Linaro CONNECT
ASIA 2013 (LCA13)

LEG – Hadoop DFS Performance

Steve Capper <steve.capper@linaro.org>

Hadoop Performance on ARM

- I concentrated on reducing the time taken for CPU bound tasks (the latency).
- My work, so far, has focussed on the underlying cluster filesystem, HDFS, as this underpins a lot of Hadoop workloads.
- The latest release-tagged Hadoop at the time of experimentation was 2.0.2-alpha. So this was the version used. (2.0.3-alpha has just come out).
- Hadoop installations consist of rather a lot of moving parts.
- So please let me know if I should be concentrating on something else! :-)

Hadoop Distributed Filesystem – Overview

- HDFS is the default Hadoop distributed filesystem.
- Consists of multiple “nodes”:
  - **Namenode** – holds the metadata for the filesystem and keeps track of the datanodes.
    - Only one namenode is active at a time per **namespace**.
    - Thus there is a high memory requirement for a namenode.
  - **Datanodes** – store the file's blocks.
    - The default block size is 64MB.
  - (Optional) **Passive namenodes** – To maintain an High Availability configuration, one can set up shared storage (via NFS or by journalnodes) and have namenodes on standby to failover.
- Filesystem blocks are replicated between datanodes.
  - Datanodes can be “rack aware”; data will be distributed between racks.
- Nodes can run on the same machine or on different machines.

Data Integrity in HDFS

- Hadoop mitigates against hardware failure in the following ways:
  - Metadata can be saved to multiple filesystems and regular snapshots are usually taken to allow for disaster recovery.
    - On some HA configurations, multiple journalnodes maintain a quorum of the metadata.
  - Data blocks are replicated across multiple datanodes (preferably to different racks).
- Data blocks are regularly transmitted:
  - All data streams are checksummed.
  - The default checksum algorithm is CRC32c.
  - The default number of bytes per checksum is 512 bytes.

Test Configuration

- So far I have been micro-benchmarking single machine workloads (1 name node, 1 data node, 1 workload).
- I have been working on an A9 platform.
- TestDFSIO `-read` and `-write` have been tested with a 10GB filesize.
- Soft-float Oracle JDK 1.7.0_10 was used.
- Performance was measured using Linux `perf -a`, Java samples were taken with the inbuilt profiler:

