Linux Server Profiling

M. Edward (Ed) Borasky

http://linuxcapacityplanning.com

LinuxCon, September 23, 2009

Where Is This Stuff?

http://github.com/znmeb/LinuxCon2009/tree/master/Linux_Server_Profiling_Using_Open_Source_Tools_for_Bottleneck_Analysis/

Why Profile Linux Servers?

Server profiling is an integral part of your comprehensive software performance engineering / capacity planning process, which is itself integrated with your organization's mission.

Why Profile Linux Servers?

Your server tanked, customers are screaming, nobody did *any* software performance engineering or capacity planning, all of the subject matter experts are on vacation and *you* are on call.

Why Profile Linux Servers?

It's fun, and you can do it all with open source tools.

sysstat Overview

- Available in nearly all Linux distributions
- Usually *not* installed by default
- Main user-level tools are *sar* and *iostat*
- Can be set up to automatically collect data and generate reports

Should I Use *sar* or *iostat*?

- *iostat* only collects overall processor utilization and disk data
- *sar* collects data for each processor and other performance metrics
- *iostat* collects merges and both reads and writes per second
- *sar* does not collect merges and only collects total operations per second
- *iostat* only gives a report, and requires some parsing to post-process
- *sar* can collect binary data using *sadc*, then format results as a report (*sar*), a database input format or a Perl-parsable data file (*sadf*)

Partition Statistics in *sysstat*

- Only available in some kernels
  - Most 2.4 kernels
  - Not in 2.6 kernels before 2.6.25
  - Re-introduced in 2.6.25
- Partition statistics make the operational analysis of swapping (and other things) easier

Primary Bottleneck

Definition

The *primary bottleneck* is the processor or disk with the highest utilization.

Starting *iostat*

```
$ iostat -cdmxt 2 > iostat.log
```

- -c: report CPU usage
- -d: report device usage
- -m: report device transfer rates in megabytes
- -x: report extended device statistics
- -t: time stamp the output
- 2: number of seconds between samples

Sample *iostat* Report

Linux 2.6.27.29-0.1-default (DreamScape) 09/20/2009 _x86_64_
Time: 10:33:17 PM

avg-cpu:	%user	%nice	%system	%iowait	%steal	%idle
	1.05	0.00	1.56	4.72	0.00	92.70

Device:	rrqm/s	wrqm/s	r/s	w/s	rMB/s	wMB/s	avgrq-sz	avgqu-sz	await	svctm	%util
sda	11.22	10.57	11.76	2.69	0.24	0.05	41.23	0.24	16.46	6.12	8.84
sda1	2.21	0.00	0.45	0.01	0.01	0.00	46.58	0.00	7.96	7.62	0.35
sda2	0.60	0.00	0.09	0.00	0.00	0.00	35.90	0.00	20.60	17.40	0.15
sda3	8.33	10.57	10.91	2.68	0.23	0.05	41.93	0.23	16.79	6.33	8.60
sdb	19.19	0.05	0.92	0.04	0.02	0.00	37.51	0.01	6.53	4.47	0.43
sdb1	14.66	0.03	0.40	0.01	0.01	0.00	37.40	0.00	5.31	3.96	0.16
sdb2	0.60	0.00	0.04	0.00	0.00	0.00	16.56	0.00	15.11	12.00	0.05
sdb3	3.63	0.02	0.41	0.03	0.01	0.00	38.74	0.00	6.91	6.33	0.28

What Do We Want From Above?

