Petabyte-scale Distributed File Systems in OSS: KFS evolution

Sriram Rao

Partner Scientist Manager, CISL@Microsoft

September 20, 2015

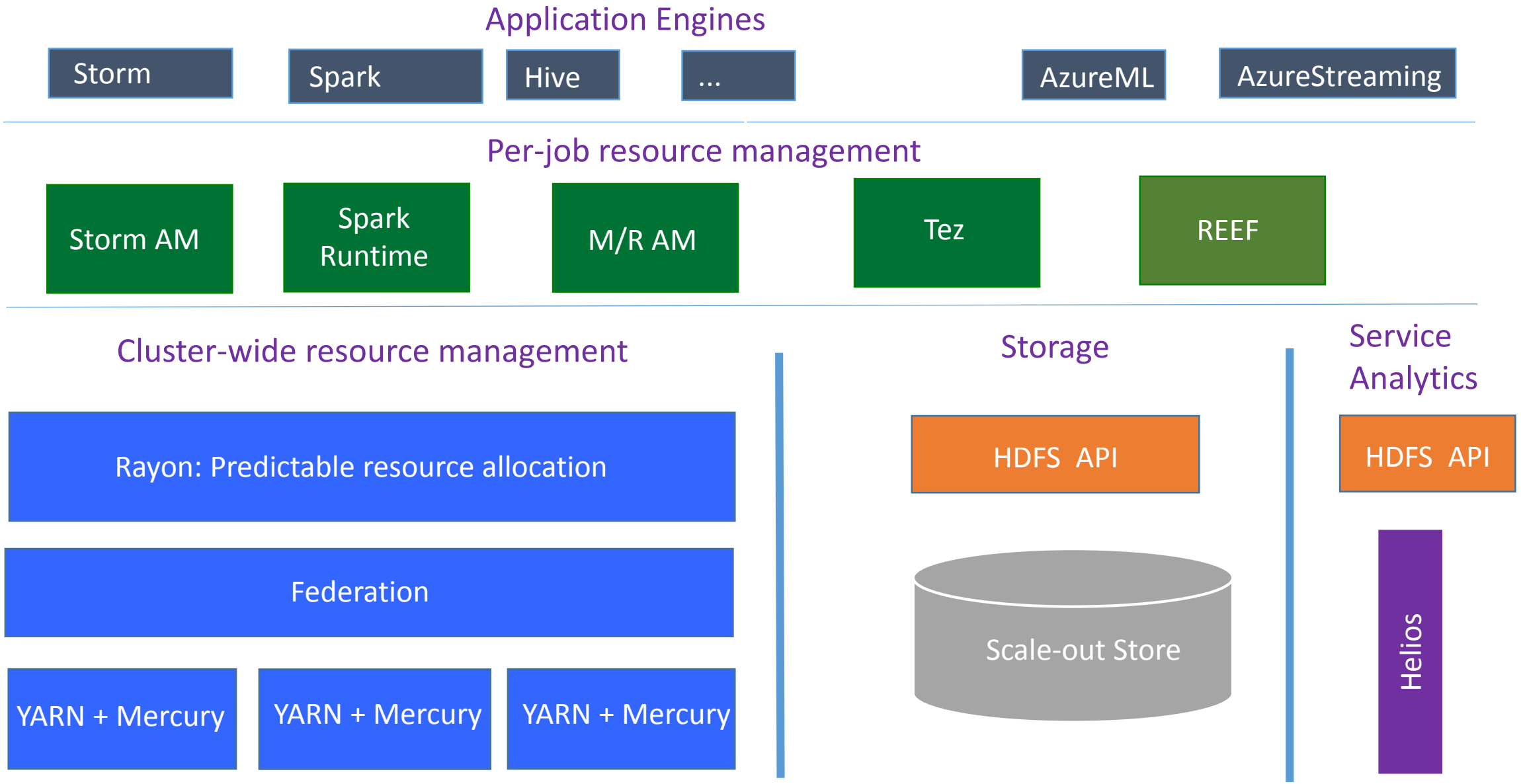
Talk Overview

- CISL and what we do
- KFS (Kosmos Filesystem)