  `-Xrunhprof:cpu=samples,depth=10`

Plain TestDFSIO write tests – Samples

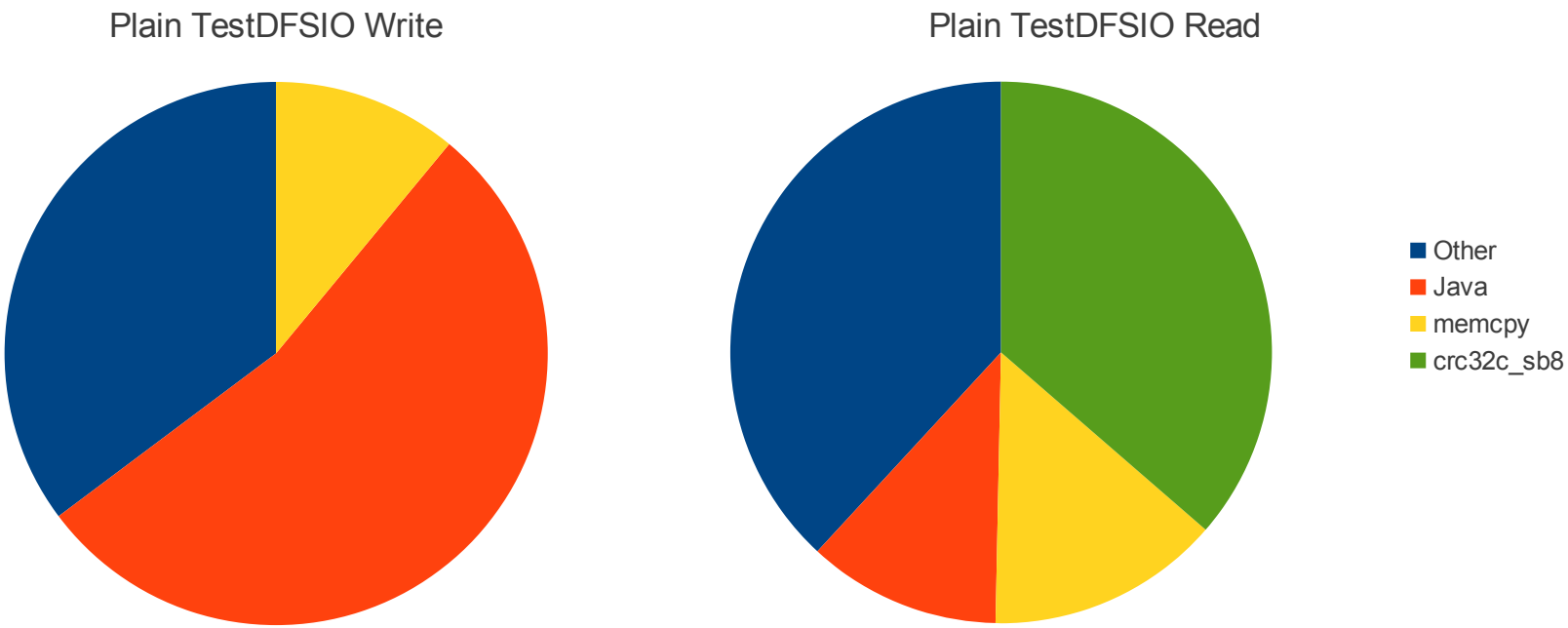
- Namenode Java samples:

```
CPU SAMPLES BEGIN (total = 164879)
rank   self  accum   count trace method
   1 24.85% 24.85%   40970 301219 sun.nio.ch.EPollArrayWrapper.epollWait
   2 24.79% 49.64%   40870 301362 sun.nio.ch.EPollArrayWrapper.epollWait
   3 24.79% 74.42%   40866 301363 sun.nio.ch.EPollArrayWrapper.epollWait
   4 24.78% 99.20%   40859 301358 sun.nio.ch.EPollArrayWrapper.epollWait
   5  0.07% 99.27%     118 300073 java.lang.ClassLoader.defineClass1
   6  0.03% 99.31%      57 300020 java.util.zip.ZipFile.open
   7  0.03% 99.34%      55 300179 java.util.zip.ZipFile.getEntry
   8  0.03% 99.38%      54 300062 java.util.zip.ZipFile.read
   9  0.02% 99.39%      31 300076 java.util.zip.Inflater.inflateBytes
  10  0.01% 99.41%      23 301165 sun.nio.ch.FileDispatcherImpl.force0
```

- Linux perf results:

```
+   33.80%            java  perf-13641.map          [.] 0xb4258538
+   19.98%            java  perf-13534.map          [.] 0xb403e980
+    5.37%            java  libc-2.15.so            [.] memcpy
+    3.02%            java  [kernel.kallsyms]       [k] __copy_from_user
+    2.60%            java  [kernel.kallsyms]       [k] __copy_to_user_std
+    1.86%            java  [kernel.kallsyms]       [k] _raw_spin_unlock_irqrestore
+    1.01%         swapper  [kernel.kallsyms]       [k] finish_task_switch
+    0.66%         swapper  [kernel.kallsyms]       [k] tick_nohz_idle_enter
+    0.55%            java  [kernel.kallsyms]       [k] get_page_from_freelist
+    0.44%            java  [kernel.kallsyms]       [k] free_hot_cold_page
+    0.42%         swapper  [kernel.kallsyms]       [k] default_idle
```

Plain TestDFSIO read & write CPU usage



Some analysis of plain TestDFSIO

- The namenode doesn't do much when it's in charge of 1 datanode. (this **will** change when things scale up).
- When writing data:
  - Most of the Java time is spent in **PureJavaCrc32C.update**
  - Most of the CPU time is spent in Java land.
- When reading data:
  - Most of the Java time is spent in **NativeCrc32.nativeVerifyChunkedSums**.
  - Most of the CPU time is in the native function: **crc32c_sb8**
- For both reading and writing data:
  - ~10% of the CPU time is spent copying memory.

