Advanced Systems Lab Report

Autumn Semester 2018

Name: Pirmin Schmid

Appendix to the report

Contents

Appendices **2**

A Appendix to section 1: system overview **2**
A.1 Appendix to 1.3: Middleware 2
A.2 Appendix to 1.7.2: Operational laws 2

B Appendix to section 2: Baseline without middleware **4**
B.1 Appendix to 2.1: One server 4
B.2 Appendix to 2.2: Two servers 4

C Appendix to section 4: throughput for writes **5**
C.1 Appendix to 4.1: Summary: Example dynamic change in the system 5
C.2 Appendix to 4.2: Summary: Additional experiment `e820` 6

Appendices

A Appendix to section 1: system overview

A.1 Appendix to 1.3: Middleware

Implementation guarantees Explicitly, the middleware implementation assures the *sequential execution* of all steps in the correct designated thread (see details in 1.3.1) as defined in the project description and required for modeling.

- All client connections are accepted in the *ClientThread*.
- Each request from all clients is read and fully parsed in the *ClientThread* before being enqueued as a job.
- Memtier already guarantees that a virtual client does not send a fresh request before it has received the reply to the former request.
- Blocking functions are used to avoid any busy polling/idling in the threads: `selector.select()` for network I/O and `queue.take()` to dequeue a job from the queue.
- A worker works only on one job at a time (despite the fact that typical async NIO could and would handle more than one job at a time).
- The following sequence is guaranteed in workers: (1) job type is recognized and then handled properly; (2) all requests are sent to servers (1-3 dependent on experiment configuration and job type); (3) the worker waits for server replies and parses them; (4) only after having received all replies, the worker sends the reply to the client; (5) instrumentation data is processed; (6) worker state is reset to be ready to dequeue a new job.

A.2 Appendix to 1.7.2: Operational laws

Used variables Table 1 explains the variables used for utilization and bottleneck analysis in sections 2-6 based on operational laws. Asymptotic bounds can be used to find the knee in the throughput graphs and to define $N^* = \frac{D+Z}{D_{max}}$ at the knee.[1] But they are not useful in finding

[1] "If the number of jobs is more than N^*, then we can say with certainty that there is queueing somewhere in the system", page 565 in [1]

Table 1: Variables of the operational laws, from box 33.1 [1], extended and with comments

Variable	Explanation
D_i	total service demand per job for device i [a], $= S_i \cdot V_i$
D	sum of service demands on all devices, $= \sum_i D_i$
D_{max}	service demand on the bottleneck device, $= max_i\{D_i\}$
N	number of jobs in the system ($\approx$ number of load generating virtual clients [b])
Q_i	number of jobs [c] in device i
R	system response time
R_i	response time per visit to device i
S_i	service time [d] per visit to device i
S	total service time per job for all visited devices, $= \sum_i S_i$
U_i	utilization [d] of device i
U_{max}	utilization of the bottleneck device
V_i	number of visits per job to device i, in $(0.0, 1.0]$ for the systems under test
X	system throughput
X_i	throughput of device i
X_{max}	max. observed throughput in a window
$X_{network\_limit}$	added: max. possible throughput due to network bandwidth limitation (policy) [e]
Z	thinking time

[a] in the report, i refers to *client*, *middleware*, *server* (only one instance or average).
[b] $\approx$ is used instead of $=$ in case multiple clients are processing replies/preparing new jobs at the same time; for all discussion in the report $=$ is assumed.
[c] this includes both, jobs in the queue of this device and jobs being processed by worker threads of this device
[d] there are 2 views for the middleware: (1) use $ServiceTime$ as measured in the middleware, which covers also all waiting time for the replies of the servers; (2) detailed bottleneck analysis for middleware and server instances, and the network connection between them, which is more informative, used in the report and described below.
[e] using 4096 + 20 bytes (see protocol overhead)

the saturation of the more complex systems with middleware, where it is not of interest whether any queueing starts in the system but interested to see when relevant queueing starts in the middleware.

