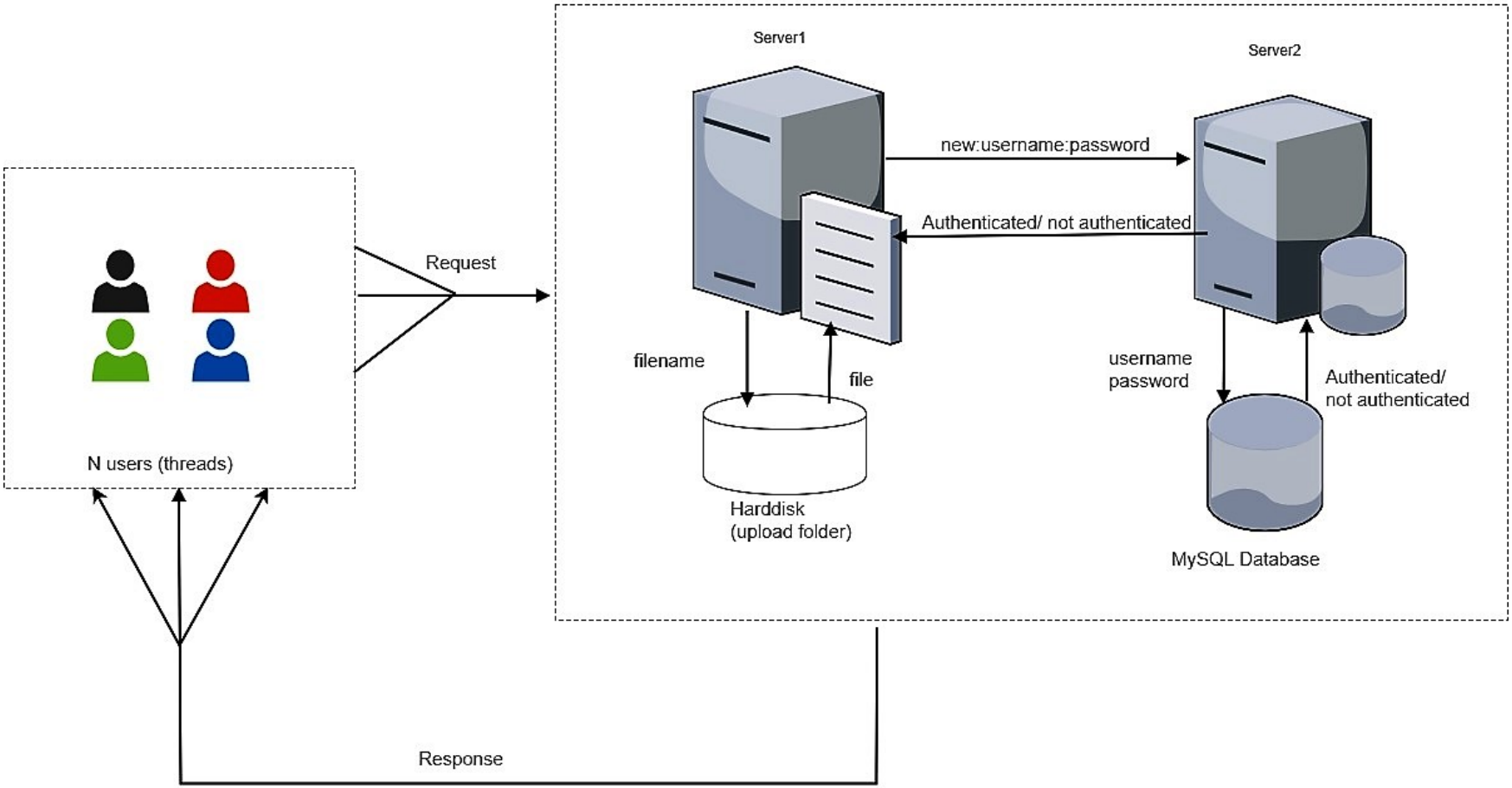


Figure: System Architecture

Load Generator



Load Generator

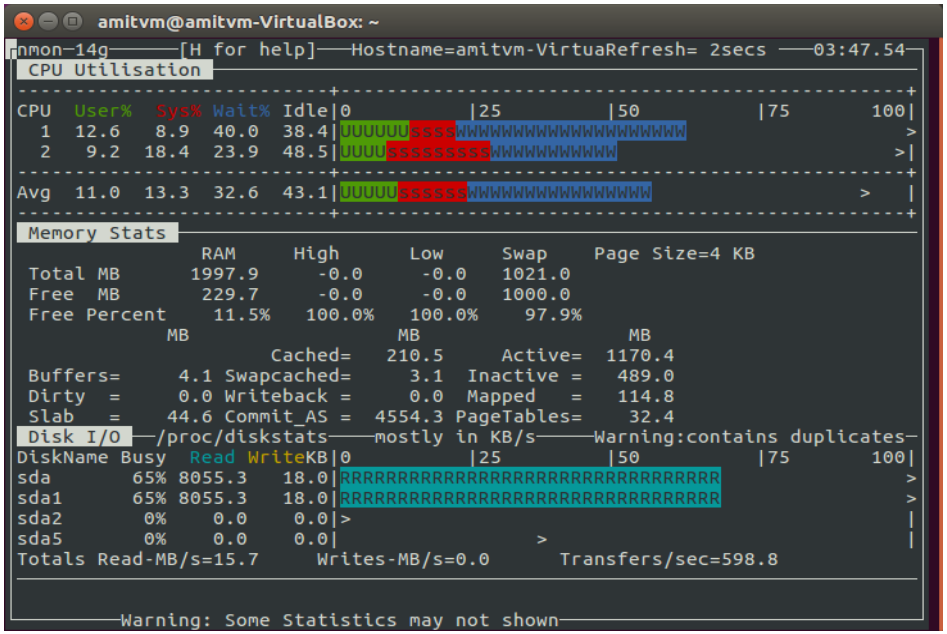
- Closed loop test ie new job arrivals are only triggered by job completions.
- N threads created to simulate N concurrent users. Each of the N threads emulate one user by issuing one request, waiting for it to complete, and issues the next request immediately afterwards.There is no think time between requests.
- Load generator can generate two types of load (or requests): **To create new user account** and **To request file from server1**.
- For each N, we defined either total runtime ie 100-120 seconds or defined number of total fetch file requests.For instance, 2000 file request for each user. So, N=1,there were total 2000 requests,for N=2,total 4000 requests, for N=10, total 20000 requests and so on.

