`btrecord` and `btreplay` User Guide

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

February 24, 2009

Abstract

The `btrecord` and `btreplay` tools provide the ability to record and replay IOs captured by the `blktrace` utility. Attempts are made to maintain ordering, CPU mappings and time-separation of IOs. The general workflow is expected to be:

1. Initiate `blktrace` to capture traces
2. Generate traces...
3. Stop `blktrace`
4. Run `btrecord` to convert traces into IO records
5. Utilize `btreplay` to replay IOs

This document will discuss the operating characteristics of `btreplay` and provide detailed command line option descriptions.

Contents

1 Introduction

The `btrecord` and `btreplay` tools provide the ability to record and replay IOs captured by the `blktrace` utility. Attempts are made to maintain ordering, CPU mappings and time-separation of IOs. The general workflow is expected to be:

1. Initiate `blktrace` to capture traces
2. Generate traces...
3. Stop `blktrace`
4. Run `btrecord` to convert traces into IO records
5. Utilize `btreplay` to replay IOs

This document will discuss the operating characteristics of `btreplay` and provide detailed command line option descriptions.

This document presents the command line overview for `btrecord` and `btreplay`, and shows some commonly used example usages of it in everyday work here at OSLO's Scalability and Performance Group.

Build Note

To build these tools, one needs to place the source directory next to a valid `blktrace`[1] directory, as it includes `../blktrace` in the `Makefile`.

[1] `git://git.kernel.dk/blktrace.git`

2 `btrecord` and `btreplay` Operating Model

The `blktrace` utility provides the ability to collect detailed traces from the kernel for each IO processed by the block IO layer. The traces provide a complete timeline for each IO processed, including detailed information concerning when an IO was first received by the block IO layer – indicating the device, CPU number, time stamp, IO direction, sector number and IO size (number of sectors). Using this information, one is able to *replay* the IO again on the same machine or another set up entirely.

2.1 Basic Workflow

The basic operating work-flow to replay IOs would be something like:

1. Run `blktrace` to collect traces. Here you specify the device or devices that you wish to trace and later replay IOs upon. Note: the only traces you are interested in are *QUEUE* requests – thus, to save system resources (including storage for traces), one could specify the `-a queue` command line option to `blktrace`.

2. While `blktrace` is running, you run the workload that you are interested in.

3. When the work load has completed, you stop the `blktrace` utility (thus saving all traces over the complete workload).

4. You extract the pertinent IO information from the traces saved by `blktrace` using the `btrecord` utility. This will parse each trace file created by `blktrace`, and craft IO descriptions to be used in the next phase of the workload processing.

5. Once `btrecord` has successfully created a series of data files to be processed, you can run the `btreplay` utility which attempts to generate the same IOs seen during the sample workload phase.

2.2 IO Stream Replay Characteristics

The major characteristics of the IO stream that are kept intact include:

Device The IOs are replayed on the same device as was seen during the sample workload.

IO direction The same IO direction (read/write) is maintained.

IO offset The same device offset is maintained.

IO size The same number of sectors are transferred.

Time differential The time stamps stored during the `blktrace` run are used to determine the amount of time between IOs during the sample workload. `btreplay` *attempts* to maintain the same time differential between IOs, but no guarantees as to complete accuracy are provided by the utility.

Device IO Stream Ordering All IOs on a device are submitted in the precise order they were seen during the sample workload run.

As noted above, the time between IOs may not be accurately maintained during replays. In addition the actual ordering of IOs *between* devices is not necessarily maintained. (Each device with an IO stream maintains its own concept of time, and thus there may be slippage of the time kept between managing threads.)

> We have prototyped a different approach, wherein a single managing thread handles all IOs across all devices. This approach, while guaranteeing correct ordering of IOs across all devices, resulted in much worse timing on a per IO basis.