- Do “man iostat” for a full explanation of all the fields
- CPU utilization: %user + %nice + %system + %steal
  - This is the average over all processors / cores
- For each device, device utilization: %util
- Ignoring partition statistics

I Love It When A Plan Comes Together

- Collect *iostat* data
- Write a parser in Perl to convert to CSV
- Read the CSV data into R and make a boxplot

Boxplot Notes

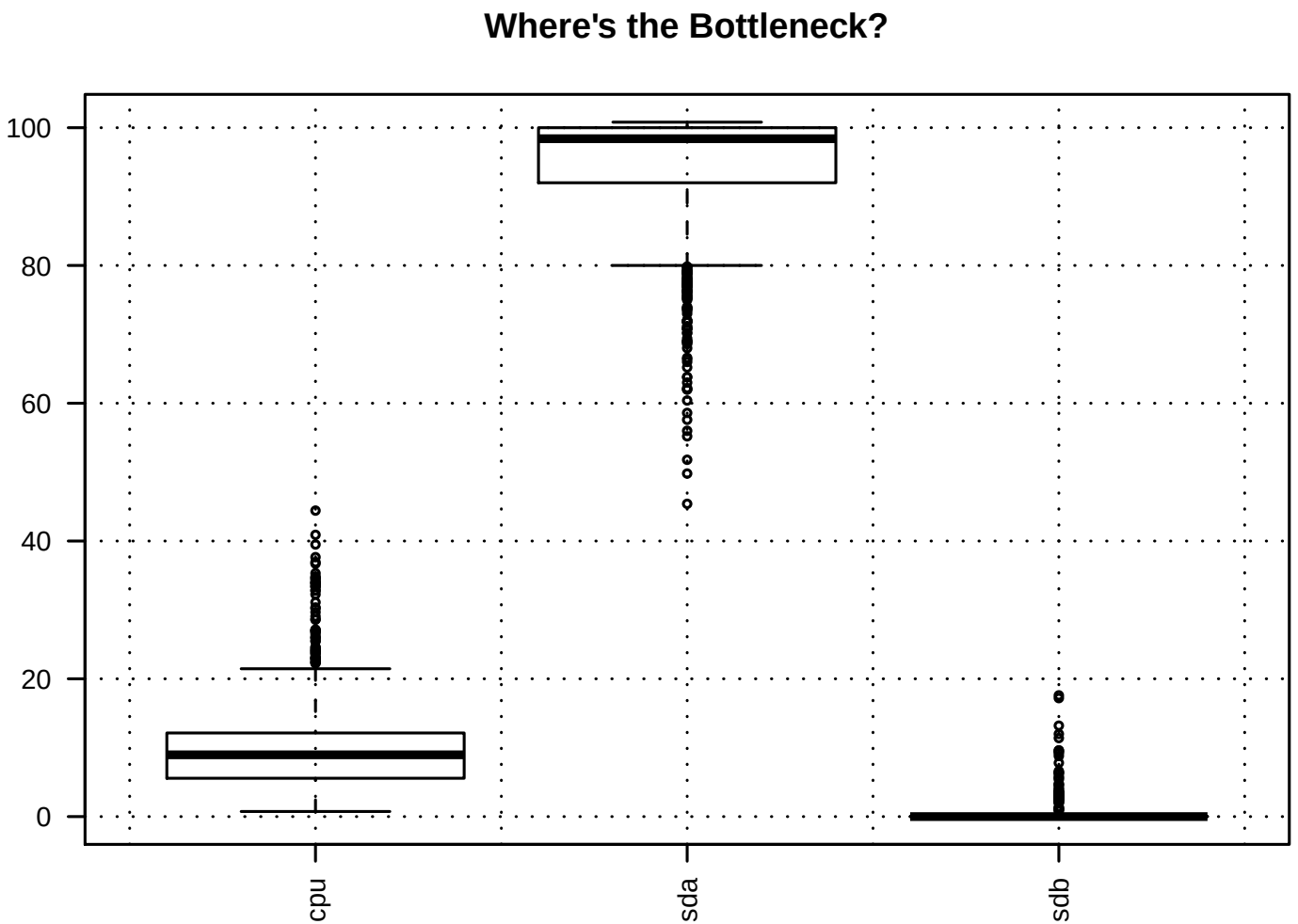
- Bottom whisker is approximately 5th percentile
- Bottom of box is 25th percentile
- Center line is median
- Top of box is 75th percentile
- Top whisker is approximately 95th percentile
- Circles are outliers