CISL: Cloud and Information Services Lab

- Started in May 2012
- What is CISL?
  - CISL is an applied research group with two sub-teams
    - CISL Systems: Sriram and team
    - CISL Data Science
- What does CISL Systems do?
  - Build prototypes, publish papers, write production code and contribute to OSS
    - Team members have Hadoop committer privileges
  - CISL systems works closely with Microsoft Big Data teams

BigData Analytics Stack



Building KFS, Releasing it as OSS, and Evolving it for massive scale

Talk Outline

- KFS Overview
- KFS as OSS project
- Filesystem support for Sorting data: atomic append
- Lessons from deploying KFS in production settings
- Summary

KFS Origins

- Kosmix (now, part of @Walmart labs) needed backend infrastructure for search engine
  - Kosmix was a search engine startup focused on “vertical search”
  - “Distributed Filesystem” for storing web-map was a set of Perl scripts
    - Login to machine; copy partitions that were needed
- Choices in 2006: Nothing we could take and use…
  - Google filesystem: Proprietary; but paper is published
  - HDFS: Yahoo was starting to build
    - Hadoop…what is that???
  - Cosmos: Didn’t know about it…
- So…at Kosmix, we chose to build our own based on ideas from GFS paper

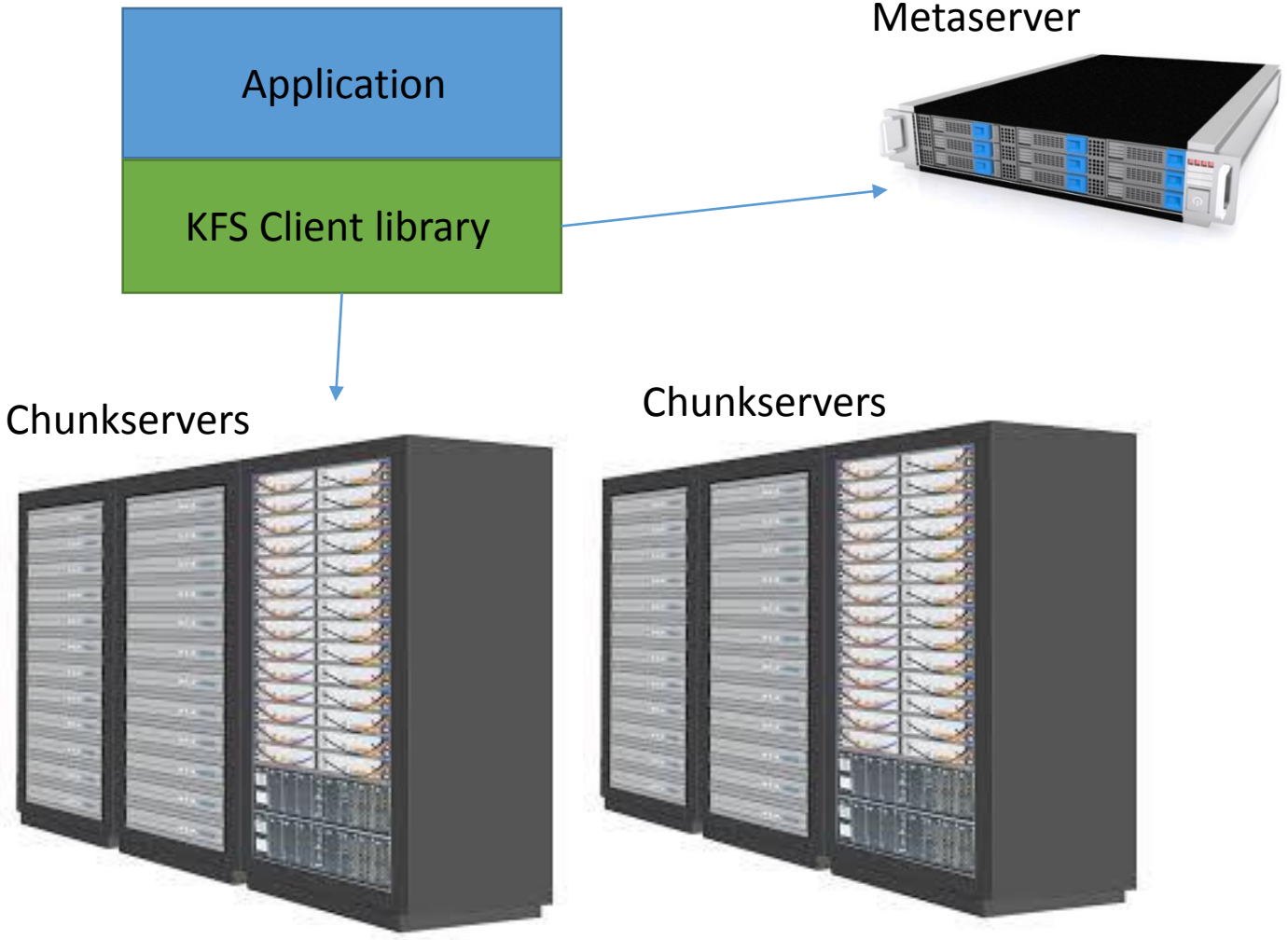