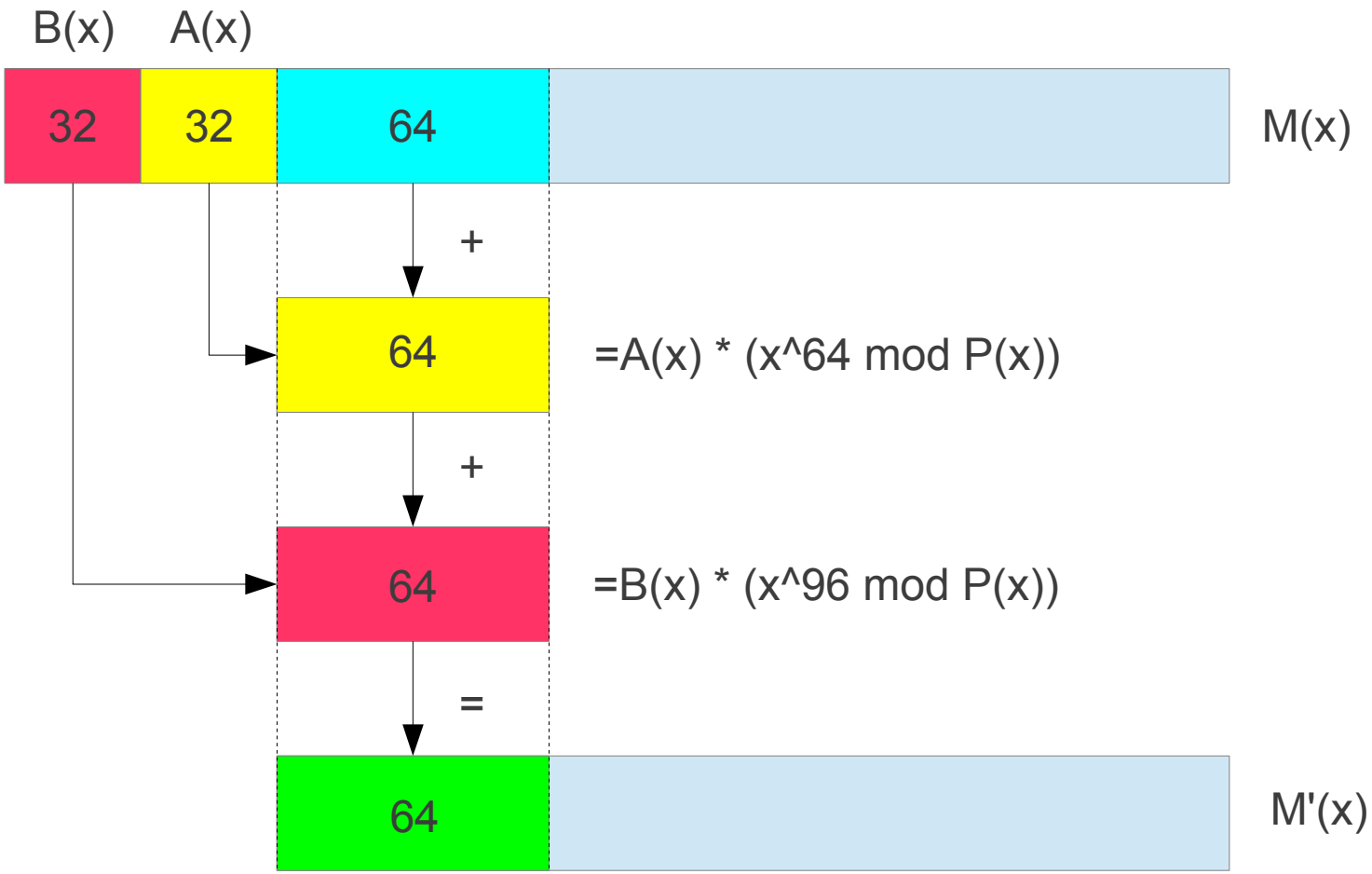
Optimising the read and write paths

- Trevor Robinson has proposed a patch to improve the write path:
  - HDFS-3529 “Use direct buffers for data in write path”
- I have been working on speeding up the computation of CRC32c checksums, by using NEON.
- Also, I've performed some very preliminary experiments on replacing PureJavaCrc32C with a JNI class that references the NEON CRC code.

NEON optimisation of the CRC32c algorithm

- I worked on an algorithm Q4 last year:
  - https://wiki.linaro.org/LEG/Engineering/CRC
- Given an input buffer, we perform polynomial multiplication and addition to give a slighter smaller buffer with the same CRC. This is referred to as “folding”.
- The algorithm reduced the buffer by 32 bytes at a time.
- The final buffer of 32 bytes was then processed by slice-by-8 as normal.
- I found that the vast majority of CRCs in Hadoop (in both **NativeCrc32.nativeVerifyChunkedSums** and **PureJavaCrc32C.update**) were computed for 512 byte buffers.