Interpreting Boxplots

- *Location* (how high is the median?)
  - Is smaller or larger better? It depends on the metric!
- *Scale* (how wide is the box?)
  - smaller usually better
- *Practical range* (how far apart are the whiskers?)
  - smaller is usually better
- Are there *outliers*?
  - "bad" outliers may represent things you need to fix
  - "good" outliers may represent "lucky breaks" you can't depend on
- Limitation: does not work well if distribution is bi-modal

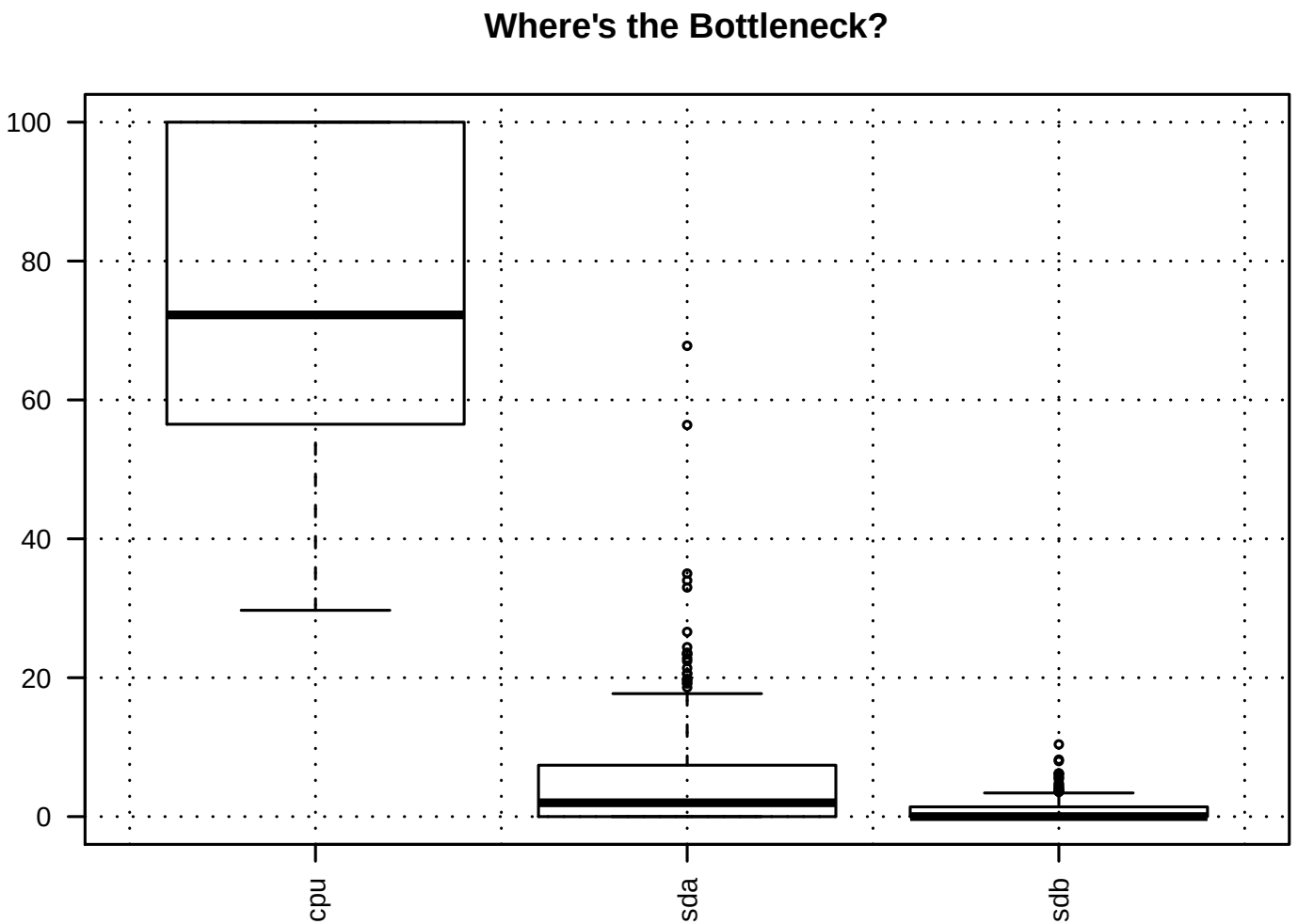
I/O Bound System – Utilization Boxplot

iozone benchmark on a SATA drive



CPU Bound System – Utilization Boxplot

Recompiling R from Source



Summary

1. Install *sysstat.*
2. Install *R.*
3. Start *iostat* and collect data while workload is running.
4. Stop *iostat* and parse log.
5. Make boxplot with *R.*
6. Are you CPU or I/O bound?

