`btt` User Guide

Alan D. Brunelle (Alan.Brunelle@hp.com)

12 February 2008

1 Introduction

`btt` is a post-processing tool for the block layer IO tracing tool called blktrace. As noted in its Users Guide, blktrace

> is a block layer IO tracing mechanism which provides detailed information about request queue operations up to user space.

blktrace is capable of producing tremendous amounts of output in the form of multiple individual traces per IO executed during the traced run. It is also capable of producing some general statistics concerning IO rates and the like. `btt` goes further and produces a variety of overall statistics about each of the individual handling of IOs, and provides data we believe is useful to plot to provide visual comparisons for evaluation.

This document will discuss `btt` usage, provide some sample output, and also show some interesting plots generated from the data provided by the `btt` utility.

A short note on the ordering of this document – the actual command-line usage section occurs relatively late in the document (see section 11), as we felt that discussing some of the capabilities and output formats would make the parameter discussion easier.

This document refers to the output formats generated by `btt` version 2.00. However, the descriptions are general enough to cover output formats prior to that.

Contents

2 Getting Started

The simple pipeline to get going with `btt` is to perform the following steps:

1. Run `blktrace`, specifying whatever devices and other parameters you want. You must save the traces to disk in this step, btt does not work in live mode.

2. After tracing completes, run `blkrawverify`, specifying all devices that were traced (or at least on all devices that you will use `btt` with – section 11.5 shows how you can dictate which devices to use with btt). If blkrawverify finds errors in the trace streams saved, it is best to recapture the data – utilizing `btt` on *unclean* trace files produces inconsistent results.

   While this step is optional, we have found that performing this helps to ensure data coming from `btt` makes the most sense.

3. Run `blkparse` with the `-d` option specifying a file to store the combined binary stream. (e.g.: `blkparse -d bp.bin ...`).

   `blktrace` produces a series of binary files containing parallel trace streams – one file per CPU per device. `blkparse` provides the ability to combine all the files into one time-ordered stream of traces for all devices.

4. Run `btt` specifying the file produced by `blkparse` utilizing the `-i` option (e.g.: `btt -i bp.bin ...`).

3 Output Overview

The major default areas of output provided by `btt` include:

average component times across all IOs The time line of each IO is broken down into 3 major regions:

1. Time needed to insert or merge an incoming IO onto the request queue. This is the average time from when the IO enters the block IO layer (queue trace) until it is inserted (insert trace).

   This is denoted as *Q2I* time.

   This is also broken down into two component times:

   Q2G Time needed to *get* a request (get request trace).

   G2I Time needed to put that request onto the request queue (insert trace).

   For *merged* requests – an incoming request that is merged with a previously submitted request – we calculate *Q2M*, the amount of time between the queue trace and the merge trace.

2. Time spent on the request queue. The average time from when the IO is inserted or merged onto the request queue, until it is issued (issue trace) to the lower level driver.

   Referred to as *I2D* time[1].

3. Driver and device time – the average time from when the actual IO was issued to the driver until is completed (completion trace) back to the block IO layer.

   This is referred to as the *D2C* time

Two other sets of results are presented in this section:

1. *Q2Q* which measures the time between queue traces in the system. This provides some idea as to how quickly IOs are being handed to the block IO layer.
2. *Q2C* which measures the times for the complete life cycle of IOs during the run[2]

For each row in this output, we provide a minimum, average, maximum (which are all presented in seconds), and overall count. As an example[3]:

[1] The *issue* trace is represented by a D in the blkparse output, hence its usage in btt to refer to issue traces. Note that an I is used to refer to *insert* traces.

[2] One of the areas that needs some work in `btt` is to better understand the multiplex nature of IOs during a run. In theory, one would like $Q2I + I2D + D2C = Q2C$ however, typically there are multiple queue traces that are combined via merges into a single IO issued and completed. We currently average the queue-to-insert and queue-to-merge times, and thus tend to be quite close to the expected equation.

[3] As with this display, the author has taken some liberty in reformatting the output for better display on the printed page.

ALL	MIN	AVG	MAX	N
Q2Q	0.000000058	0.000012761	9.547941661	2262310
Q2I	0.000000272	0.000005995	0.104588839	2262311
I2D	0.000001446	0.094992714	0.239636864	2262311
D2C	0.000193721	0.030406554	1.634221408	2262311
Q2C	0.000207665	0.125405263	1.830917198	2262311

When tracking *device mapper* devices, we also break down the *Q2A* and *Q2C* times for those IOs.

Device Overhead Using the data from the previous chart, we can then provide some idea as to where IO spend most of the time on average. The following output shows the percentage of time spent in each of the phases of an IO[4]

DEV	Q2G	G2I	Q2M	I2D	D2C
(8, 80)	0.0013%	0.0004%	0.0006%	88.5005%	11.4988%
Overall	0.0003%	0.0001%	0.0041%	21.4998%	78.4958%

Device Merge Information A key measurement when making changes in the system (software *or* hardware) is to understand the block IO layer ends up merging incoming requests into fewer, but larger, IOs to the underlying driver. In this section, we show the number of incoming requests (Q), the number of issued requests (D) and the resultant ratio. We also provide values for the minimum, average and maximum IOs generated.