A single 64 bit fold



CRC(M'(x)) = CRC(M(x))

NEON implementations

- With ARMv7 NEON we can only perform polynomial multiplication on 8 bit lanes:
  - We need to be able to multiply at least 32 bits, thus multiple vmull.p8's were chained together to achieve this).
  - There are 16 vmull.p8's per fold, so a little register pressure!
- A gcc intrinsic version has been coded up:
  - This is considerably simpler to look at.
  - Unfortunately, it runs a little slower than hand-optimised assembler.
  - A test case has been sent to the Linaro Toolchain Working group (a couple of weeks ago) and this is being analysed by them.

Replacing PureJavaCrc32C

- PureJavaCrc32C has two noteworthy methods:
  - public void update(byte[] b, int off, int len) – called mostly for lengths of 512.
  - public void update(int b) – seldom called.
- I created a new class implementing the Checksum interface that:
  - Used the same implementation of: update(int b) as PureJavaCrc32C.
  - Called straight into JNI for update(byte[] b, int off, int len).
- The class name NativeCrc32 was already taken by something else, so I chose a rather silly temporary name: HyperCrc32C

Dealing with byte [] in JNI

- We are only reading from the byte[] array, and only for a very short time.
- Thus I pinned the buffer in memory with: **GetPrimitiveArrayCritical**
- Then subsequently released the buffer with **ReleasePrimitiveArrayCritical(..., JNI_ABORT)**
- This worked for me, but perhaps a better long term solution would be to change the backing data type to a ByteBuffer?
  - We could also be clever and change the alignment of these?

- Rather than test every optimisation individually in this talk; I am going to put them all together.

TestDFSIO read & write CPU usage

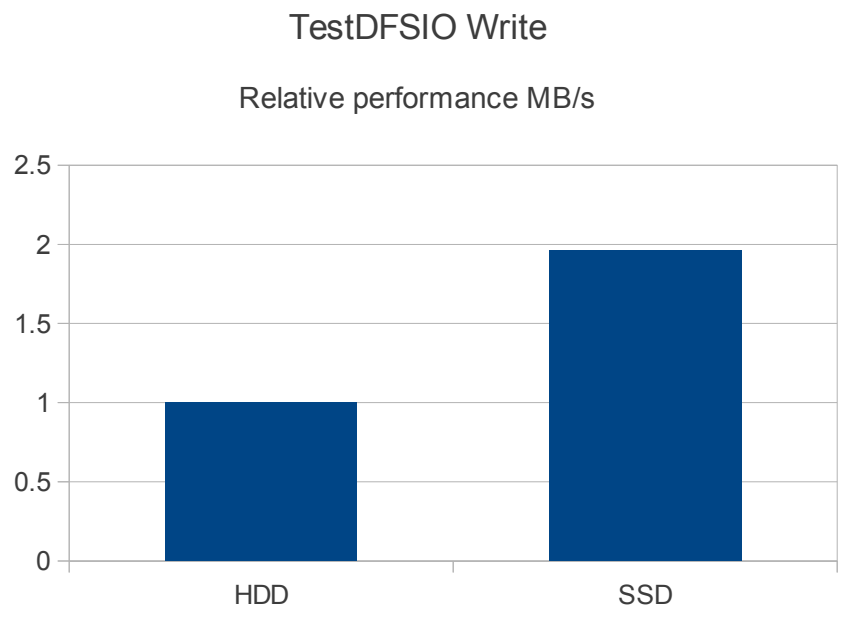
Some Analysis of the New TestDFSIO runs

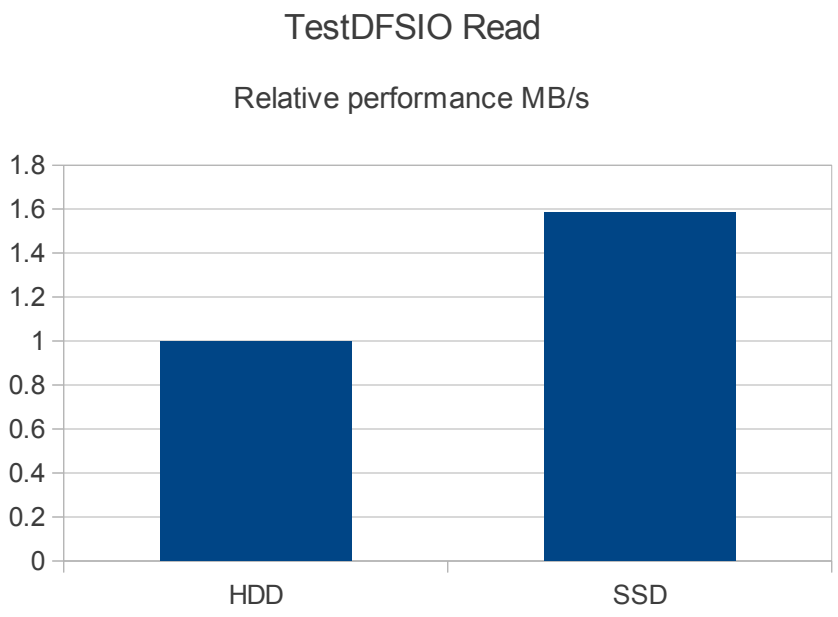
- The name node samples are unchanged.
- For the write path:
  - Most of the time is now spent running native code rather than Java code.
  - There is a noticeable reduction in Hadoop user CPU usage.
- For the read path:
  - Most of the time is still spent running native code.
  - There is again a reduction in Hadoop user CPU usage.
- For both the read and write paths there is a significant amount of CPU time spent copying memory around.