Why *blktrace* and *seekwatcher*?

- I/O subsystems are expensive and complicated
- Processor capacity increasing faster than hard drive bandwidth
- Linux I/O is complicated
  - Four different schedulers
  - Tradeoffs between page cache and other uses of RAM
  - Four major journaled filesystems, and more on the way
  - Dozens of tunable parameters
  - Logical volume managers, hardware and software RAID, storage area networks, solid-state disks
- Many applications are I/O bound
  - Especially interested in large PostgreSQL databases
- *blktrace and seekwatcher let you capture, analyze and visualize behavior of real applications on real I/O subsystems!*

Software

- openSUSE 11.1 Linux 2.6.27 kernel
- Default scheduler – completely fair queuing (CFQ)
- *iozone* 3.321
- *blktrace* 2.0.0 from developer Git tree
- *seekwatcher* 0.12
- Data located in an XFS partition towards the outside of the disk

```
Filesystem Size Used Avail Use% Mounted on
/dev/sda5  20G  13G  7.6G  63%  /
```

Who?

- *blktrace*
  - Jens Axboe (Oracle)
    - Also maintains the Linux block I/O layer
    - Wrote most of the kernel-level code
  - Alan Brunelle (HP)
    - Wrote most of the analysis packages
    - Wrote the documentation
- *seekwatcher*
  - Chris Mason (Oracle)
    - Also developing *btrfs*

Overview Of *blktrace*

- Brunelle, Alan P. (2006) "Block I/O Layer Tracing: blktrace" `http://www.gelato.org/pdf/apr2006/gelato_ICE06apr_blktrace_brunelle_hp.pdf`

When?

- Underlying kernel mechanisms introduced 2.6.16 – 2.6.17
  - Between Red Hat Enterprise Linux 4 (2.6.9) and Red Hat Enterprise Linux 5 (2.6.18)
- Requires 2.6.17 or later kernel
- Git repository for *blktrace* goes back to August of 2005
- *1.0.0 release October 31, 2008*

How?

1. Install *blktrace*
2. Run your application and gather data
3. Post-process / analyze the data
4. Remove a bottleneck
5. Is it fast enough yet?
   - Yes? Ship it!
   - No? Go back to 2.

Installing From Source

- 1.0.0 release tarball is at `http://brick.kernel.dk/snaps/blktrace-1.0.0.tar.bz2`
  - October 30, 2008
- "Latest" Git tarball is at `http://brick.kernel.dk/snaps/blktrace-git-latest.tar.gz`
  - Latest is September 1, 2009
  - I used Feburary 18, 2009 tarball
- Git repository is at *git://git.kernel.dk/blktrace.git*

  git clone git://git.kernel.dk/blktrace.git
  http://git.kernel.dk/?p=blktrace.git;a=summary
- Mailing list is *linux-btrace@vger.kernel.org*

bash Script To Build *blktrace*, *btt*, *btrecord*, *btreplay* And Documents

```
#! /bin/bash -v
# build from upstream Git source
rm -fr blktrace
git clone git://git.kernel.dk/blktrace.git
cd blktrace
make 2>&1 | tee ../make.log
make docs 2>&1 | tee ../docs.log
sudo make install 2>&1 | tee ../install.log
cd ..
```

Client and Server

- *blktrace* records *every* major event in the life of *every* I/O
- That includes its *own* I/Os!
- This skews the results and adds overhead
- So:
  - We capture trace data on a system under test
  - Ship the trace data to another machine over the network
- *blktrace calls the system under test the* **client** *and the other machine the* **server**

Server Setup

- Install *blktrace*
- Open up port 8462 in your firewall!
- Set aside disk space for the traces
- Type

  blktrace -l

- You don't need to be *root*!