Looking at the following example:

DEV	#Q	#D	Ratio	BLKmin	BLKavg	BLKmax	Total
(68, 64)	2262311	18178	124.5	2	124	128	2262382

we see that (on average) the block IO layer is combining upwards of 125 incoming requests into a single request down the IO stack. The resultant average IO size is 124 blocks.

[4]It should be noted that incoming requests either go through:

1. Q2G + Q2I
   or
2. Q2M

before proceeding to I2D and D2C.

Device Seek Information Another useful measure is the variability in the sector distances between consecutively *received* – *queued* and *submitted* – *issued* IOs. The next two sections provides some rudimentary statistics to gauge the general nature of the sector differences between IOs. Values provided include the number of seeks (number of IOs submitted to lower level drivers), the *mean* distance between IOs, the *median* value for all seeks, and the *mode* - the value(s) and the counts are provided for the latter.

The first of the two sections displays values for Q2Q seek distances – providing a set of indicators showing how close incoming IO requests are to each other. The second section shows D2D seek distances – providing a set of indicators showing how close the IO requests are that are handled by underlying drivers.

```
      DEV | NSEEKS    MEAN MEDIAN | MODE
--------- | ------ ------- ------ | -------
( 68, 64) |  18178 19611.3      0 | 0(17522)
```

We have almost exclusively seen median and mode values of 0, indicating that seeks tend to have an equal amount of forward and backwards seeks. The larger the count for the mode in comparison to the total number of seeks is indicative as to how many IOs are coming out of the block IO layer in adjacent sectors. (Obviously, the higher this percentage, the better the underlying subsystems can handle them.)

Request Queue Plug Information During normal operation, requests queues are *plugged* and during such times the IO request queue elements are not able to be processed by underlying drivers. The next section shows how often the request queue was in such a state.

```
      DEV | # Plugs # Timer Us  | % Time Q Plugged
--------- | ------- ----------  | ----------------
( 68, 64) |     833(         0) |   0.356511895%
```

There are two major reasons why request queues are unplugged, and both are represented in the above table.

1. Explicit unplug request from some subsystem in the kernel.
2. Timed unplugs, due to a request queue exceeding some temporal limit for being plugged.

The total number of unplugs is equal to the number of plugs less the ones due to timer unplugs.

IOs per Unplug & Unplugs-due-to-timeout In this subsection one can see the average number of IOs on the request queue at the time of an unplug or unplug due to a timeout. The following sample shows a sample of both unplug sections:

```
=================== Plug Information ===================

       DEV |    # Plugs # Timer Us  | % Time Q Plugged
---------- | ---------- ----------  | ----------------
 (  8,  0) |       1171(       123) |   0.280946640%
 (  8, 32) |          4(         0) |   0.000325469%
---------- | ---------- ----------  | ----------------
   Overall |    # Plugs # Timer Us  | % Time Q Plugged
   Average |        587(        61) |   0.140636055%

       DEV |    IOs/Unp   IOs/Unp(to)
---------- | ----------   ----------
 (  8,  0) |        9.2          8.8
 (  8, 32) |        2.5          0.0
---------- | ----------   ----------
       DEV |    IOs/Unp   IOs/Unp(to)
   Overall |        9.2          8.8
```

This table and the preceding one have to be considered together – in the sample output in the immediately preceding table one can see how the larger number of data values for device (8,0) dominates in the overall average.

Active Requests At Q Information An important consideration when analyzing block IO schedulers is to know how many requests the scheduler has to work with. The metric provided in this section details how many requests (on average) were being held by the IO scheduler when an incoming IO request was being handled. To determine this, `btt` keeps track of how many Q requests came in, and subtracts requests that have been issued (D).

Here is a sample output of this sections:

```
================== Active Requests At Q Information ==================

       DEV |  Avg Reqs @ Q
---------- | -------------
 ( 65, 80) |          12.0
 ( 65,240) |          16.9
...
 ( 66,112) |          44.2
---------- | -------------
```

```
Overall | Avgs Reqs @ Q
Average |          17.4
```

Detailed Data

In addition to the default sections output, if one supplies the `--all-data` or `-A` argument (see section 11.2) to `btt` further sections are output:

Per Process As traces are emitted, they are tagged with the process ID of the currently running thread in the kernel. The process names are also preserved, and mapped to the ID. For each of the parts of the time line discussed above on page 4, a chart is provided which breaks down the traces according to process ID (name).

One must be aware, however, that the process ID may not have anything to do with the originating IO. For example, if an application is doing buffered IO, then the actual submitted IOs will most likely come from some page buffer management daemon thread (like pdflush, or kjournald for example). Similarly, completion traces are rarely (if ever?) going to be associated with the process which submitted the IO in the first place.

Here is a sample portion of this type of chart, showing Q2Q times per process:

```
          Q2Q         MIN         AVG         MAX       N
------------ ----------- ----------- ----------- -------
mkfs.ext3     0.000000778 0.000009074 1.797176188 1899371
mount         0.000000885 0.000672513 0.030638128      73
pdflush       0.000000790 0.000006752 0.247231307  179791
```

Per Process Averages The average columns from the above charts, are also presented in their own chart.