Phase2 bottleneck

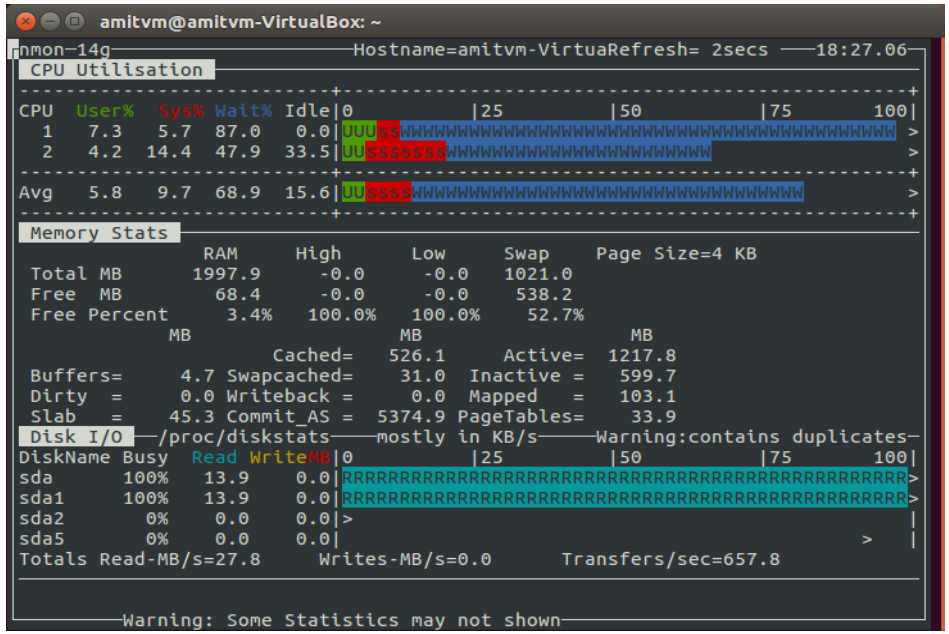
Load type	File size	N*	Mean Response Time at N=N*	Max Throughput at N=N*	Bottleneck
new		800	0.219622	3363.49	Disk I/O (Write) and MySQL
get	1KB	17	0.0054039	3093.7	Disk I/O (Read)
get	10KB	10	0.00390524	2547.3	Disk I/O (Read)
get	100KB	5	0.0113418	430.355	Disk I/O (Read)
get	1MB	2	0.0265555	75.1187	Disk I/O (Read)
get	10MB	1	0.0984429	10.1575	Disk I/O (Read)
get	100MB	1	0.880579	1.1356	Disk I/O (Read)