Web-scale workloads

- Workloads characteristics:
  - Few millions of large MB/GB size files
  - Files are accessed (mostly) sequentially
    - Random I/O is rare
  - Files are written once; read many, many times
    - Streaming algorithms that do few passes over massive data sets
- Dataset (typically) will not fit in RAM
    - Need function shipping and data shipping
- Need massive storage and compute capability at low cost => Build using commodity hardware
  - At scale, 1K cluster with 20TB/node => 2PB of space

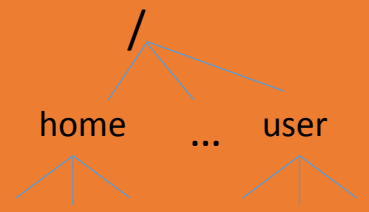
Key ideas from GFS Paper

- Construct a global namespace by decoupling storage from filesystem namespace
  - Build a scale-out filesystem by aggregating the storage from individual nodes in the cluster
  - To improve performance, stripe a file across multiple nodes in the cluster
    - Blocks of a file are “chunks”
    - Chunks are fixed in size (typically, 64MB)
  - Use replication for tolerating failures
    - Chunks are replicated across nodes
    - Can recover from bit errors, drive failures, node failures
- Filesystem runs as a user-land application

KFS Architecture

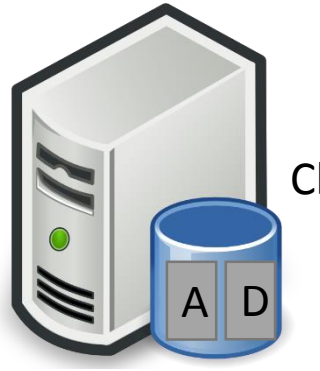


Key data structures in RAM



File Id	Blocks
10	A, b, c
11	D, e, f
12	g

Block Id	Locations
A	C1, C2, C3
D	C1, C4, C5
...	



