RHEL Performance Tuning:

Using Zabbix to identify problems

Andrew Nelson

Consultant, Red Hat

September, 21, 2012

Performance Tuning Agenda

- whoami
- Why tune?
- Little's Law
- Instrumentation
- Real world example

$ whoami

- Andrew Nelson

  anelson@redhat.com
- Consultant with Red Hat North America
- Active in the Zabbix community for approximately 8 years
  - Known as “nelsonab” in forums and IRC
- Author of Zabcon and Zbxapi

Why Tune

- Performance tuning is something we've all needed to do at some point.
  - Minimize wait time
  - Increase performance (page views for web server)
  - Increase longevity of old hardware
  - Power saving
  - “It's too slow” (Users are always so specific)
- AKA: Maximize Return on Investment

Little's Law

- Performance tuning is something we have all had to do at some point.
- Performance tuning is defined by a formula known as Little's Law

  $L=\lambda h$

  - Published in 1961, essentially a restatement of the Erlang formula which described call queuing and congestion in telephone networks.
  - L Total number of Customers (Queue Depth)
  - λ Arrival Rate
  - h Average time to service (customer) requests

Little's Law

$L = \lambda h$

Assuming all other values are constant:

- ↑L Able to service more items, latency increases
- ↓L Fewer items serviced, latency decreases
- ↑λ Queue becomes full faster
- ↓λ More time required to fill queue
- ↑h Queue drains faster, latency decreases
- ↓h Queue becomes full faster, latency increases

Little's Law

$L = \lambda h$

- Most often we directly control only two of the three values.
  - L = Queue length. Most often buffers.
  - h = Time to service a request. This can include tuning the efficiency of a component, or using a more efficient algorithm.
- Often we can't control λ, arrival rate, our end users take care of that.

Instrumentation

- We can only tune if we can instrument what's going on
- Common OS Instrumentation sources
  - /proc
  - /sys
  - SystemTap

Instrumentation

- How can SystemTap be useful?
- Example for Zabbix already in the forums:

  http://www.zabbix.com/forum/showthread.php?t=23752

  Example output:

```
COMMAND         DEV     XMIT_PK RECV_PK XMIT_KB RECV_KB
swapper         eth1       2425    5644     138    6702
firefox         eth1       2847    3897     354    4501
X               eth1         77     162       4     191
zabbix_agentd   lo           16      16       1       0
zabbix_server   lo            8       8       0       0
lxpanel         eth1         12      17       0      18
scsi_eh_1       eth1         10      25       0      32
ata/0           eth1         10      24       0      31
vmtoolsd        eth1          2      19       0      23
gconfd-2        eth1         10      11       0      12
vmware-user-loa eth1          3       5       0       3
swapper         eth2          0       5       0       0
zabbix_agentd   eth1          3       5       0       4
```

Instrumentation

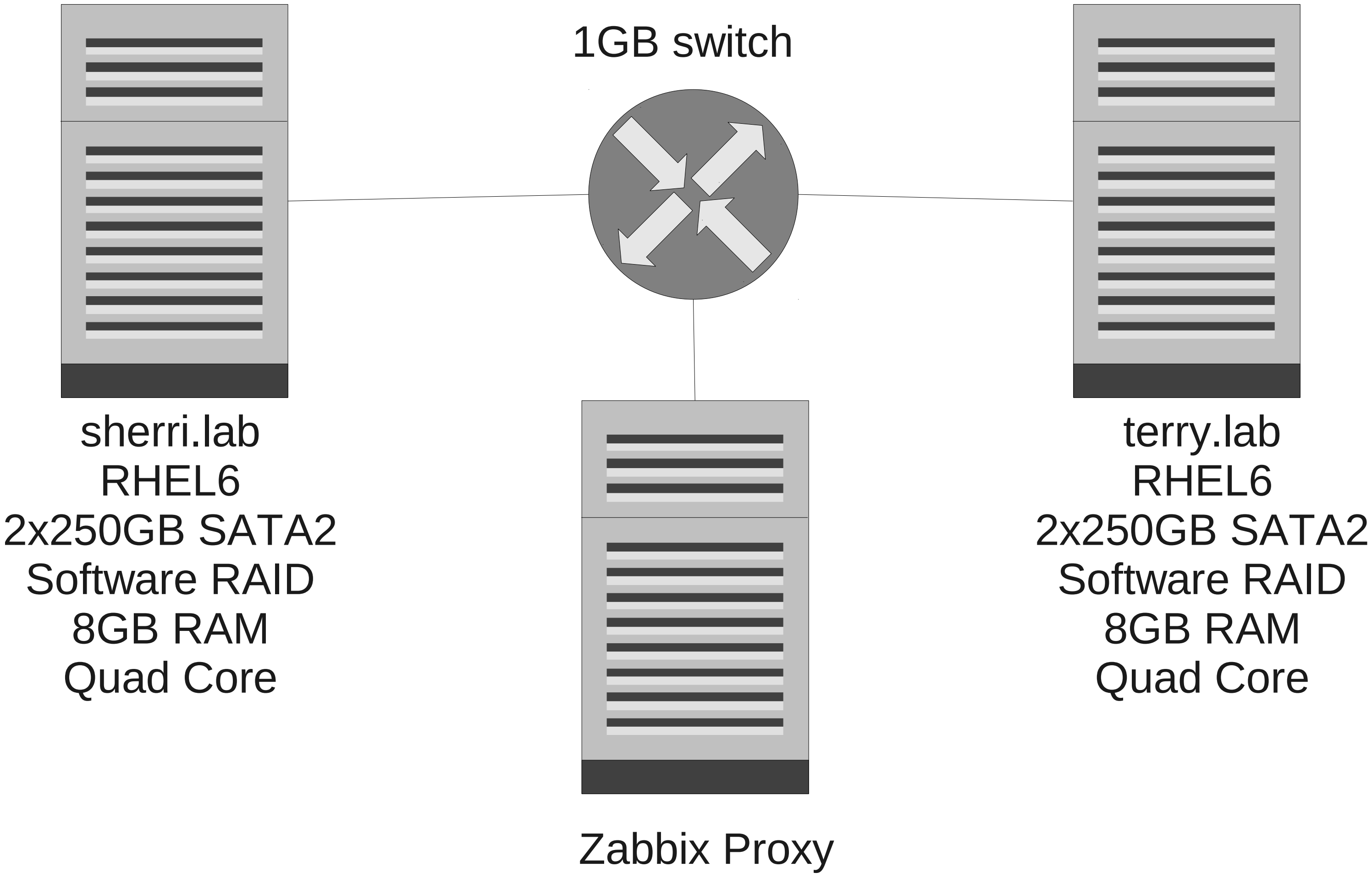
- Where can I learn more?
  - Kernel documentation (kernel-doc rpm)
  - RHEL 6 Performance Tuning Guide
  - Red Hat Documentation for RHEL 6
  - Red Hat Knowledgebase
  - Red Hat Training (Performance Tuning RH443)

Real World Example

- This theory stuff is nice, but show me the numbers!
- NFS Performance tuning
  - Users are complaining of poor performance
  - Using a simple test using dd to copy a 10GB file the following speeds are noted:
  - Client → Server 287.912 s, 37.3 MB/s
  - Server → Client 167.834 s, 64.0 MB/s
  - Server → Server (via client) 413.595 s, 26.0 MB/s

Real World Example

- The test environment



1GB switch

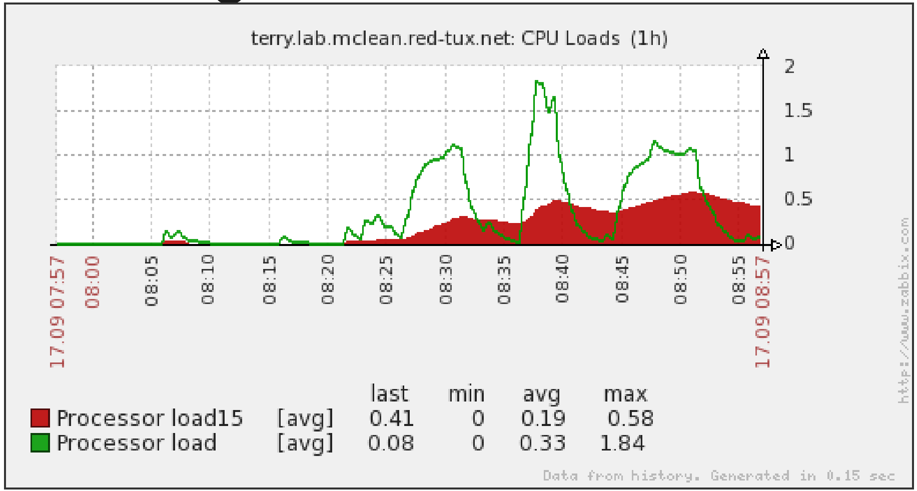
sherri.lab
RHEL6
2x250GB SATA2
Software RAID
8GB RAM
Quad Core

