Column Cache: Buffer Cache for Columnar Storage on HDFS

Takeshi Yoshimura, Tatsuhiro Chiba, Hiroshi Horii

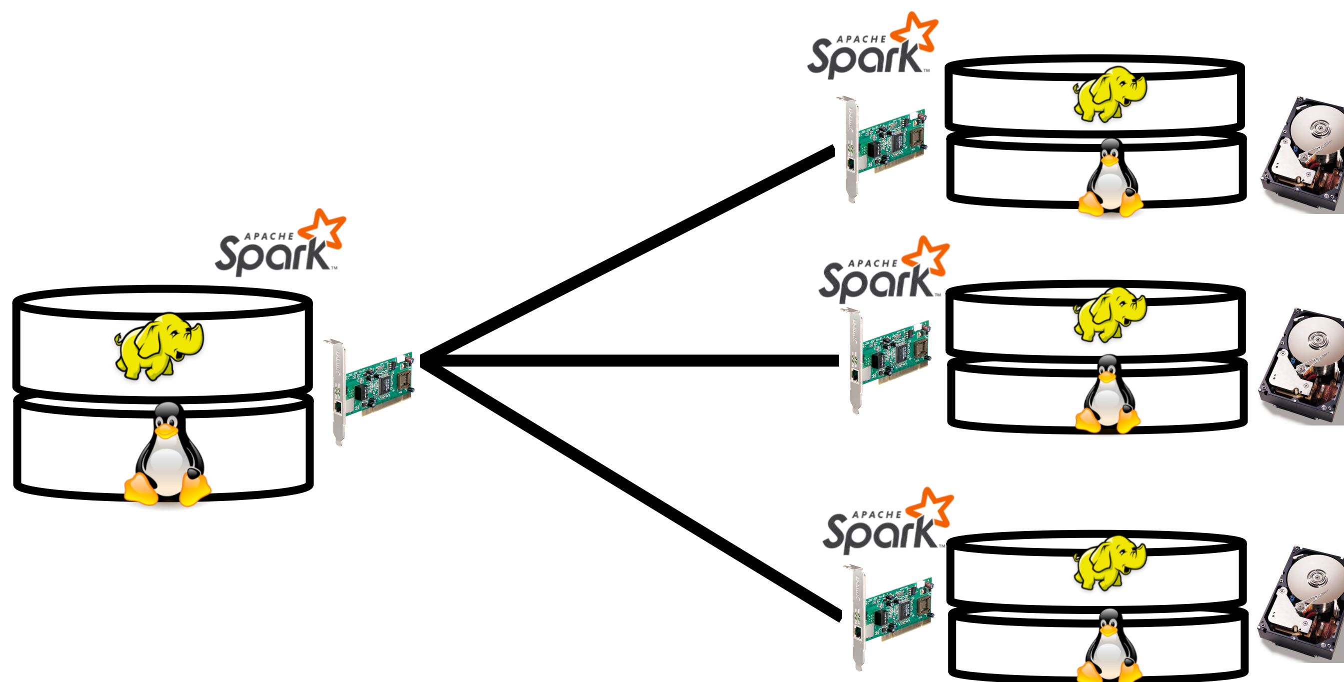
IBM Research – Tokyo

IEEE BigData 2018

© 2018 IBM Corporation

Hadoop distributed file system (HDFS)

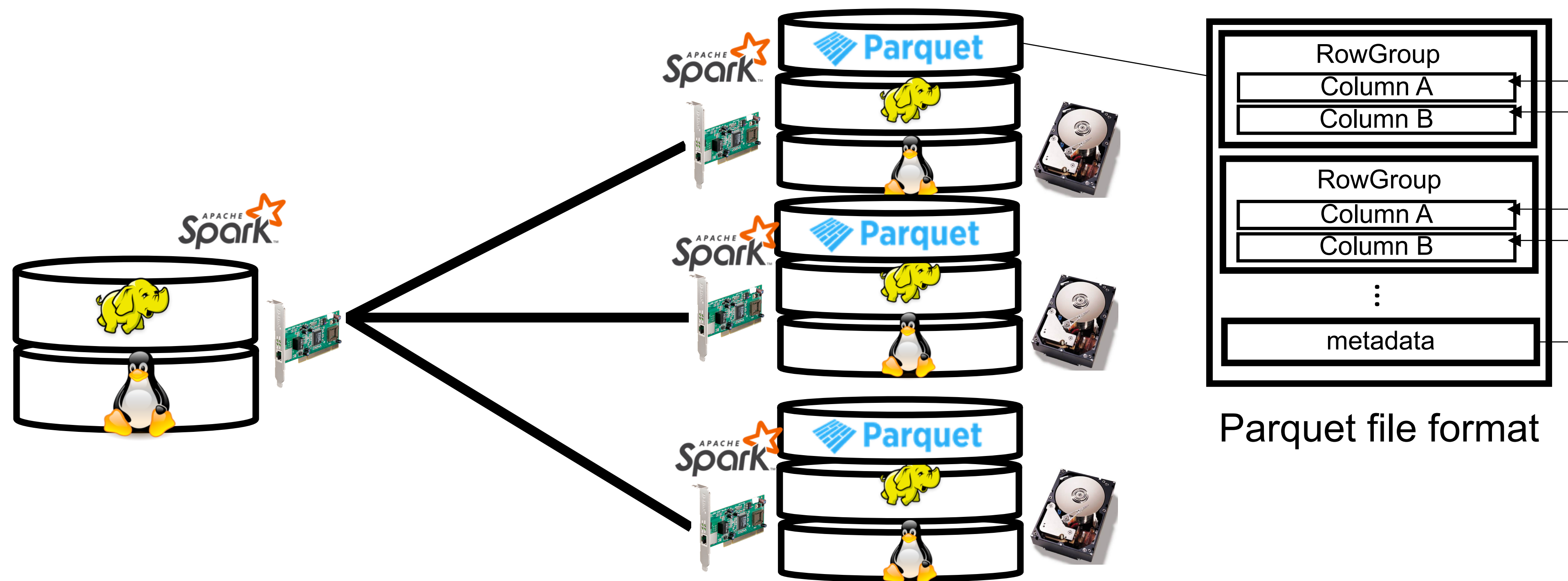
- A major data store of scale-out architecture in big data processing
  - –Offers high scalability and availability to parallel distributed computing
- Typical use-case: relational query processing and machine learning
  - –e.g., Spark SQL/ML, Apache Tez, Apache Mahout



© 2018 IBM Corporation

Columnar Storage on HDFS

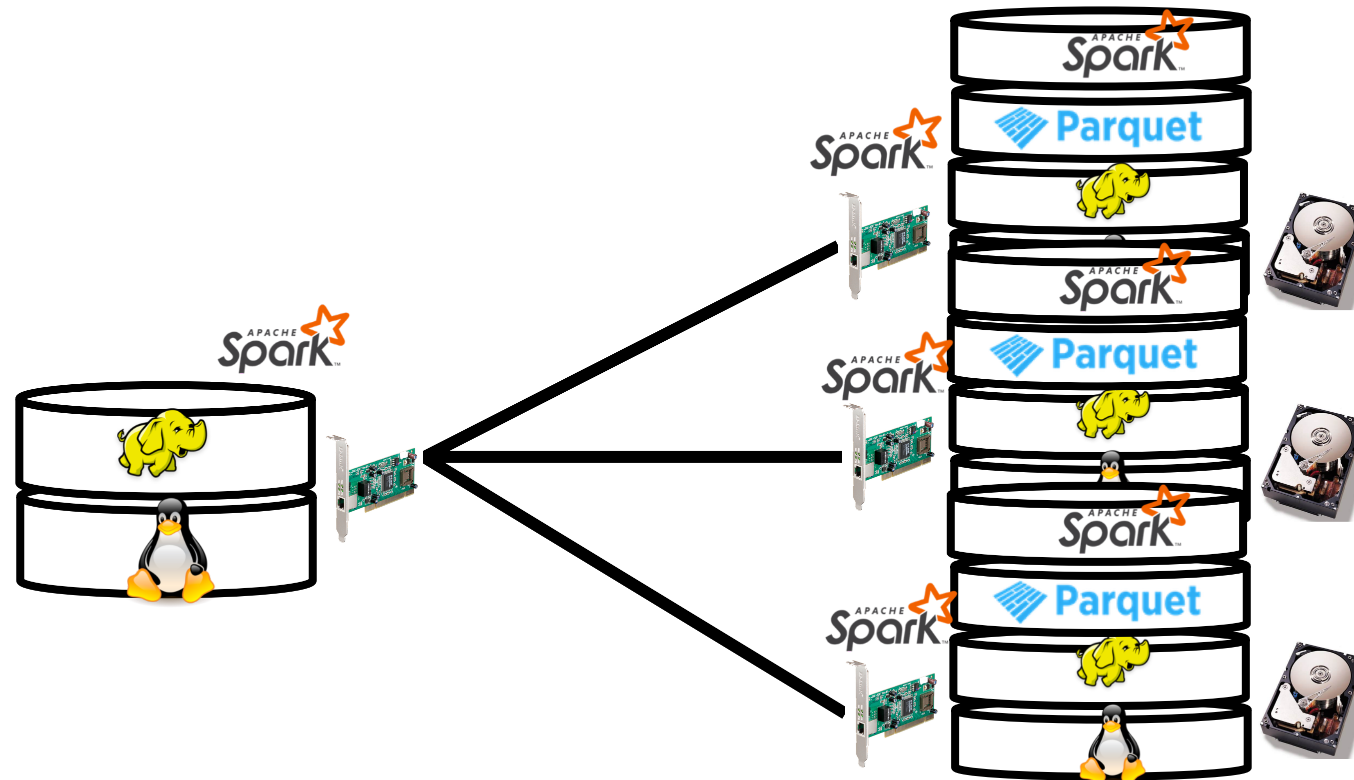
- Designed as a special data representation for a HDFS file
  - –Provides highly compressed data by columnar-wise data layout
  - –Preserves high scalability and availability of HDFS
- Optimizes read heavy queries