Chunkserver

Stores chunk files on disk

KFS and Other FS'es

- KFS metaserver ≡ HDFS Namenode
- KFS chunkserver ≡ HDFS Datanode
- KFS client library ≡ HDFS Filesystem client

Feature	GFS	HDFS	KFS	Cosmos
Multi-writer atomic append	Yes	No	• Multiple writers per file • Multiple blocks can be concurrently appended to (for Sorting)	Multiple writers per stream (for ingest)
Erasure coding	?	No	• **Yes**: Replicate data using erasure codes	No
Redundant meta data servers	Yes	Yes	No	Yes
Snapshots	?	Yes	No	No

KFS Timeline…

- Started at Kosmix in March 2006
- Initial implementation done by November 2006
  - Supported 3x replication, leases, scalable metaserver, …
  - Tested on a 8-node cluster ☺…Woo hoo…Ready for “big data”
- KFS work “paused” in February 2007
- Revived and released as OSS in September 2007
  - Added a “HDFS shim” over KFS to enable Hadoop M/R jobs to run with KFS
  - To play nice with Hadoop ecosystem, released with Apache license
- And then…learn the OSS rhythm…

Skrentablog

« My compiler vs. the monkeys | Main | Beautiful presentations: Jon Bentley's quicksort video »

Kosmix releases Google GFS workalike 'KFS' as open source

Search startup Kosmix has released a C++ implementation of the Google File System as open source. This parallels the existing Hadoop/HDFS project which is written in Java. The Kosmix team has deep engineering talent, including a strong track record, and having recently built a web-scale crawler and search engine from scratch. Google has a set of tools that the rest of the industry needs in order to compete...it's cool that folks are stepping up to the task and leveraging the open source model to try to provide some balance.

Every startup that scales beyond a single machine needs platform technology to build their application and run their cluster. If enough folks adopt the code and contribute, the hope is that it could become something like the gcc/linux/perl of the cluster storage layer.

Posted on September 27, 2007 10:12 PM | Permalink

KFS as OSS project (in 2007…)

- I tried evangelizing KFS…
- Presented KFS to folks at Powerset, Facebook, …
- Hadoop and HDFS were becoming the rage
  - Hadoop was at version 0.15.3 by the time KFS was OSS'ed
  - Yahoo was a big supporter of Hadoop
- Hard lesson: Building a community in OSS without a corporate sponsor can be hard ☹

Konrad Feldman Dec 9, 2007

To sriramsrao@yahoo.com

Sriram,

Hi, sorry for the out of the blue email. I heard about the Hadoop FS project you've been working through a friend [redacted] and it sounded like there could be some interesting overlap in the things that we are also working on.

If you have some time available and would like to hear a little more about Quantcast let me know and we can set something up.

Regards,

Konrad

Konrad Feldman
CEO | Quantcast Corp.
917.302.8999

Quantcast is the world's first open Internet ratings service.
See your stats at www.quantcast.com.

Reply, **Reply All** or **Forward** | **More**

quαntcast

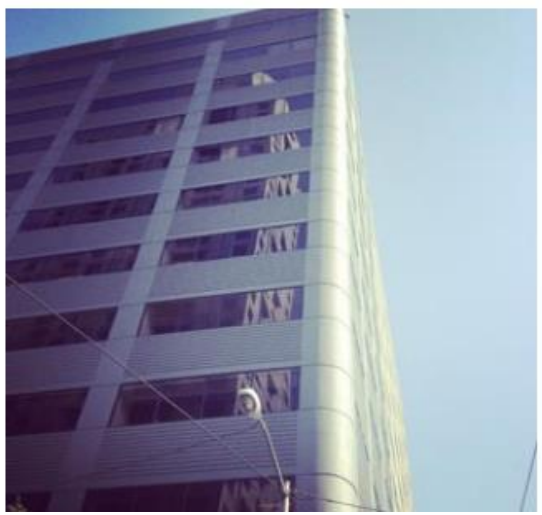
Explore

Search Quantcast Profiles

Company

About | Careers | Engineering | Legal | Inquiries

About Quantcast



Quantcast is a technology company specialized in real-time advertising and audience measurement. As the pioneer of direct audience measurement in 2006, Quantcast has today the most in-depth understanding of digital audiences across web and mobile, allowing marketers and publishers to make the smartest choices as they buy and sell the most effective targeted advertising on the market. Quantcast is dedicated to making display as relevant and effective as search, and currently delivers outstanding advertising campaigns for the world's leading advertisers and publishers, and brings accurate audience measurement to over 100 million digital destinations.