terry.lab
RHEL6
2x250GB SATA2
Software RAID
8GB RAM
Quad Core

Zabbix Proxy

Real World Example

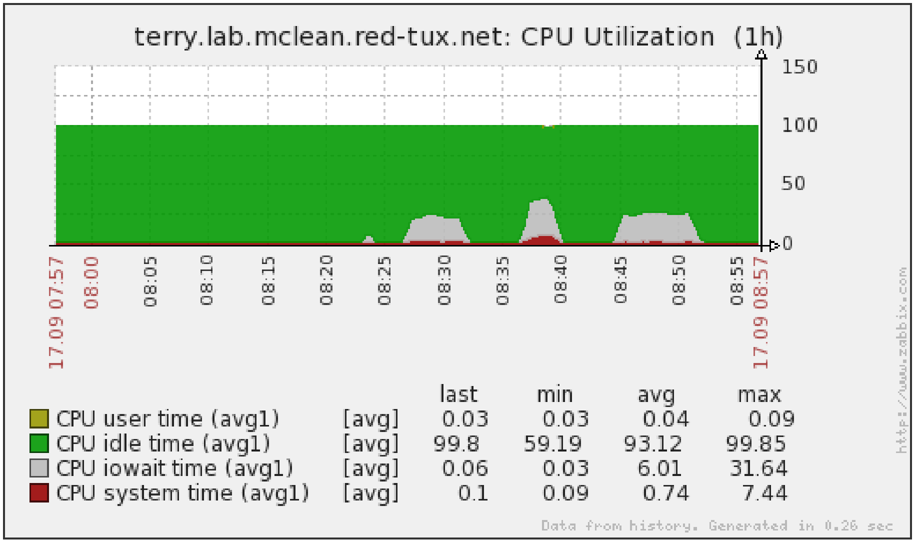
- Logging into Zabbix the following is noted on the client during the tests:



terry.lab.mclean.red-tux.net: CPU Loads (1h)

			last	min	avg	max
🟥	Processor load15	[avg]	0.41	0	0.19	0.58
🟩	Processor load	[avg]	0.08	0	0.33	1.84

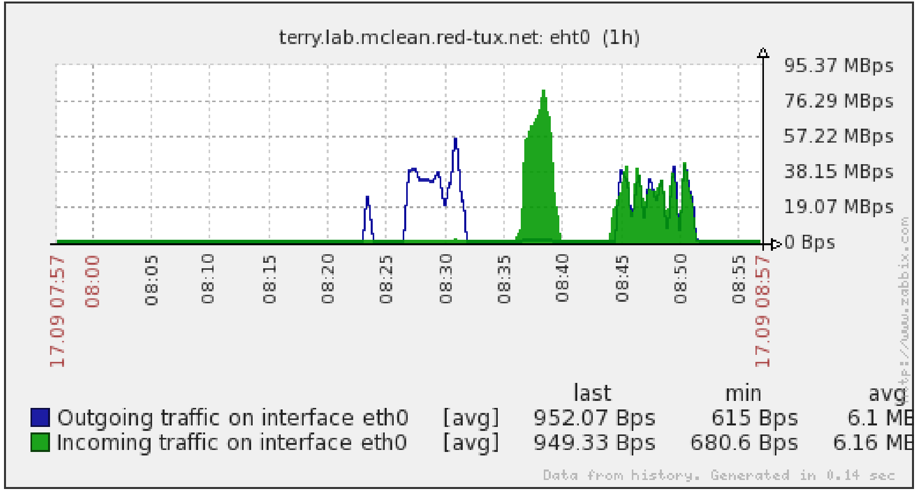
Data from history. Generated in 0.15 sec



terry.lab.mclean.red-tux.net: CPU Utilization (1h)

			last	min	avg	max
■	CPU user time (avg1)	[avg]	0.03	0.03	0.04	0.09
🟩	CPU idle time (avg1)	[avg]	99.8	59.19	93.12	99.85
□	CPU iowait time (avg1)	[avg]	0.06	0.03	6.01	31.64
🟥	CPU system time (avg1)	[avg]	0.1	0.09	0.74	7.44

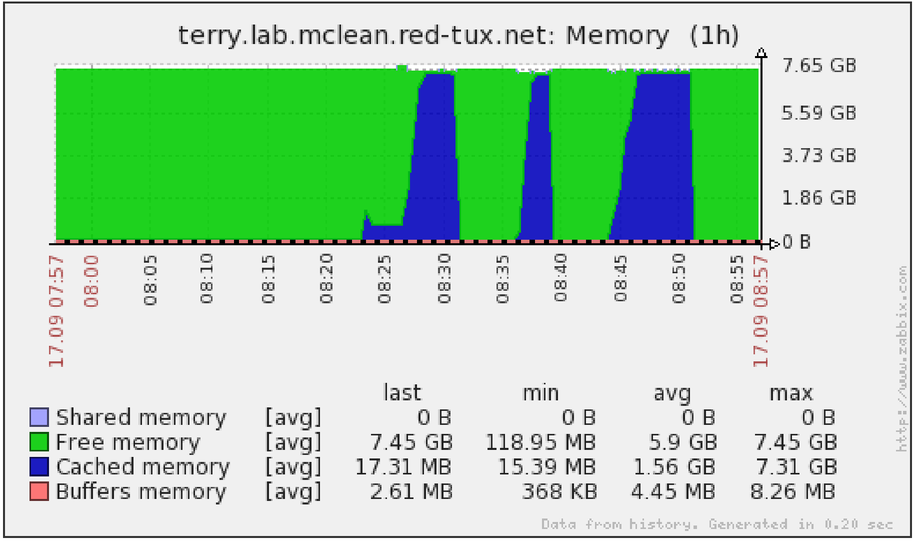
Data from history. Generated in 0.26 sec



terry.lab.mclean.red-tux.net: eht0 (1h)

			last	min	avg
🟦	Outgoing traffic on interface eth0	[avg]	952.07 Bps	615 Bps	6.1 MB
🟩	Incoming traffic on interface eth0	[avg]	949.33 Bps	680.6 Bps	6.16 MB

Data from history. Generated in 0.14 sec



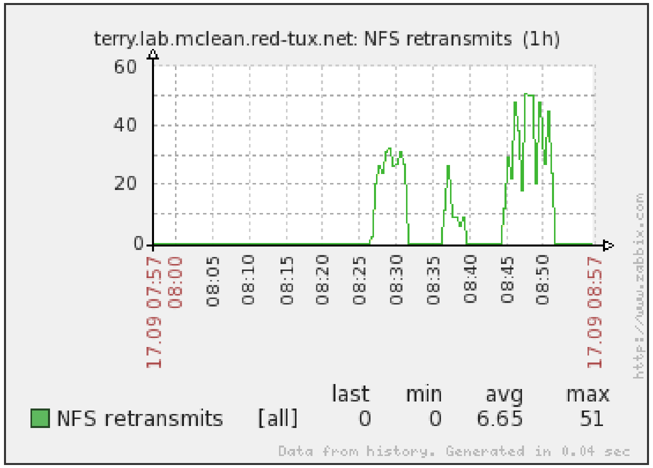
terry.lab.mclean.red-tux.net: Memory (1h)

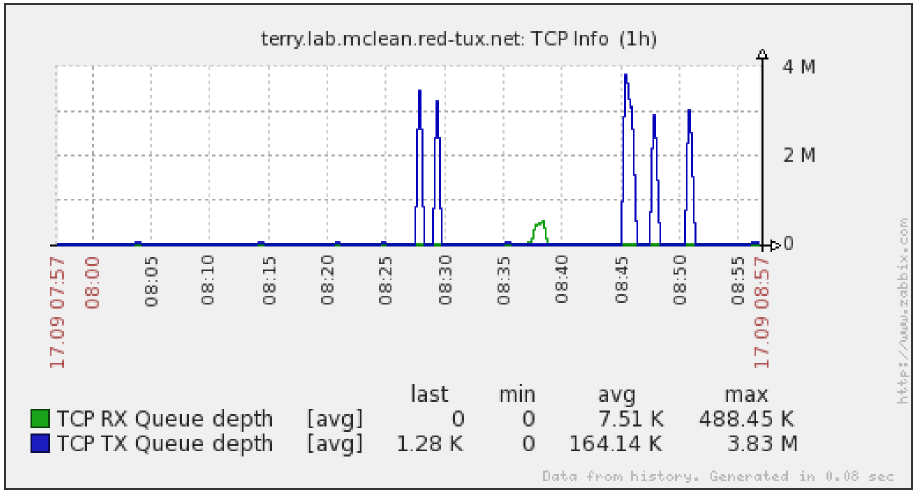
			last	min	avg	max
■	Shared memory	[avg]	0 B	0 B	0 B	0 B
🟩	Free memory	[avg]	7.45 GB	118.95 MB	5.9 GB	7.45 GB
🟦	Cached memory	[avg]	17.31 MB	15.39 MB	1.56 GB	7.31 GB
🟥	Buffers memory	[avg]	2.61 MB	368 KB	4.45 MB	8.26 MB