Per Device Similar to the per-process display, `btt` will also break down the various parts of an IOs time line based upon a per-device criteria. Here's a portion of this area, displayed showing the issued to complete times (D2C).

```
      D2C         MIN         AVG         MAX      N
--------- ----------- ----------- ----------- ------
( 65, 80) 0.000140488 0.001076906 0.149739869 169112
( 65, 96) 0.000142762 0.001215221 0.173263182 155488
( 65,112) 0.000145221 0.001254966 0.124929936 165726
( 65,128) 0.000141896 0.001159596 0.775231052 169015
( 65,144) 0.000140832 0.001290985 0.211384698 210661
( 65,160) 0.000139915 0.001175554 0.073512063 133973
( 65,176) 0.000141254 0.001104870 0.073231310 145764
( 65,192) 0.000141453 0.001234460 0.167622507 140618
...
```

Per Device Averages The average columns from the above charts, are also presented in their own chart.

Q2D Histogram A display of histogram buckets for the Q to D times – basically, from where an IO enters the block IO layer for a given device, and when it is dispatched. The buckets are arranged via the time in seconds, as in:

```
==================== Q2D Histogram ====================

       DEV | <.005 <.010 <.025 <.050 <.075 <.100 <.250 <.500 < 1.0 >=1.0
 --------- | ===== ===== ===== ===== ===== ===== ===== ===== ===== =====
 ( 66, 80) |  61.2   7.9  12.1   7.9   3.0   1.4   1.5   0.2   0.0   4.6
 ( 65,192) |  42.3   5.0   8.7  30.0   8.9   3.0   1.8   0.1   0.0   0.1
 ( 65,128) |  34.3   5.3   8.9  32.0   9.7   3.7   5.3   0.6   0.0   0.1
...
 ( 65, 64) |  59.9   4.2   6.0  24.6   4.2   0.8   0.1   0.0   0.0   0.1
 ( 66, 64) |  62.6   8.1  12.7   7.9   2.4   0.6   0.1   0.0   0.0   5.4
========== | ===== ===== ===== ===== ===== ===== ===== ===== ===== =====
       AVG |  52.9   6.2  10.0  20.1   5.3   1.7   1.4   0.2   0.0   2.1
```

4 Data Files Output

Besides the averages output by default, the following 3 files are also created with data points which may be plotted.

***file*.dat** This file provides a notion of *activity* for the system, devices and processes. The details of this file are provided in section 5.

***file*_qhist.dat** Provides histogram data for the size of incoming IO requests, for more information see section 6.

***file*_dhist.dat** Provides histogram data for the size of IO requests submitted to lower layer drivers, for more information see section 6.

Besides the default data files output, there are optional data files which can be generated by btt. These include:

iostat iostat-like data can be distilled by btt, and is described in section 7.

per IO detail Each and every IO traced can be output in a form that shows each of the IO components on consecutive lines (rather than grepping through a blkparse output file for example). The details on this file is included in section 8.

iostat Latency information – both Q2C and D2C – on a per-IO basis can be generated. These are described in sections 9 and 9.

seek details A set of data files containing all IO-to-IO sector differences can be output, with details found in section 10.

unplug histogram details A data file per device containing histogram output for the amount of IOs released at unplug time. Section 11.18 has more details.

5 Activity Data File

The activity data file contains a series of data values that indicate those periods of time when queue and complete traces are being processed. The values happen to be in a format easily handled by xmgrace[5], but is easy to parse for other plotting and/or analysis programs.

The file is split into pairs of sets of data points, where each pair contains a set of queue activity and a set of completion activity. The points are presented with the first column (X values) being the time (in seconds), and the second column (Y values) providing an on/off type of setting. For each pair, the Y values have two settings off (low) and on (high). For example, here is a snippet of a file showing some Q activity:

```
# Total System
#     Total System : q activity
0.000000000   0.0
0.000000000   0.4
0.000070381   0.4
0.000070381   0.0
1.023482637   0.0
1.023482637   0.4
6.998746618   0.4
6.998746618   0.0
7.103336799   0.0
7.103336799   0.4
17.235419786   0.4
17.235419786   0.0
26.783361447   0.0
26.783361447   0.4
26.832454929   0.4
26.832454929   0.0
28.870431266   0.0
28.870431266   0.4
28.870431266   0.4
28.870431266   0.0
```

What this indicates is that there was q activity for the system from 0.000000000 through 0.000070381, but was inactive from there to 1.023482637, and so on. Section 11.4 contains details on how to adjust btt's notion of what constitutes activity.

The pairs are arranged as follows:

- First there is the total system activity – meaning activity in either queue or completion traces across all devices.

[5] `http://plasma-gate.weizmann.ac.il/Grace/` "Grace is a WYSIWYG 2D plotting tool for the X Window System and M*tif."

- Next comes per-device activity information – for each device being traced, that request queues Q and C traces are presented.
- Last we present pairs per-process.