2.3 `btrecord/btreplay` Method of Operation

As noted above, `btrecord` extracts `QUEUE` operations from `blktrace` output. These `QUEUE` operations indicate the entrance of IOs into the block IO layer. In order to replay these IOs with some accuracy in regards to ordering and timeliness, we decided to take multiple sequential (in time) IOs and put them in a single *bunch* of IOs that will be processed as a single *asynchronous IO* call to the kernel[2]. To manage the size of the *bunches*, the `btrecord` utility provides you with two controlling knobs:

`--max-bunch-time` This is the amount of time to encompass in one bunch – only IOs within the time specified are eligible for *bunching*. The default time is 10 milliseconds (10,000,000 nanoseconds). Refer to section 3.1.6 on page 9 for more information.

`--max-pkts` A *bunch* size can be anywhere from 1 to 512 packets in size and by default we max a bunch to contain no more than 8 individual IOs. With this option, one can increase or decrease the maximum *bunch* size. Refer to section 3.1.7 on page 9 for more information.

Each input data file (one per device per CPU) results in a new record data file (again, one per device per CPU) which contains information about *bunches* of IOs to be replayed. `btreplay` operates on these record data files by spawning a new pair of threads per file. One thread manages the submitting of AIOs per

[2]Attempts to do them individually resulted in too large of a turnaround time penalty (user-space to kernel and back). Note that in a number of workloads, the IOs are coming in from the page cache handling code, and thus are submitted to the block IO layer with *very small* time intervals between issues.

bunch in the record data file, while the other thread manages reclaiming AIOs completed[3].

Each submitting thread simply reads the input file of *bunches* recorded by `btrecord`, and attempts to faithfully reproduce the ordering and timing of IOs seen during the sample workload. The reclaiming thread simply waits for AIO completions, freeing up resources for the submitting thread to utilize to submit new AIOs.

The number of CPUs being used on the replay system can be different from the number on the recorded system. To help with mappings here the `--cpus` option allows one to state how many CPUs on the replay system to utilize. If the number of CPUs on the replay system is less than on the recording system, we wrap CPU IDs. This *may* result in an overload of CPU processing capabilities on the replay system. (Refer to section 3.2.1 on page 11 for more details about the `--cpus` option.)

[3]We have found that having the same thread do both results in a further reduction in replay timing accuracy.

2.4 Known Deficiencies and Proposed Possible Fixes

The overall known deficiencies with this current set of utilities is outlined here, in some cases ideas on additions and/or improvements are included as well.

1. Lack of IO ordering across devices.

   We could institute the notion of global time across threads, and thus ensure IO ordering across devices, with some reduction in timing accuracy.

2. Lack of IO timing accuracy – additional time between IO bunches.

   This is the primary problem with any IO replay mechanism – how to guarantee per-IO timing accuracy with respect to other replayed IOs? One idea to reduce errors in this area would be to push the IO replay into the kernel, where you may *receive more responsive timings.*

3. Bunching of IOs results in reduced time amongst IOs within a bunch.

   The user has some *control over this (via the* `--max-pkts` *option). One* could *simply specify* `-max-pkts=1` *and then each IO would be treated individually. Of course, this would probably then run into the problem of excessive inter-IO times.*

4. 1-to-1 mapping of devices – for now the devices on the replay machine must be the same as on the recording machine.

   It should be relatively trivial to add in the notion of mapping – simply include a file that is read which maps devices on one machine to devices (with offsets and sizes) on the replay machine[4]*.*

   One could also add in the notion of CPU mappings as well – device D_{rec} managed by CPU C_{rec} on the recorded system shall be replayed on device D_{rep} and CPU C_{rep} on the replay machine.

   With version 0.9.1 we now support the `-M` option to do this – see section 3.2.8 on page 12 for more information on device mapping.

[4] The notion of an offset and device size to replay on could be used to both allow for a single device to masquerade as more than one device, and could be utilized in case the replay device is smaller than the recorded device.