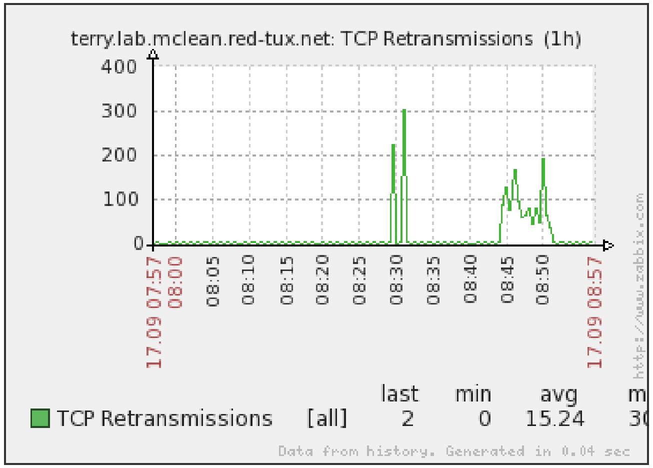
Data from history. Generated in 0.20 sec

Real World Example

- Let's add some more graphs (Network)



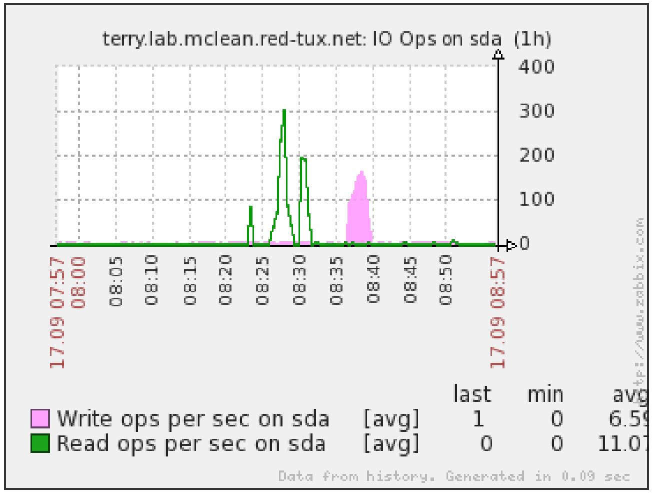


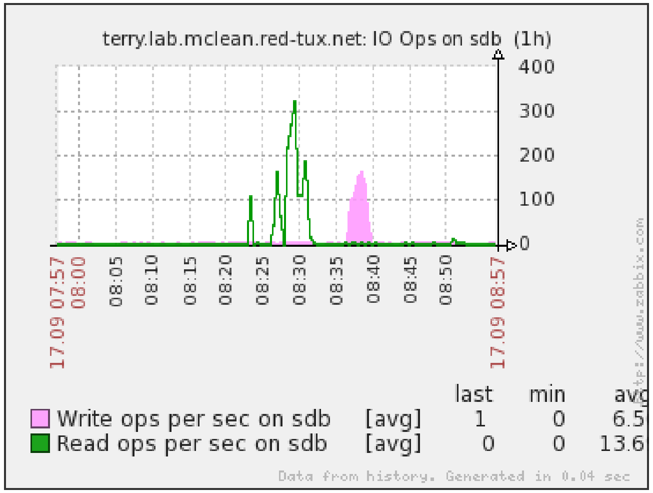


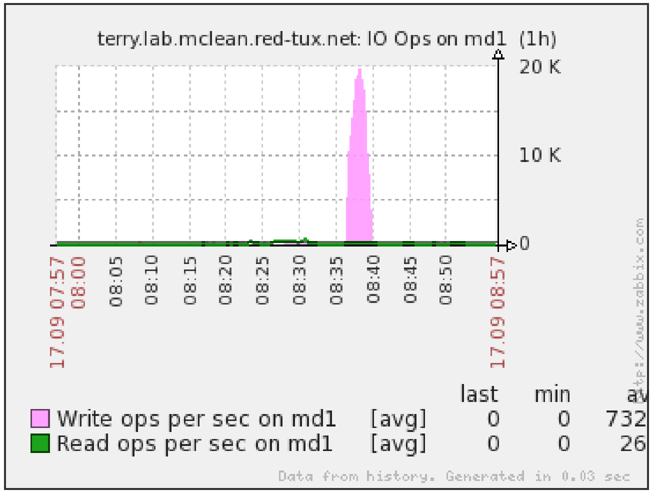
- UserParameter=NFS.retrans,grep rpc /proc/net/rpc/nfs | awk '{print $3}'
- UserParameter=TCP.retrans,/var/lib/zabbix/proc_netstat.sh snmp Tcp RetransSegs
- UserParameter=TCP.rx_queue,ss -m | grep mem | sed 's/.*r\([0-9]*\),.*/\1/'|awk '{sum+=$1}END{print sum}'
- UserParameter=TCP.tx_queue,ss -m | grep mem | sed 's/.*w\([0-9]*\),.*/\1/'|awk '{sum+=$1}END{print sum}'

Real World Example

- Let's add some more graphs (Disk)



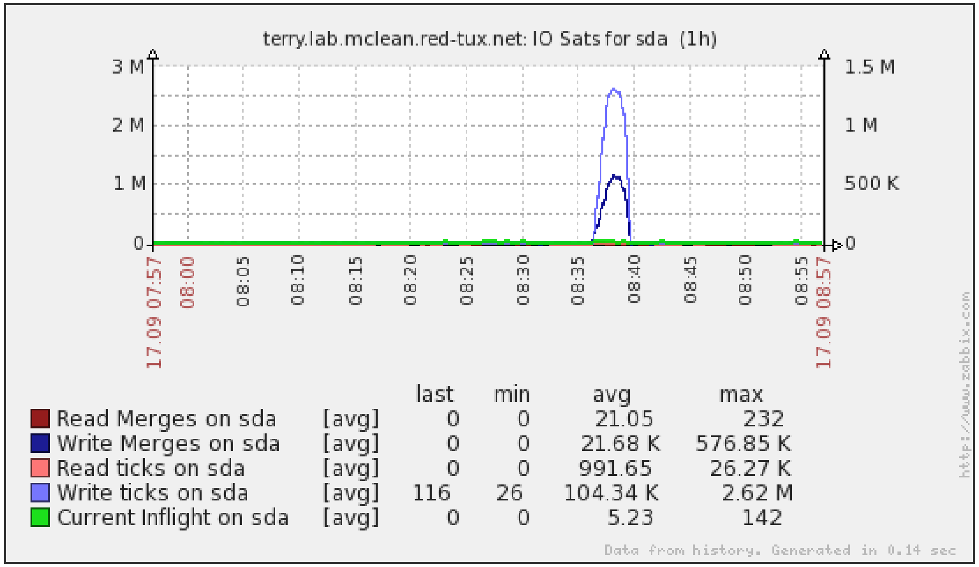




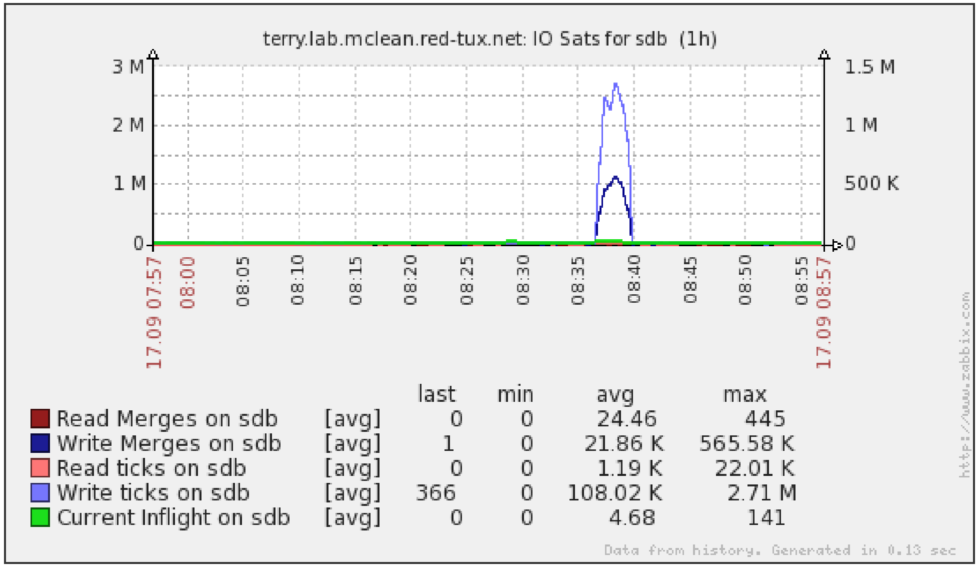
- vfs.dev.read[<disk>,operations]