Using this, one is then able to plot regions of activity versus inactivity – and one can gather a sense of deltas between the queueing of IOs and when they are completed. Figure 1 shows a very simplistic chart showing some activity:

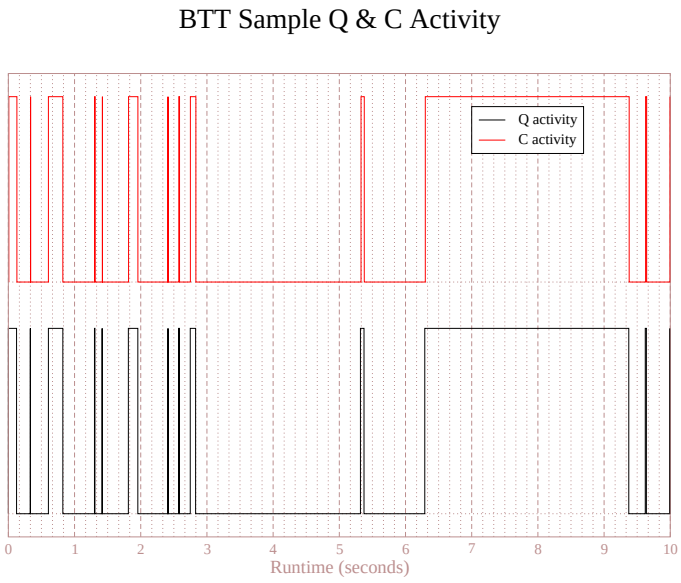

Figure 1: Simple Activity Chart

When the black line (system Q activity) is *high*, then the system is seeing relatively continuous incoming queues. Conversely, when it is low, it represents an extended period of time where no queue requests were coming in. Similarly for the red line and C activity.

6 Histogram Data Files

The histogram data files provide information concerning incoming and outgoing IO sizes (in blocks). For simplicity, the histogram buckets are one-for-one for sizes up to 1,024 blocks in the IO, and then a single bucket for all sizes greater than or equal to 1,024 blocks.

The files are again in grace-friendly format, with the first set containing data for the first 1,023 buckets, and a separate set representing sizes ≥ 1024 blocks. (This is done so that one can easily use a separate formatting specification for the latter set.)

The first column (X values) is the various IO sizes, and the second column (Y values) represents the number of IOs of that size.

Q Histogram Data File

Figure 2 is a sample graph generated from data used during some real-world analysis[6]. With the visual representation provided by this, one can quickly discern some different characteristics between the 3 runs – in particular, one can see that there is only a single red point (representing 8 blocks per IO), whereas the other two had multiple data points greater than 8 blocks.

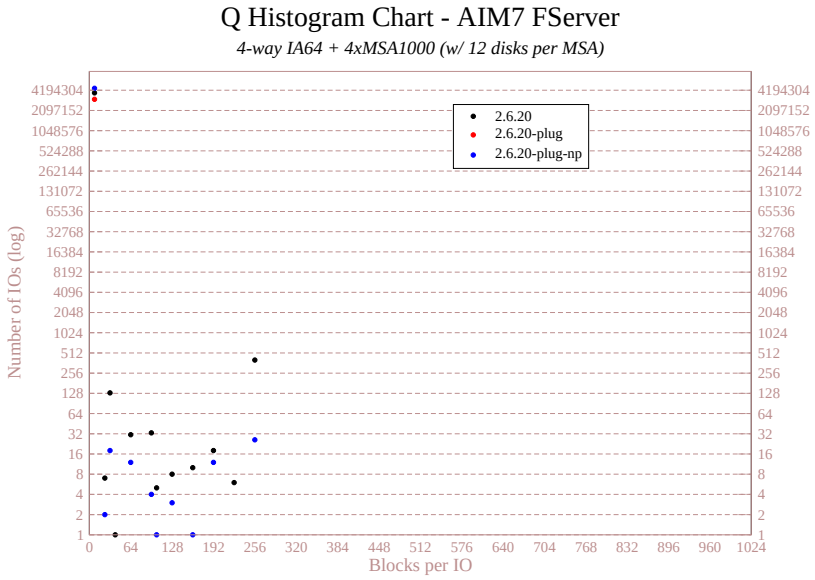

Figure 2: Q Histogram

[6]Note the logarithmic nature of the Y axis for this chart.

D Histogram Data File

Figure 3 is a sample graph generated from data used during some real-world analysis[7]. Again, visually, one can see that the black and blue dots are somewhat similar below about 192 blocks per IO going out. And then one can make the broad generalization of higher reds, lower blues and blacks in the middle.

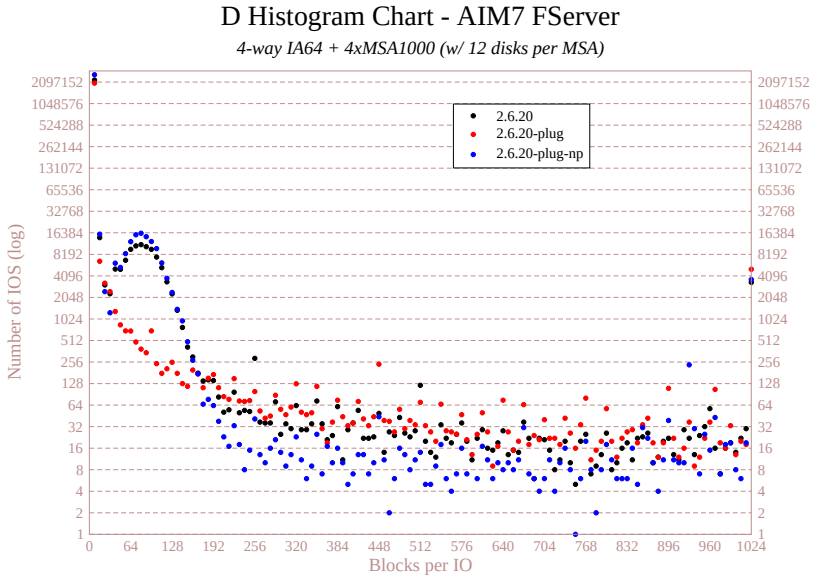

Figure 3: D Histogram

[7]Note the logarithmic nature of the Y axis for this chart.