© 2018 IBM Corporation

…And Spark wraps all with RDD

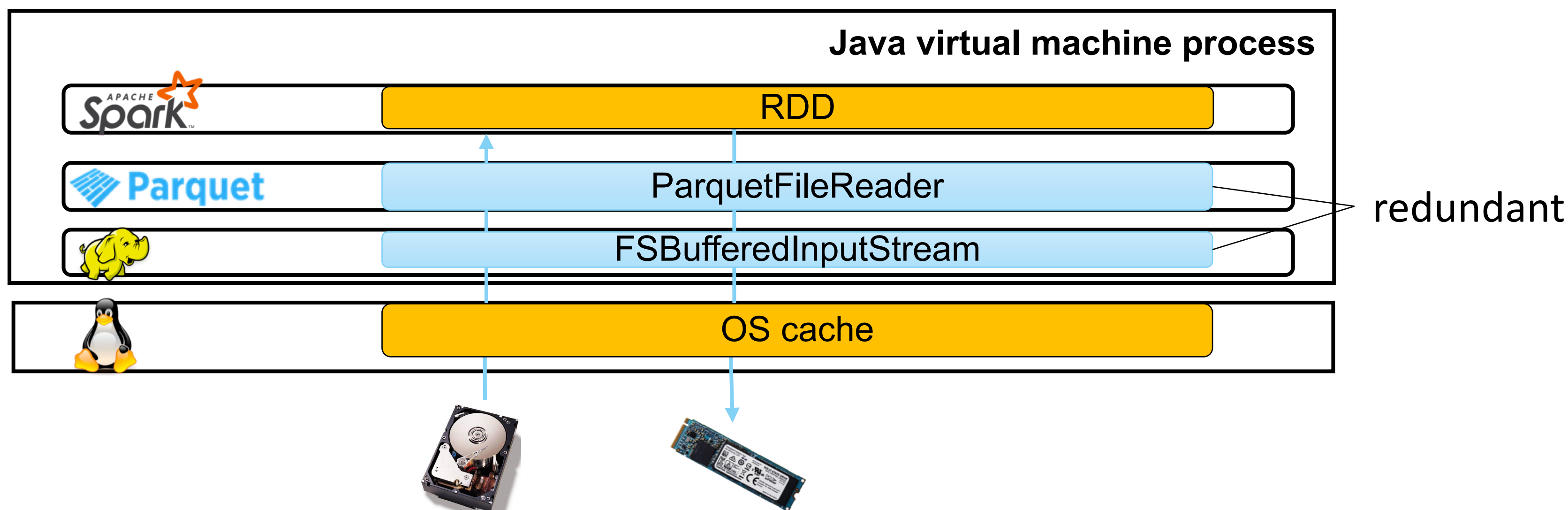
IBM



© 2018 IBM Corporation

Problem: Deep software layering

- Disk I/O for data analytics is processed by multiple software layers
  - Redundant buffers cause inefficient memory usage
- Every software layer does not coordinate each other
  - User memory increases memory pressure and OS cache eviction

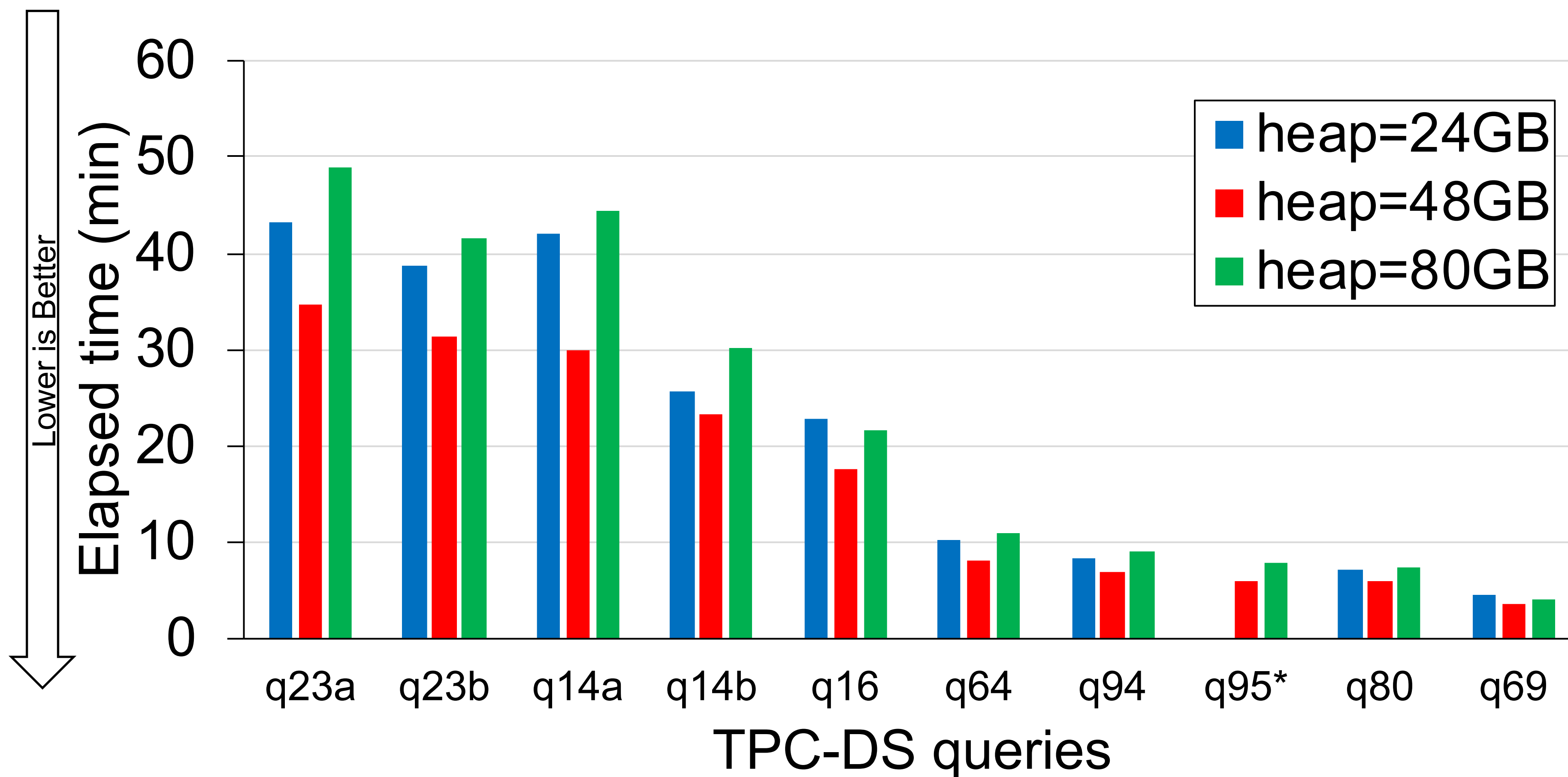


© 2018 IBM Corporation

Preliminary experiments with TPC-DS

IBM

- Large JVM heap caused ~10 mins slowdown in TPC-DS on Spark
  - System memory: 128 GB, Scale factor: 1TB



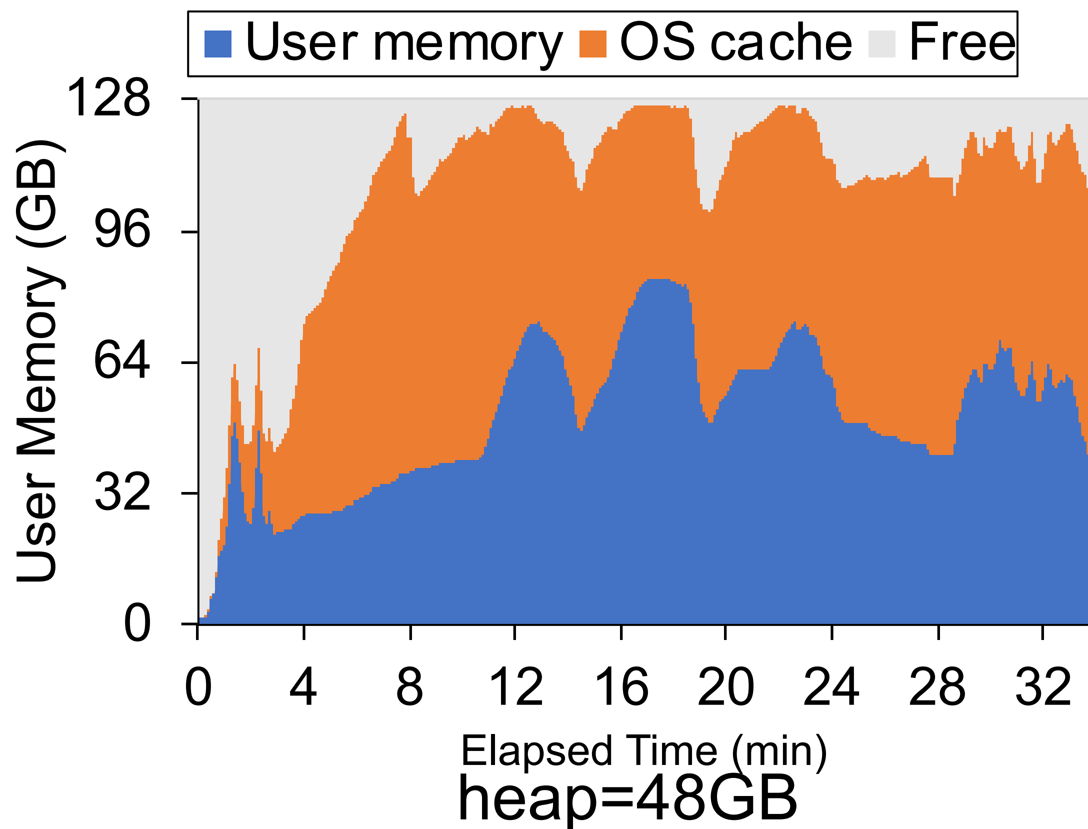
Environment: 1master, 5 slaves x86 Ubuntu16 in IBM Cloud, 32 vCPUs 128 GB RAM, 1 TB SAN disk, TPC-DS scale factor=1000