3 Command Line Options

3.1 `btrecord` Command Line Options

```
Usage: btrecord -- version 0.9.3

[ -d <dir>  : --input-directory=<dir> ] Default: .
[ -D <dir>  : --output-directory=<dir>] Default: .
[ -F        : --find-traces           ] Default: Off
[ -h        : --help                  ] Default: Off
[ -m <nsec> : --max-bunch-time=<nsec> ] Default: 10 msec
[ -M <pkts> : --max-pkts=<pkts>       ] Default: 8
[ -o <base> : --output-base=<base>    ] Default: replay
[ -v        : --verbose               ] Default: Off
[ -V        : --version               ] Default: Off
<dev>...                                Default: None
```

Figure 1: `btrecord --help` Output

3.1.1 `-d` or `--input-directory` Set Input Directory

The `-d` option requires a single parameter providing the directory name for where input files are to be found. The default directory is the current directory (`.`).

3.1.2 `-D` or `--output-directory` Set Output Directory

The `-D` option requires a single parameter providing the directory name for where output files are to be placed. The default directory is the current directory (`.`).

3.1.3 `-F` or `--find-traces` Find Trace Files Automatically

The `-F` option instructs `btrecord` to go find all the trace files in the directory specified (either via the `-d` option, or in the default directory '.').

3.1.4 `-h` or `--help`
Display Help Message

3.1.5 `-V` or `--version`
Display `btrecord`Version

The `-h` option displays the command line options and defaults, as presented in figure 1 on page 8.

The `-V` option displays the `btreplay` version, as shown here:

```
$ btrecord --version
btrecord -- version 0.9.0
```

Both commands exit immediately after processing the option.

3.1.6 `-m` or `--max-bunch-time`
Set Maximum Time Per Bunch

The `-m` option requires a single parameter which specifies an amount of time (in nanoseconds) to include in any one bunch of IOs that are to be processed. The smaller the value, the smaller the number of IOs processed at one time – perhaps yielding in more realistic replay. However, after a certain point the amount of overhead per bunch may result in additional real replay time, thus yielding less accurate replay times.

The default value is 10,000,000 nanoseconds (10 milliseconds).

3.1.7 `-M` or `--max-pkts`
Set Maximum Packets Per Bunch

The `-M` option requires a single parameter which specifies the maximum number of IOs to store in a single bunch. As with the `-m` option (section 3.1.6), smaller values *may* or *may not* yield more accurate replay times.

The default value is 8, with a maximum value of up to 512 being supported.

3.1.8 `-o` or `--output-base`
Set Base Name for Output Files

Each output file has 3 fields:

1. Device identifier (taken directly from the device name of the `blktrace` output file).
2. `btrecord` base name – by default "replay".
3. And the CPU number (again, taken directly from the `blktrace` output file name).

This option requires a single parameter that will override the default name (replay), and replace it with the specified value.

3.1.9 `-v` or `--verbose` Select Verbose Output

This option will output some simple statistics at the end of a successful run. Figure 2 (page 10) shows an example of some output, while figure 3 (page 10) shows what the fields mean.

```
sdab:0: 580661 pkts (tot), 126030 pkts (replay), 89809 bunches, 1.4 pkts/bunch
sdab:1: 2559775 pkts (tot), 430172 pkts (replay), 293029 bunches, 1.5 pkts/bunch
sdab:2: 653559 pkts (tot), 136522 pkts (replay), 102288 bunches, 1.3 pkts/bunch
sdab:3: 474773 pkts (tot), 117849 pkts (replay), 69572 bunches, 1.7 pkts/bunch
```

Figure 2: Verbose Output Example

Field 1 The first field contains the device name and CPU identifier. Thus: `sdab:0:` means the device `sdab` and traces on CPU 0.

Field 2 The second field contains the total number of packets processed for each device file.

Field 3 The next field shows the number of packets eligible for replay.

Field 4 The fourth field contains the total number of IO bunches.

Field 5 The last field shows the average number of IOs per bunch recorded.