Table: Test results before optimisation

Phase2 bottleneck



100KB



1MB

Phase2 bottleneck

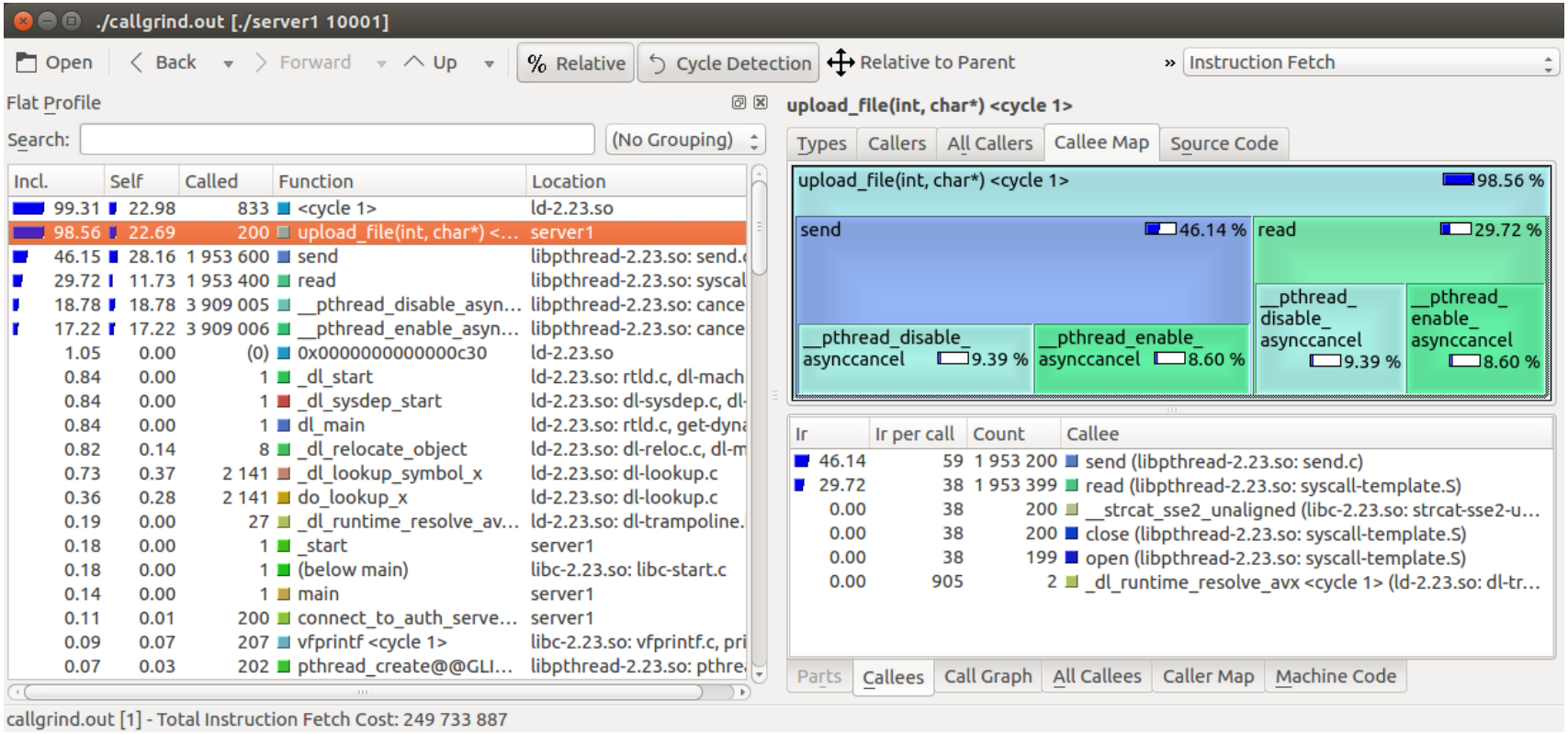


Figure: Profiling the server code

Optimisation

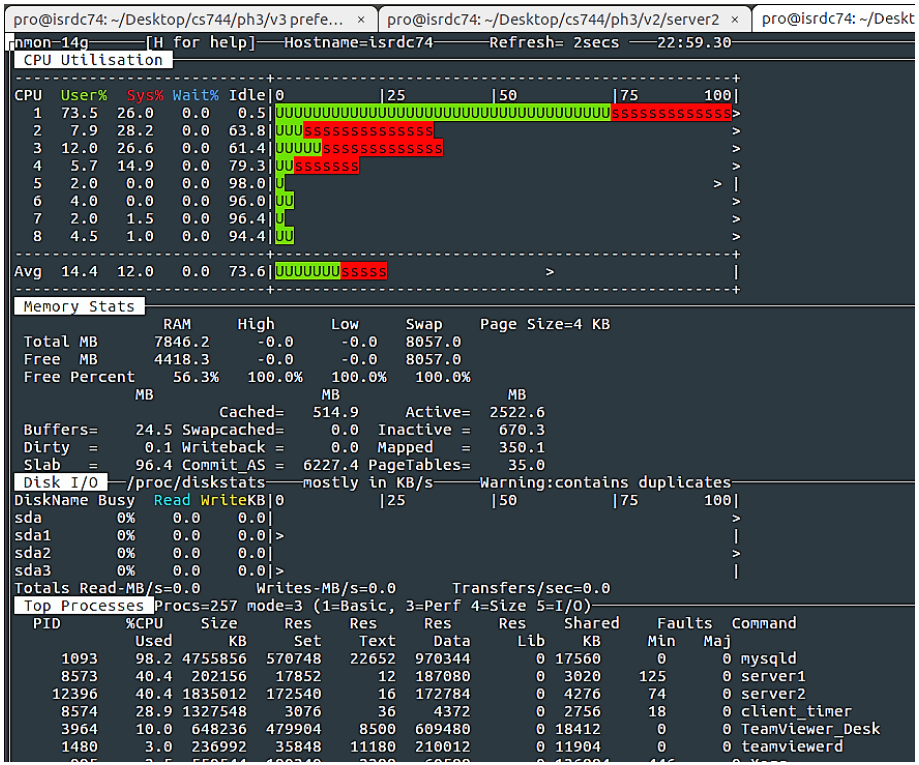
- **Design change**.Changed Server2 from Multi-process to multi-threaded.
- **Disk IO optimization**.Opened files in O_DIRECT mode and read every file in one I/O, used malloc to store file contents when server started and client request were fetched without disk access.
- **Code optimization**.Keeping variables that will be accessed together (or right after another) next to each other in memory by declaring them together. Also, created structure to include items accessed together. Removing loops to access contents of malloc(ed) files by using memcopy with offset.
- **Database optimisation using Redis Database**.Created username-password as key-value in Redis database and changed server2 code to connect to Redis-server.

Bottleneck after prefetching using MySQL

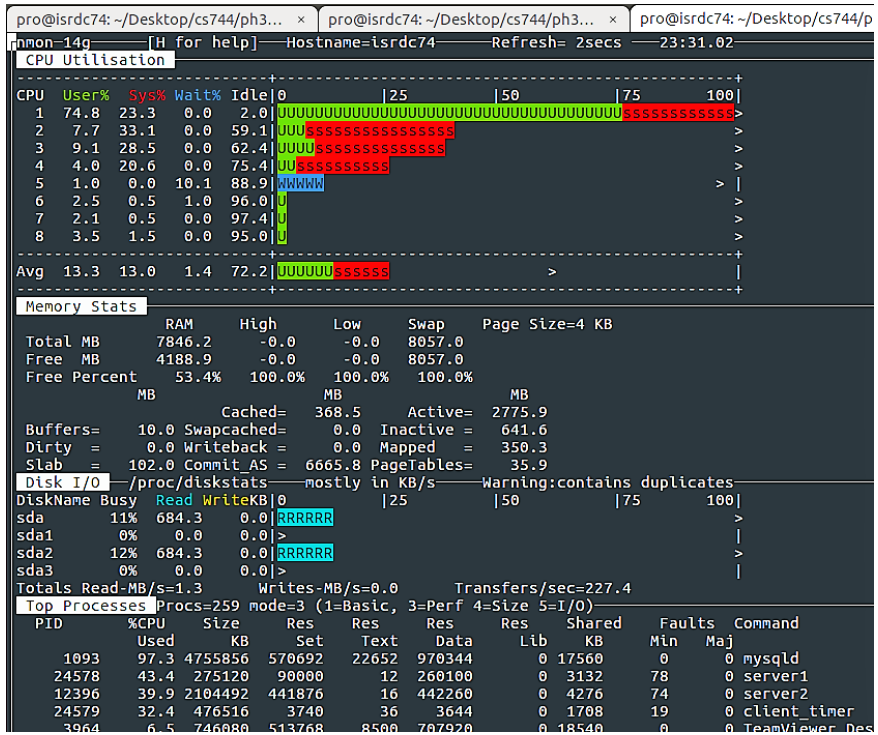
Load type	File size	Max Throughput	Bottleneck
new		6806.03	Disk I/O (Write) and MySQL
get	1KB	9932.58	MySQL
get	10KB	9973.76	MySQL
get	100KB	9804.96	MySQL/ CPU
get	1MB	1633.88	CPU
get	10MB	176.932	CPU
get	100MB	18.0733	CPU