Number of jobs in the system In the closed system under test, the number of jobs in the system can be considered to be equal to the number of virtual clients (Table 1 comment b). With given latencies and payload size, only few are in transit in the network[2] Some are in the queue middleware. However, many jobs are "in" the workers of the middleware, which is relevant in particular for configurations with lots of worker threads. This explains why doubling the number of workers needs more clients (= more jobs) before queueing can be observed in the middleware (see e.g. subsection 3.1).

[2] As an approximation, a 100 Mbit/s bandwidth network connection may contain up to 1.5 requests with 4 KiB payload; connections with larger bandwidths accordingly more.

B Appendix to section 2: Baseline without middleware

B.1 Appendix to 2.1: One server

Table 2: Operational laws variables: knee and device utilization bounds one server

Variable	Calculation	write-only (set)	read-only (get)
network bandwidth limit		3 x 200 Mbit/s	1 x 100 Mbit/s
deduced $X_{network\_limit}$		18.2 kop/s	3.0 kop/s
Z	configuration	0.0 ms	0.0 ms
X_{max} [a]	measured	16.2 kop/s	3.5 kop/s
R_{min} [a]	measured	0.825 ms	1.875 ms
D_{max}	$\frac{1}{X_{max}}$	0.062 ms	0.289 ms
$D_{client,uc}$ [b,c]	IL	0.007 ms	0.003 ms
D_{server} [d]		$D_{max} = 0.062$ ms	$\leq D_{max} = 0.289$ ms [e]
"$D_{network\_limit}$"		0.055 ms	0.329 ms
D [f]	$\leq R_{min}$	0.825 ms	1.875 ms
N^*	$\frac{D+Z}{D_{max}}$	≤ 13.3	≤ 6.5
N_{uc} [c]	see report Figure 6 (a)	72 ($VC = 12$)	6 ($VC = 1$)
$X(N_{uc})$ [c]	measured	14.071 ± 0.270 kop/s	2.954 ± 0.008 kop/s
$R(N_{uc})$ [c]	measured	5.076 ± 0.114 ms	2.015 ± 0.006 ms
$U_{client,uc}$ [c]	$X(N_{uc}) * D_{client}$	10 %	0.8 %
$U_{server,uc}$ [c,d]	$X(N_{uc}) * D_{server}$	**87 % (bottleneck)**	≤ 85 % [g]
"$U_{network\_limit,uc}$" [c]	$X(N_{uc}) * D_{network\_limit}$	77 %	**97 % (bottleneck)**
$CPU\_usage_{client,uc}$ [c]	measured	15.1 ± 0.8 %	5.2 ± 0.1 %
$CPU\_usage_{server,uc}$ [c]	measured	87.5 ± 1.9 %	10.4 ± 0.7 %

[a] measured in at least one 1 s window; average throughput is lower; average response time is longer

[b] $D_{client} \approx R_{client,expected} - R_{client,measured}$ (increasing for increasing number of virtual clients per thread) and conceptually $D_{client} \leq D_{server} \leq D_{max}$ in this setting

[c] *uc* stands for *usable_capacity*

[d] including receiving network stack, memcached service, sending network stack (bandwidth limit) of the server; detailed view of $D_{memcached}$ would be lower

[e] D_{server} without the network send bandwidth limitation would be lower, probably much closer to D_{server} of the set operations (see discussion)

[f] D is defined as $\sum_i D_i$ for all devices i, which includes client, server, network connections here; due to limited availability of S_{client} and S_{server}, that are already approximated, this sum would be associated with large error margins; thus, the following bound is used $D \leq S \leq R_{min}$ (see subsection 1.7)

[g] this is a clear over-estimation due to coupling with bandwidth limit policy; $U_{server,uc}$ (for kernel and memcached) is closer to 18 % using D_{server} for write-only; see also CPU usage

Table with detailed calculations for operational laws, including utilization and bottleneck analysis (Table 2).

B.2 Appendix to 2.2: Two servers

See Table 3

Table with detailed calculations for operational laws, including utilization and bottleneck analysis (Table 3). Note: the conceptually correct $U_{client,bound}$ (≤ 74 %) overestimates the client utilization, and $U_{client,uc}$ (2 %) probably underestimates client utilization due to noise in the used raw values for calculation creating only a wide bounding interval [2, 74] %.

