cloudera
Ask Bigger Questions

Feeding the Elephant: Optimizing the Read Path of the Hadoop Distributed Filesystem

by Colin P. McCabe

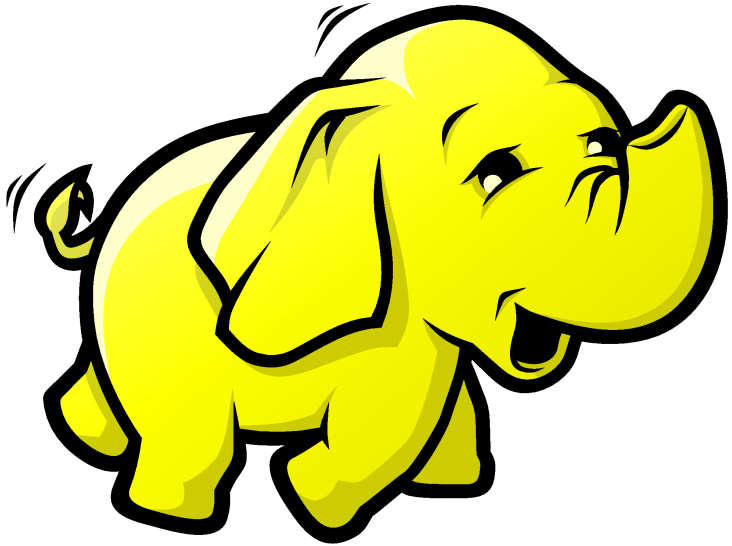
About Me

- I work on HDFS and related storage technologies at Cloudera.
- Committer on the HDFS and Hadoop projects.
- Previously worked on the Ceph distributed filesystem

About Hadoop

The open source framework for big data

Started by Doug Cutting and Mike Cafarella in 2005.



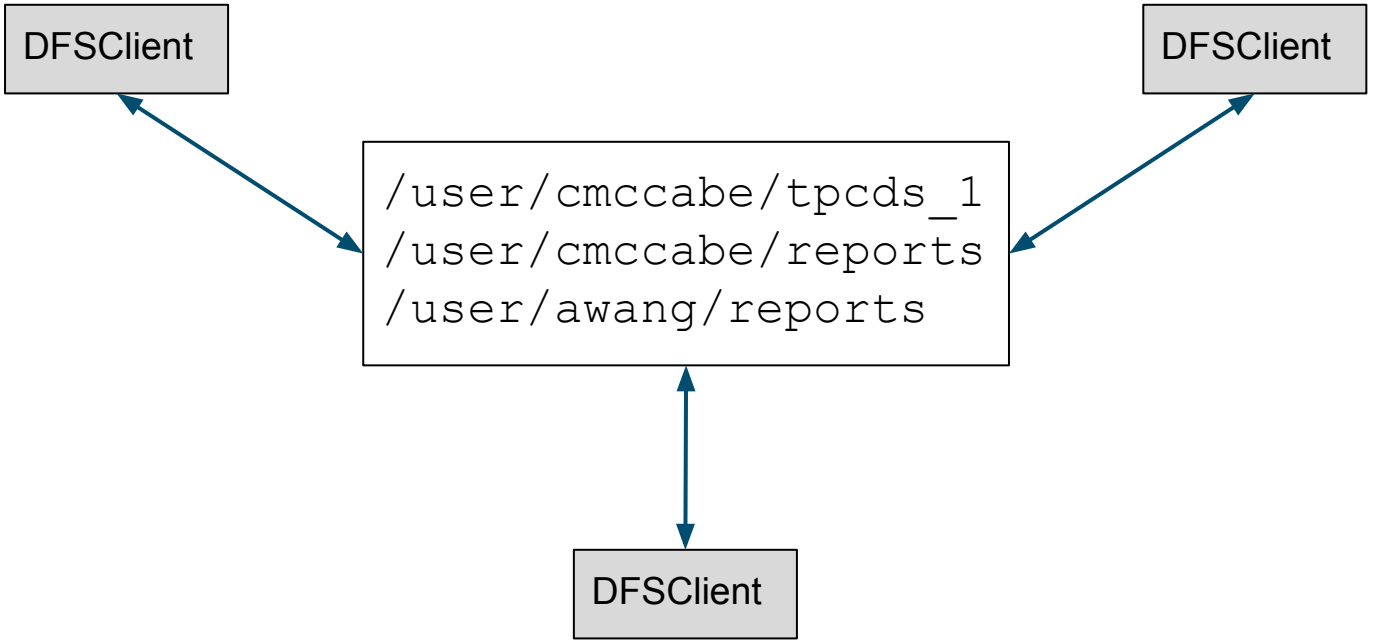
Big Ideas Behind Hadoop

- Distributed
  - Avoid single-node points of failure
  - Avoid losing data
- Bring the computation to the data
  - Cheaper to move computation than data.
- Commodity hardware, not specialized hardware

HDFS Architecture

- HDFS decides where and how to store data in Hadoop.
- Nodes
  - DataNodes: store data
  - NameNodes: handle metadata
  - JournalNodes: store metadata

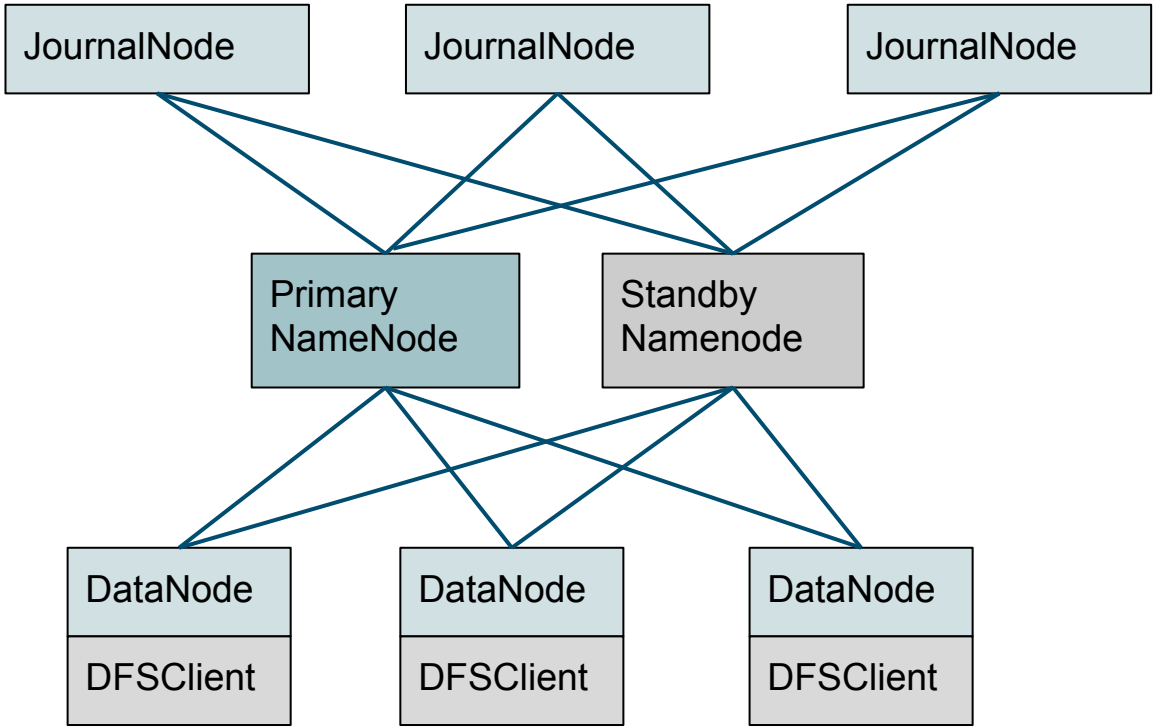
Shared Namespace



HDFS Clients

- But… where data is located still matters!
- The HDFS client library is called “the DFSClient.”
- Usually DFSClients are co-located with DataNodes, to minimize network I/O.
- “Move the computation to the data.”