Table: Test results after prefetching using MySQL

Bottleneck after prefetching using MySQL

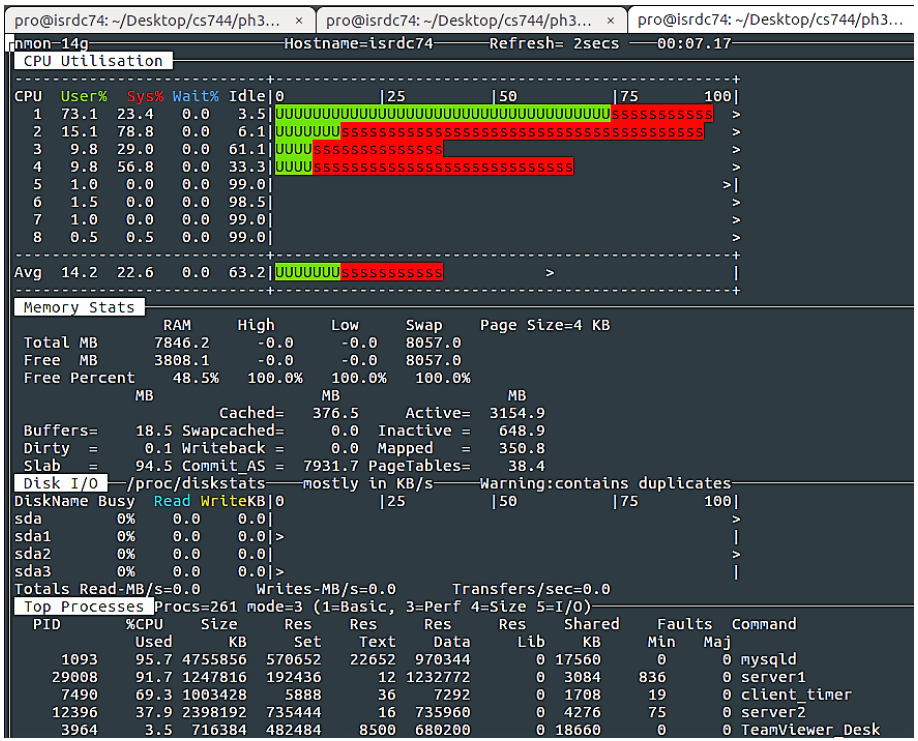


1KB

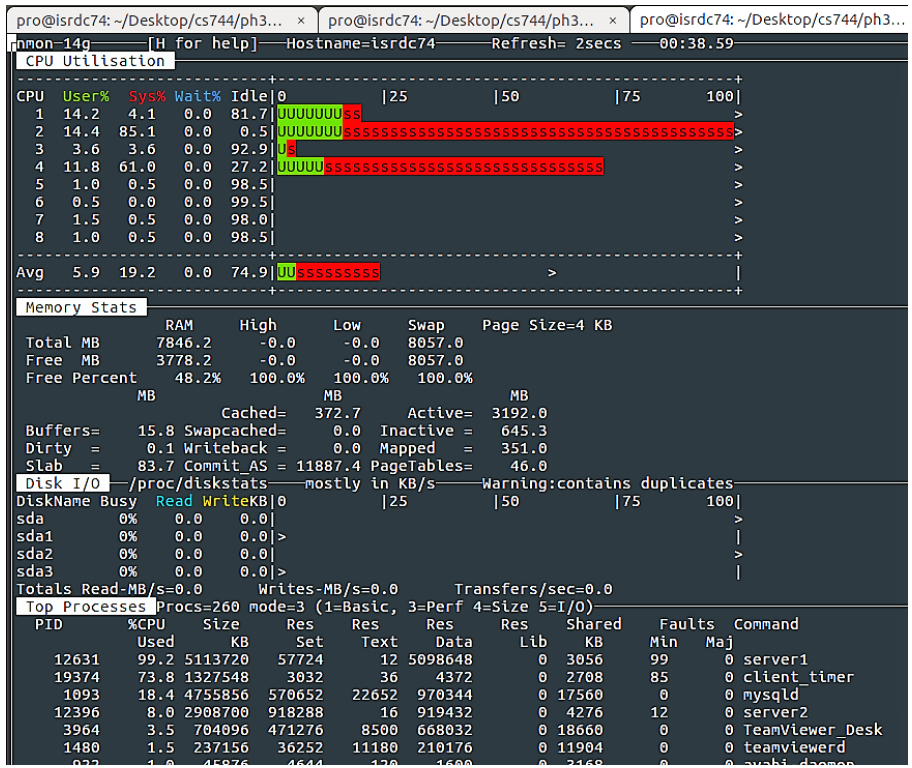


10KB

Bottleneck after prefetching using MySQL



100KB



1MB

Bottleneck after prefetching using MySQL

1KB

```
Totals Read-MB/s=0.0      Writes-MB/s=0.0      Transfers/sec=0.0
 Top Processes Procs=257 mode=3 (1=Basic, 3=Perf 4=Size 5=I/O)
  PID       %CPU    Size     Res    Res     Res     Res    Shared    Faults  Command
            Used      KB     Set   Text    Data     Lib     KB      Min   Maj
    1093    98.2 4755856  570748  22652  970344       0 17560        0     0 mysqld
    8573    40.4  202156   17852     12  187080       0  3020      125     0 server1
   12396    40.4 1835012  172540     16  172784       0  4276       74     0 server2
    8574    28.9 1327548    3076     36    4372       0  2756       18     0 client_timer
    3964    10.0  648236  479904   8500  609480       0 18412        0     0 TeamViewer_Desk
    1480     3.0  236992   35848  11180  210012       0 11904        0     0 teamviewerd
```