Conclusions and Further Work

- The CPU usage required for TestDFSIO runs has been reduced for both read/write paths.
  - This gives us more CPU cycles to run Hadoop jobs with!
- Hadoop is known to be very sensitive to underlying disk IO.
  - To optimise HDFS IO, it would make sense to optimise disk/filesystem IO as much as possible and re-run these benchmarks.
- As Hadoop runs under Java:
  - It makes sense to keep track of JVMs.
  - A beta hard-float JVM has been released.
- The CPU usage for memcpy is making me uneasy!

We should also consider our hardware...





Plain TestDFSIO write tests – Some Samples

- Workload Java samples:

```
CPU SAMPLES BEGIN (total = 81097)
rank   self  accum   count trace method
   1 36.88% 36.88%   29906 300854 sun.nio.ch.EPollArrayWrapper.epollWait
   2 32.95% 69.83%   26723 301107 sun.nio.ch.EPollArrayWrapper.epollWait
   3 16.48% 86.31%   13363 301109 org.apache.hadoop.util.PureJavaCrc32C.update
   4  2.74% 89.05%    2225 301113 sun.misc.Unsafe.copyMemory
   5  1.67% 90.72%    1357 301115 sun.nio.ch.FileDispatcherImpl.write0
   6  1.09% 91.81%     884 301128 org.apache.hadoop.hdfs.DFSOutputStream.writeChunk
   7  0.70% 92.51%     564 301110 sun.nio.ch.FileDispatcherImpl.read0
   8  0.42% 92.93%     342 301123 org.apache.hadoop.hdfs.DFSOutputStream.writeChunk
   9  0.40% 93.33%     326 301202
org.apache.hadoop.hdfs.DFSOutputStream.waitAndQueueCurrentPacket
  10  0.33% 93.66%     266 301124 java.lang.System.arraycopy
```

- Datanode Java samples:

```
CPU SAMPLES BEGIN (total = 227245)
rank   self  accum   count trace method
   1 16.85% 16.85%   38293 301110 sun.nio.ch.EPollArrayWrapper.epollWait
   2 16.82% 33.67%   38228 301313 sun.nio.ch.EPollArrayWrapper.epollWait
   3 16.80% 50.48%   38184 301374 sun.nio.ch.EPollArrayWrapper.epollWait
   4 16.80% 67.28%   38184 301375 sun.nio.ch.EPollArrayWrapper.epollWait
   5 16.80% 84.08%   38169 301373 sun.nio.ch.ServerSocketChannelImpl.accept0
   6  6.75% 90.83%   15342 301646 java.io.FileOutputStream.writeBytes
   7  2.77% 93.60%    6294 301642 org.apache.hadoop.util.DataChecksum.verifyChunkedSums
   8  2.49% 96.08%    5653 301664 java.io.FileOutputStream.writeBytes
   9  1.14% 97.22%    2587 301640 sun.nio.ch.FileDispatcherImpl.read0
  10  0.35% 97.57%     798 301691 sun.nio.ch.FileDispatcherImpl.write0
```