7 iostat Data File

`btt` attempts to produce the results from running an `iostat -x` command in parallel with the system as it is being traced. The fields (columns) generated by the `--iostat` or `-I` option can be seen from the following output snippet – note that the line has been split to fit on the printed page:

```
Device:       rrqm/s   wrqm/s     r/s     w/s    rsec/s    wsec/s
             rkB/s     wkB/s avgrq-sz avgqu-sz   await   svctm  %util   Stamp
...
(  8, 16)       0.00     0.00    0.00 1005.30      0.00 152806.36
              0.00  76403.18   152.00    31.00    0.00    0.00   0.00   71.79
...
(  8, 16)       1.02     5.80    0.34    1.07      4.03     55.62
              2.02     27.81    42.13     0.61    0.00   21.90   0.00   TOTAL
```

Note that the STAMP field contains the runtime (in seconds) for that line of data.

8 Per-IO Data File

`btt` can produce a text file containing time line data for each IO processed. The time line data contains rudimentary information for the following stages:

- queue traces
- get request traces
- insert traces
- merge traces
- issue traces
- completion traces
- remap traces

The *–per-io-dump* or *-p* option triggers this behavior, and will produce a file containing streams of IOs (separated by blank spaces). As an example, here is a snippet of 4 IOs that were merged together, you will note there are 3 merged IOs, and 1 inserted in the stream. The issue and completion traces are replicated per IO.

```
66,0  :     0.763283556 Q       6208+8
            0.763300157 I       6208+8
            0.763296365 G       6208+8
            0.763338848 D       6208+32
            0.763705760 C       6208+32

66,0  :     0.763314550 Q       6224+8
            0.763315341 M       6224+8
            0.763338848 D       6208+32
            0.763705760 C       6208+32

66,0  :     0.763321010 Q       6232+8
            0.763321775 M       6232+8
            0.763338848 D       6208+32
            0.763705760 C       6208+32

65,240:     0.763244173 Q       6216+8
            0.763244974 M       6216+8
            0.763374288 D       6208+32
            0.763826610 C       6208+32
```

The columns provide the following information:

1. Device major/minor.

2. Time of the trace (seconds from the start of the run)
3. Trace type
4. start block + number of blocks

9 Latency Data Files

The latency data files which can be optionally produced by `btt` provide per-IO latency information, one for total IO time (Q2C) and one for latencies induced by lower layer drivers and devices (D2C).

In both cases, the first column (X values) represent runtime (seconds), while the second column (Y values) shows the actual latency for a command at that time (either Q2C or D2C).

10 Seek Data File

`btt` can also produce two data files containing all IO-to-IO sector deltas, providing seek information which can then be plotted. The produced data file contains 3 sets of data:

1. Combined data – all read and write IOs
2. Read data – just seek deltas for reads
3. Write data – just seek deltas for writes

The format of the output file names is to have the name generated by the following fields separated by underscores (_):

- The prefix provided as the argument to the `-s` option.
- The major and minor numbers of the device separated by a comma.
- The string `q2q` or `d2d`, indicating the Q2Q or D2D seeks, respectively.
- One of the following characters:

  r For read (device to system) IOs

  w For write (system to device) IOs

  c Combined – both read and write IOs

An example name would be after specifying `-s seek` would be: `seek_065,048_q2q_w.dat`.

The format of the data is to have the runtime values (seconds since the start of the run) in column 1 (X values); and the difference in sectors from the previous IO in column 2 (Y values). Here is a snippet of the first few items from a file:

```
# Combined
     0.000034733          35283790.0
     0.000106453          35283790.0
     0.005239009          35283950.0
     0.006968575          35283886.0
     0.007218709          35283694.0
     0.012145393          35283566.0
     0.014980835         -35848914.0
     0.024239323         -35848914.0
     0.024249402         -35848914.0
     0.025707095         -35849072.0
     ...
```

Figure 4 shows a simple graph that can be produced which provides visual details concerning seek patterns.

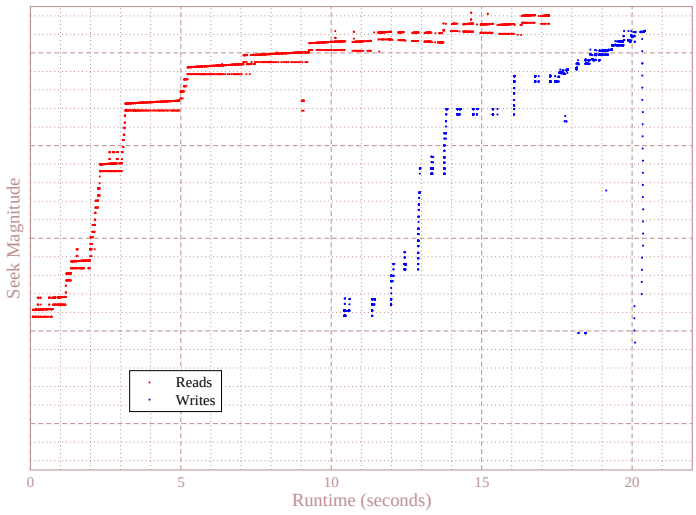

Figure 4: Seek Chart

The seek difference is calculated in one of two ways:

default By default, the seek distance is calculated as the *closest* distance between the previous IO and this IO. The concept of *closeness* means that it could either be the *end* of the previous IO and the beginning of the next, or the end of this IO and the start of the next.