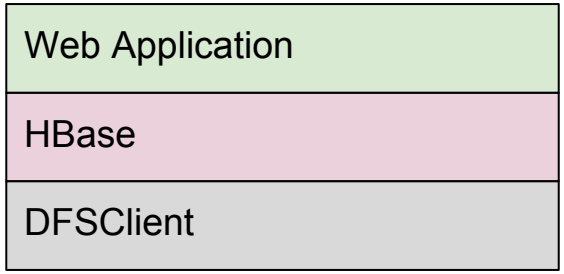
Typical HDFS Setup

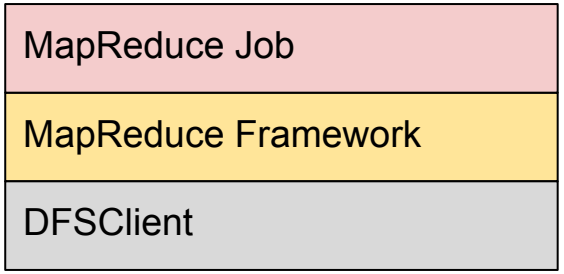


HDFS Storage Stacks

- HDFS “Clients” are often daemons like MapReduce, Impala, or HBase, not user programs.
- The performance of the DFSClient can be a limiting factor for the performance of the rest of the stack.

HDFS Storage Stacks



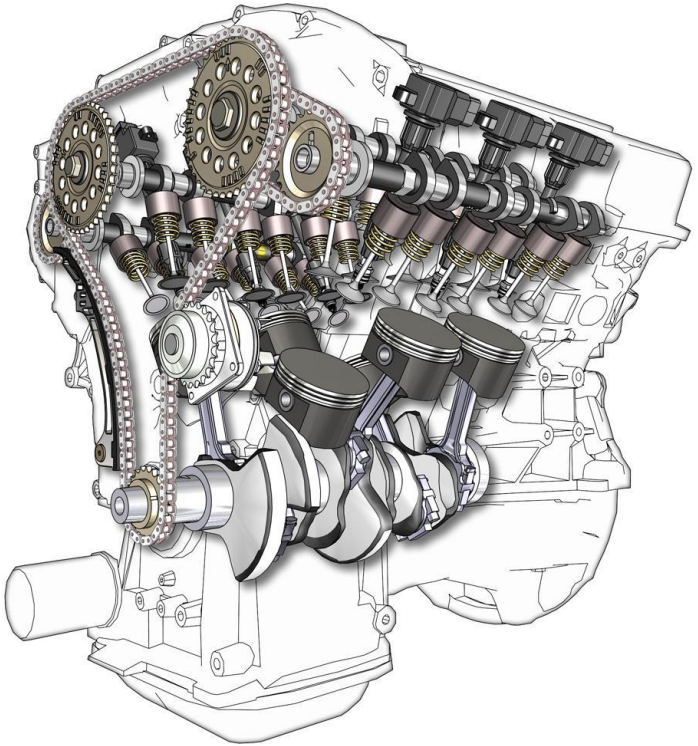


DFSClient Performance

- DFSClient performance is a big part of HDFS performance, since HDFS files are often large.
- Reads and writes don't go through the NameNode… they go through the DFSClient.
- How can we improve DFSClient performance?
- First, we have to quantify it.

Quantifying Performance

- CPU utilization
- Memory bandwidth
- Latency
- Disk Bandwidth
- Scalability



Quantifying Performance

- Workload-dependent
  - Amdahl's Law
  - What's the bottleneck?
- Cluster dependent
  - 1 gigabit ethernet? 10 gigabit?
  - How many disks? CPUs?

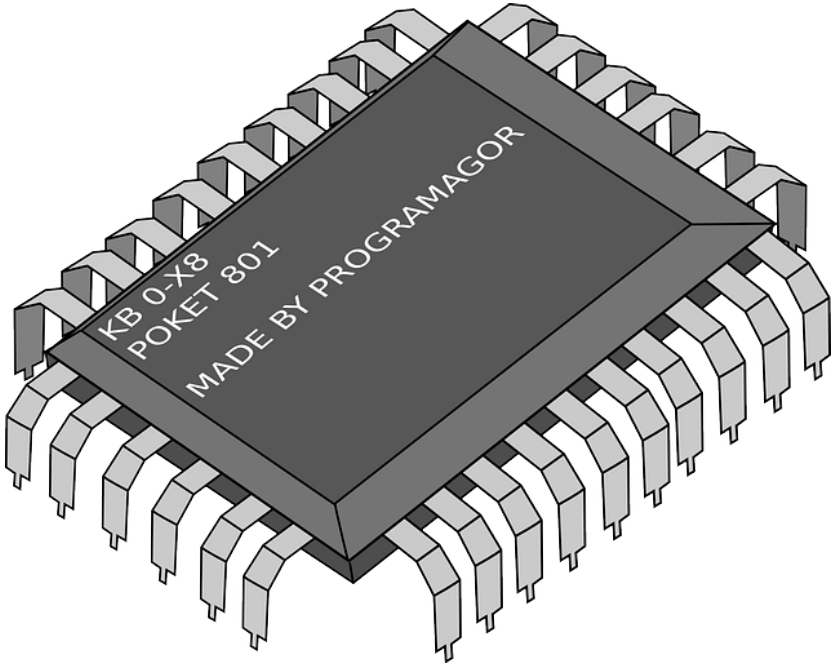
DFSClient Overheads

- CPU overhead of checksumming
- Copying data from buffer to buffer
- The TCP 3-way handshake
- Making system calls to the operating system
- Stragglers
- Garbage collection

Saving CPU

The more CPU we use in the DFSClient and the DataNode, the less is available to clients.

How can we save CPU cycles?