Plain TestDFSIO read tests – Some Samples

- Workload Java samples:

```
CPU SAMPLES BEGIN (total = 13668)
rank   self  accum   count trace method
   1 36.82% 36.82%    5033 301083 org.apache.hadoop.util.NativeCrc32.nativeVerifyChunkedSums
   2 31.37% 68.19%    4287 301080 sun.nio.ch.EPollArrayWrapper.epollWait
   3 14.00% 82.18%    1913 301086 sun.nio.ch.FileDispatcherImpl.read0
   4  4.82% 87.01%     659 301082 sun.misc.Unsafe.copyMemory
   5  2.00% 89.00%     273 301118 sun.nio.ch.SocketChannelImpl.isConnected
   6  1.87% 90.88%     256 301089 sun.nio.ch.FileDispatcherImpl.read0
   7  0.76% 91.64%     104 300072 java.lang.ClassLoader.defineClass1
   8  0.61% 92.25%      84 301105 sun.nio.ch.SocketChannelImpl.isConnected
   9  0.51% 92.76%      70 300841 sun.nio.ch.EPollArrayWrapper.epollWait
  10  0.49% 93.25%      67 301127 sun.misc.Unsafe.copyMemory
```

- Datanode Java samples:

```
CPU SAMPLES BEGIN (total = 118437)
rank   self  accum   count trace method
   1 17.57% 17.57%   20810 301107 sun.nio.ch.EPollArrayWrapper.epollWait
   2 17.51% 35.08%   20741 301304 sun.nio.ch.EPollArrayWrapper.epollWait
   3 17.48% 52.56%   20697 301368 sun.nio.ch.EPollArrayWrapper.epollWait
   4 17.48% 70.03%   20697 301369 sun.nio.ch.EPollArrayWrapper.epollWait
   5 17.47% 87.51%   20695 301367 sun.nio.ch.ServerSocketChannelImpl.accept0
   6  8.71% 96.22%   10320 301593 sun.nio.ch.FileChannelImpl.transferTo0
   7  0.48% 96.70%     569 301524 org.apache.hadoop.io.nativeio.NativeIO.posix_fadvise
   8  0.40% 97.10%     471 301388 sun.nio.ch.EPollArrayWrapper.epollWait
   9  0.31% 97.41%     364 301595 sun.nio.ch.FileDispatcherImpl.write0
  10  0.29% 97.69%     339 301648 java.io.FileInputStream.readBytes
```

Plain TestDFSIO read tests – More Samples

- Namenode Java samples:

```
CPU SAMPLES BEGIN (total = 92048)
rank   self  accum   count trace method
   1 24.81% 24.81%   22833 301222 sun.nio.ch.EPollArrayWrapper.epollWait
   2 24.68% 49.48%   22715 301361 sun.nio.ch.EPollArrayWrapper.epollWait
   3 24.68% 74.16%   22713 301362 sun.nio.ch.EPollArrayWrapper.epollWait
   4 24.67% 98.83%   22708 301360 sun.nio.ch.EPollArrayWrapper.epollWait
   5  0.16% 98.99%     147 300075 java.lang.ClassLoader.defineClass1
   6  0.05% 99.04%      47 300084 java.util.zip.ZipFile.getEntry
   7  0.03% 99.07%      32 300078 java.util.zip.Inflater.inflateBytes
   8  0.03% 99.11%      30 300123 java.util.zip.ZipFile.read
   9  0.02% 99.13%      21 301170 sun.nio.ch.FileDispatcherImpl.force0
  10  0.02% 99.15%      16 301165
org.apache.hadoop.hdfs.server.namenode.EditLogFileOutputStream.<clinit>
```

- Linux perf results:

```
+  36.37%            java  libhadoop.so.1.0.0      [.] crc32c_sb8
+   9.40%            java  [kernel.kallsyms]       [k] __copy_to_user_std
+   6.02%            java  perf-14362.map           [.] 0xb40d5040
+   5.52%            java  perf-15039.map           [.] 0xb440aba0
+   4.92%            java  libc-2.15.so             [.] memcpy
+   2.09%            java  [kernel.kallsyms]       [k] kmap_high
+   1.51%            java  [kernel.kallsyms]       [k] _raw_spin_unlock_irqrestore
+   1.10%         swapper  [kernel.kallsyms]       [k] default_idle
+   0.70%         kswapd0  [kernel.kallsyms]       [k] __remove_mapping
+   0.68%            java  [kernel.kallsyms]       [k] get_page_from_freelist
```