`-a` If the `-a` or `--seek-absolute` option is specified, then the seek distance is simply the difference between the end of the previous IO and the start of this IO.

11 Command Line

```
Usage: btt 2.01
[ -a               | --seek-absolute ]
[ -A               | --all-data ]
[ -B <output name> | --dump-blocknos=<output name> ]
[ -d <seconds>     | --range-delta=<seconds> ]
[ -D <dev;...>     | --devices=<dev;...> ]
[ -e <exe,...>     | --exes=<exe,...>  ]
[ -h               | --help ]
[ -i <input name>  | --input-file=<input name> ]
[ -I <output name> | --iostat=<output name> ]
[ -l <output name> | --d2c-latencies=<output name> ]
[ -M <dev map>     | --dev-maps=<dev map>
[ -o <output name> | --output-file=<output name> ]
[ -p <output name> | --per-io-dump=<output name> ]
[ -q <output name> | --q2c-latencies=<output name> ]
[ -s <output name> | --seeks=<output name> ]
[ -S <interval>    | --iostat-interval=<interval> ]
[ -t <sec>         | --time-start=<sec> ]
[ -T <sec>         | --time-end=<sec> ]
[ -u <output name> | --unplug-hist=<output name> ]
[ -V               | --version ]
[ -v               | --verbose ]
```

11.1 --seek-absolute/-a

When specified on the command line, this directs btt to calculate seek distances based solely upon the ending block address of one IO, and the start of the next. By default `btt` uses the concept of the closeness to either the beginning or end of the previous IO. See section 10 for more details about seek distances.

11.2 --all-data/-A

Normally `btt` will not print out verbose information concerning per-process and per-device data (as outlined in section 3). If you desire that level of detail you can specify this option.

11.3 --dump-blocknos/-B

This option will output absolute block numbers to three files prefixed by the specified output name:

***prefix_device*_r.dat** All read block numbers are output, first column is time (seconds), second is the block number, and the third column is the ending block number.

***prefix_device_*w.dat** All write block numbers are output, first column is time (seconds), second is the block number, and the third column is the ending block number.

***prefix_device_*c.dat** All block numbers (read and write) are output, first column is time (seconds), second is the block number, and the third column is the ending block number.

11.4 `--range-delta/-d`

Section 5 discussed how `btt` outputs a file containing Q and C activity, the notion of *active* traces simply means that there are Q or C traces occurring within a certain period of each other. The default values is 0.1 seconds; with this option allowing one to change that granularity. The smaller the value, the more data points provided.

11.5 `--devices/-D`

Normally, `btt` will produce data for all devices detected in the traces parsed. With this option, one can reduce the analysis to one or more devices provided in the string passed to this option. The device identifiers are the major and minor number of each device, and each device identifier is separated by a colon (:). A valid specifier for devices 8,0 and 8,8 would then be: `"8,0:8,8"`.

11.6 `--exes/-e`

Likewise, `btt` will produce data for all processes (executables) found in the traces. With this option, one can specify which processes you want displayed in the output. The format of the string passed is a list of executable *names* separated by commas (,). An example would be `"-e mkfs.ext3,mount"`.

11.7 `--help/-h`

Prints out the simple help information, as seen at the top of section 11.

11.8 `--input-file/-i`

Specifies the binary input file that `btt` will interpret traces in. See section 2 for information concerning binary trace files.

11.9 `--iostat/-I`

This option triggers `btt` to generate iostat-like output to the file specified. Refer to section 7 for more information on the output produced.

11.10 --d2c-latencies/-l

This option instructs `btt` to generate the D2C latency file discussed in section 9.

11.11 --dev-maps/-M

Internal option, still under construction.

11.12 --output-file/-o

Normally `btt` sends the statistical output (covered in section 3) to standard out, if you specify this option this data is redirected to the file specified.

11.13 --per-io-dump/-p

This option tells `btt` to generate the per IO dump file as discussed in section 8.

11.14 --q2c-latencies/-q

This option instructs `btt` to generate the Q2C latency file discussed in section 9.

11.15 --seeks/-s

This option instructs `btt` to generate the seek data file discussed in section 10.

11.16 --iostat-interval/-S

The normal `iostat` command allows one to specify the snapshot interval, likewise, `btt` allows one to specify how many seconds between its generation of snapshots of the data via this option. Details about the iostat-like capabilities of `btt` may be found in section 7.

11.17 --time-start/-t and --time-end/T

> *This `btt` capability is still under construction, results are not always consistent at this point in time.*

These options allow one to dictate to `btt` when to start and stop parsing of trace data in terms of seconds since the start of the run. The trace chosen will be between the start time (or 0.0 if not specified) and end time (or the end of the run) specified.

11.18 --unplug-hist/-u

This option instructs `btt` to generate a data file containing histogram information for *unplug* traces on a per device basis. It shows how many times an unplug was hit with a specified number of IOs released. There are 21 output values into the file, as follows:

X value	Representing Counts
0	0...4
1	5...9
2	10...14
...	.....
19	95...99
20	100+

The file name(s) generated use the text string passed as an argument for the prefix, followed by the device identifier in `major,minor` form, with a `.dat` extension (as an example, with `-u up_hist` specified on the command line: `up_hist_008,032.dat`.