Client Setup

System Under Test

- Get some hardware :)
- Install Linux, *blktrace*, application and data
- Open up port 8462 on your firewall
- Start up *blktrace*
  - Here you do need to be *root*

Basic *blktrace* Command Line

sudo blktrace -s -h *server* *list-of-devices*

- *server* is the name or IP address of the server
- *list-of-devices* is a list of the devices you want to trace
  - List needs to be in quotes
- *Remember: you get a trace record for every major event in the life of every I/O!*
- Start your application up
- Order more disk space for your *blktrace* server :)

Client Example

- Client (SUT) is 192.168.1.101, server is 192.168.1.100
- Device is */dev/sda*
- So, on 192.168.1.101, typing

  > sudo blktrace -s -h 192.168.1.100 /dev/sda

- Yields

  blktrace: connecting to 192.168.1.100
  blktrace: connected!

On The Server

```
> blktrace -l
server: waiting for connections
server: connection from 192.168.1.101
```

Now Do Some I/O On The System Under Test

- Your benchmark goes here

Stop The Client

```
^CDevice: /dev/sda
  CPU  0: 0 events, 3455 KiB data
  CPU  1: 0 events, 715 KiB data
  Total: 0 events (dropped 0), 4170 KiB data
```

Stop The Server

```
server: end of run for sda
Device: sda
  CPU  0: 0 events, 3455 KiB data
  CPU  1: 0 events, 715 KiB data
  Total: 0 events (dropped 0), 4170 KiB data
^C
```

Where Are The Trace Files On The Server?

```
> ls -1tF
192.168.1.101-2009-01-11-03:55:04/
> cd 192.168.1.101-2009-01-11-03\:55\:04/
> ls -lt
total 4172
-rw-r--r-- 1 znmeb users 3537555 2009-01-10 20:19 sda.blktrace.0
-rw-r--r-- 1 znmeb users  731868 2009-01-10 20:19 sda.blktrace.1
```

What's That Stuff?

- Directory name gives client IP address and time stamp
- About 766 megabytes for my *iozone* run!
- One file for each CPU
  - Dual-core Athlon64 X2
  - *blktrace* tracks I/Os by the CPU that initiated them
- These are binary files

Post-Processing Options

- *blkparse*: formats the raw events
  - pages and pages ... not really useful without filtering
  - useful as input to other processing steps, however
  - *seekwatcher* runs this to get data, for example
- *btt*: analysis tool that comes with the package
  - flexible
  - designed by I/O engineers
  - can make some graph input files
- *seekwatcher*
  - makes pictures and movies from raw data

seekwatcher

- Written by Chris Mason (Oracle)
- Python script
- Gives a quick look
- Makes still plots or movies
  - Depends on *mplayer* to make movies
  - Depends on Python, Python *python-matplotlib* package
- Not in most distros – source is at `http://oss.oracle.com/~mason/seekwatcher/`