10KB

```
sda3       0%    0.0    0.0|>                                                  |
Totals Read-MB/s=1.3      Writes-MB/s=0.0      Transfers/sec=227.4
 Top Processes Procs=259 mode=3 (1=Basic, 3=Perf 4=Size 5=I/O)
  PID       %CPU    Size     Res    Res     Res     Res    Shared    Faults  Command
            Used      KB     Set   Text    Data     Lib     KB      Min   Maj
    1093    97.3 4755856  570692  22652  970344       0 17560        0     0 mysqld
   24578    43.4  275120   90000     12  260100       0  3132       78     0 server1
   12396    39.9 2104492  441876     16  442260       0  4276       74     0 server2
   24579    32.4  476516    3740     36    3644       0  1708       19     0 client_timer
```

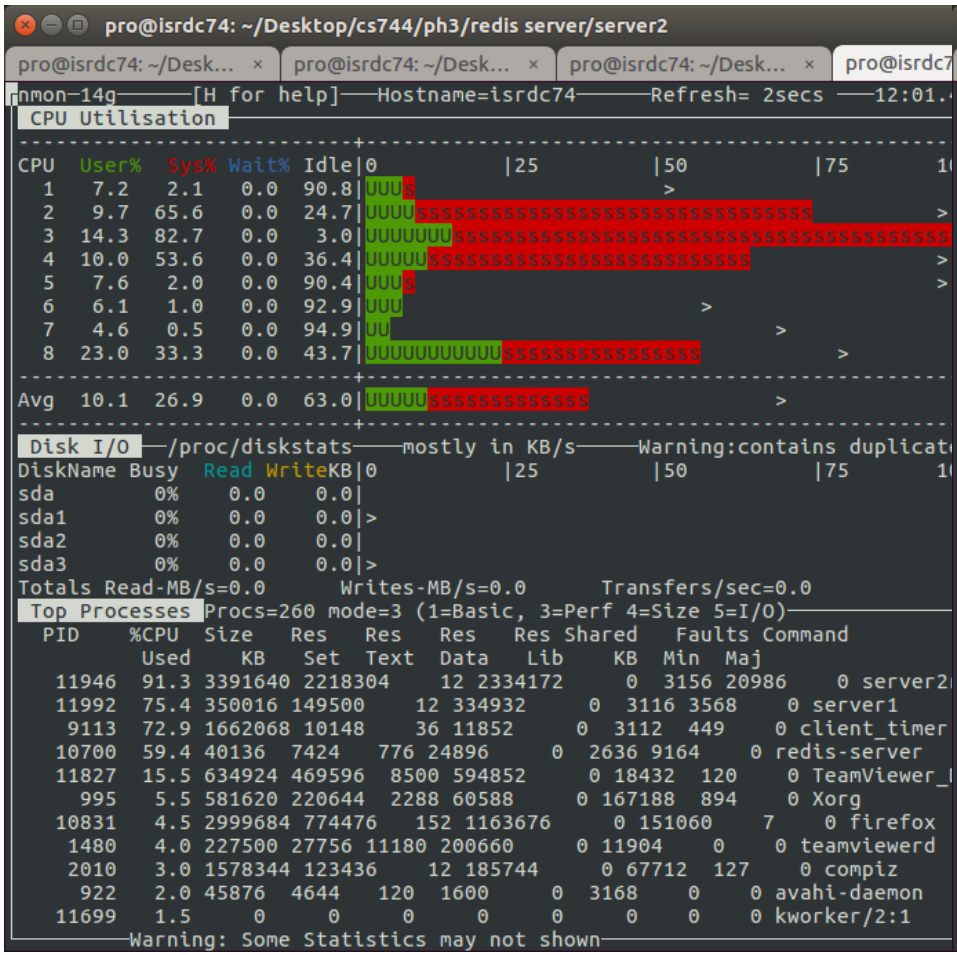
100KB

```
sda3       0%    0.0    0.0|>                                                  |
Totals Read-MB/s=0.0      Writes-MB/s=0.0      Transfers/sec=0.0
 Top Processes Procs=261 mode=3 (1=Basic, 3=Perf 4=Size 5=I/O)
  PID       %CPU    Size     Res    Res     Res     Res    Shared    Faults  Command
            Used      KB     Set   Text    Data     Lib     KB      Min   Maj
    1093    95.7 4755856  570652  22652  970344       0 17560        0     0 mysqld
   29008    91.7 1247816  192436     12 1232772       0  3084      836     0 server1
    7490    69.3 1003428    5888     36    7292       0  1708       19     0 client_timer
   12396    37.9 2398192  735444     16  735960       0  4276       75     0 server2
    3964     3.5  716384  482484   8500  680200       0 18660        0     0 TeamViewer_Desk
```