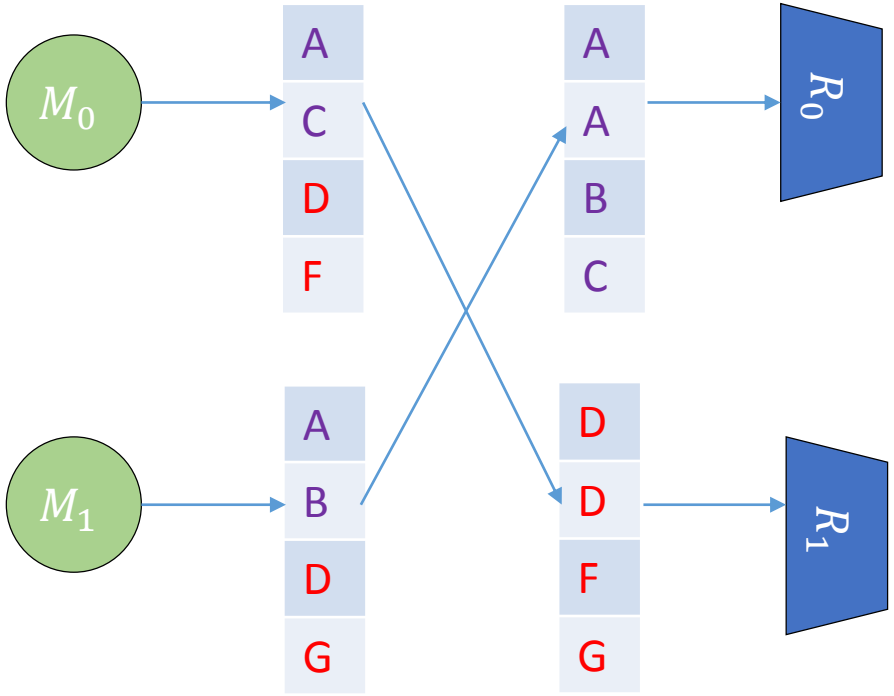
KFS@Quantcast

- In June 2008 they built a ~1000 node cluster with “flat” connectivity
  - Planned to run KFS and HDFS
  - Machine config: 4 cores, 8GB RAM, 4x1 TB drives, 1Gbps NIC
- KFS deployed, initially, on a 120-node cluster for storing logs (~100TB)
- Over the next few months…
  - Fixed the first set of KFS performance issues
  - Deployed KFS on the full cluster to be used as a “backup” system: archive data from HDFS to KFS
- Built trust in KFS…
- But, how to get KFS to “first class” citizenry status?

Building Trust in KFS…

- Focus on the one application that the cluster was used for…
- Hadoop Map-Reduce!
- Heart of Map-Reduce is essentially a distributed merge sort
  - Map output is partitioned by key
  - Reducer input is ordered by key

What if we built a sort engine?