Saving CPU

- HDFS-2080: Optimized native checksum implementation
  - Uses Intel's built-in SSE4.2 CRC32 instruction
  - Checksum overhead went from ~50% CPU to ~15%

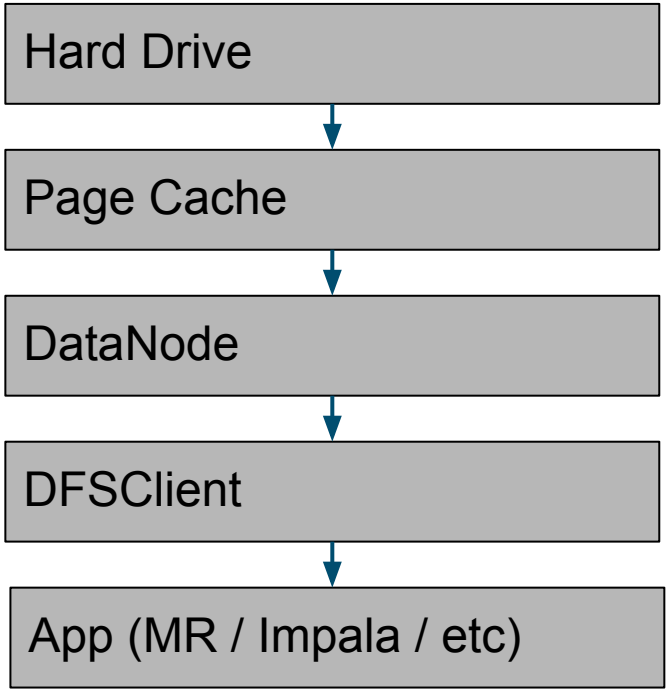
Saving CPU

- HDFS-5276: Thread-Local statistics in FileSystem
  - A big improvement for massively multithreaded programs

Avoiding Copies

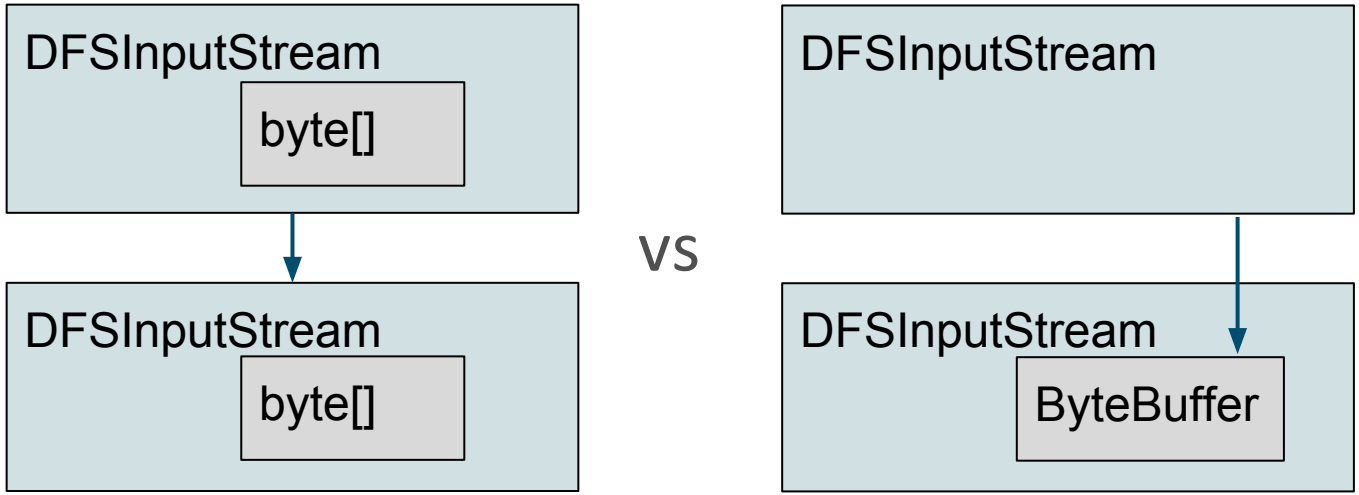
Can we reduce the number of times we copy the data?

(Simplified view ignoring copies due to readahead, decompression, etc.)



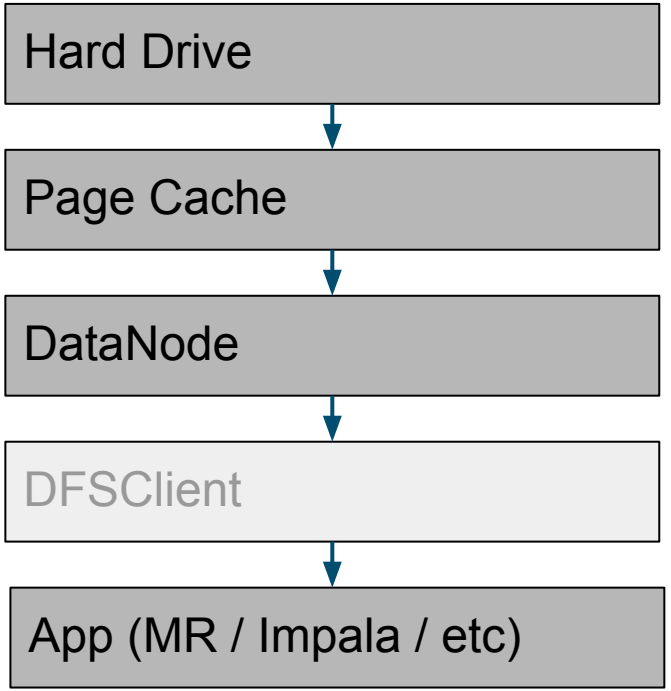
Direct Reads

HDFS-2834: Use Java's ByteBuffer abstraction in DFSClient to avoid a copy inside DFSClient.



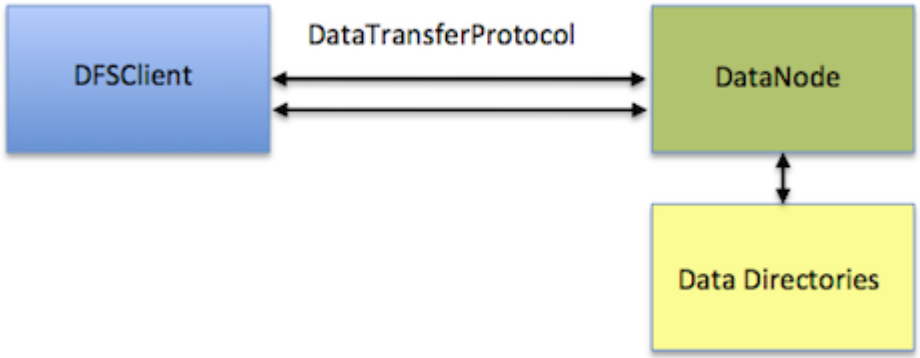
Direct Reads

Now we don’t have to copy from a byte array to another byte array in the DFSClient.