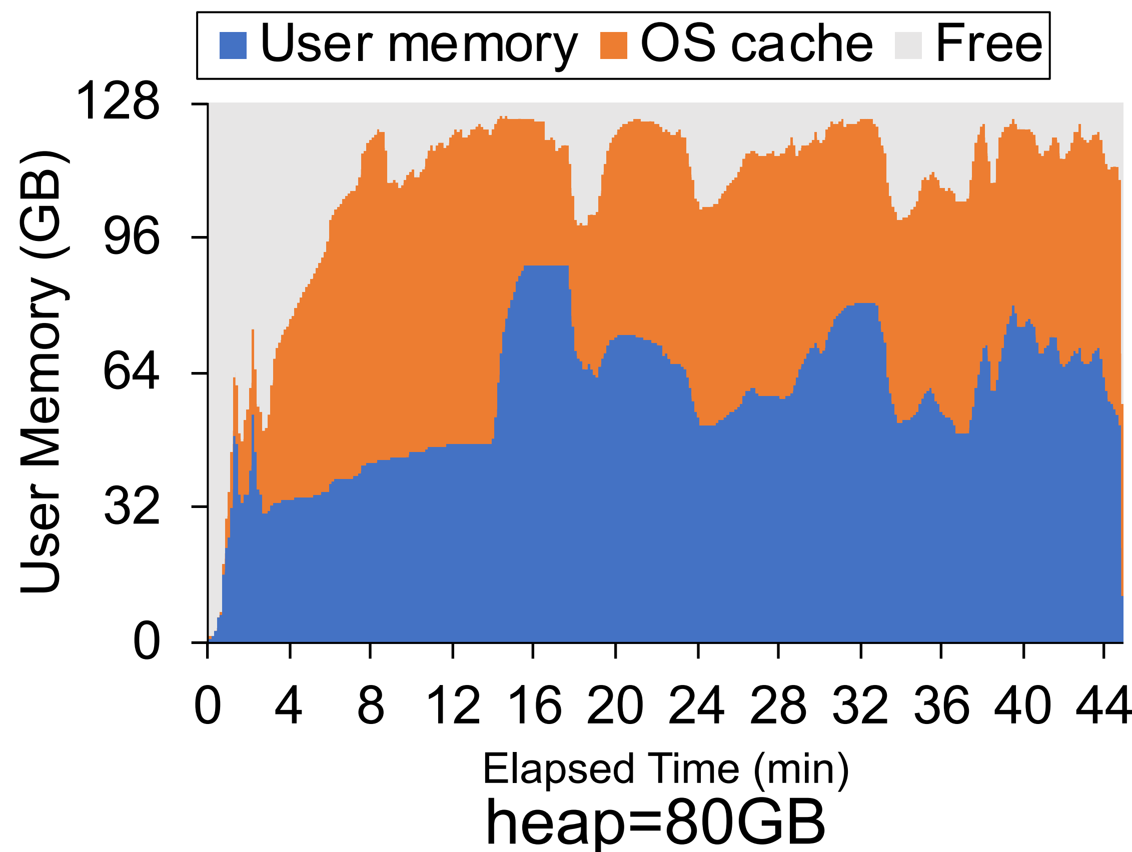
© 2018 IBM Corporation

Slowdowns due to OS-level cache (TPC-DS q23a)

- Large heap evicted OS cache that is for Spark's temporary files
  - Smaller heap caused fewer cache misses but insufficient memory for Spark

- Difficult to coordinate memory management in OS and Spark





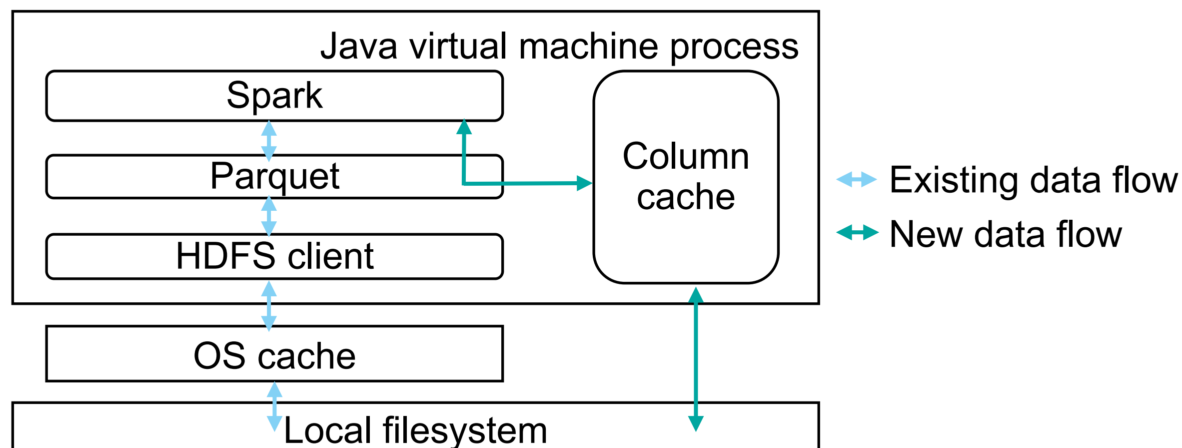
Environment: 1master, 5 slaves x86 Ubuntu16 in IBM Cloud, 32 vCPUs 128 GB RAM, 1 TB SAN disk, TPC-DS scale factor=1000 Spark offheap: 48 GB

© 2018 IBM Corporation

Column cache: Buffer cache for columnar storage on HDFS

IBM

- Unifies buffering and caching logic from OS to Parquet
  - –Leverages query plans to eagerly evict user memory and increase OS cache
  - –Exploits HDFS's optimization for direct I/O on HDFS files
- Provided as a Java library with Parquet compatible APIs
  - –Required 88 LoC changes in Spark
- Modifies only disk *read* for Parquet files in HDFS client processes
  - –Does not require restarting HDFS servers

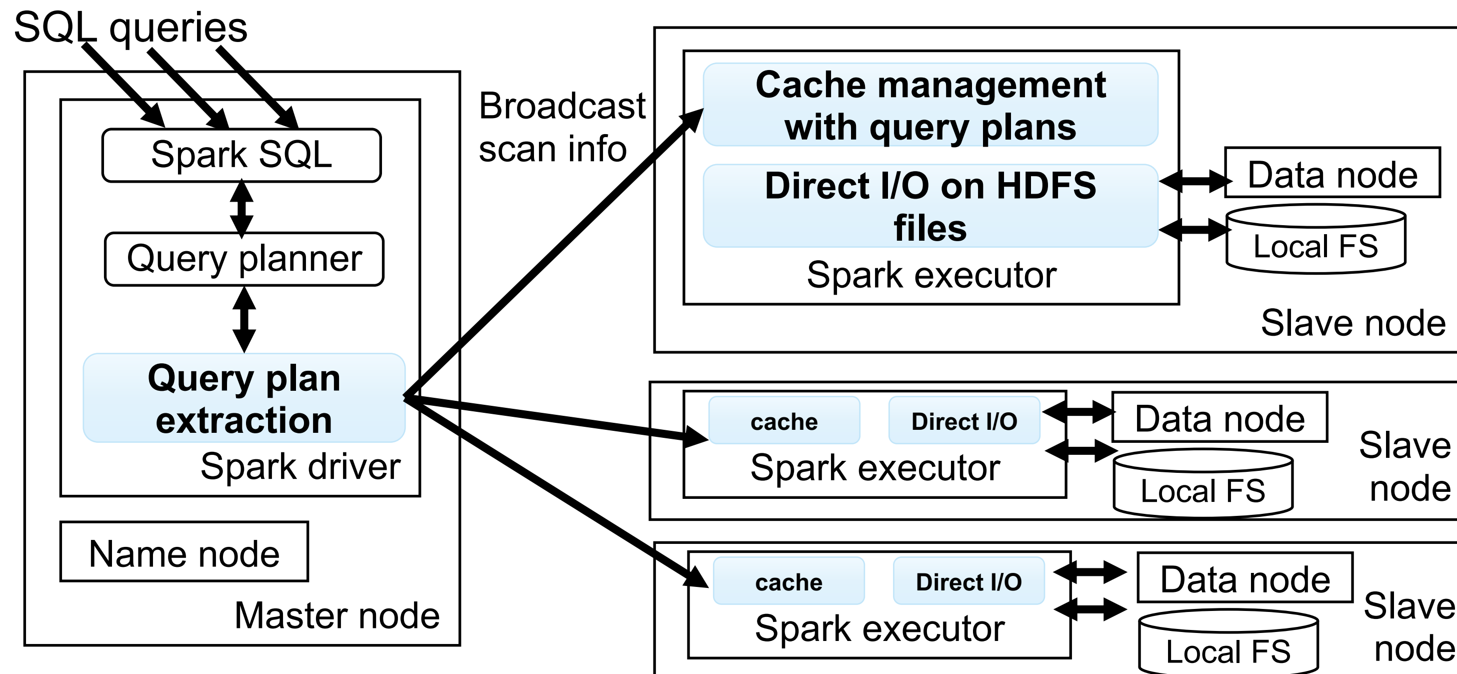


© 2018 IBM Corporation

Approach

IBM

- Cache management with query plans
  - Master node extracts and broadcasts scan info from query plans
  - Use anonymous mmap()/munmap() system calls for explicit memory management
- User-level buffer cache with direct I/O on HDFS files
  - Use JNA to call open(), pread(), fcntl(), close() system calls from Java processes