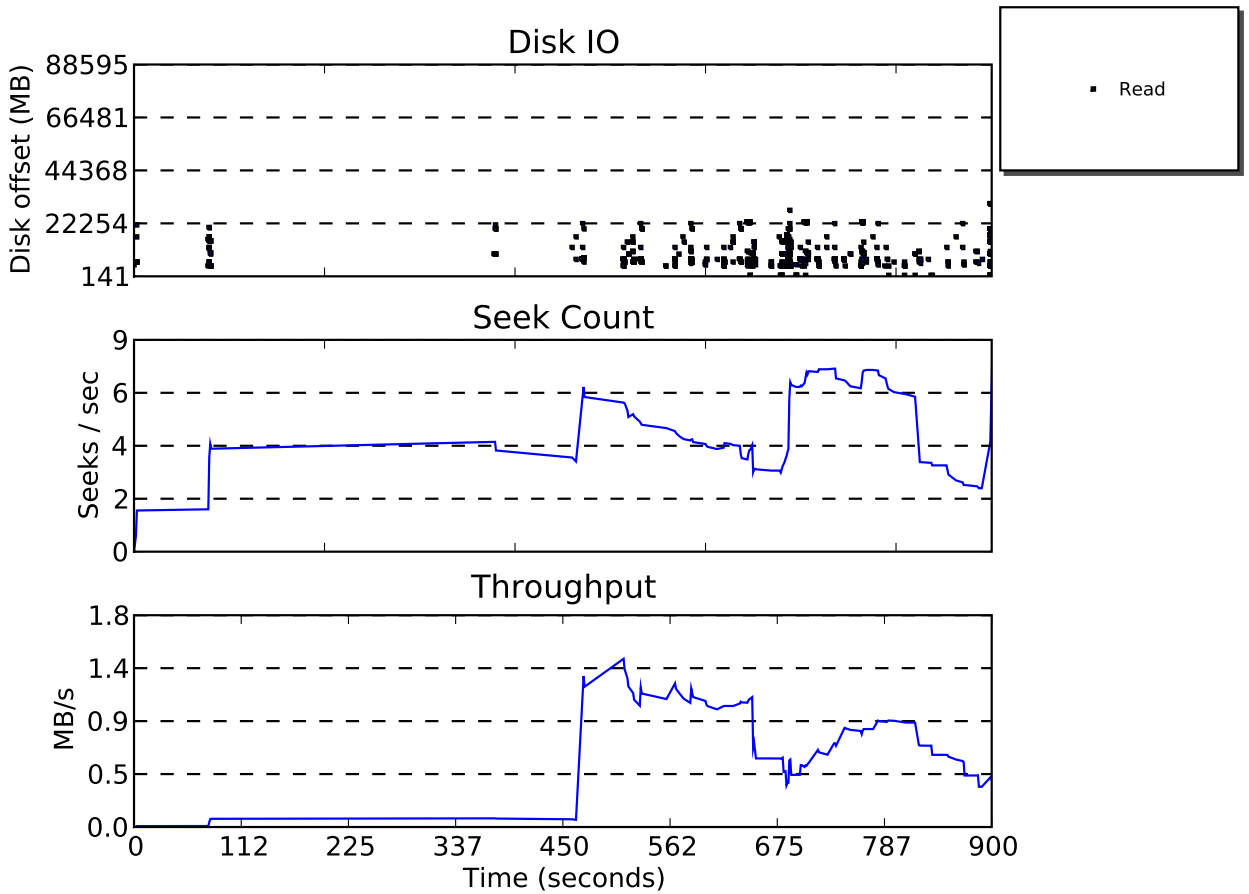
btt

- Can create a report and raw data files for further processing
- Documentation created by “make docs”
- Requires knowledge of Linux block I/O layer for interpretation
- A bit tricky to specify the command line