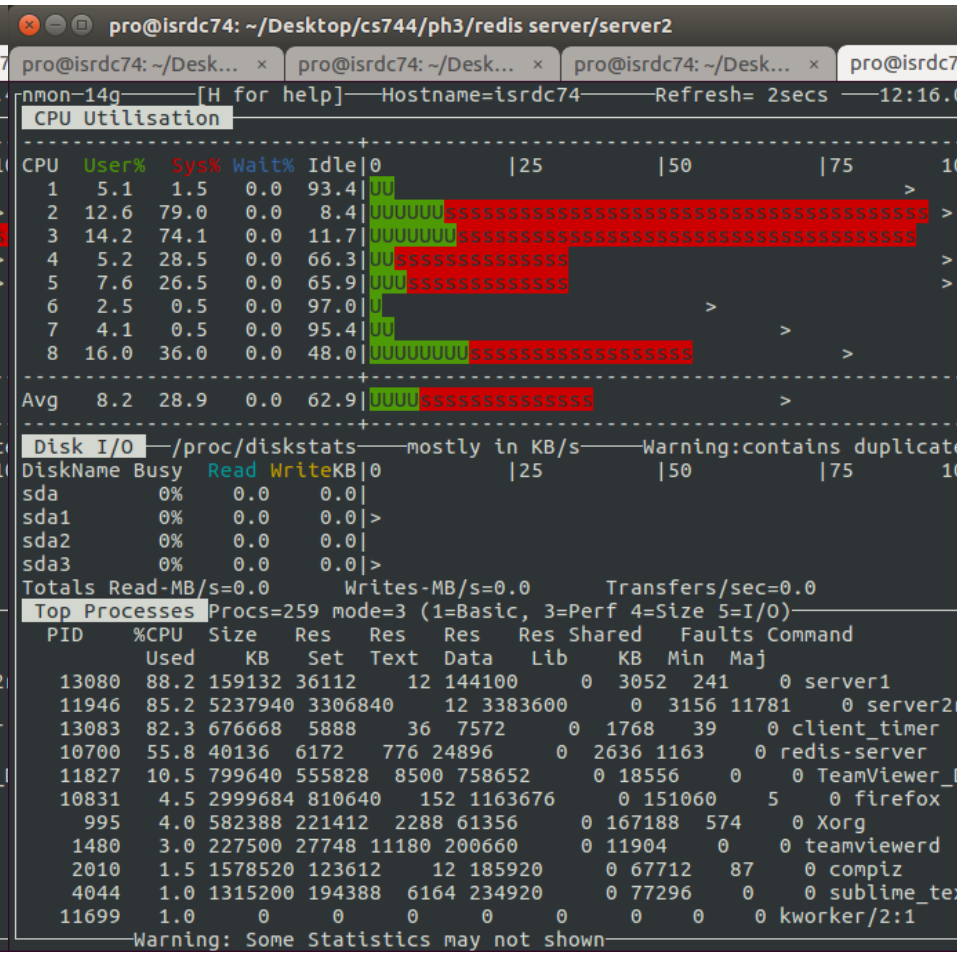
1MB

```
Totals Read-MB/s=0.0      Writes-MB/s=0.0      Transfers/sec=0.0
 Top Processes Procs=260 mode=3 (1=Basic, 3=Perf 4=Size 5=I/O)
  PID       %CPU    Size     Res    Res     Res     Res    Shared    Faults  Command
            Used      KB     Set   Text    Data     Lib     KB      Min   Maj
   12631    99.2 5113720   57724     12 5098648       0  3056       99     0 server1
   19374    73.8 1327548    3032     36    4372       0  2708       85     0 client_timer
    1093    18.4 4755856  570652  22652  970344       0 17560        0     0 mysqld
   12396     8.0 2908700  918288     16  919432       0  4276       12     0 server2
    3964     3.5  704096  471276   8500  668032       0 18660        0     0 TeamViewer_Desk
    1480     1.5  237156   36252  11180  210176       0 11904        0     0 teamviewerd
```