11.19 --version/-V

Prints out the `btt` version, and exits.

11.20 --verbose/-v

While `btt` is processing data, it will put out periodic (1-second granularity) values describing the progress it is making through the input trace stream. The value describes how many traces have been processed. At the end of the run, the overall number of traces, trace rate (number of thousands of traces per second), and the real time for trace processing and output are displayed. Example (note: the interim trace counts are put out with carriage returns, hence, they overwrite each time):

```
# btt -i bp.bin -o btt -v
Sending range data to bttX.dat
Sending stats data to bttX.avg
 287857 t
1414173 t
1691581 t
...
4581291 traces @ 279.7 Ktps
16.379036+0.000005=16.379041
```

12 bno_plot.py

Included with the distribution is a simple 3D plotting utility based upon the block numbers output when `-B` is specified (see section 11.3 for more details about the `-B option`). The display will display *each* IO generated, with the time (seconds) along the X-axis, the block number (start) along the Y-axis and the number of blocks transferred in the IO represented along the Z-axis.

The script requires Python[8] and gnuplot[9], and will enter interactive mode after the image is produced. In this interactive mode one can enter gnuplot commands at the `'gnuplot>'` prompt, and/or can change the viewpoint within the 3D image by *left-click-hold* and moving the mouse. A sample screen shot can be seen in figure 5 on page 27.

`bno_plot.py` Command Line Options

```
$ bno_plot.py --help

bno_plot.py
[ -h | --help       ]
[ -K | --keys-below ]
[ -v | --verbose    ]
[ <file...>         ]

Utilizes gnuplot to generate a 3D plot of the block number
output from btt.  If no <files> are specified, it will
utilize all files generated after btt was run with -B
blknos (meaning: all files of the form blknos*[rw].dat).

The -K option forces bno_plot.py to put the keys below the
graph, typically all keys for input files are put in the
upper right corner of the graph. If the number of devices
exceed 10, then bno_plot.py will automatically push the
keys under the graph.

To exit the plotter, enter 'quit' or ^D at the 'gnuplot> '
prompt.
```