Real World Example

- Let's add some more graphs (Disk)



		last	min	avg	max
🟥 Read Merges on sda	[avg]	0	0	21.05	232
🟦 Write Merges on sda	[avg]	0	0	21.68 K	576.85 K
🟥 Read ticks on sda	[avg]	0	0	991.65	26.27 K
🟦 Write ticks on sda	[avg]	116	26	104.34 K	2.62 M
🟩 Current Inflight on sda	[avg]	0	0	5.23	142



		last	min	avg	max
🟥 Read Merges on sdb	[avg]	0	0	24.46	445
🟦 Write Merges on sdb	[avg]	1	0	21.86 K	565.58 K
🟥 Read ticks on sdb	[avg]	0	0	1.19 K	22.01 K
🟦 Write ticks on sdb	[avg]	366	0	108.02 K	2.71 M
🟩 Current Inflight on sdb	[avg]	0	0	4.68	141

- UserParameter=IOStat.rmerges[*],awk '{print $$2}' /sys/block/$1/stat
- UserParameter=IOStat.rticks[*],awk '{print $$4}' /sys/block/$1/stat
- UserParameter=IOStat.wmerges[*],awk '{print $$6}' /sys/block/$1/stat
- UserParameter=IOStat.wticks[*],awk '{print $$8}' /sys/block/$1/stat
- UserParameter=IOStat.inflight[*],awk '{print $$9}' /sys/block/$1/stat

Real World Example

- Let's change some kernel values and see how our performance improves. (Values are stored in /etc/sysctl.conf)
- Set the amount of memory in bytes available per socket
  - net.core.wmem_max = 131071 → 10485760
  - net.core.rmem_max = 131071 → 10485760
- Set the default socket buffer in bytes
  - net.core.wmem_default = 8192
  - net.core.rmem_default = 8192

Real World Example

Changed values, continued

- Increase the total number of pages (4096 bytes) available to the TCP Stack
  - net.ipv4.tcp_mem

    2561 4096 5120 → 171552 228736 343104
- Increase the number of bytes available per TCP connection
  - net.ipv4.tcp_wmem and net.ipv4.tcp_rmem

4096 16384 4194304 → 8192 8388608 10485760

Real World Example

Changed values, continued

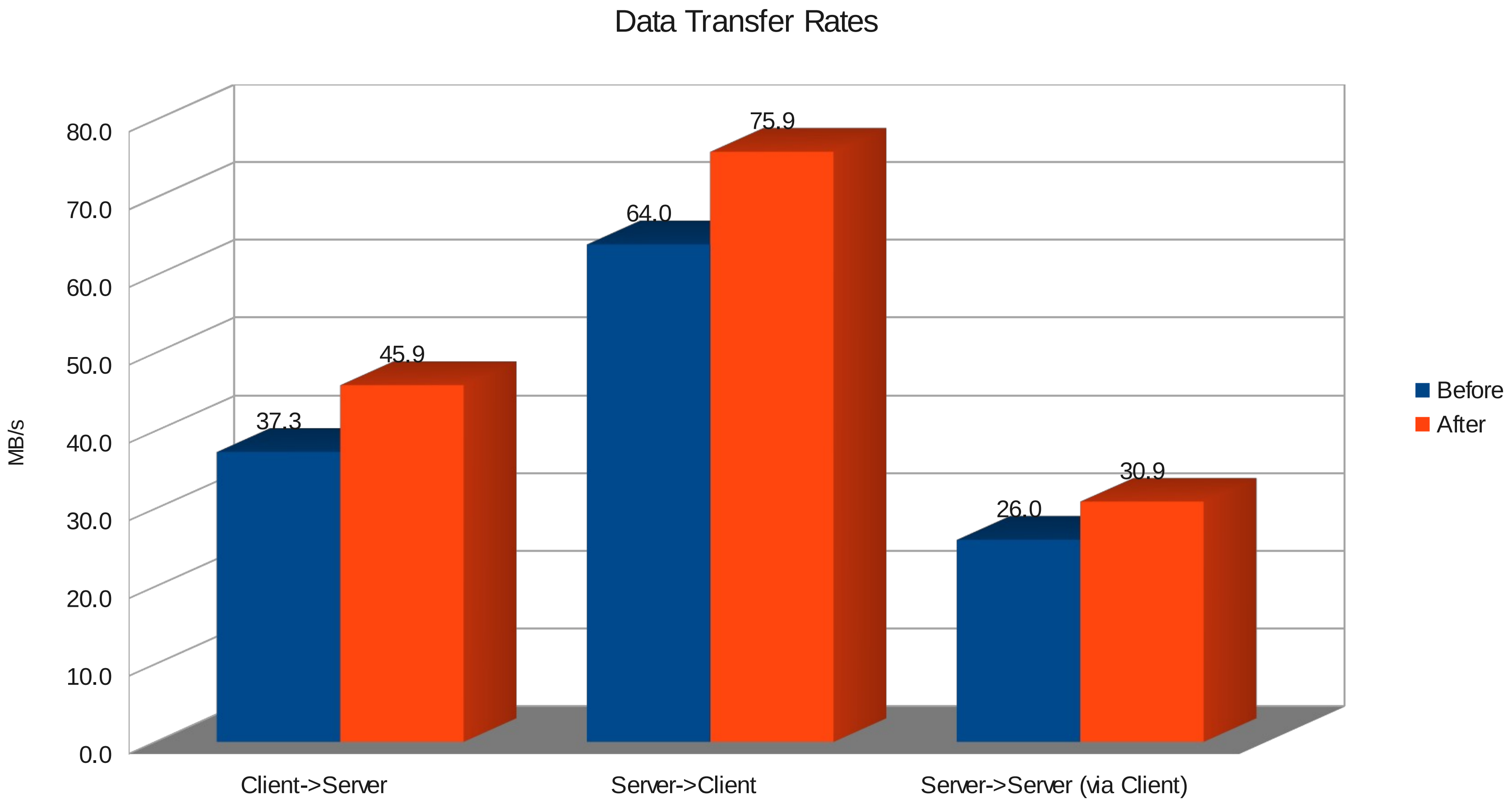
- /etc/fstab mount options

  intr,timeo=1 → intr,timeo=150

  - timeo is the timeout for NFS requests in tenths of a second
  - intr allows for the killing of IO requests to NFS hard mounts should there be a network failure.
    - This option can save you much anguish!

Real World Example

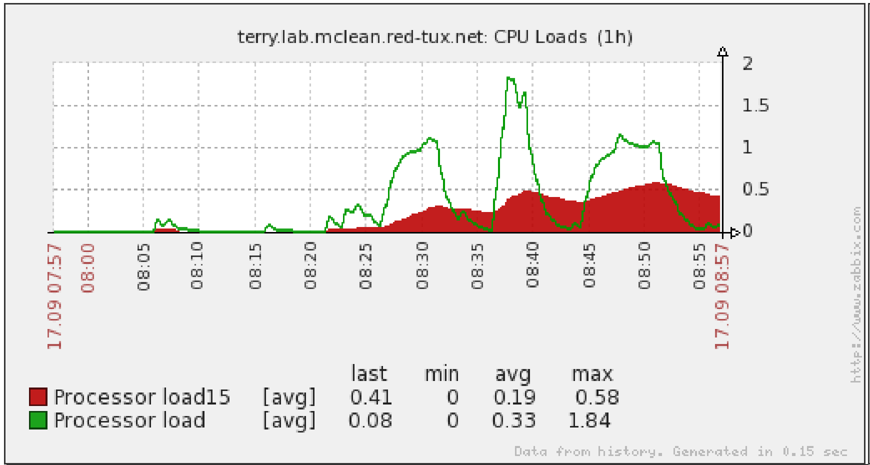
- The results:



Real World Example

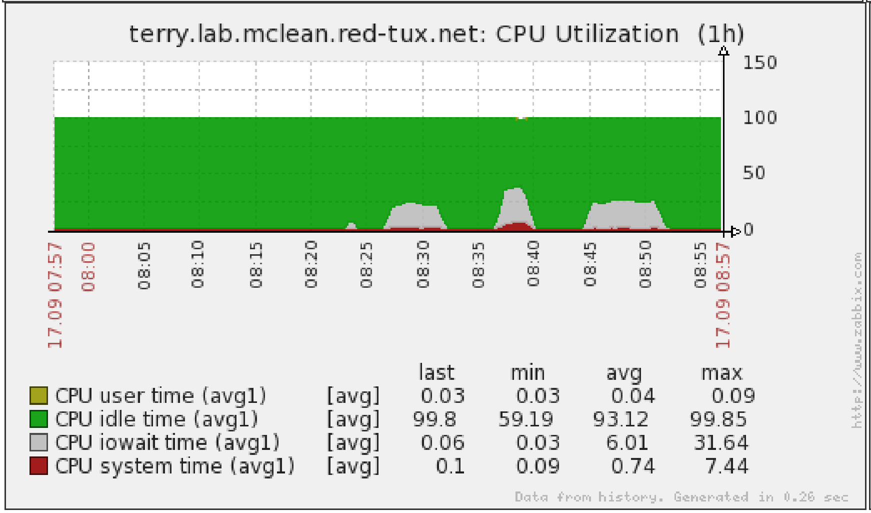
- What do the graphs show?