Bottleneck after prefetching using Redis

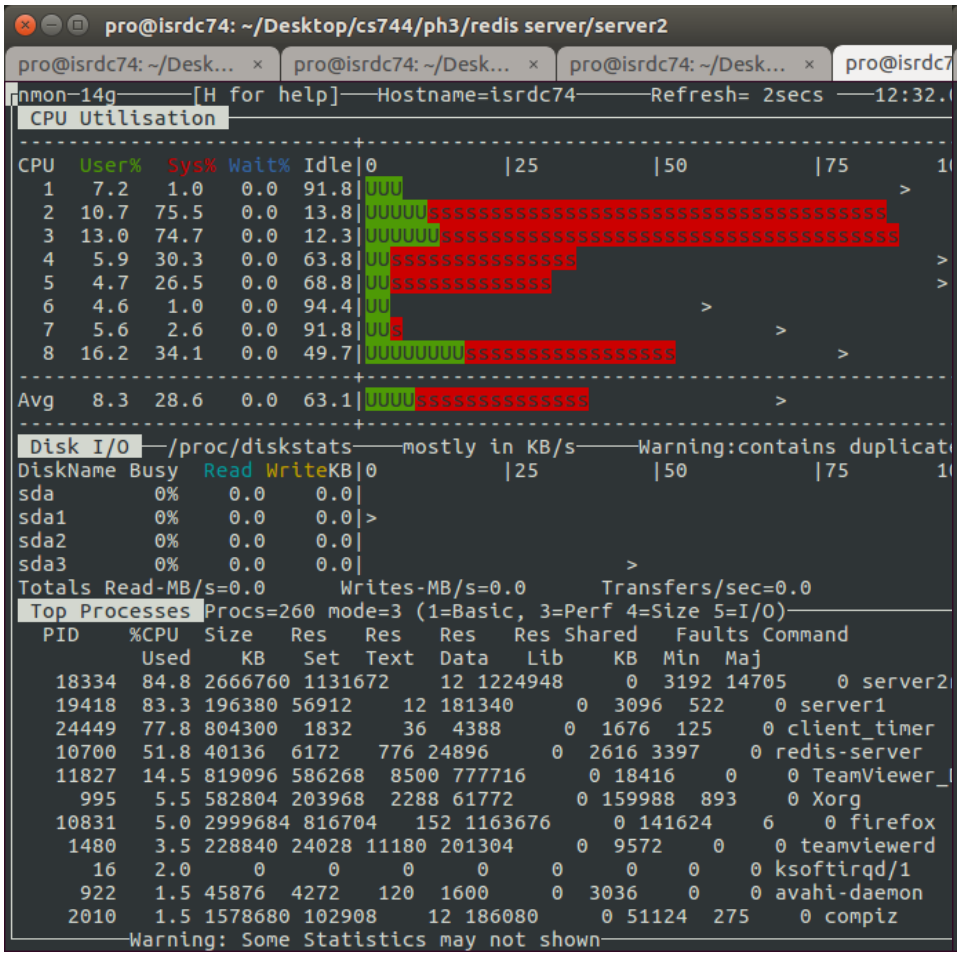


1KB



10KB

Bottleneck after prefetching using Redis



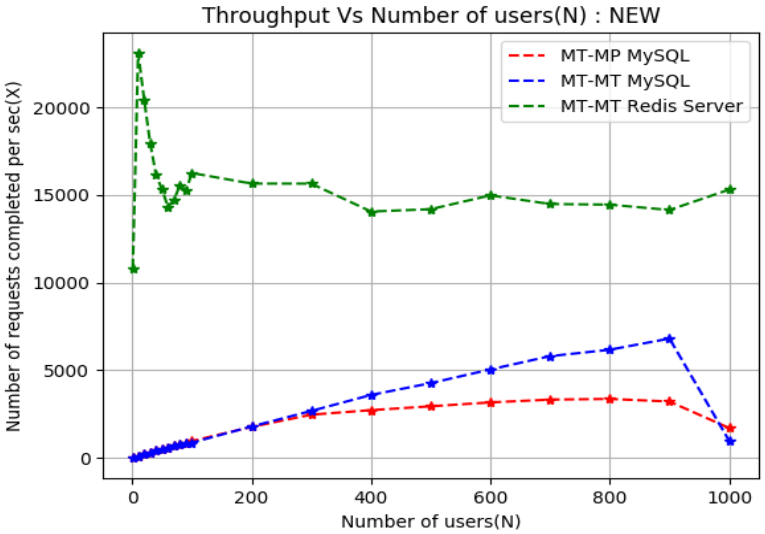
100KB 1MB

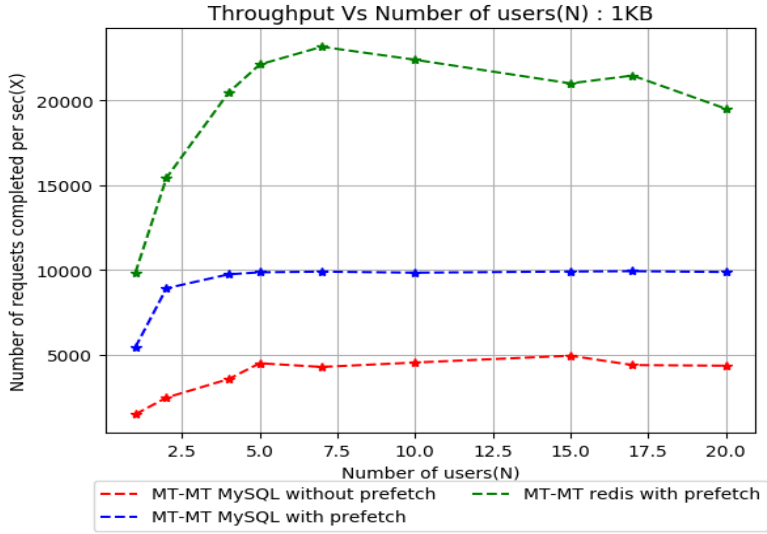
Summary of results

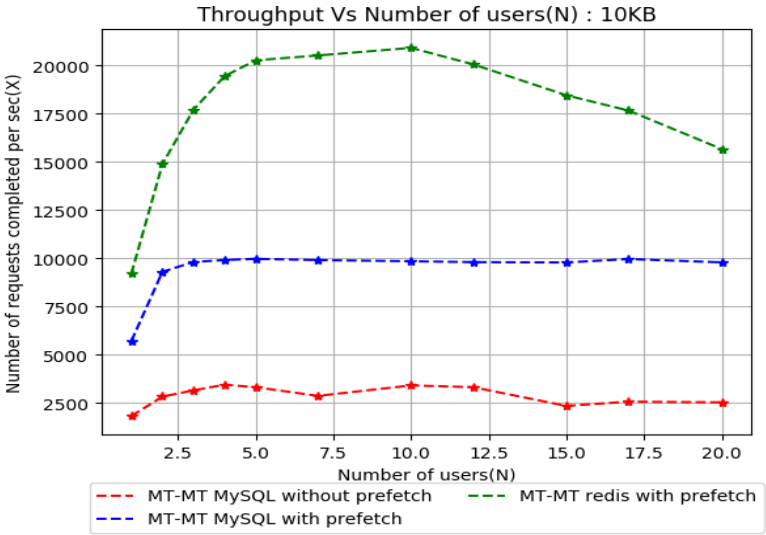
Load type	File size	Max Throughput			
		MT-MP, no prefetching, MySQL	**MT-MT, no prefetching, MySQL**	**MT-MT, prefetching, MySQL**	**MT-MT, prefetching, Redis**
new		3363.49	6806.03	6806.03	23089.8
get	1KB	1136.3921	2556.86	9932.58	23154.2
get	10KB	914.86	1559.36	9973.76	20907.1
get	100KB	697.947	832.57	9804.96	9604.82
get	1MB	132.954	136.1965	1633.88	1624.99
get	10MB	16.8549	16.9793	176.932	175.399
get	100MB	1.9439	1.95548	18.0733	17.6434