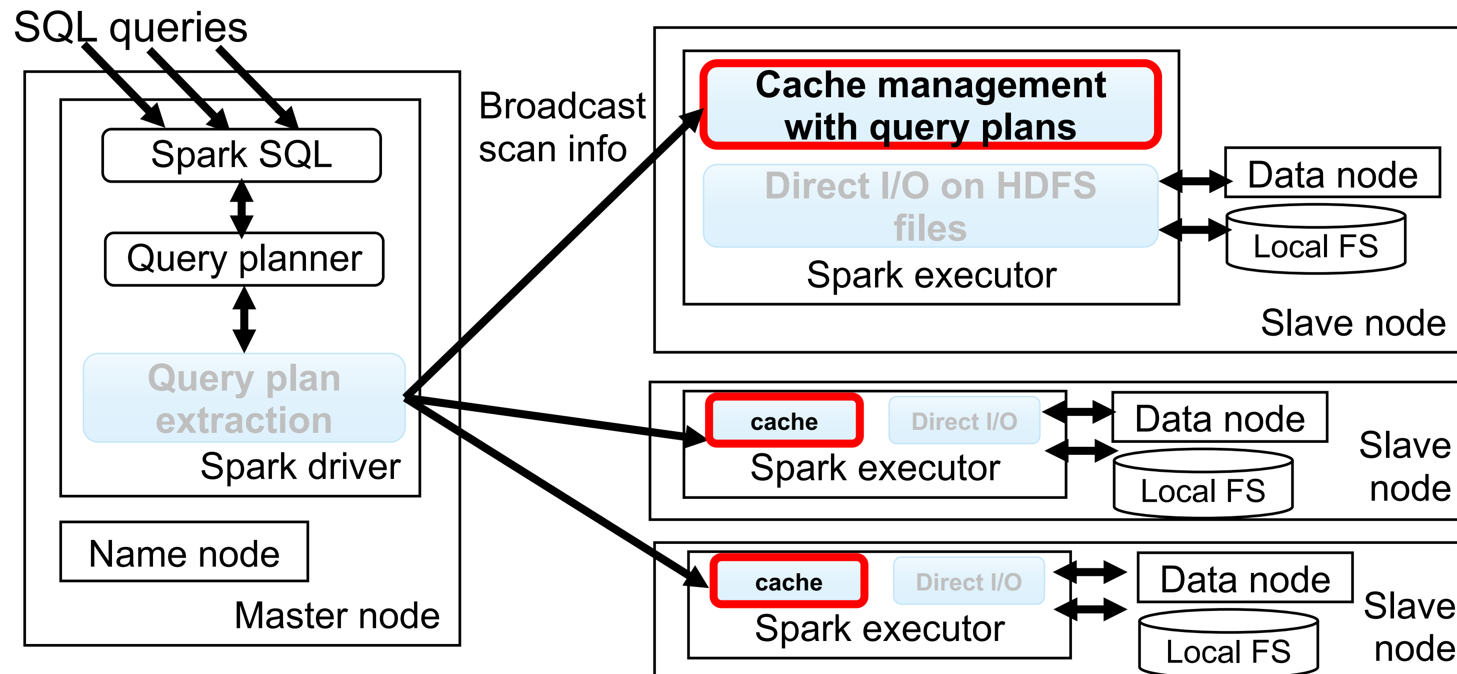


© 2018 IBM Corporation

Approach

IBM

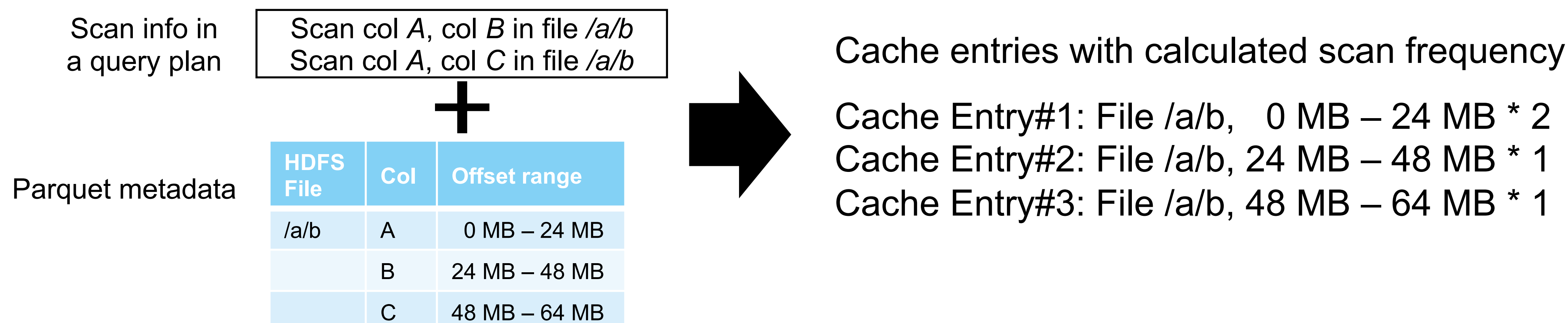
- Cache management with query plans
  - Master node extracts and broadcasts scan info from query plans
  - Use anonymous mmap()/munmap() system calls for explicit memory management
- User-level buffer cache with direct I/O on HDFS files
  - Use JNA to call open(), pread(), fcntl(), close() system calls from Java processes



© 2018 IBM Corporation

Cache management with query plans

- Build cache entries by associating direct I/O result with corresponding offset ranges of HDFS files
- Estimates the frequency of scanned file ranges within query plans
  - Parquet metadata contains how each column is physically placed in an HDFS file
  - Mark a cache entry releasable if its scanned count reaches the frequency
- Monitor system memory and free releasable cache under its pressure
  - Apply LRU policy if two entries have the same frequency

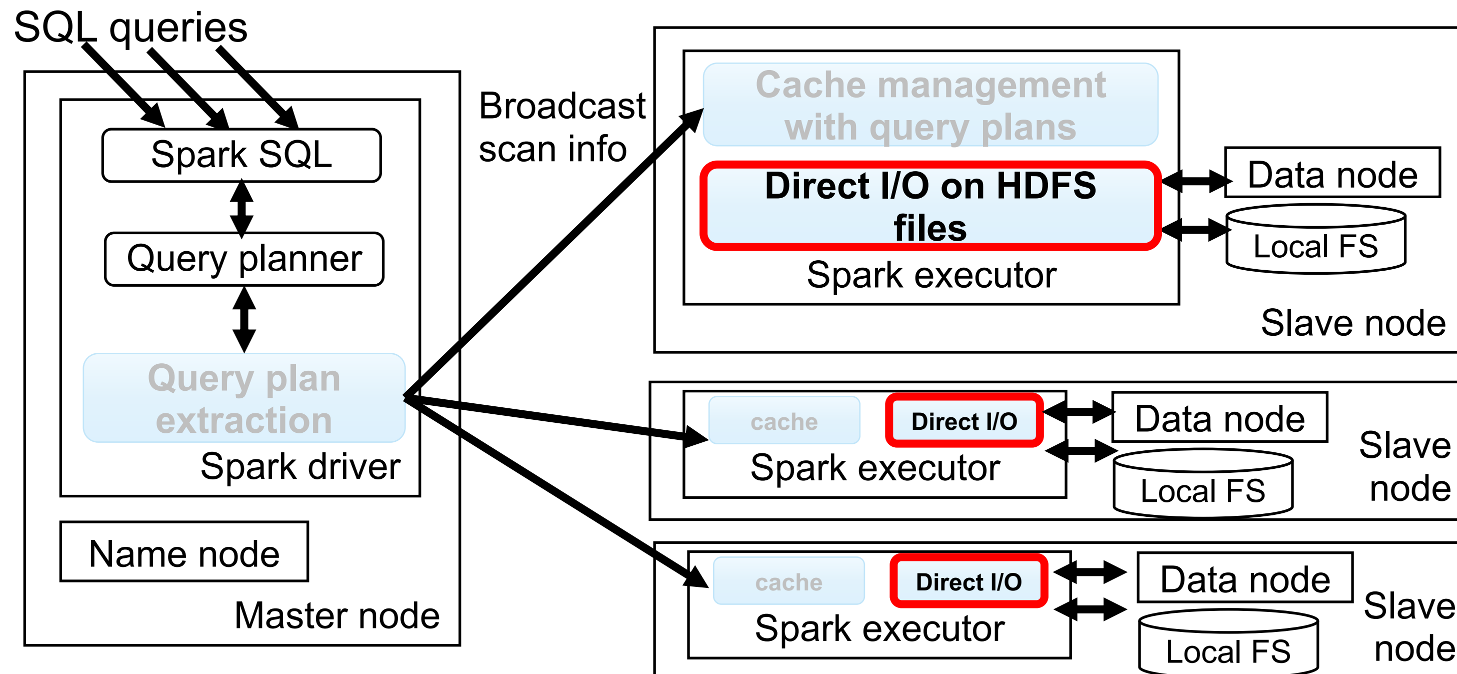


© 2018 IBM Corporation

Approach

IBM

- Cache management with query plans
  - Master node extracts and broadcasts scan info from query plans
  - Use anonymous mmap()/munmap() system calls for explicit memory management
- User-level buffer cache with direct I/O on HDFS files
  - Use JNA to call open(), pread(), fcntl(), close() system calls from Java processes



© 2018 IBM Corporation

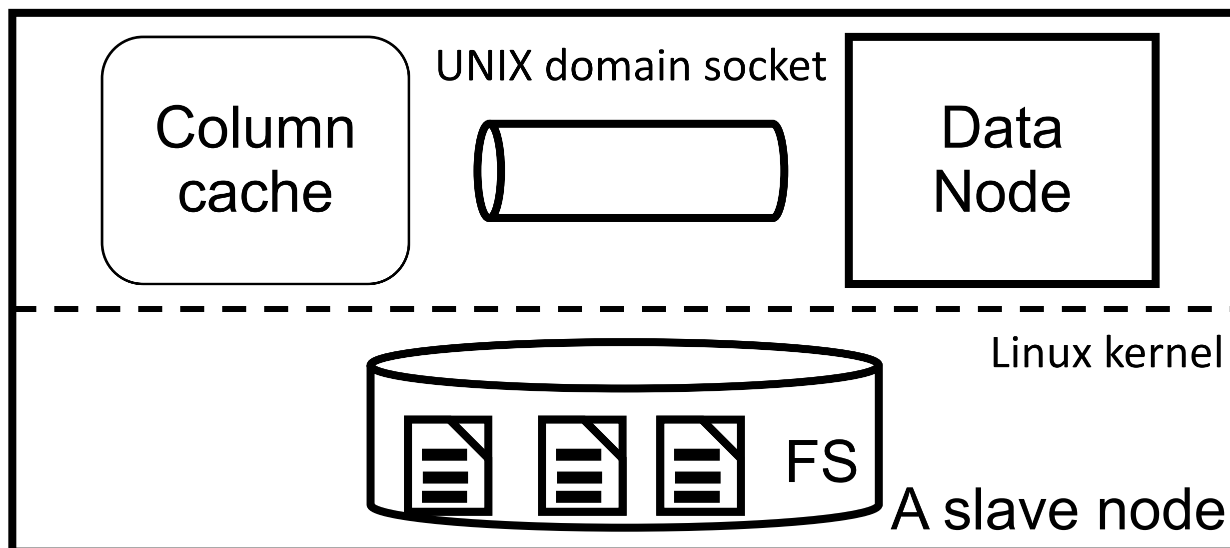
Challenge: direct I/O limitation

- Direct I/O requires a file descriptor that is opened with O_DIRECT
  - HDFS maintains local file information for each HDFS block
- Direct I/O does not merge multiple read() calls unlike normal FS reads
  - Linux page cache merges/splits multiple read() requests at block I/O layer
  - Splitting multiple read() calls hurts performance