Figure 3: Verbose Field Definitions

3.2 `btreplay` Command Line Options

```
Usage: btreplay -- version 0.9.3

[ -c <cpus> : --cpus=<cpus>           ] Default: 1
[ -d <dir>  : --input-directory=<dir> ] Default: .
[ -F        : --find-records          ] Default: Off
[ -h        : --help                  ] Default: Off
[ -i <base> : --input-base=<base>     ] Default: replay
[ -I <iters>: --iterations=<iters>    ] Default: 1
[ -M <file> : --map-devs=<file>       ] Default: None
[ -N        : --no-stalls             ] Default: Off
[ -x <int>  : --acc-factor=<int>      ] Default: 1
[ -v        : --verbose               ] Default: Off
[ -V        : --version               ] Default: Off
[ -W        : --write-enable          ] Default: Off
<dev...>                                Default: None
```

Figure 4: `btreplay --help` Output

3.2.1 `-c` or `--cpus`
Set Number of CPUs to Use

3.2.2 `-d` or `--input-directory`
Set Input Directory

The `-d` option requires a single parameter providing the directory name for where input files are to be found. The default directory is the current directory (`.`).

3.2.3 `-F` or `--find-records`
Find RecordFiles Automatically

The `-F` option instructs `btreplay` to go find all the record files in the directory specified (either via the `-d` option, or in the default directory '.').

3.2.4 `-h` or `--help`
Display Help Message

3.2.5 `-V` or `--version`
Display `btreplay`Version

The `-h` option displays the command line options and defaults, as presented in figure 4 on page 11.

The `-V` option displays the `btreplay` version, as show here:

```
$ btreplay --version
btreplay -- version 0.9.0
```

Both commands exit immediately after processing the option.

3.2.6 `-i` or `--input-base` Set Base Name for Input Files

Each input file has 3 fields:

1. Device identifier (taken directly from the device name of the `blktrace` output file).
2. `btrecord` base name – by default "replay".
3. And the CPU number (again, taken directly from the `blktrace` output file name).

This option requires a single parameter that will override the default name (replay), and replace it with the specified value.

3.2.7 `-I` or `--iterations` Set Number of Iterations to Run

This option requires a single parameter which specifies the number of times to run through the input files. The default value is 1.

3.2.8 `-M` or `map-devs` Specify Device Mappings

This option requires a single parameter which specifies the name of a file containing device mappings. The file must be very simply managed, with just two pieces of data per line:

1. The device name on the recorded system (with the `'/dev/'` removed). Example: `/dev/sda` would just be `sda`.
2. The device name on the replay system to use (again, without the `'/dev/'` path prepended).

An example file for when one would map devices `/dev/sda` and `/dev/sdb` on the recorded system to `dev/sdg` and `sdh` on the replay system would be:

```
sda sdg
sdb sdh
```

The only entries in the file that are allowed are these two element lines – we do not (yet?) support the notion of blank lines, or comment lines, or the like.

The utility *does* allow for multiple `-M` options to be supplied on the command line.

3.2.9 `-N` or `--no-stalls`
Disable Pre-bunch Stalls

When specified on the command line, all pre-bunch stall indicators will be ignored. IOs will be replayed without inter-bunch delays.

3.2.10 `-x` or `--acc-factor`
Acceleration Factor

While the `--no-stalls` option allows the traces to be replayed with no waiting time, this option specifies some acceleration factor to be used. If the value of two is used, then the stall time is divided by half resulting in a reduction of the execution time by this factor. Note that if this number is too high, the results will be equivalent of not having stall.

3.2.11 `-v` or `--verbose`
Select Verbose Output

When specified on the command line, this option instructs `btreplay` to store information concerning each *stall* and IO operation performed by `btreplay`. The name of each file so created will be the input file name used with an extension of `.rep` appended onto it. Thus, an input file of the name `sdab.replay.3` would generate a verbose output file with the name `sdab.replay.3.rep` in the directory specified for input files.

In addition, `btreplay` will also output to `stderr` the names of the input files being processed.

3.2.12 `-W` or `--write-enable`
Enable Writing During Replay

As a precautionary measure, by default `btreplay` will *not* process *write* requests. In order to enable `btreplay` to actually *write* to devices one must explicitly specify the `-W` option.