New TestDFSIO write tests – Some Samples

- Workload Java samples:

```
CPU SAMPLES BEGIN (total = 52774)
rank   self  accum   count trace method
   1 38.67% 38.67%   20407 300839 sun.nio.ch.EPollArrayWrapper.epollWait
   2 32.56% 71.23%   17184 301074 sun.nio.ch.EPollArrayWrapper.epollWait
   3 11.51% 82.74%    6076 301070 org.apache.hadoop.util.HyperCrc32C.nativeUpdate
   4  4.14% 86.88%    2184 301071 sun.misc.Unsafe.copyMemory
   5  2.69% 89.57%    1419 301076 sun.nio.ch.FileDispatcherImpl.write0
   6  0.65% 90.22%     341 301102 sun.nio.ch.FileDispatcherImpl.read0
   7  0.45% 90.67%     240 301143 java.lang.System.arraycopy
   8  0.44% 91.11%     231 300804 sun.nio.ch.EPollArrayWrapper.epollWait
   9  0.38% 91.49%     203 301154 sun.nio.ch.EPollArrayWrapper.epollCtl
  10  0.38% 91.88%     202 301080 org.apache.hadoop.hdfs.DFSOutputStream.writeChunk
```

- Datanode Java samples:

```
CPU SAMPLES BEGIN (total = 234447)
rank   self  accum   count trace method
   1 17.80% 17.80%   41734 301113 sun.nio.ch.EPollArrayWrapper.epollWait
   2 17.77% 35.58%   41672 301307 sun.nio.ch.EPollArrayWrapper.epollWait
   3 17.76% 53.33%   41629 301365 sun.nio.ch.EPollArrayWrapper.epollWait
   4 17.76% 71.09%   41628 301368 sun.nio.ch.EPollArrayWrapper.epollWait
   5 17.75% 88.84%   41621 301364 sun.nio.ch.ServerSocketChannelImpl.accept0
   6  3.33% 92.17%    7796 301634 sun.nio.ch.FileDispatcherImpl.write0
   7  2.73% 94.89%    6390 301657 java.io.FileOutputStream.writeBytes
   8  1.33% 96.22%    3113 301640 sun.nio.ch.FileDispatcherImpl.read0
   9  1.30% 97.52%    3038 301631 org.apache.hadoop.util.NativeCrc32.nativeVerifyChunkedSums
  10  0.33% 97.84%     766 301641 sun.nio.ch.FileDispatcherImpl.write0
```

New TestDFSIO write tests – More Samples

- Namenode Java samples:

```
CPU SAMPLES BEGIN (total = 180508)
rank   self  accum   count trace method
   1 24.87% 24.87%   44890 301198 sun.nio.ch.EPollArrayWrapper.epollWait
   2 24.82% 49.69%   44808 301337 sun.nio.ch.EPollArrayWrapper.epollWait
   3 24.82% 74.51%   44806 301338 sun.nio.ch.EPollArrayWrapper.epollWait
   4 24.81% 99.33%   44792 301334 sun.nio.ch.EPollArrayWrapper.epollWait
   5  0.07% 99.40%     135 300075 java.lang.ClassLoader.defineClass1
   6  0.03% 99.43%      46 300080 java.util.zip.ZipFile.getEntry
   7  0.02% 99.45%      41 300079 java.util.zip.Inflater.inflateBytes
   8  0.01% 99.47%      27 300548 java.io.FileInputStream.readBytes
   9  0.01% 99.48%      24 301230 java.lang.UNIXProcess.waitForProcessExit
  10  0.01% 99.49%      22 301145 sun.nio.ch.FileDispatcherImpl.force0
```

- Linux perf results:

```
+  18.04%            java  libhadoop.so.1.0.0      [.] fold
+  14.38%            java  perf-16729.map          [.] 0xb42c0840
+   6.45%            java  libc-2.15.so            [.] memcpy
+   5.77%            java  [kernel.kallsyms]       [k] __copy_from_user
+   3.89%            java  [kernel.kallsyms]       [k] __copy_to_user_std
+   2.73%            java  perf-16647.map          [.] 0xb4142560
+   2.42%            java  [kernel.kallsyms]       [k] _raw_spin_unlock_irqrestore
+   1.91%            java  libhadoop.so.1.0.0      [.] crc32c_sb8
+   1.28%         swapper  [kernel.kallsyms]       [k] finish_task_switch
+   0.96%         swapper  [kernel.kallsyms]       [k] tick_nohz_idle_enter
+   0.88%            java  libhadoop.so.1.0.0      [.] crc32c_neon
```