© 2018 IBM Corporation

Enabling direct I/O on HDFS files

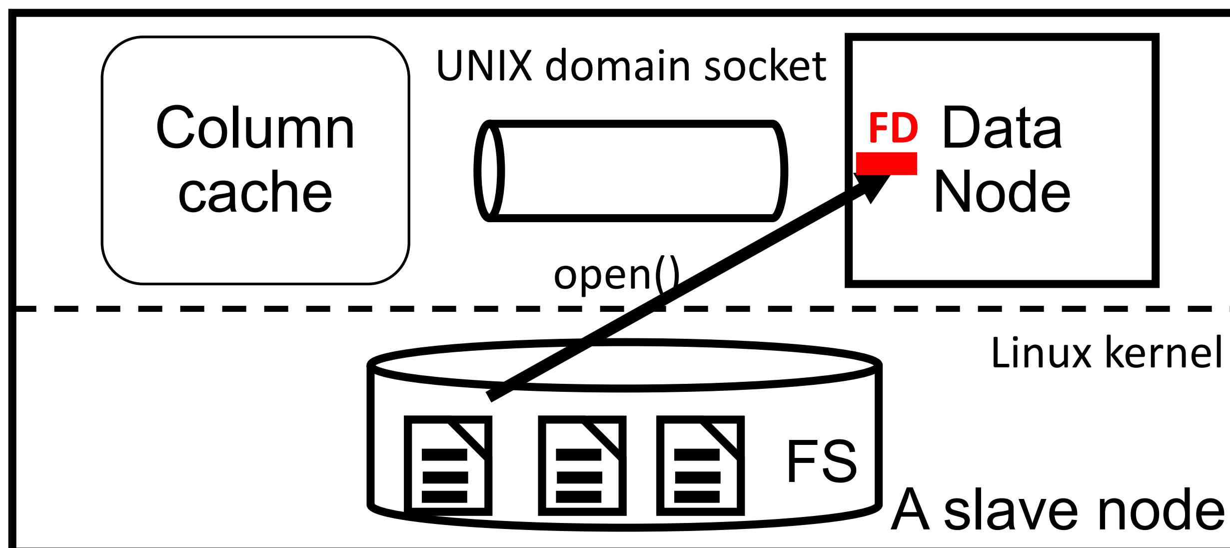
- Enable direct I/O on HDFS files by exploiting HDFS's optimization
  - Short-circuit local reads that optimize reading HDFS files within a local node
  - Data node passes a requested file as an FD via UNIX domain socket
- Use fcntl() to enable O_DIRECT on a file descriptor (FD)
  - Use Java reflection to directly access an FD in an input stream



© 2018 IBM Corporation

Enabling direct I/O on HDFS files

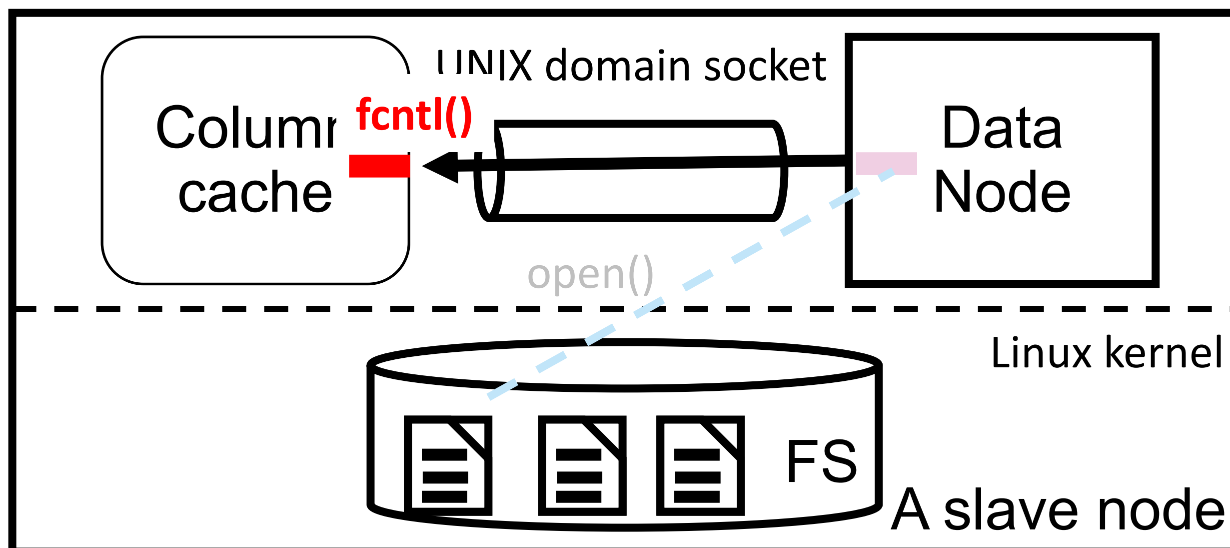
- Enable direct I/O on HDFS files by exploiting HDFS's optimization
  - Short-circuit local reads that optimize reading HDFS files within a local node
  - Data node passes a requested file as an FD via UNIX domain socket
- Use fcntl() to enable O_DIRECT on a file descriptor (FD)
  - Use Java reflection to directly access an FD in an input stream



© 2018 IBM Corporation

Enabling direct I/O on HDFS files

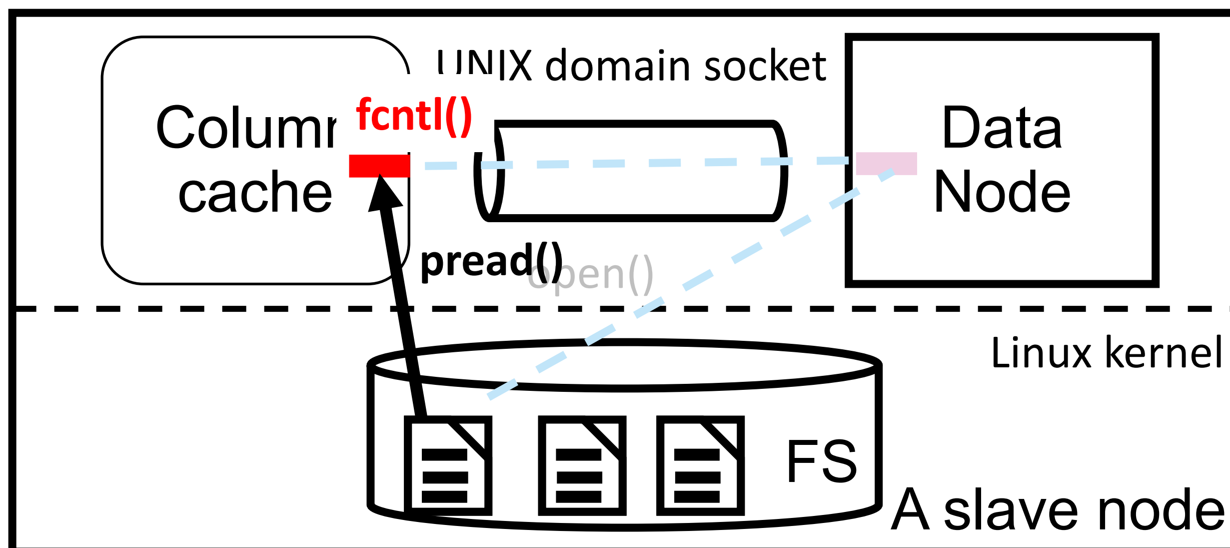
- Enable direct I/O on HDFS files by exploiting HDFS's optimization
  - Short-circuit local reads that optimize reading HDFS files within a local node
  - Data node passes a requested file as an FD via UNIX domain socket
- Use fcntl() to enable O_DIRECT on a file descriptor (FD)
  - Use Java reflection to directly access an FD in an input stream



© 2018 IBM Corporation

Enabling direct I/O on HDFS files

- Enable direct I/O on HDFS files by exploiting HDFS's optimization
  - Short-circuit local reads that optimize reading HDFS files within a local node
  - Data node passes a requested file as an FD via UNIX domain socket
- Use fcntl() to enable O_DIRECT on a file descriptor (FD)
  - Use Java reflection to directly access an FD in an input stream

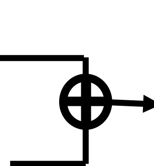


© 2018 IBM Corporation

Optimizing direct I/O on HDFS files

- Explicitly aggregate pread() requests into a single call
  - Existing buffering logic calls multiple pread() for a single column
- Track metadata for Parquet and HDFS in advance of a column scan
  - Parquet metadata contains how each column is physically placed in an HDFS file
- Calculate the exact offset and size of local file reads for each column
- Call a single pread() on the calculated consecutive file area

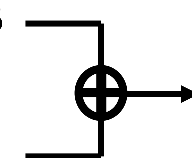
Parquet read: /a/b, column=B



HDFS read:
/a/b, offset=24 MB,length=24 MB

HDFS File	Col	Offset range
/a/b	A	0 MB – 24 MB
	B	24 MB – 48 MB
	C	48 MB – 64 MB

Parquet metadata



Local file read:
/c/blk2, offset=0, length=24 MB

HDFS File	Offset range	Node	Local File
/a/b	0 MB – 24 MB	node1	/c/blk1
	24 MB – 48 MB	node2	/c/blk2