Before



		last	min	avg	max
🟥 Processor load15	[avg]	0.41	0	0.19	0.58
🟩 Processor load	[avg]	0.08	0	0.33	1.84

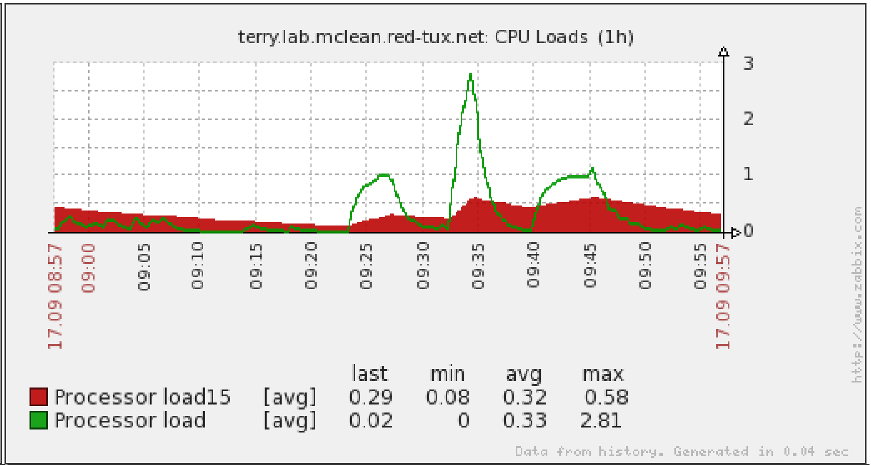
Data from history. Generated in 0.15 sec



		last	min	avg	max
🟨 CPU user time (avg1)	[avg]	0.03	0.03	0.04	0.09
🟩 CPU idle time (avg1)	[avg]	99.8	59.19	93.12	99.85
⬜ CPU iowait time (avg1)	[avg]	0.06	0.03	6.01	31.64
🟥 CPU system time (avg1)	[avg]	0.1	0.09	0.74	7.44

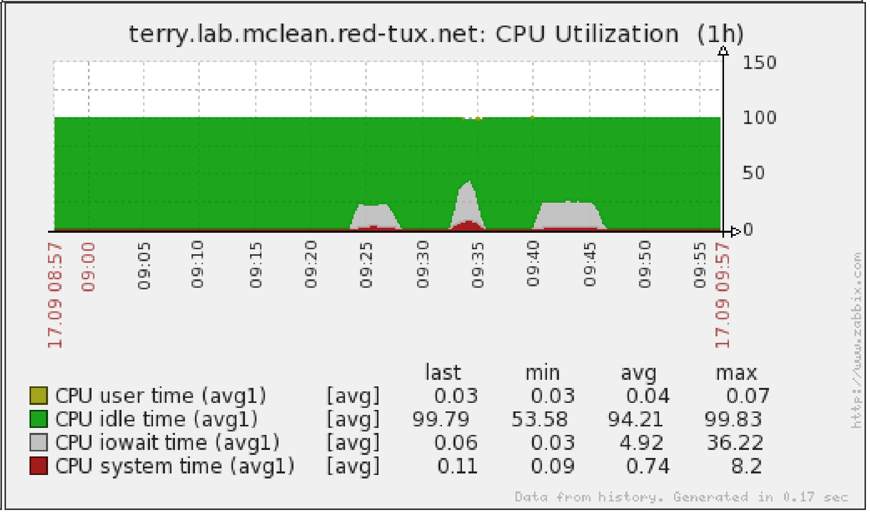
Data from history. Generated in 0.26 sec

After



		last	min	avg	max
🟥 Processor load15	[avg]	0.29	0.08	0.32	0.58
🟩 Processor load	[avg]	0.02	0	0.33	2.81

Data from history. Generated in 0.04 sec

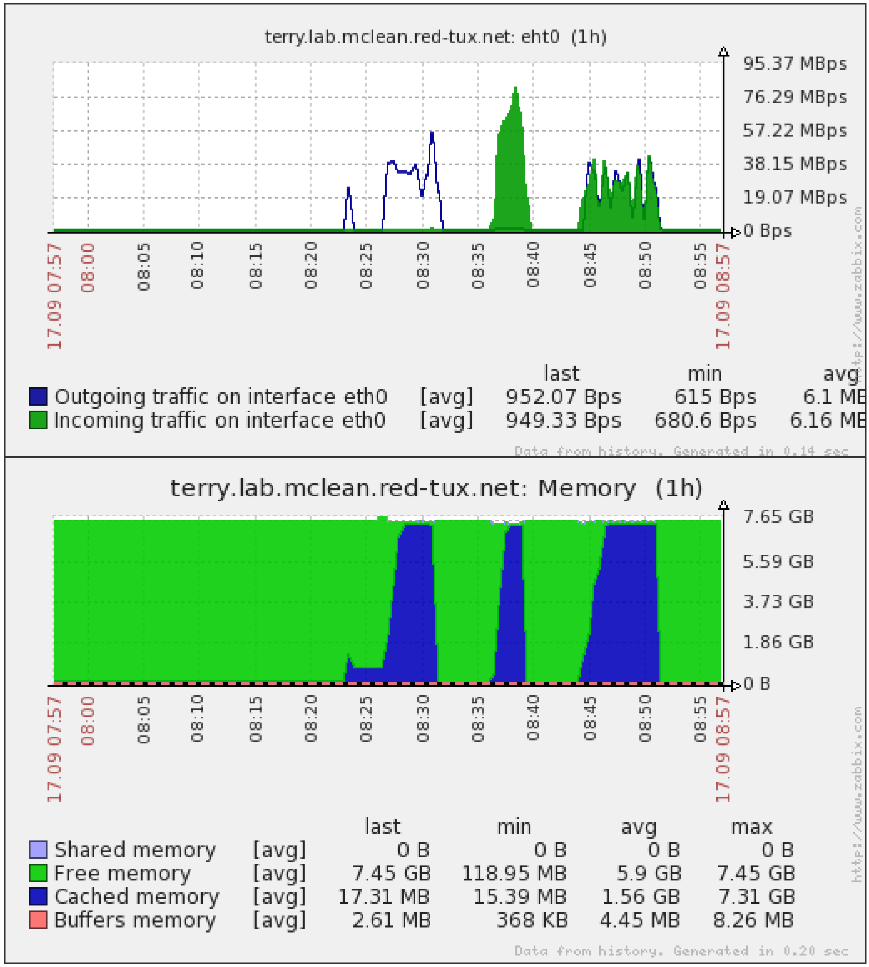


		last	min	avg	max
🟨 CPU user time (avg1)	[avg]	0.03	0.03	0.04	0.07
🟩 CPU idle time (avg1)	[avg]	99.79	53.58	94.21	99.83
⬜ CPU iowait time (avg1)	[avg]	0.06	0.03	4.92	36.22
🟥 CPU system time (avg1)	[avg]	0.11	0.09	0.74	8.2