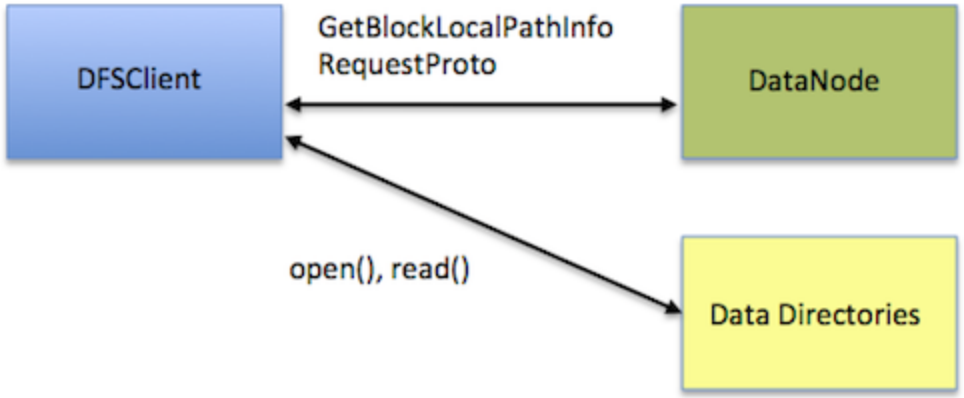
Short-Circuit Local Reads

Original read path:



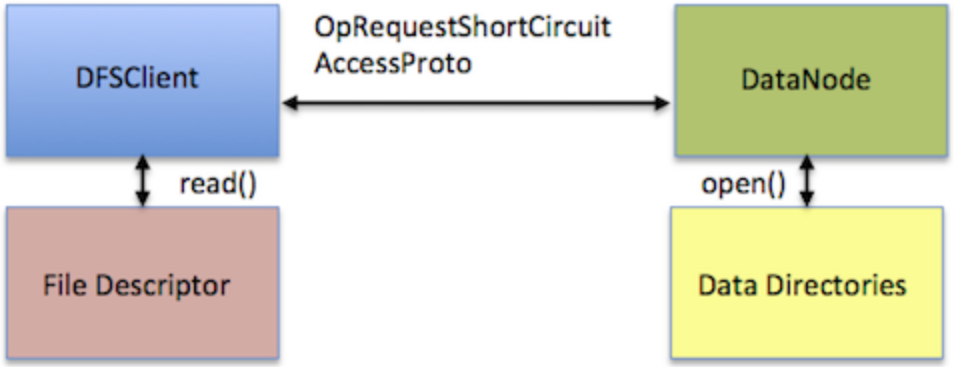
Short-Circuit Local Reads

HDFS-2246 Short-circuit read path:



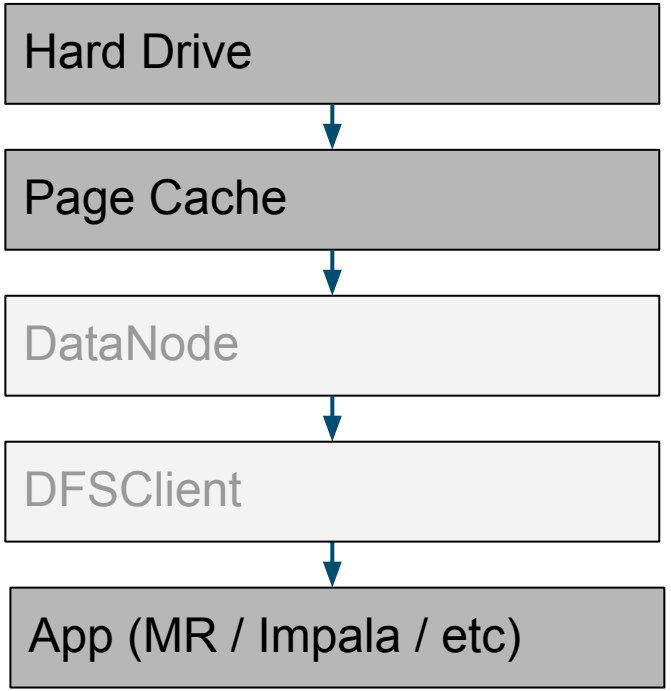
Short-Circuit Local Reads

HDFS-347 Secure Short-circuit read path:

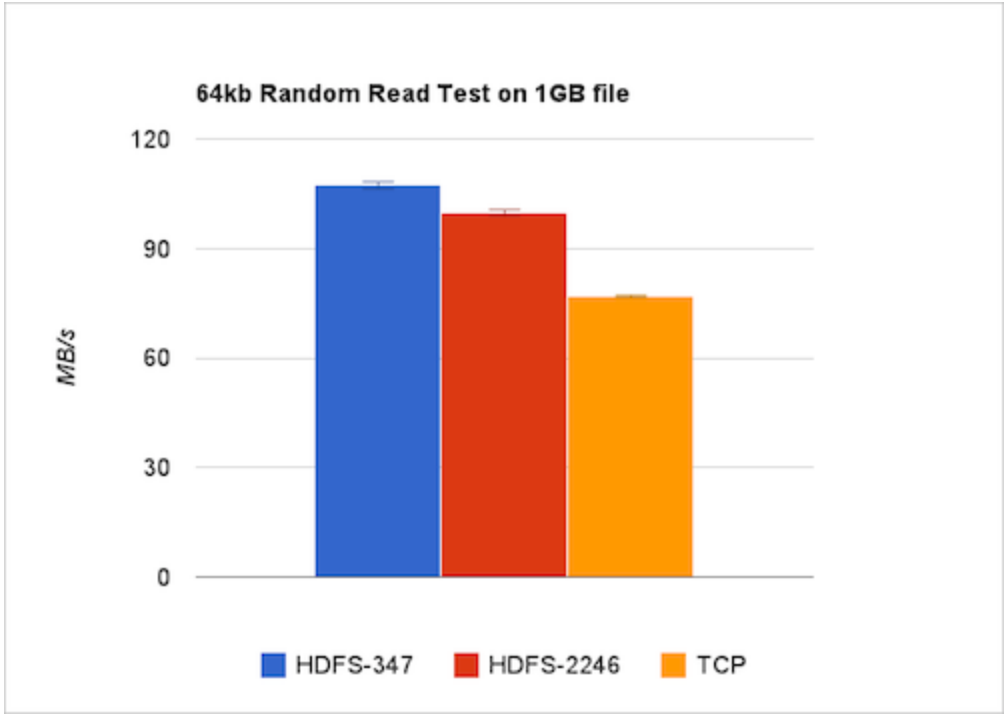


Short-Circuit Local Reads

Now we aren't copying the data into the DataNode process any more, for local blocks.



HDFS-347 Performance Improvements

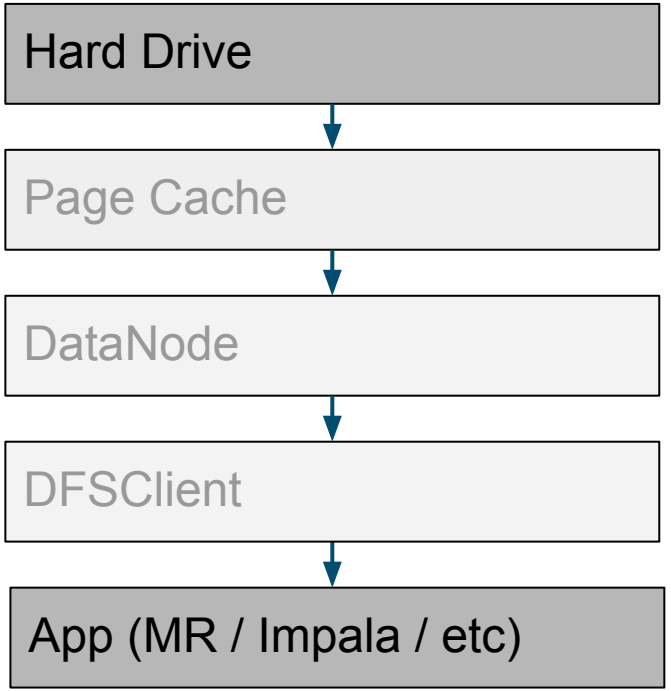


HDFS-4953: Zero-copy Reads

- Normal reads copy the data into the OS page cache, and then copy it again into the process' memory space
- Zero-copy reads use mmap to directly map the memory into the process memory space.
- Must be able to skip checksum