HDFS metadata

© 2018 IBM Corporation

Evaluation

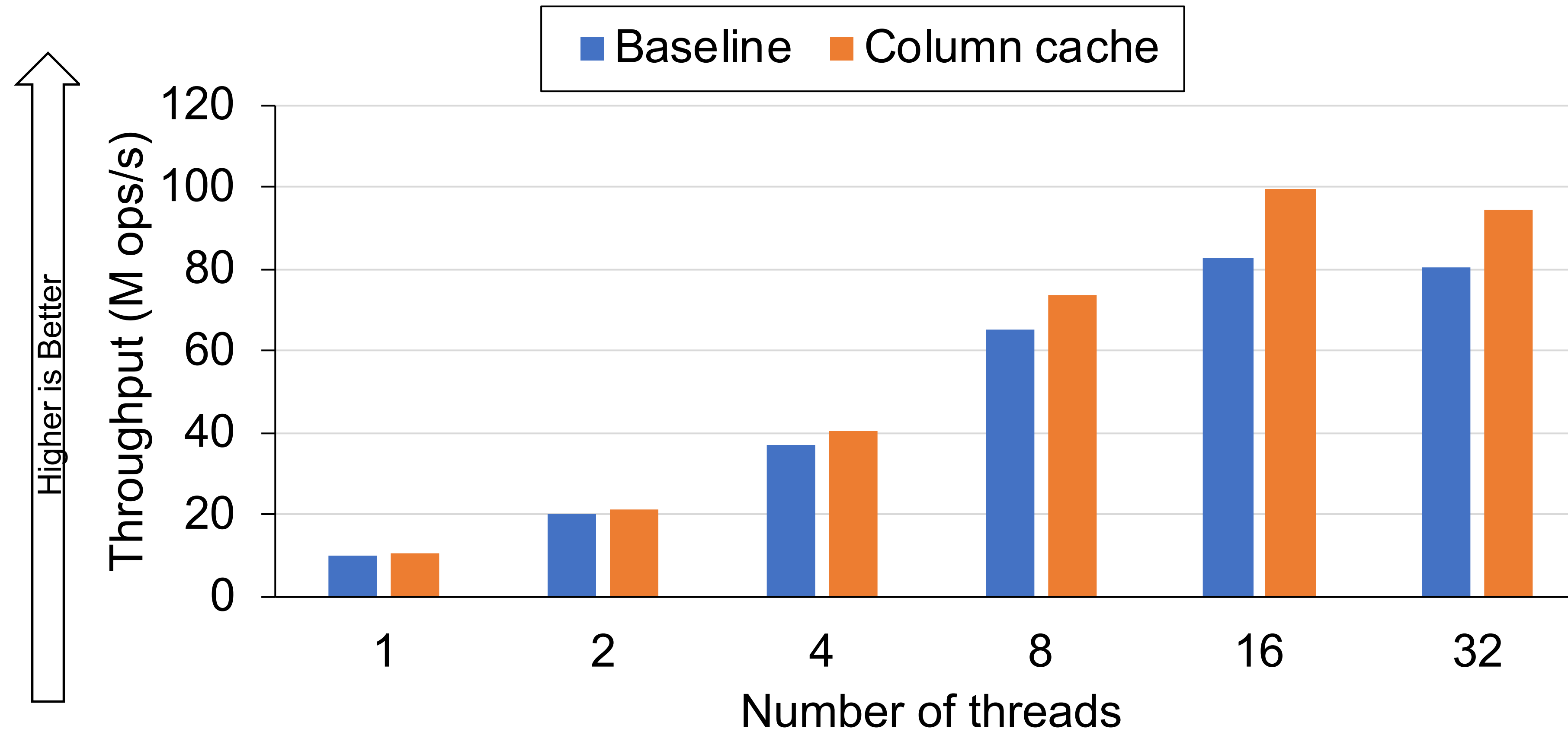
- Microbenchmark
- Single TPC-DS query
- Seven concurrent TPC-DS queries

© 2018 IBM Corporation

Microbenchmark (parallel file read with 256 Parquet files)

IBM

- Column cache speeded up parallel file read with 256 Parquet files
  - Showed 1.2x throughput in 16 threads



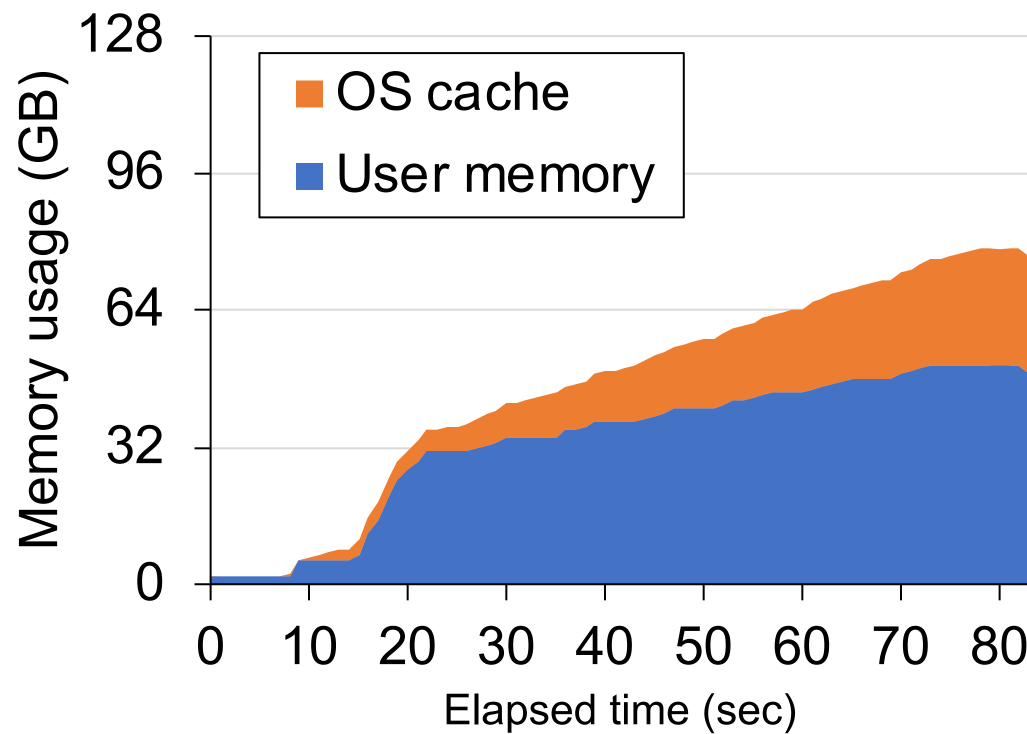
Environment: 1 node, x86 Ubuntu16 in IBM Cloud, 32 vCPUs 128 GB RAM, 1 TB SAN disk, 48 GB JVM heap

© 2018 IBM Corporation

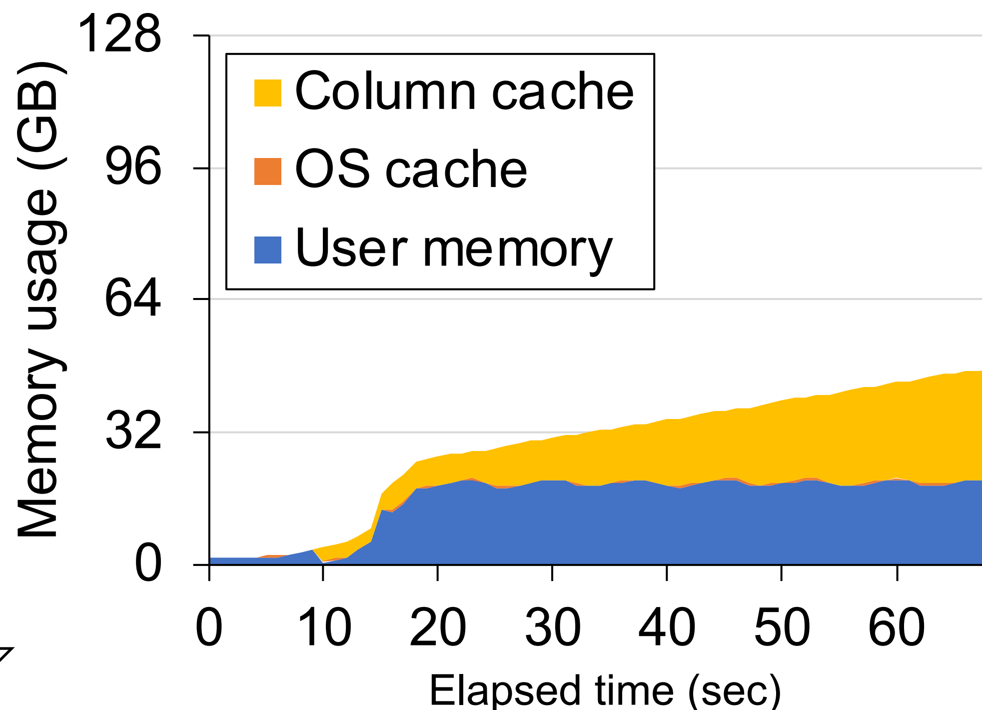
Microbenchmark: Breakdown of Memory usage

IBM

- Column cache reduced memory usage by 14GB
  - Direct I/O removed page cache interleaving during Parquet file reads
  - No cache eviction but the speedup occurred