Data from history. Generated in 0.17 sec

Real World Example

Before



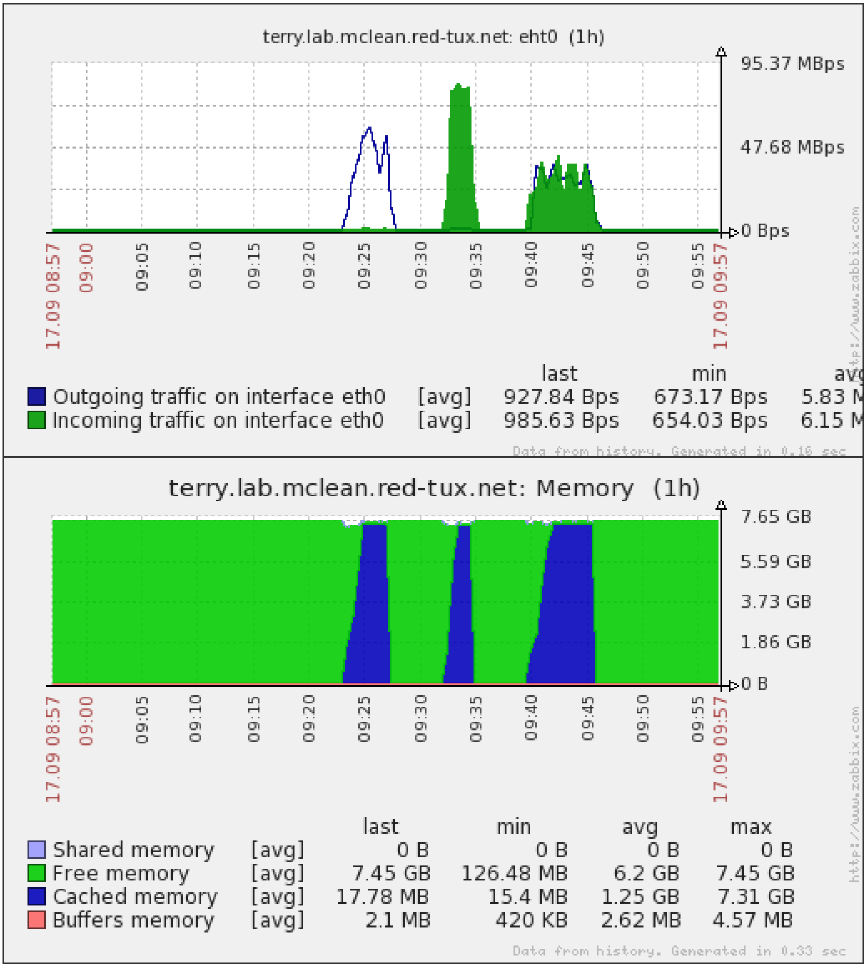
		last	min	avg
🟦 Outgoing traffic on interface eth0	[avg]	952.07 Bps	615 Bps	6.1 MB
🟩 Incoming traffic on interface eth0	[avg]	949.33 Bps	680.6 Bps	6.16 MB

Data from history. Generated in 0.14 sec

		last	min	avg	max
🟪 Shared memory	[avg]	0 B	0 B	0 B	0 B
🟩 Free memory	[avg]	7.45 GB	118.95 MB	5.9 GB	7.45 GB
🟦 Cached memory	[avg]	17.31 MB	15.39 MB	1.56 GB	7.31 GB
🟥 Buffers memory	[avg]	2.61 MB	368 KB	4.45 MB	8.26 MB

Data from history. Generated in 0.20 sec

After



		last	min	avg
🟦 Outgoing traffic on interface eth0	[avg]	927.84 Bps	673.17 Bps	5.83 M
🟩 Incoming traffic on interface eth0	[avg]	985.63 Bps	654.03 Bps	6.15 M

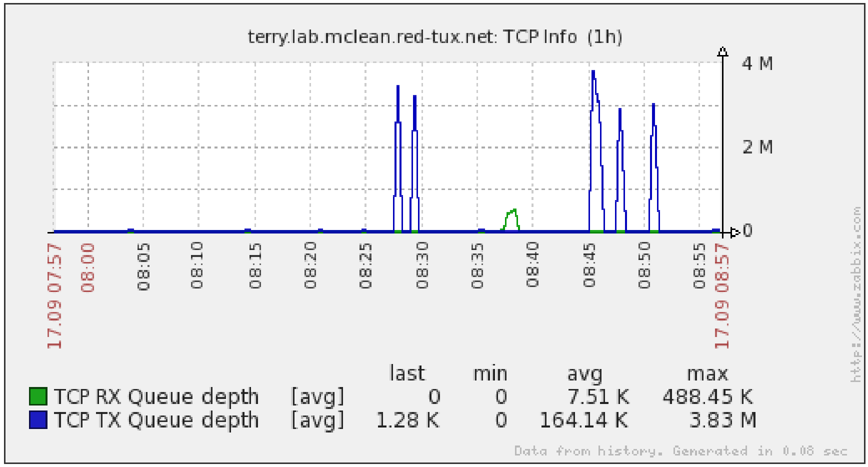
Data from history. Generated in 0.16 sec

		last	min	avg	max
🟪 Shared memory	[avg]	0 B	0 B	0 B	0 B
🟩 Free memory	[avg]	7.45 GB	126.48 MB	6.2 GB	7.45 GB
🟦 Cached memory	[avg]	17.78 MB	15.4 MB	1.25 GB	7.31 GB
🟥 Buffers memory	[avg]	2.1 MB	420 KB	2.62 MB	4.57 MB

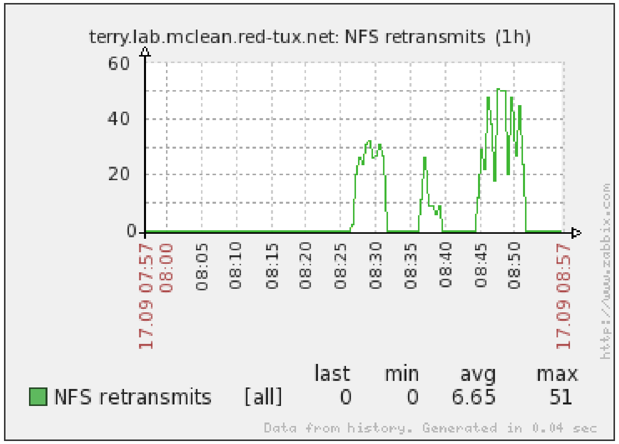
Data from history. Generated in 0.33 sec

Real World Example

Before

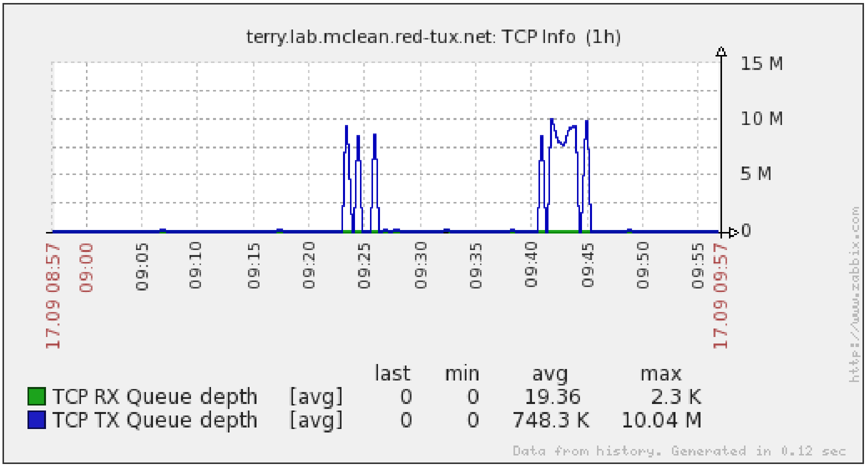


		last	min	avg	max
🟩 TCP RX Queue depth	[avg]	0	0	7.51 K	488.45 K
🟦 TCP TX Queue depth	[avg]	1.28 K	0	164.14 K	3.83 M

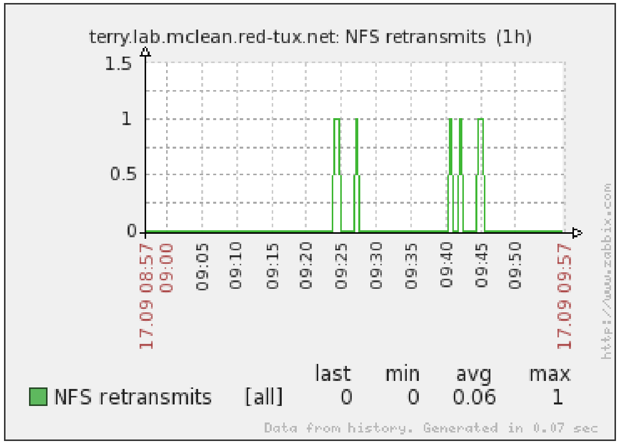


		last	min	avg	max
🟩 NFS retransmits	[all]	0	0	6.65	51

After



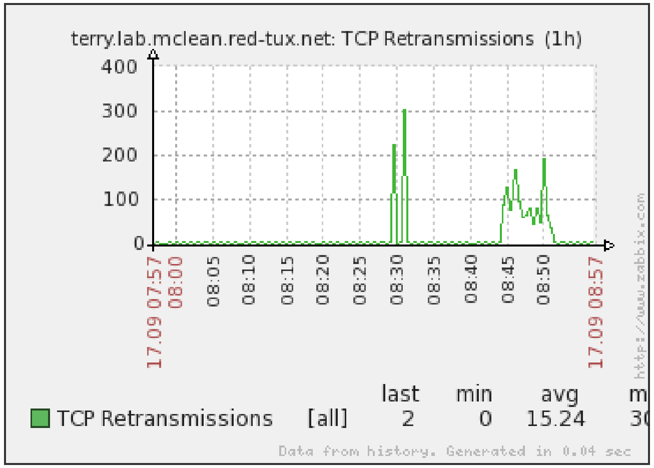
		last	min	avg	max
🟩 TCP RX Queue depth	[avg]	0	0	19.36	2.3 K
🟦 TCP TX Queue depth	[avg]	0	0	748.3 K	10.04 M