Zero-Copy Reads

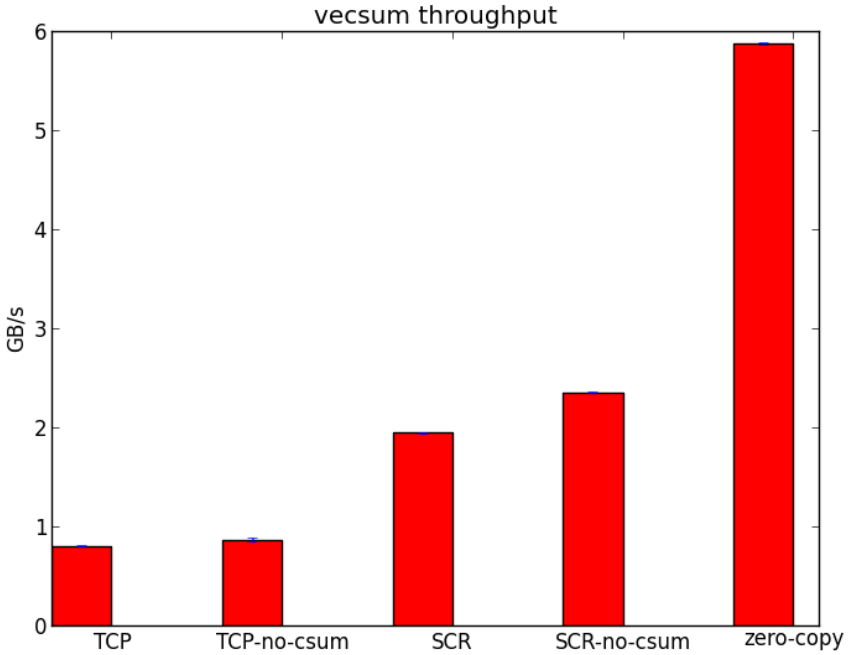
Now the page cache memory is mapped into our address space, avoiding another copy.



Throughput Using Zero-Copy Reads

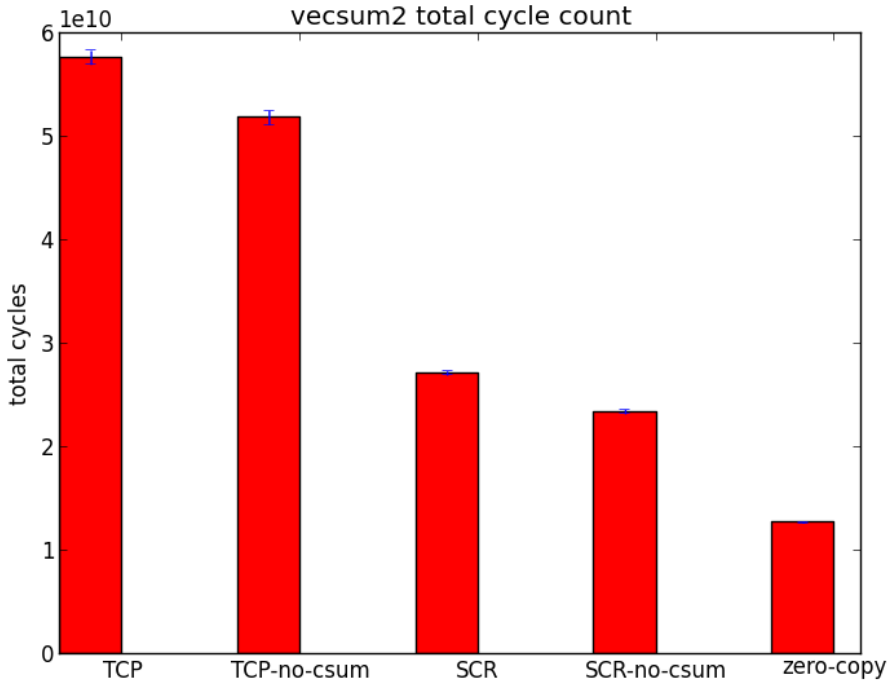
vecsum: a highly optimized libhdfs program on CDH5

Computes the sum of a 1GB file containing floating point numbers 20 times.



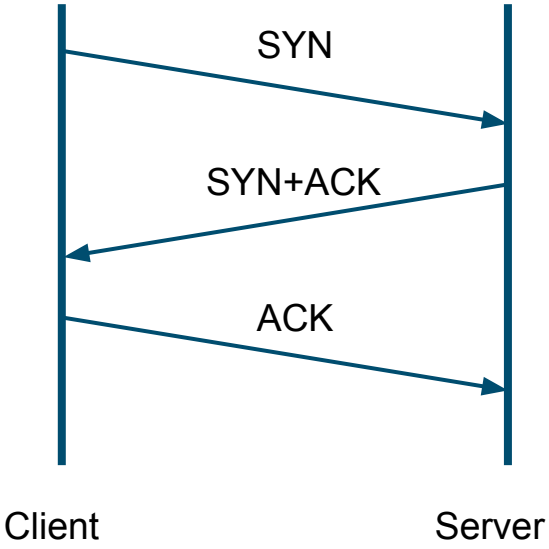
CPU Consumption Using Zero-Copy Reads

Total CPU cycles consumed when reading 1 GB 20 times



Avoiding TCP overheads

How can we avoid TCP overheads?



Avoiding TCP overheads

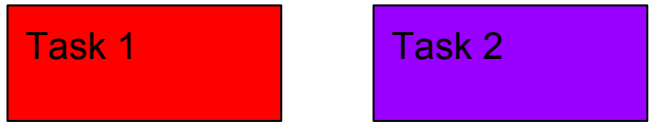
- HDFS-347: Avoid TCP entirely via short-circuit reads
- HDFS-941: Re-use TCP connections to the DataNode
  - Avoid paying overhead of 3-way handshake and connection setup each time
  - Requires socket cache and DataXceiver changes

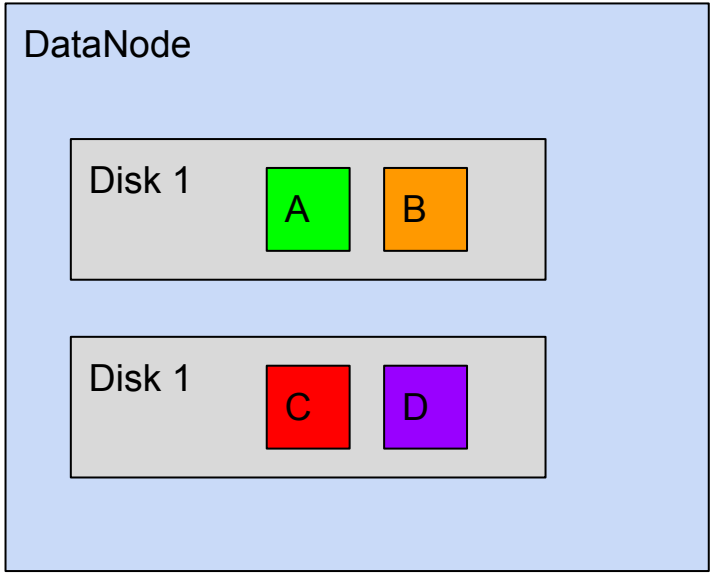
Higher Level Improvements

- More than just optimizing our own code, how do we let clients make better decisions about where to schedule and how to cache data?
  - Impala
  - MapReduce

HDFS-3672: Expose Disk Location Information

Scheduling multiple jobs that need the same hard disk at the same time makes things slower.