Baseline



Column cache

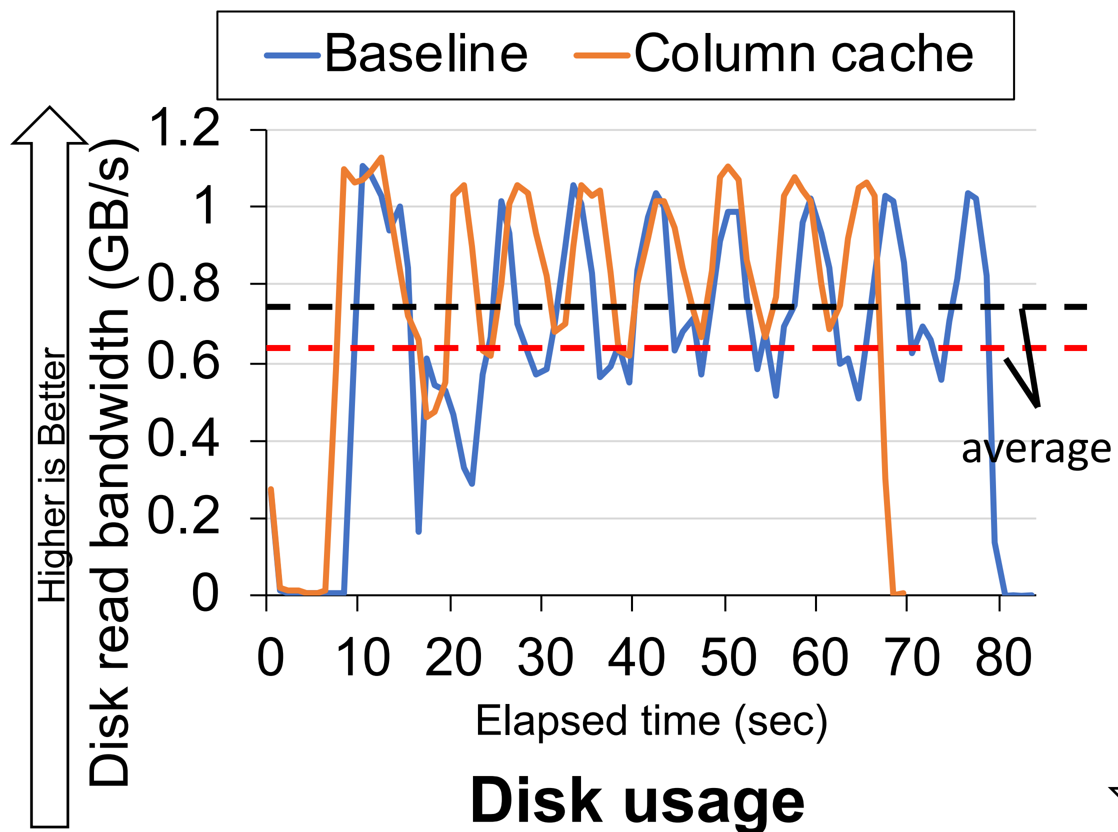
Environment : 1 node, x86 Ubuntu16 in IBM Cloud, 32 vCPUs 128 GB RAM, 1 TB SAN disk, 48 GB JVM heap

© 2018 IBM Corporation

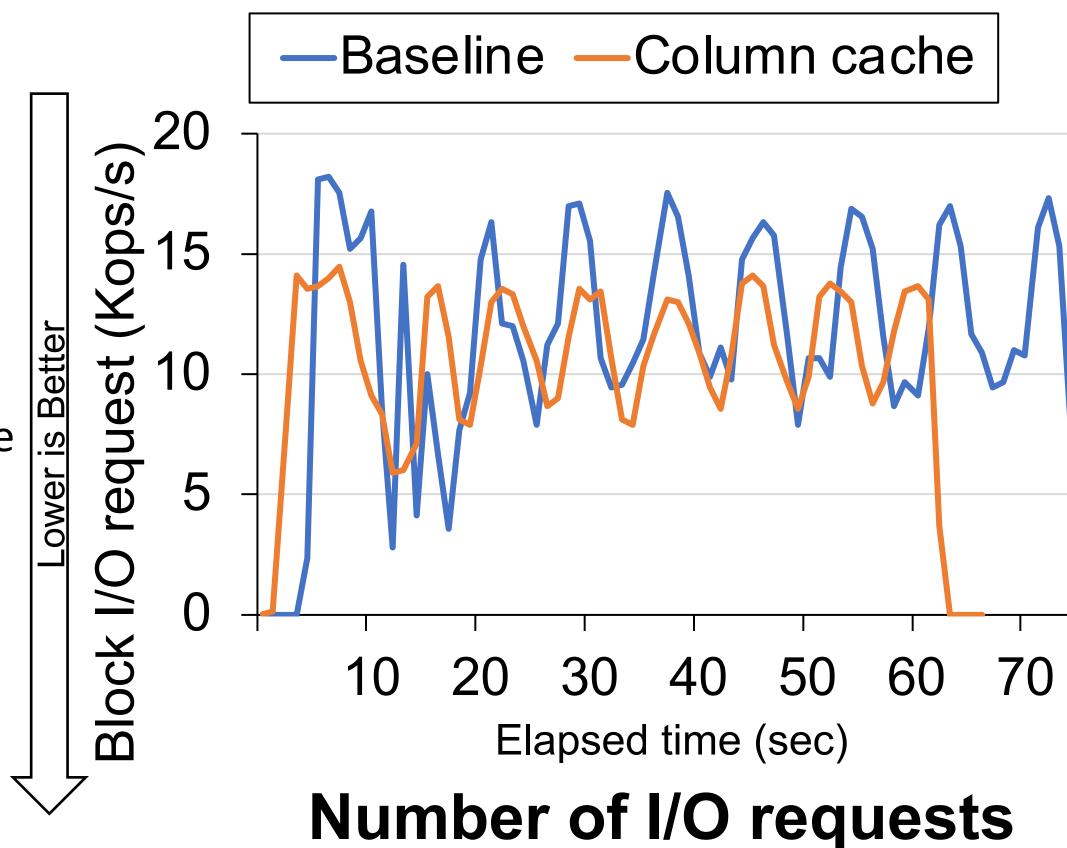
Microbenchmark: Breakdown of disk usage

IBM

- Column cache increased average disk I/O bandwidth
  - –Aggregated direct I/O decreased number of requests per second



Disk usage



Number of I/O requests

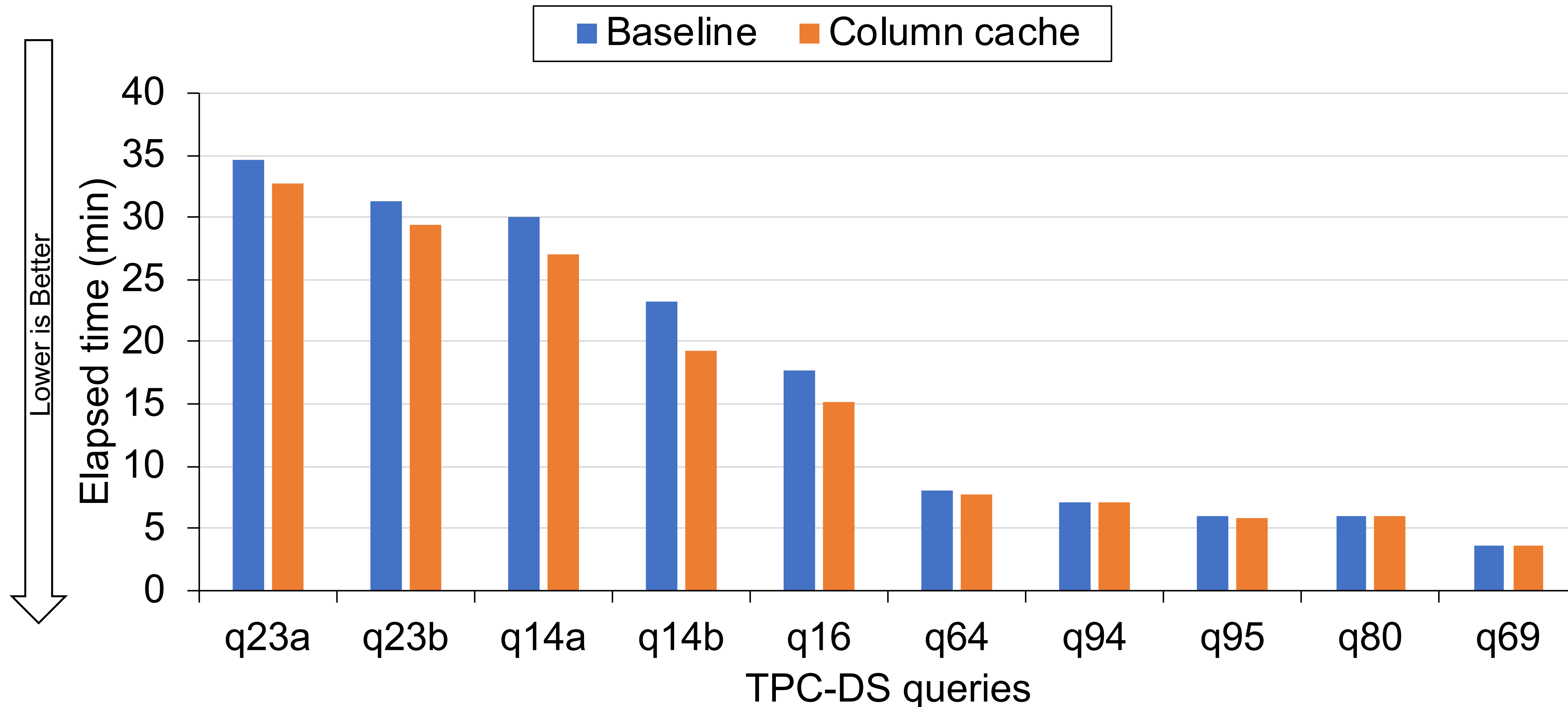
Environment : 1 node, x86 Ubuntu16 in IBM Cloud, 32 vCPUs 128 GB RAM, 1 TB SAN disk, 48 GB JVM heap

© 2018 IBM Corporation

TPC-DS queries

IBM

- Column cache reduced query processing time by up to 4 min compared to heap=48GB