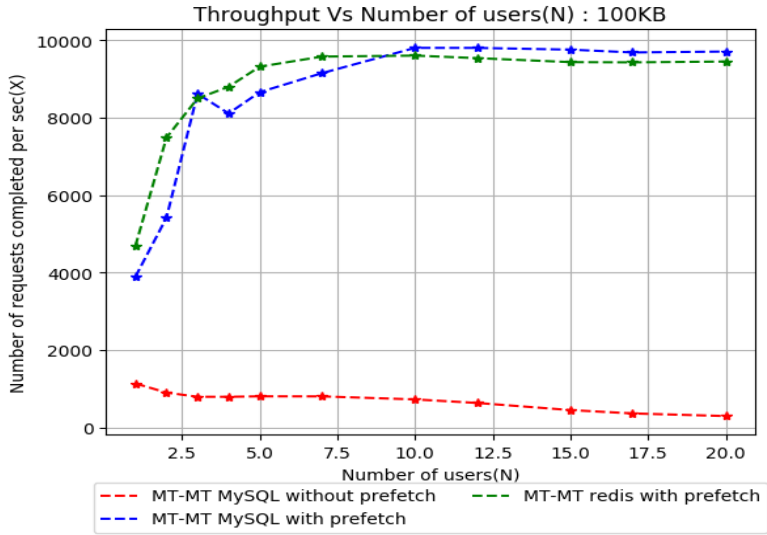
Table: Summary of results

Summary of results

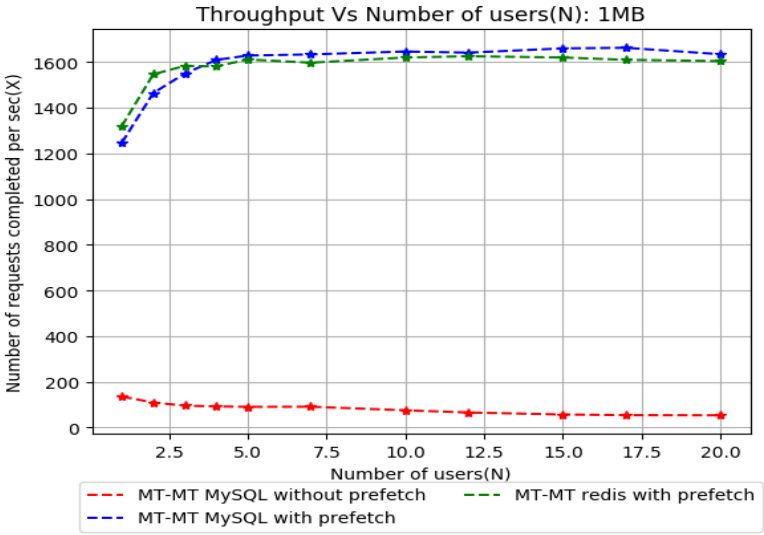


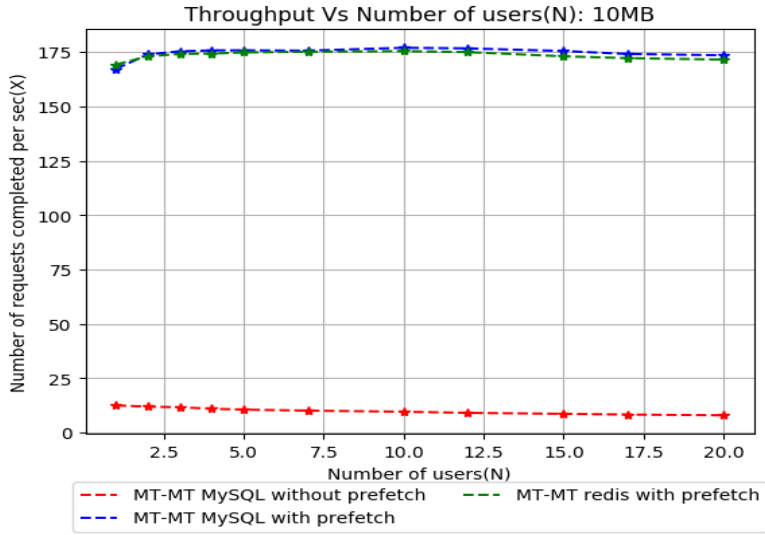


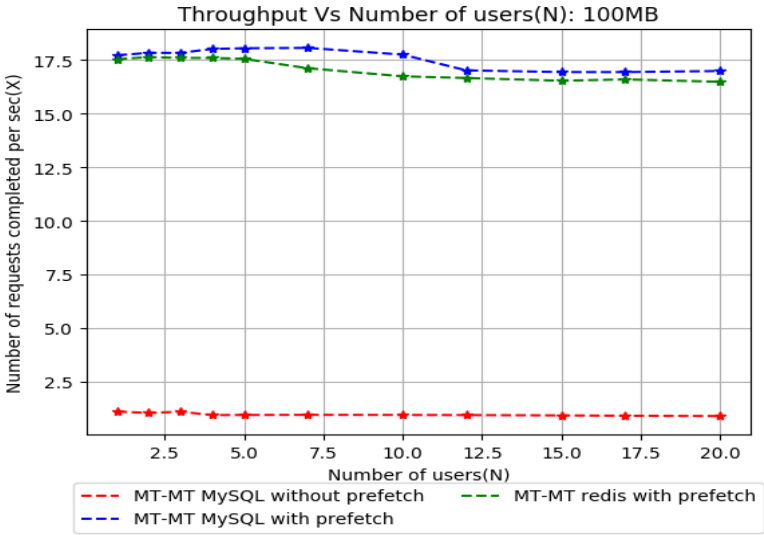




Summary of results







Lessons learnt

- Multi-threaded systems give better performance when there is no shared data and overhead of synchronization.
- Pre-fetching the files into user space by allocating memory to file contents using malloc removes disk as bottleneck and increases throughput.
- In-memory database like Redis server gives better throughput performance for small file size when MySQL acts as bottleneck.
- Difficult to remove network bottleneck as it needs hardware changes.
- For optimising performance and identifying hardware bottlenecks, there is requirement to tune sockets, system and MySQL parameters.

Thank You!