Table 3: Operational laws variables: knee and device utilization bounds two servers, one client

Variable	Calculation	write-only (set)	read-only (get)
network bandwidth limit		1 x 200 Mbit/s	2 x 100 Mbit/s
deduced $X_{network\_limit}$		6.1 kop/s	6.1 kop/s
Z	configuration	0.0 ms	0.0 ms
X_{max} [a]	measured	8.0 kop/s	6.1 kop/s
R_{min} [a]	measured	0.734 ms	0.698 ms
D_{max}	$\frac{1}{X_{max}}$	0.125 ms	0.163 ms
$D_{client,uc}$ [b,c]	IL	0.003 ms	0.005 ms
$D_{client,bound}$ [d]		$\leq D_{max} = 0.125$ ms	$\leq D_{max} = 0.163$ ms
D_{server} [e,f]		$\leq D_{client,bound}$	$\leq D_{client,bound}$
"$D_{network\_limit}$"		0.165 ms	0.165 ms
D [g]	$\leq R_{min}$	0.734 ms	0.698 ms
N^*	$\frac{D+Z}{D_{max}}$	≤ 5.9	≤ 4.3
N_{uc} [c]	see report Figure 6 (b)	6 ($VC = 3$)	6 ($VC = 3$)
$X(N_{uc})$ [c]	measured	5.956 ± 0.024 kop/s	5.830 ± 0.017 kop/s
$R(N_{uc})$ [c]	measured	1.001 ± 0.005 ms	1.019 ± 0.008 ms
$U_{client,uc}$ [c]	$X(N_{uc}) * D_{client}$	2 %	3 %
$U_{client,bound}$ [c,e]	$X(N_{uc}) * D_{client,bound}$	$\leq$ 74 % [h]	$\leq$ 95 % [h]
$U_{server,uc}$ [c,e]	$X(N_{uc}) * D_{server}$	$\leq$ 74 % [h]	$\leq$ 95 % [h]
"$U_{network\_limit,uc}$" [c]	$X(N_{uc}) * D_{network\_limit}$	**98 % (bottleneck)**	**96 % (bottleneck)**
$CPU\_usage_{client,uc}$ [c]	measured	17.3 ± 0.2 %	21.7 ± 0.5 %
$CPU\_usage_{server,uc}$ [c]	measured	9.8 ± 0.2 %	9.5 ± 0.2 %

[a] measured in at least one 1 s window; average throughput is lower; average response time is longer

[b] $D_{client} \approx R_{client,expected} - R_{client,measured}$ (increasing for increasing number of virtual clients per thread)

[c] *uc* stands for *usable_capacity*

[d] upper bound for D_{client} including receiving network stack, memtier service, sending network stack (bandwidth limit) of the client VM; see conceptually $D_{server} \leq D_{client} \leq D_{max}$ in this setting; however, clear over-estimations due to network bandwidth limitations

[e] including receiving network stack, memcached service, sending network stack (bandwidth limit) of the server; detailed view of $D_{memcached}$ would be lower

[f] these deduced upper bounds are very clear over-estimations; see also conceptually $D_{server} \leq D_{client} \leq D_{max}$ in this setting

[g] D is defined as $\sum_i D_i$ for all devices i, which includes client, server, network connections here; due to limited availability of S_{client} and S_{server}, that are already approximated, this sum would be associated with large error margins; thus, the following bound is used $D \leq S \leq R_{min}$

[h] both upper bounds are clear over-estimation due to coupling with bandwidth limit policy; $U_{client,bound}$ is closer to $U_{client,uc}$; $U_{server,uc}$ (for kernel and memcached) is closer to 18 % (see write-only of subsection 2.1; see also CPU usage)

C Appendix to section 4: throughput for writes

C.1 Appendix to 4.1: Summary: Example dynamic change in the system

An example of a dynamic change in the system. Small change in server RTT has much larger effect on middleware response time due to longer queue (Figure 1). Discussion in the report in detail.