New TestDFSIO read tests – Some Samples

- Workload Java samples:

```
CPU SAMPLES BEGIN (total = 13997)
rank   self  accum   count trace method
   1 41.44% 41.44%    5800 301067 sun.nio.ch.EPollArrayWrapper.epollWait
   2 25.40% 66.84%    3555 301075
org.apache.hadoop.util.NativeCrc32.nativeVerifyChunkedSums
   3 13.15% 79.98%    1840 301074 sun.nio.ch.FileDispatcherImpl.read0
   4  4.63% 84.61%     648 301076 sun.misc.Unsafe.copyMemory
   5  2.22% 86.83%     311 301085 sun.nio.ch.SocketChannelImpl.isConnected
   6  1.86% 88.70%     261 301077 sun.nio.ch.FileDispatcherImpl.read0
   7  1.56% 90.26%     219 300827 sun.nio.ch.EPollArrayWrapper.epollWait
   8  1.15% 91.41%     161 300790 sun.nio.ch.EPollArrayWrapper.epollWait
   9  0.64% 92.05%      89 300072 java.lang.ClassLoader.defineClass1
  10  0.61% 92.66%      86 301099 sun.nio.ch.SocketChannelImpl.isConnected
```

- Datanode Java samples:

```
CPU SAMPLES BEGIN (total = 123308)
rank   self  accum   count trace method
   1 17.64% 17.64%   21751 301117 sun.nio.ch.EPollArrayWrapper.epollWait
   2 17.59% 35.23%   21686 301307 sun.nio.ch.EPollArrayWrapper.epollWait
   3 17.55% 52.78%   21643 301370 sun.nio.ch.EPollArrayWrapper.epollWait
   4 17.55% 70.33%   21643 301371 sun.nio.ch.EPollArrayWrapper.epollWait
   5 17.55% 87.88%   21640 301369 sun.nio.ch.ServerSocketChannelImpl.accept0
   6  8.05% 95.93%    9930 301604 sun.nio.ch.FileChannelImpl.transferTo0
   7  0.78% 96.71%     963 301567 java.io.FileInputStream.readBytes
   8  0.49% 97.20%     604 301580 org.apache.hadoop.io.nativeio.NativeIO.posix_fadvise
   9  0.26% 97.46%     320 301607 sun.nio.ch.FileDispatcherImpl.write0
  10  0.22% 97.68%     268 301586 sun.nio.ch.EPollArrayWrapper.epollWait
```

New TestDFSIO read tests – More Samples

- Namenode Java samples:

```
CPU SAMPLES BEGIN (total = 97523)
rank   self  accum   count trace method
   1 24.83% 24.83%   24218 301190 sun.nio.ch.EPollArrayWrapper.epollWait
   2 24.72% 49.55%   24103 301336 sun.nio.ch.EPollArrayWrapper.epollWait
   3 24.71% 74.26%   24101 301337 sun.nio.ch.EPollArrayWrapper.epollWait
   4 24.71% 98.97%   24096 301335 sun.nio.ch.EPollArrayWrapper.epollWait
   5  0.14% 99.11%     133 300073 java.lang.ClassLoader.defineClass1
   6  0.05% 99.16%      50 300085 java.util.zip.ZipFile.getEntry
   7  0.05% 99.20%      45 300078 java.util.zip.Inflater.inflateBytes
   8  0.02% 99.22%      21 300081 java.util.zip.ZipFile.read
   9  0.02% 99.25%      21 301135 sun.nio.ch.FileDispatcherImpl.force0
  10  0.01% 99.26%      13 301128
org.apache.hadoop.hdfs.server.namenode.EditLogFileOutputStream.<clinit>
```

- Linux perf results:

```
+  20.63%            java  libhadoop.so.1.0.0       [.] fold
+  10.89%            java  [kernel.kallsyms]        [k] __copy_to_user_std
+   6.58%            java  perf-15890.map           [.] 0xb44417f8
+   5.74%            java  perf-15818.map           [.] 0xb41a8de0
+   5.37%            java  libc-2.15.so             [.] memcpy
+   3.15%            java  libhadoop.so.1.0.0       [.] crc32c_sb8
+   2.64%            java  [kernel.kallsyms]        [k] kmap_high
+   2.01%            java  [kernel.kallsyms]        [k] _raw_spin_unlock_irqrestore
+   1.31%         swapper  [kernel.kallsyms]        [k] default_idle
+   1.30%            java  libhadoop.so.1.0.0       [.] crc32c_neon
+   0.88%            java  [kernel.kallsyms]        [k] get_page_from_freelist
```

More about Linaro Connect: www.linaro.org/connect/
More about Linaro: www.linaro.org/about/
More about Linaro engineering: www.linaro.org/engineering/