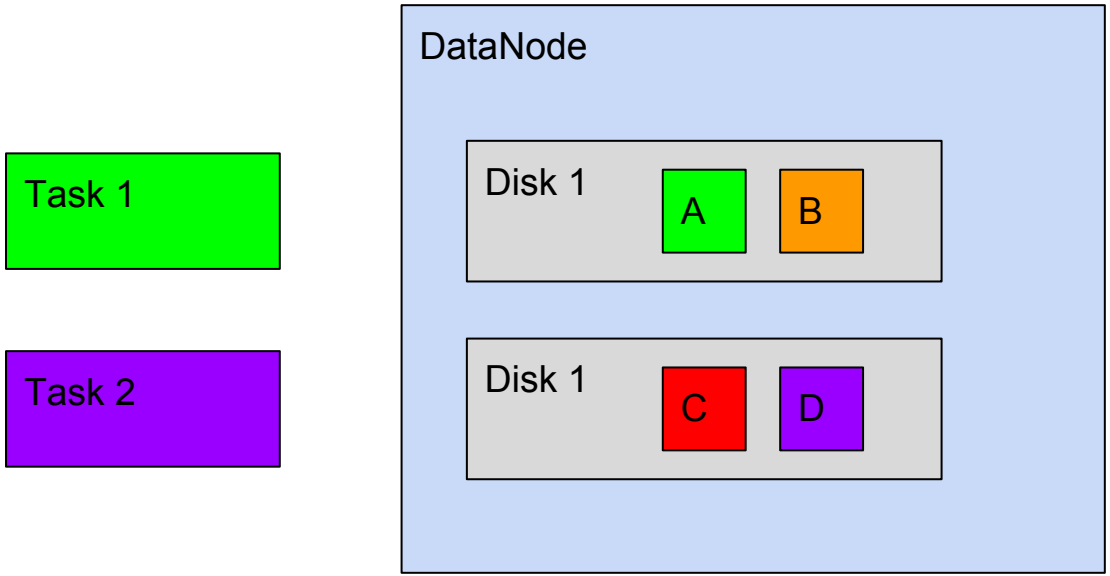


HDFS-3672: Expose Disk Location Information

- Previously: Impala knew that block B was on DataNodes D1, D2, and D3, but didn't know which drives it was on. Had to guess.
- Now: Impala can schedule work across the cluster so that it reads from every drive on each DataNode.

HDFS-3672: Expose Disk Location Information

With more information, we can schedule smarter.



HDFS-4949: HDFS Caching

- Previously: Operating system decided where to cache things. Often cached the same thing on more than one DataNode. Might kick important information out of cache during a big query or job.
- Now: can explicitly ask for certain files or directories to be cached. Can skip checksum when reading these files (or ZCR)

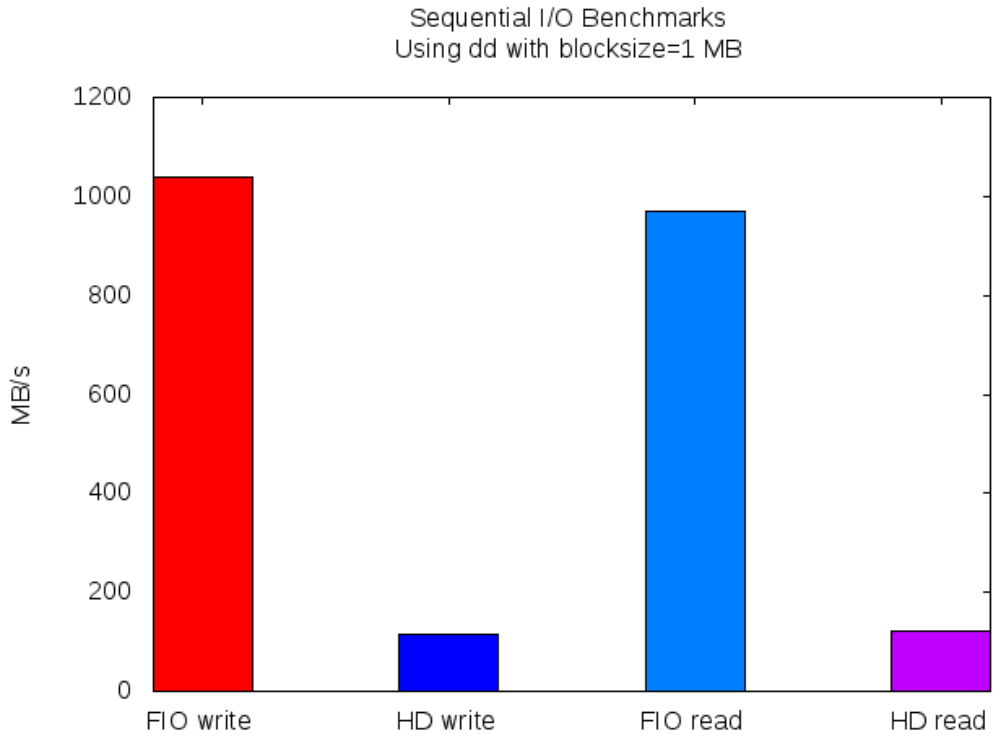
HDFS-4949: HDFS Caching

- Can “move the computation to the cache” to take advantage of memory speed. Up to an order of magnitude improvement in Impala performance.

Hardware Trends

- Solid-state disks
  - Need more read and write path improvements to best take advantage of these.
  - Need improvements in client software too.
  - The assumption that I/O is the bottleneck may not always hold!

Fusion I/O vs. Hard Disk Bandwidth



Hardware Trends

- Networks with full bisectional bandwidth
  - Rack locality stops being important.
  - Cache locality is always important, however.
    - You always want to bring the computation to the cache.
    - Memory is usually at least order of magnitude faster than network.