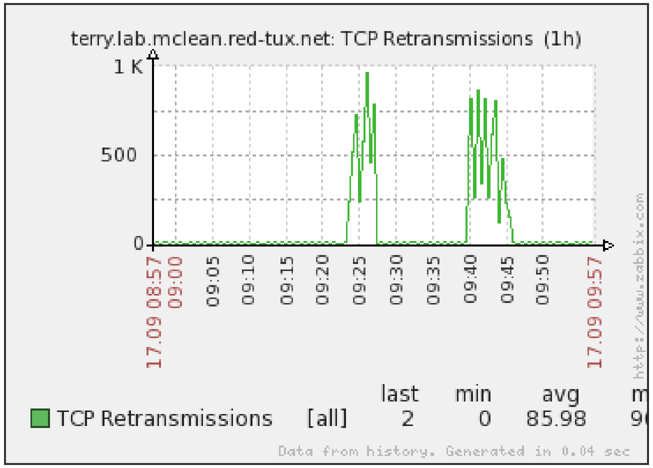
		last	min	avg	max
🟩 NFS retransmits	[all]	0	0	0.06	1

Real World Example

Before

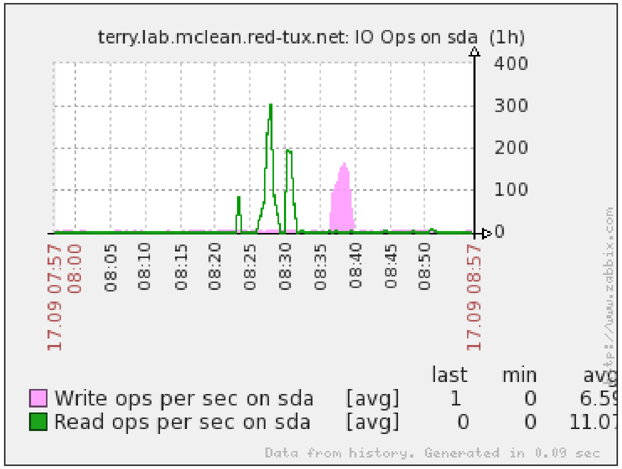


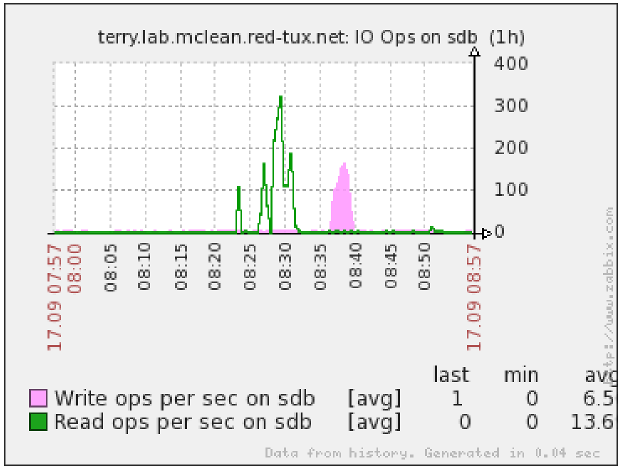
After

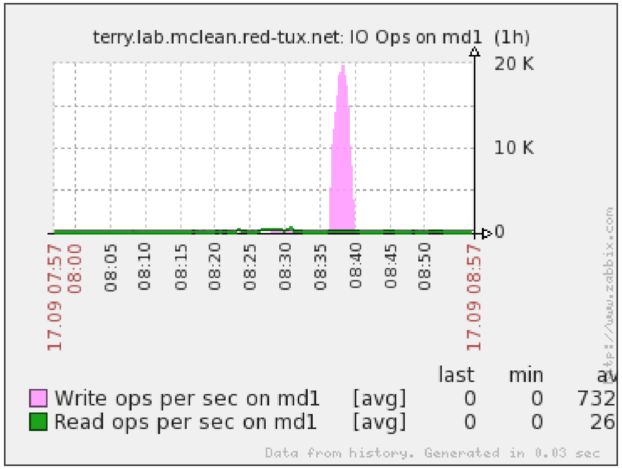


Real World Example

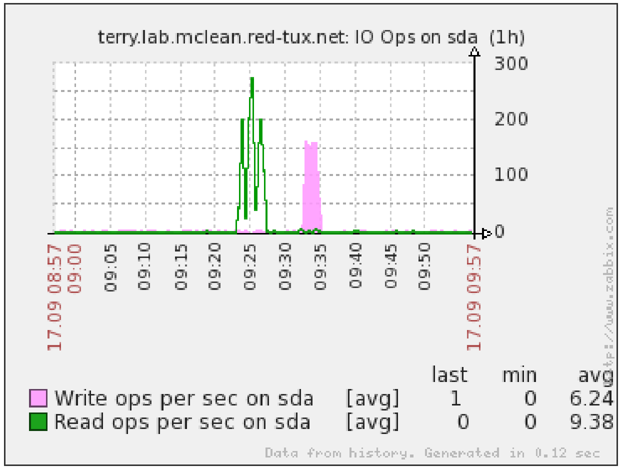
Before

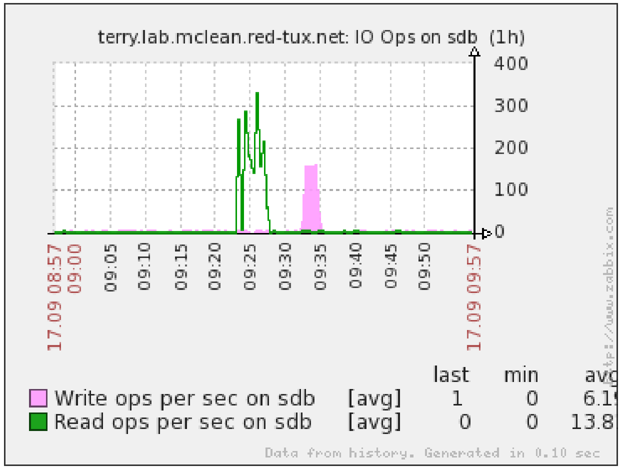


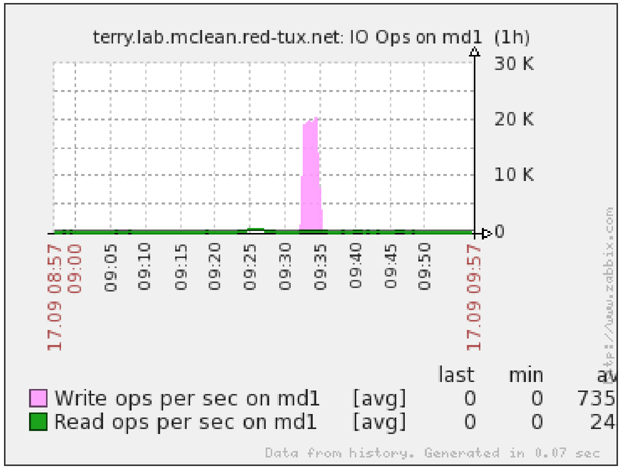




After

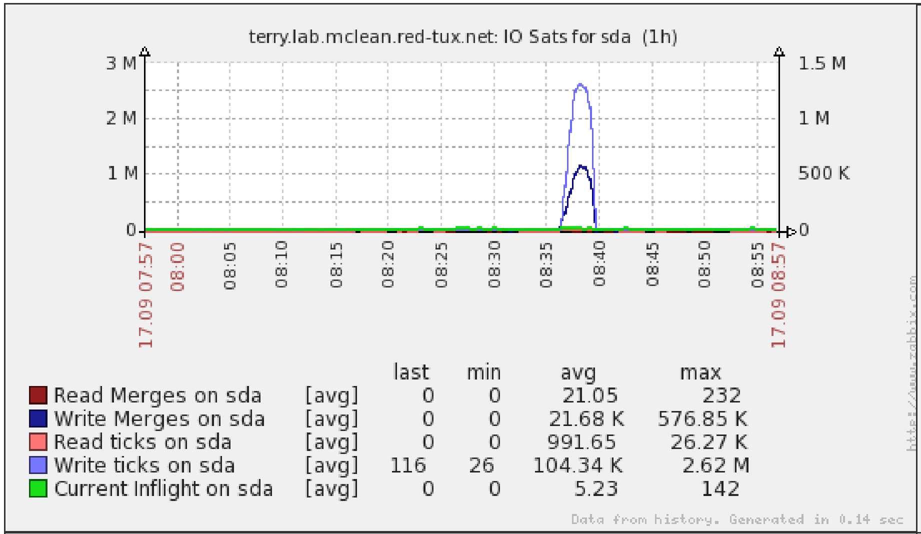




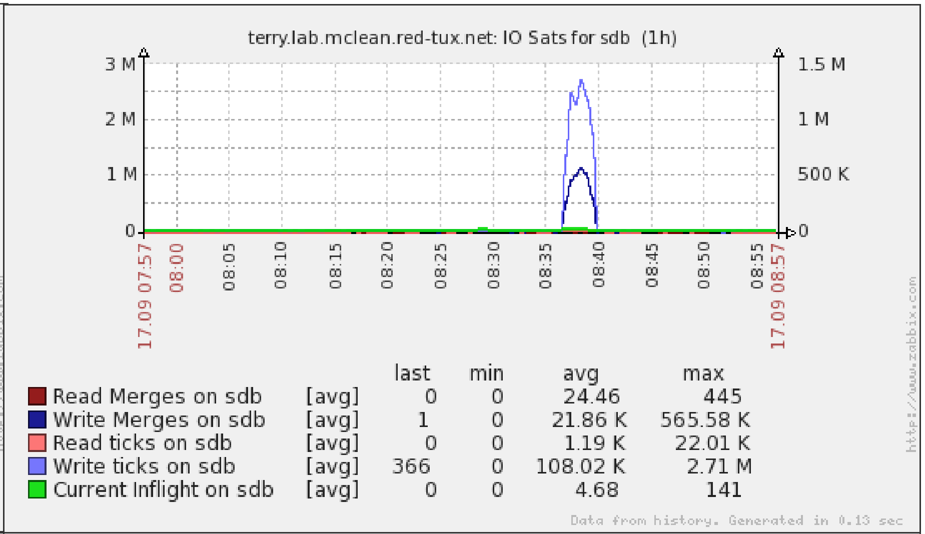


Real World Example

Before

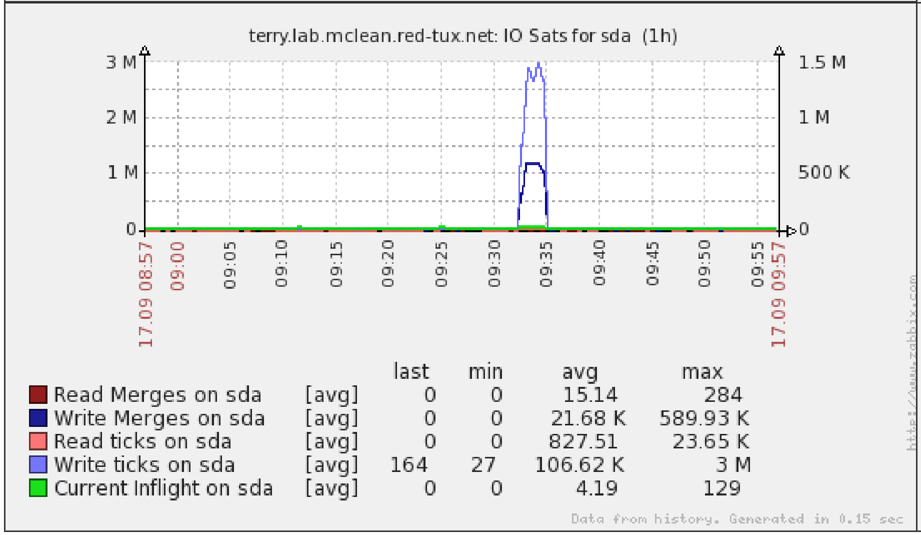

terry.lab.mclean.red-tux.net: IO Sats for sda (1h)

		last	min	avg	max
🟥 Read Merges on sda	[avg]	0	0	21.05	232
🟦 Write Merges on sda	[avg]	0	0	21.68 K	576.85 K
🟥 Read ticks on sda	[avg]	0	0	991.65	26.27 K
🟦 Write ticks on sda	[avg]	116	26	104.34 K	2.62 M
🟩 Current Inflight on sda	[avg]	0	0	5.23	142

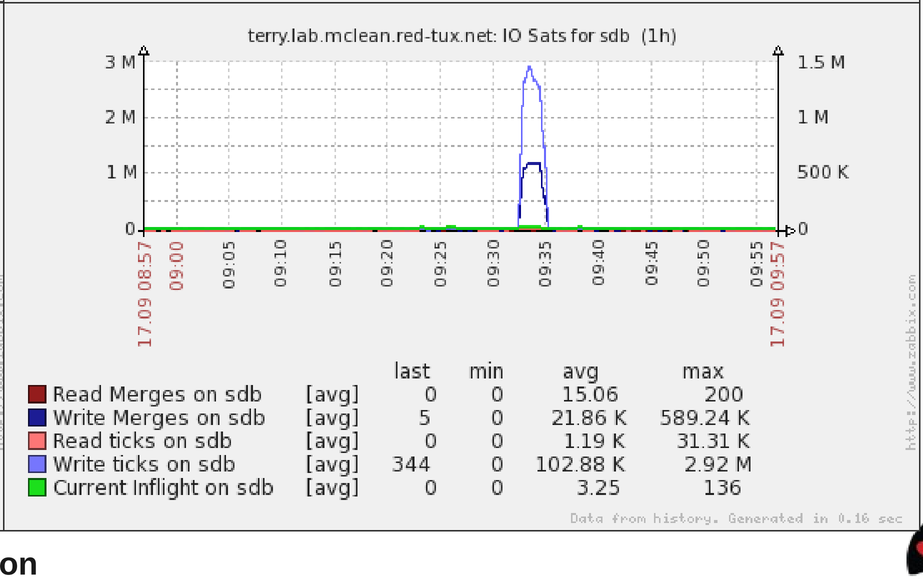

terry.lab.mclean.red-tux.net: IO Sats for sdb (1h)

		last	min	avg	max
🟥 Read Merges on sdb	[avg]	0	0	24.46	445
🟦 Write Merges on sdb	[avg]	1	0	21.86 K	565.58 K
🟥 Read ticks on sdb	[avg]	0	0	1.19 K	22.01 K
🟦 Write ticks on sdb	[avg]	366	0	108.02 K	2.71 M
🟩 Current Inflight on sdb	[avg]	0	0	4.68	141

After


terry.lab.mclean.red-tux.net: IO Sats for sda (1h)

		last	min	avg	max
🟥 Read Merges on sda	[avg]	0	0	15.14	284
🟦 Write Merges on sda	[avg]	0	0	21.68 K	589.93 K
🟥 Read ticks on sda	[avg]	0	0	827.51	23.65 K
🟦 Write ticks on sda	[avg]	164	27	106.62 K	3 M