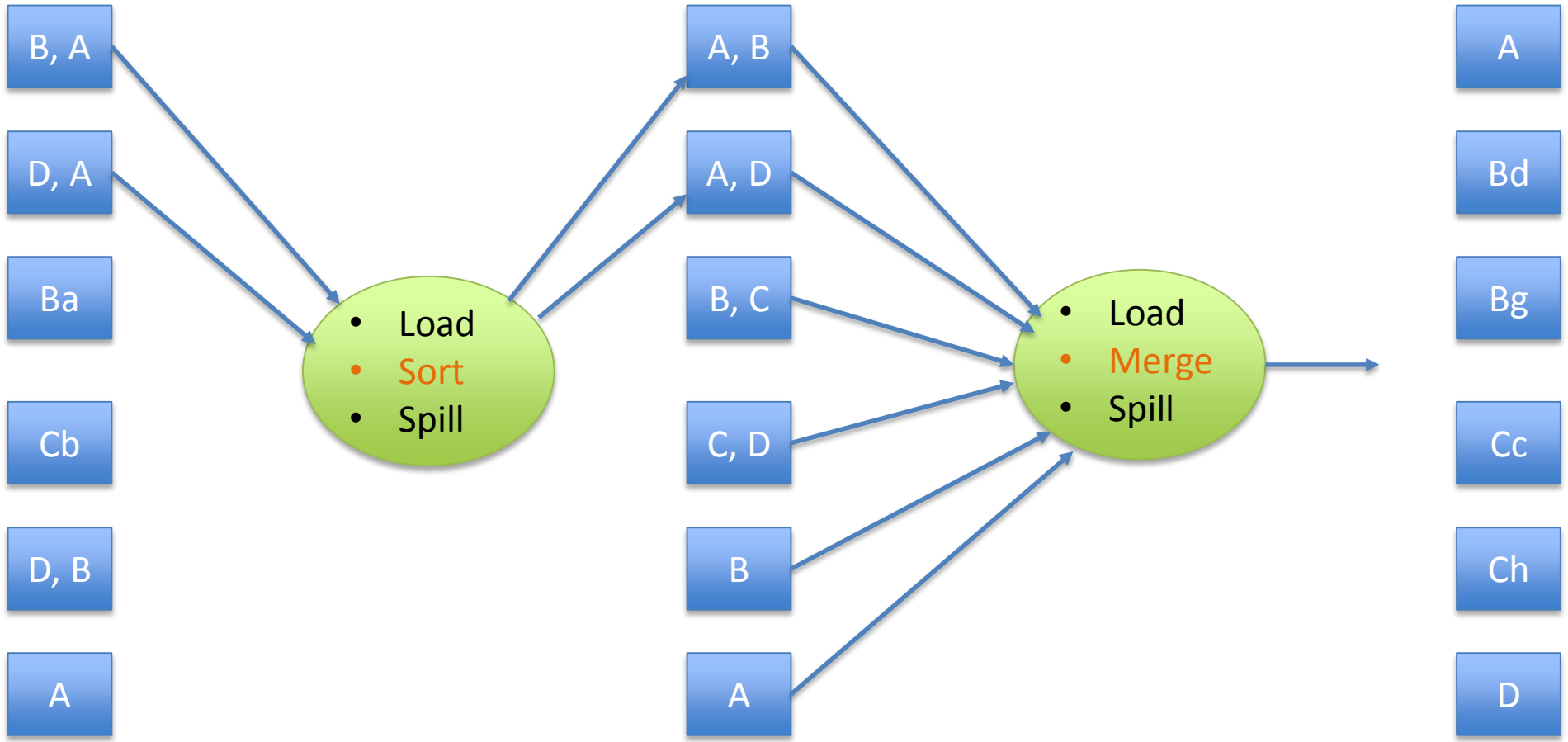
Building Trust in KFS: Sorting

- Map-Reduce jobs at Quantcast involved “large” (1-10TB) sorts
  - Few hundred jobs per day
- Typical sort volume per day (in 2008): ~100TB
- They had replaced Hadoop’s shuffle with something proprietary
  - Mostly worked…couldn’t scale beyond ~30TB in a single job
- Our plan…build a highly performant/scalable sort engine using KFS
  - Data is written to disk and read back straightaway as job progresses…
  - Mindset: We will discover problems. Iterate…Iterate…(working->working) ☺
  - Net: If KFS-sort can handle large sorts, can make it the primary FS