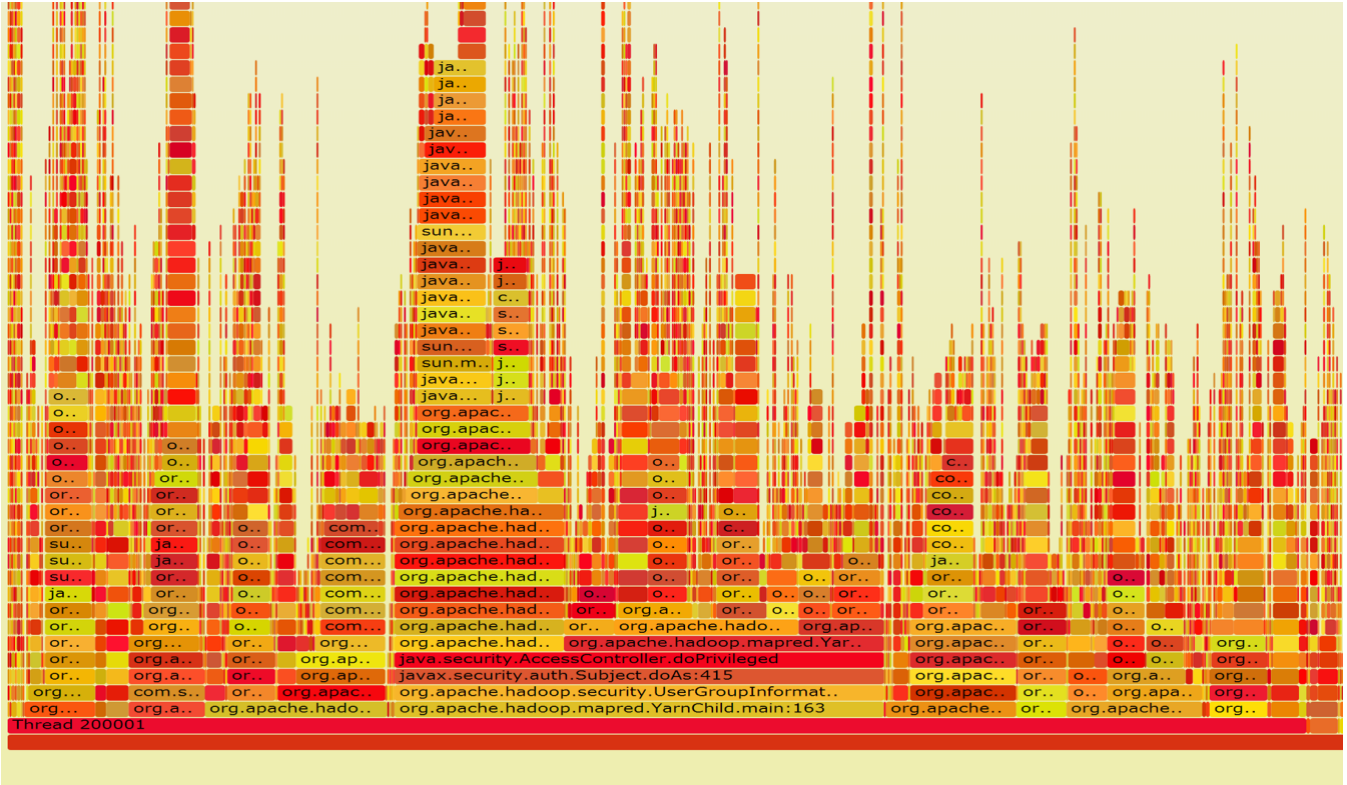
Limits to Performance Improvements

- Not all clients are bottlenecked on I/O
  - Many MapReduce jobs do a **lot** of CPU work.
  - Lifting I/O bottlenecks may only reveal other bottlenecks.

Limits to Performance Improvements

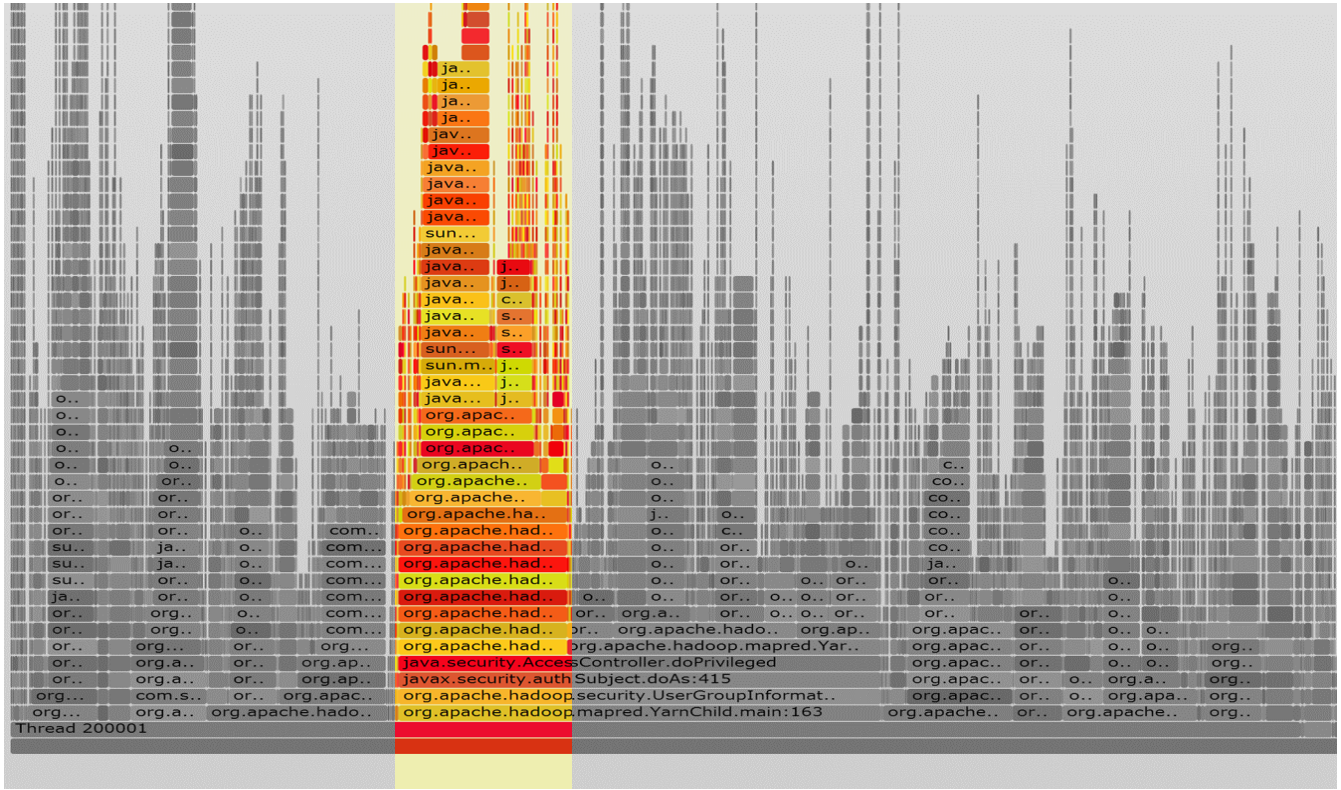
- Not all clusters are the same.
  - Cluster performance can be limited by the network, or even by something else unexpected such as DNS resolution.
  - Benchmark, benchmark, benchmark. Don't assume.