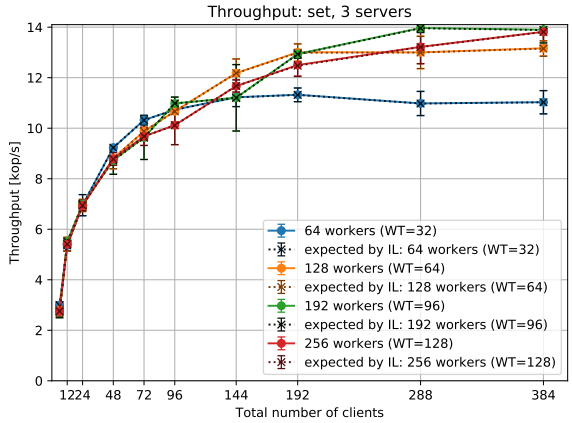
Figure 1: Detailed time measurements in the middleware. Experiment with 192 clients, 16 worker threads (WT=8), 3 servers

C.2 Appendix to 4.2: Summary: Additional experiment `e820`

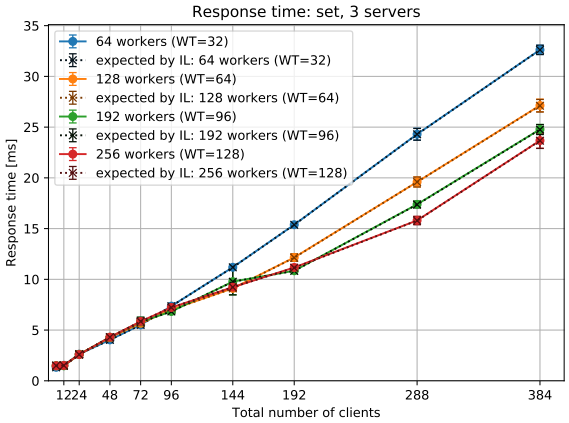
Extended experiment `e820` with 64, 128, 192, 256 workers (WT=32, 64, 96, 128): Figure 2.

References

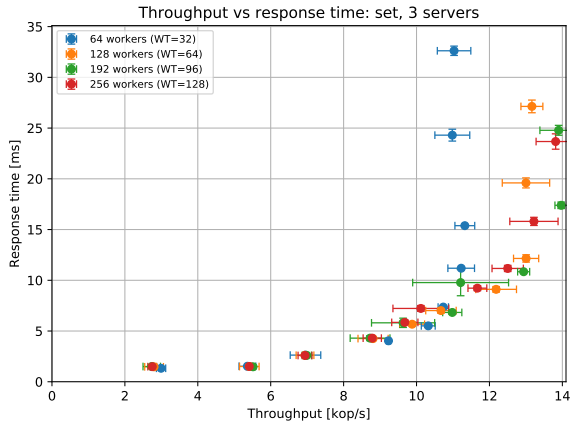
[1] Jain R. The Art of Computer Systems Performance Analysis. 1st ed. Wiley Professional Computing, 1991



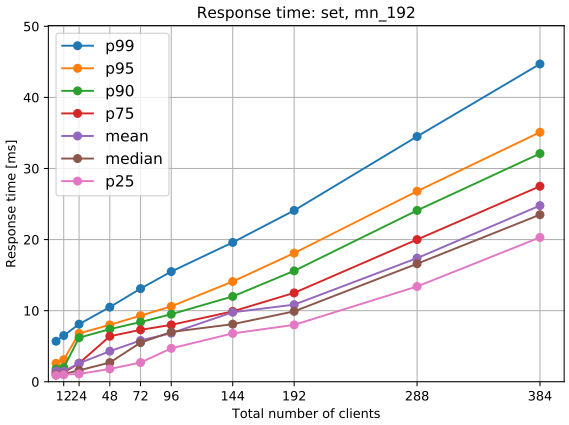
(a) Throughput



(b) Response time



(c) Throughput vs response time



(d) Response time percentiles (WT=96)

Figure 2: Write-only workload: 2 middleware and 3 memcached server instances; extended experiment `e820` with WT=32, 64, 96, 128