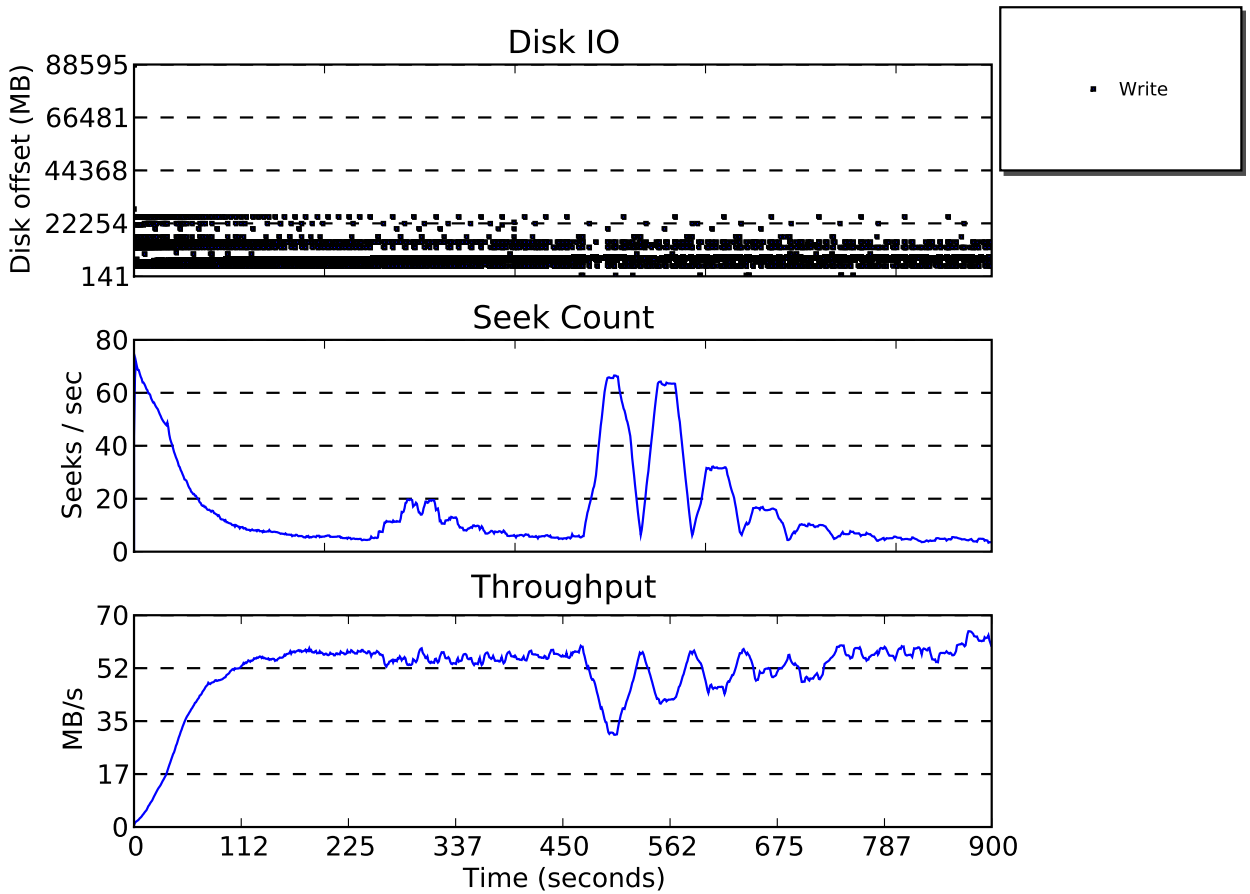
post-process.sh

```
#! /bin/bash -v
export label='sda'
export devices='/dev/sda'
blkrawverify ${label} # check data for errors
# make pictures and movies
seekwatcher -z 0:0 -R -t ${label} -o ${label}-read.eps -d ${devices}
seekwatcher -z 0:0 -W -t ${label} -o ${label}-write.eps -d ${devices}
seekwatcher -z 0:0 -t ${label} -o ${label}-both.eps -d ${devices}
seekwatcher -z 0:0 -R -t ${label} -o ${label}-read.ogg -d ${devices} –movie
seekwatcher -z 0:0 -W -t ${label} -o ${label}-write.ogg -d ${devices} –movie
seekwatcher -z 0:0 -t ${label} -o ${label}-both.ogg -d ${devices} –movie
# generate reports
blkparse -d ${label}.bin -i ${label} -O # merge the binaries
blkparse -s -h -t -i ${label} > ${label}.blkparse # (huge) text output
btt -A -i ${label}.bin > ${label}.btt # basic btt report
cd ..
exit
```

Reads Only


Disk IO
Disk offset (MB)
88595
66481
44368
22254
141
Read
Seek Count
Seeks / sec
9
6
4
2
0
Throughput
MB/s
1.8
1.4
0.9
0.5
0.0
0
112
225
337
450
562
675
787
900
Time (seconds)

Writes Only


Disk IO
Disk offset (MB)
88595
66481
44368
22254
141
Write
Seek Count
Seeks / sec
80
60
40
20
0
Throughput
MB/s
70
52
35
17
0
0
112
225
337
450
562
675
787
900
Time (seconds)

Reads and Writes