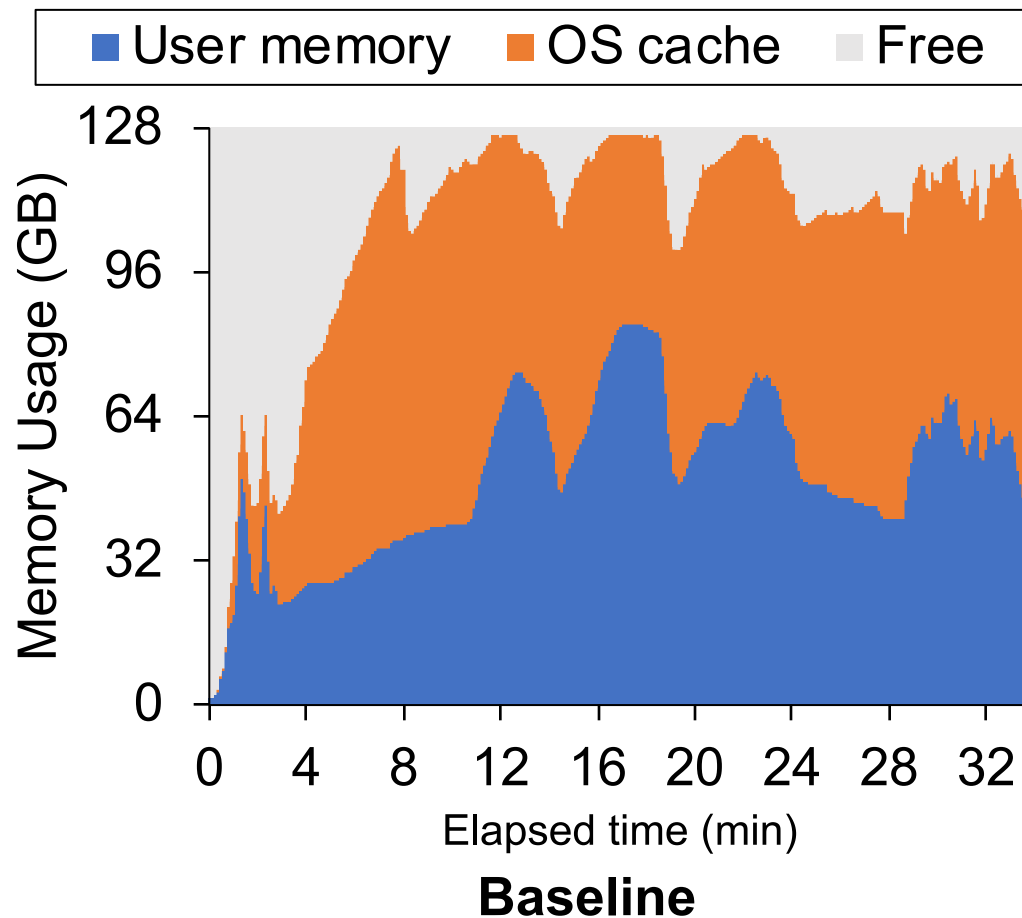


Environment: 1master, 5 slaves x86 Ubuntu16 in IBM Cloud, 32 vCPUs 128 GB RAM, 1 TB SAN disk, TPC-DS scale factor=1000, 48 GB JVM heap

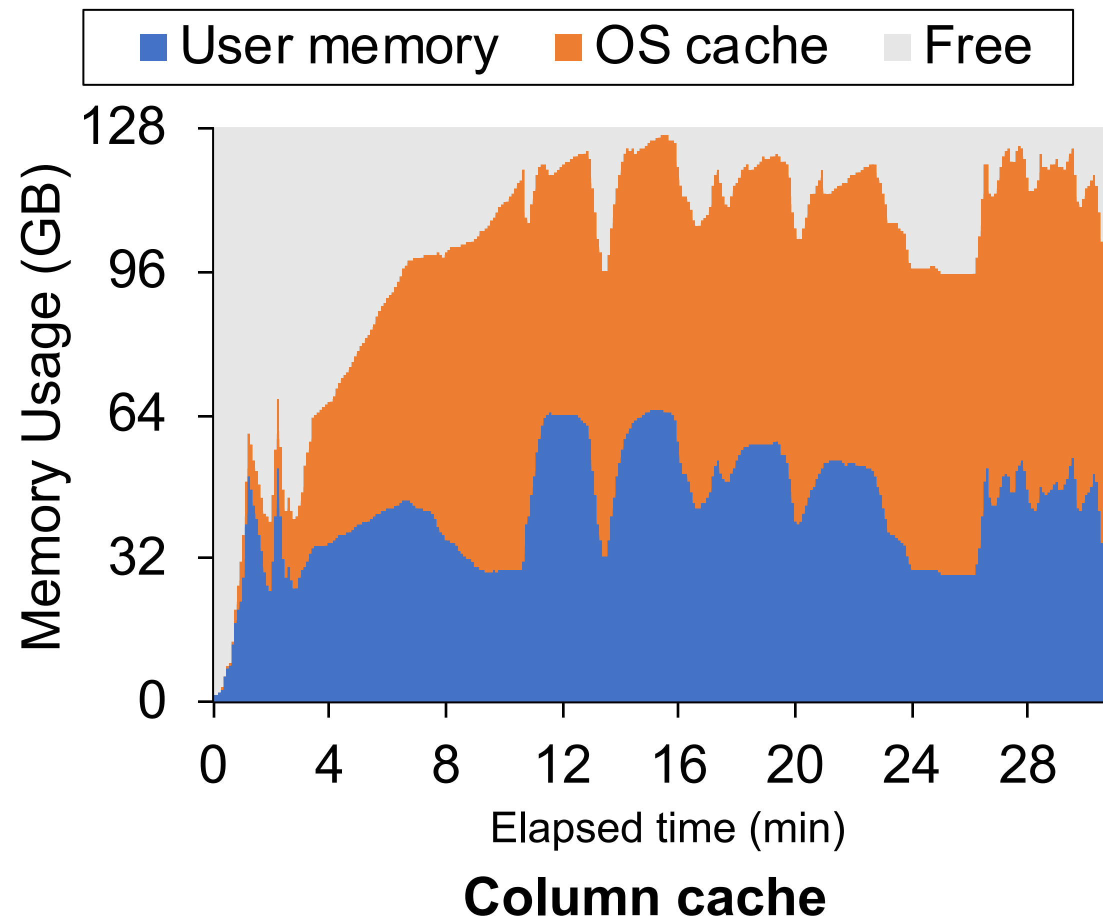
© 2018 IBM Corporation

Q23a: Memory usage

- Less-frequently-used columns are evicted from user memory
- Increased page cache by 18%, and reduced disk reads by 43%



Baseline



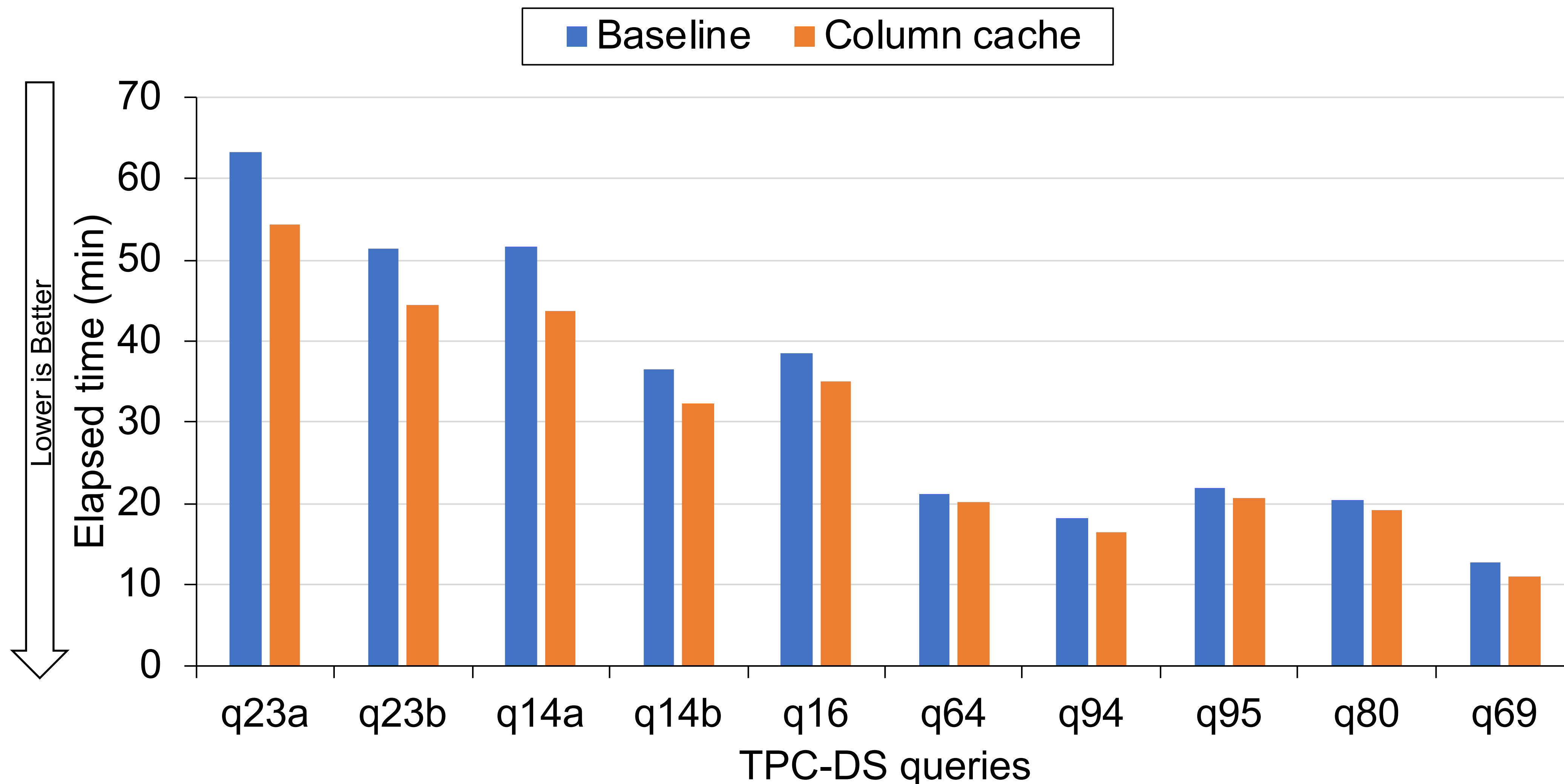
Column cache

Environment: 1master, 5 slaves x86 Ubuntu16 in IBM Cloud, 32 vCPUs 128 GB RAM, 1 TB SAN disk, TPC-DS scale factor=1000, 48 GB JVM heap

© 2018 IBM Corporation

Seven concurrent TPC-DS queries

- Column cache reduced query processing time by up to 9 min



Environment: 1master, 5 slaves x86 Ubuntu16 in IBM Cloud, 32 vCPUs 128 GB RAM, 1 TB SAN disk, TPC-DS scale factor=1000, 48 GB JVM heap

© 2018 IBM Corporation

Summary

- Introduced column cache, buffer cache for columnar storage on HDFS
  - Unifies buffering and caching logic from OS to Parquet
  - Leverages query plans to eagerly evict user memory and increase OS cache
  - Exploits HDFS's optimization for direct I/O on HDFS files
- Speeded up TPC-DS queries of Spark SQL up to 9 min
  - Increased page cache usage by 18%, and reduced disk reads by 43%

© 2018 IBM Corporation

© 2018 IBM Corporation