🟩 Current Inflight on sda	[avg]	0	0	4.19	129


terry.lab.mclean.red-tux.net: IO Sats for sdb (1h)

		last	min	avg	max
🟥 Read Merges on sdb	[avg]	0	0	15.06	200
🟦 Write Merges on sdb	[avg]	5	0	21.86 K	589.24 K
🟥 Read ticks on sdb	[avg]	0	0	1.19 K	31.31 K
🟦 Write ticks on sdb	[avg]	344	0	102.88 K	2.92 M
🟩 Current Inflight on sdb	[avg]	0	0	3.25	136

Conclusion

- Why do we performance tune?
  - Maximize ROI
- Little's Law
  - $L=\lambda h$
    - Queue Depth = Arrival Rate * Time to Service
  - Foundation of performance tuning
- SystemTap
  - A good tool for instrumenting the Linux Kernel for specific needs.

Conclusion

- Documentation
  - Kernel docs are indispensable for understanding performance tuning.
  - Red Hat Knowledge base, it's the secret sauce.
- Performance tuning requires a disciplined approach.
  - Document the before and after performance
  - Document the before and after variables
    - Kernel Parameters
    - fstab options etc.

Questions

Red Hat Customer Portal

https://access.redhat.com/home

Red Hat consulting

http://www.redhat.com/consulting/

proc_netstat.sh

```
#!/bin/bash

#$1 /proc/net/* file to be read
#$2 Grouping to use
#$3 Item to retrive

headers=( `cat /proc/net/$1 | grep $2 | head -1` )
length=${#headers[@]}
position=0
for (( i=1; i<=$length; i++)); do
  if [[ "${headers[$i]}" == "$3" ]]; then
    position=$i
    break
  fi
done

if [ $position -ne 0 ]; then
  data=( `cat /proc/net/$1 | grep $2 | tail -1` )
  echo ${data[$position]}
else
  echo "$3: not found"
fi
```