[8] `www.python.org`
[9] `www.gnuplot.info`

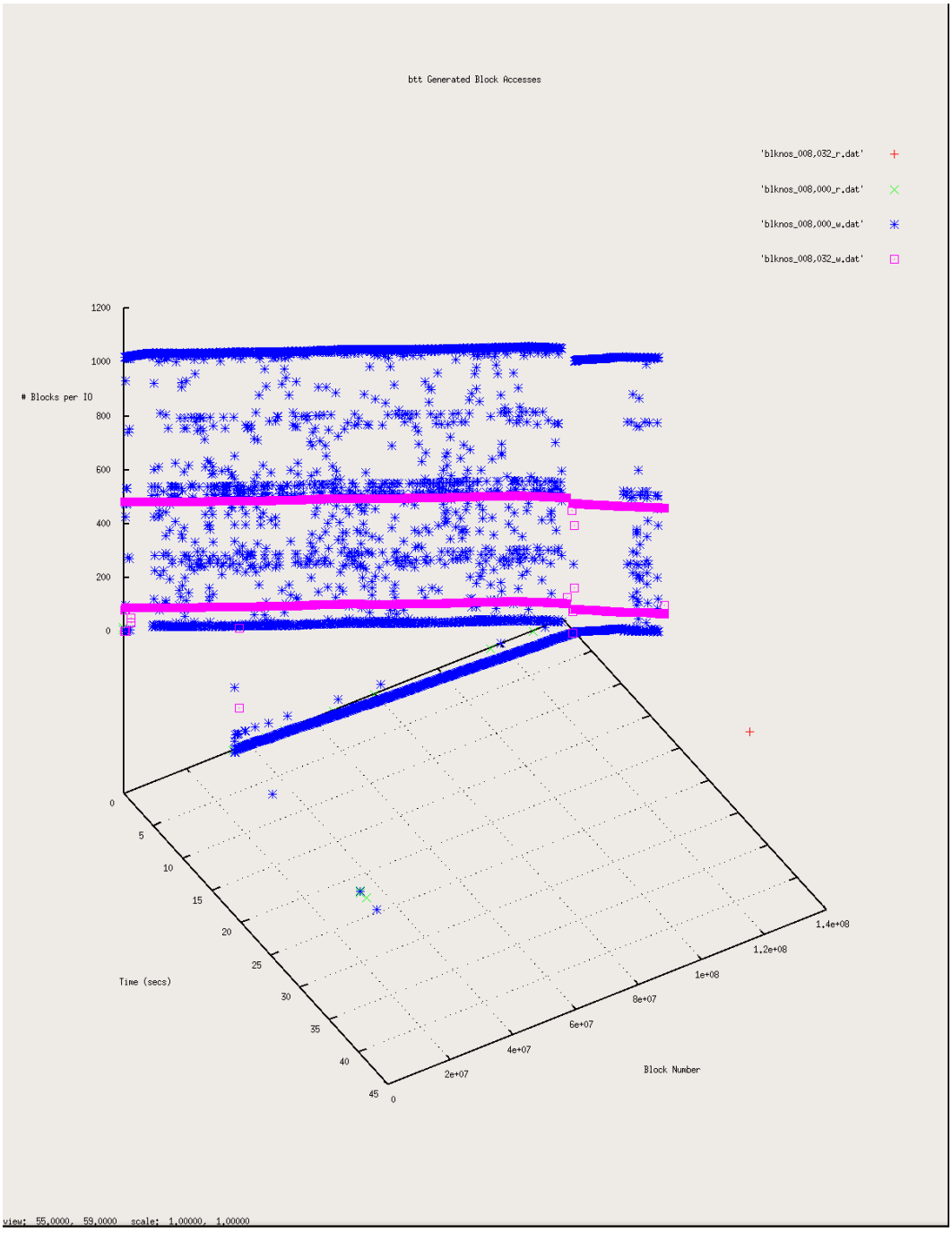

Figure 5: Sample `bno_plot.py` Screen Shot

13 Sample `btt` Output

Here is a complete output file from a btt run, illustrating a lot of the capabilities of btt.

```
==================== All Devices ====================

            ALL           MIN           AVG           MAX           N
--------------- ------------- ------------- ------------- -----------
Q2Qdm             0.000001260   0.000078915  14.504199709      491531
Q2Adm             0.000000398   0.001184212  43.228889856      491566

Q2Q               0.000000455   0.000009032   1.811181609      491566
Q2G               0.000000466   0.002396857   0.203392940        8756
G2I               0.000000142   0.000000461   0.000096257        8756
Q2M               0.000000230   0.000000433   0.000173083      482810
I2D               0.000000999   0.407286973   0.982146009        8756
M2D               0.000001251   0.356512650   0.982143260      482810
D2C               0.000057729   0.037337508   2.011319687      491566
Q2C               0.000060550   0.394797701   2.035625308      491566

==================== Device Overhead ====================

       DEV |       Q2G       G2I       Q2M       I2D       D2C
---------- | --------- --------- --------- --------- ---------
 (  8,  0) |   0.3104%   0.0004%   0.0003%  58.9079%  41.0914%
 (  8, 32) |   0.5858%   0.0001%   0.0001%  98.3852%   1.6146%
---------- | --------- --------- --------- --------- ---------
   Overall |   0.0957%   0.0000%   0.0010%  16.2534%  83.6500%
```

```
==================== Device Merge Information ====================

       DEV |       #Q       #D   Ratio |   BLKmin   BLKavg   BLKmax    Total
---------- | -------- -------- ------- | -------- -------- -------- --------
 (  8,  0) |   246812     3970    62.2 |        2      497     1024  1974490
 (  8, 32) |   244754     4786    51.1 |        8      409      488  1958032
---------- | -------- -------- ------- | -------- -------- -------- --------
       DEV |       #Q       #D   Ratio |   BLKmin   BLKavg   BLKmax    Total
     TOTAL |   491566     8756    56.1 |        2      449     1024  3932522

==================== Device Q2Q Seek Information ====================

       DEV |          NSEEKS            MEAN          MEDIAN | MODE
---------- | --------------- --------------- --------------- | ---------------
 (  9,  0) |          491532          2693.5               0 | 0(489746)
 (  8,  0) |          246812          2682.4               0 | 0(245021)
 (  8, 32) |          244754          1355.3               0 | 0(243795)
---------- | --------------- --------------- --------------- | ---------------
   Overall |          NSEEKS            MEAN          MEDIAN | MODE
   Average |          983098          2357.6               0 | 0(978562)

==================== Device D2D Seek Information ====================

       DEV |          NSEEKS            MEAN          MEDIAN | MODE
---------- | --------------- --------------- --------------- | ---------------
 (  8,  0) |            3970        219326.9               0 | 0(2177)
 (  8, 32) |            4786         69254.5               0 | 0(3828)
---------- | --------------- --------------- --------------- | ---------------
   Overall |          NSEEKS            MEAN          MEDIAN | MODE
   Average |            8756        137297.8               0 | 0(6005)
```

```
==================== Plug Information ====================

       DEV |    # Plugs # Timer Us  | % Time Q Plugged
---------- | ---------- ----------  | ----------------
 (  8,  0) |       1152(       137) |   0.107460846%
 (  8, 32) |          5(         0) |   0.000175916%
---------- | ---------- ----------  | ----------------
   Overall |    # Plugs # Timer Us  | % Time Q Plugged
   Average |        578(        68) |   0.053818381%

==================== Active Requests At Q Information ====================

       DEV |  Avg Reqs @ Q
---------- | -------------
 (  8,  0) |          11.1
 (  8, 32) |         133.0
---------- | -------------
   Overall | Avgs Reqs @ Q
   Average |          71.8

==================== Q2D Histogram ====================

       DEV | <.005 <.010 <.025 <.050 <.075 <.100 <.250 <.500 < 1.0 >=1.0
 --------- | ===== ===== ===== ===== ===== ===== ===== ===== ===== =====
 (  8,  0) |  80.5   3.5   1.0   1.0   0.7   0.6   4.0   5.3   3.4   0.0
 (  8, 32) |   0.3   0.0   0.3   0.2   0.2   0.2   1.7   2.0  95.2   0.0
========== | ===== ===== ===== ===== ===== ===== ===== ===== ===== =====
       AVG |  40.5   1.8   0.6   0.6   0.4   0.4   2.8   3.7  49.1   0.0
```