Word Count MapReduce Flame Graph



cloudera
Ask Bigger Questions

Word Count MapReduce Flame Graph: I/O



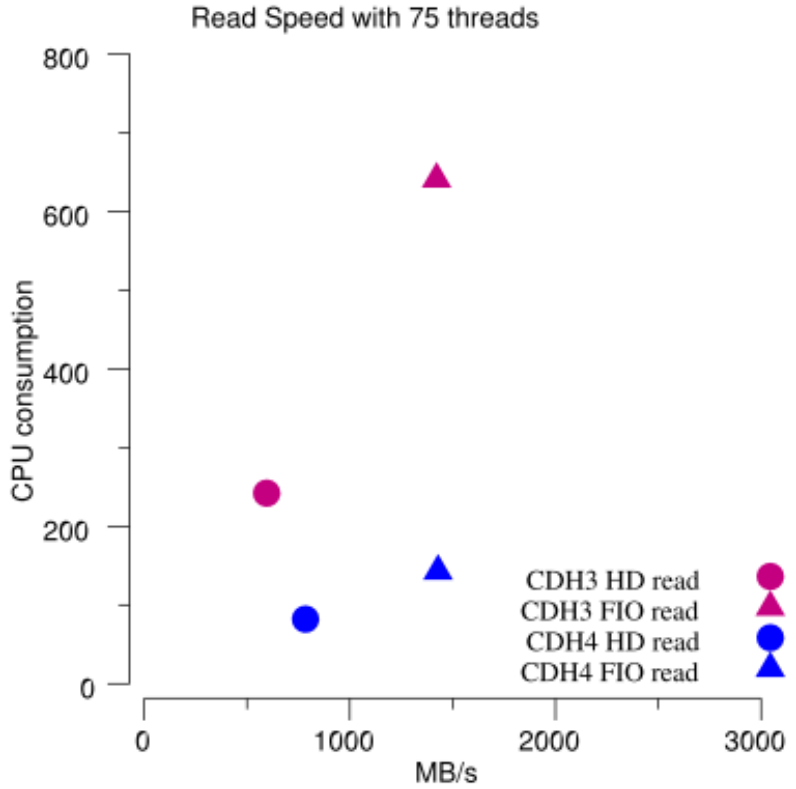
cloudera
Ask Bigger Questions

Limits to Performance Improvements

- HDFS performance is most important when the higher layers are also optimized.
- Impala can get 10x gains out of HDFS caching or using flash, but MR struggles to get 2x
- Newer frameworks like Apache Spark will help

Benchmarking Old Versus New Read Path

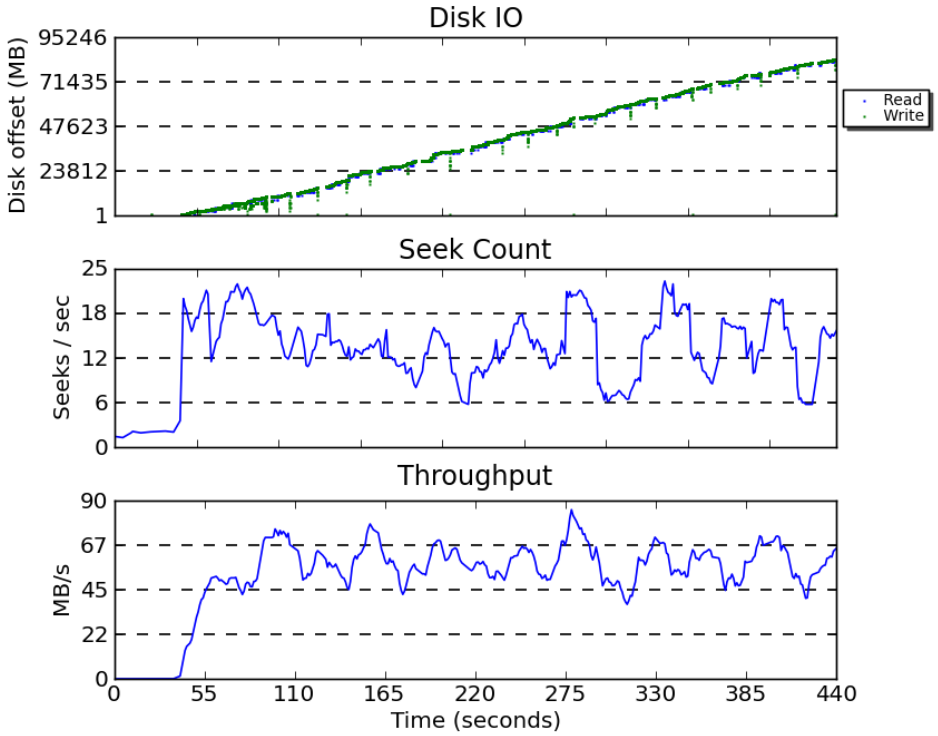
- CDH3 versus CDH4
- Simultaneous sequential reads from 75 different threads.
- 10 hard drive configuration vs. high-speed flash



cloudera
Ask Bigger Questions

Single Hard Drive Seeks and Throughput

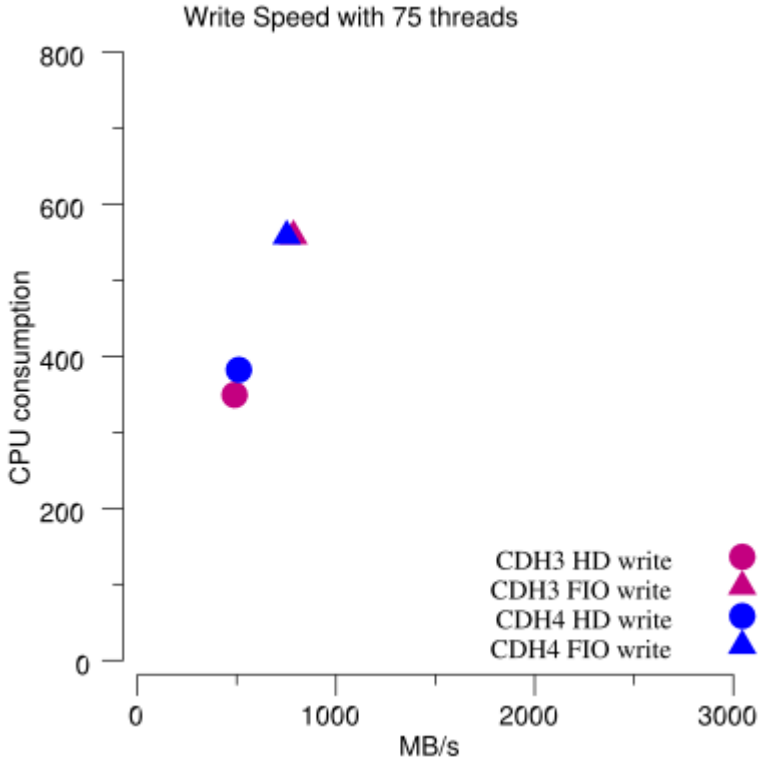
- Average seeks: 12/s
- Average throughput around 50 MB/s



cloudera
Ask Bigger Questions

Benchmarking Old Versus New Write Path

- The write path needs more improvement.
- Harder to optimize because of need to use network (3x writes)



Future Work

- Make HDFS caching useful in more scenarios
  - ❏ Sub-block caching
  - ❏ Automatic caching via LRU, LFU, etc.
- HDFS on flash
  - ❏ HDFS-2832: Heterogeneous Storage
  - ❏ Allow HDFS to manage different pools of storage (e.g. hard drives versus flash)

Future Work

- Write path efficiency improvements
  - ❑ Native checksums for write path
- Write-side caching
  - ❑ Avoid flushing temporary files to disk if it's not needed.