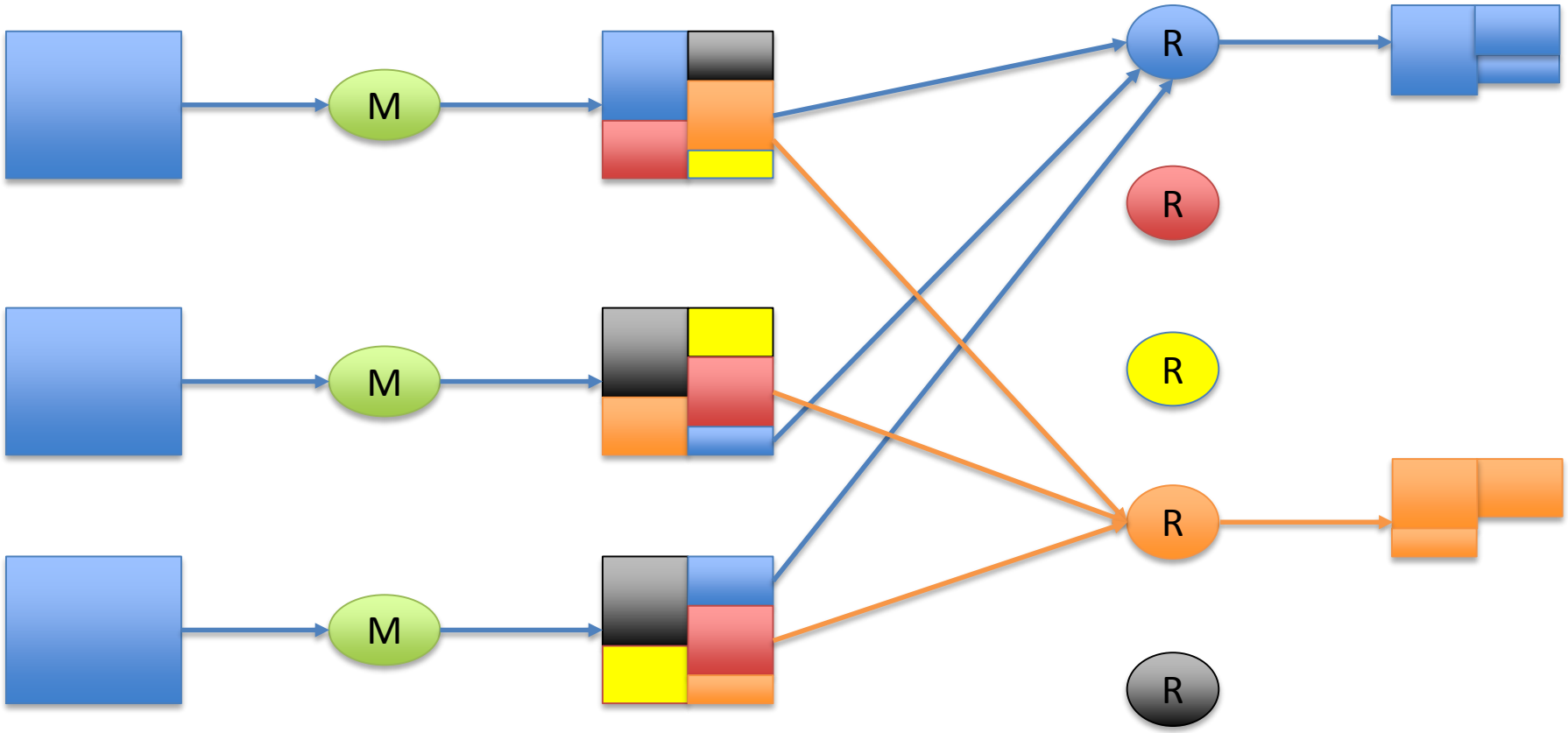
Extending KFS to handle scalable merge-sort (Sailfish)

(Joint work with Raghu Ramakrishnan, Mike Ovsiannikov, and Damian Reeves)

External Merge Sort: 1-Process



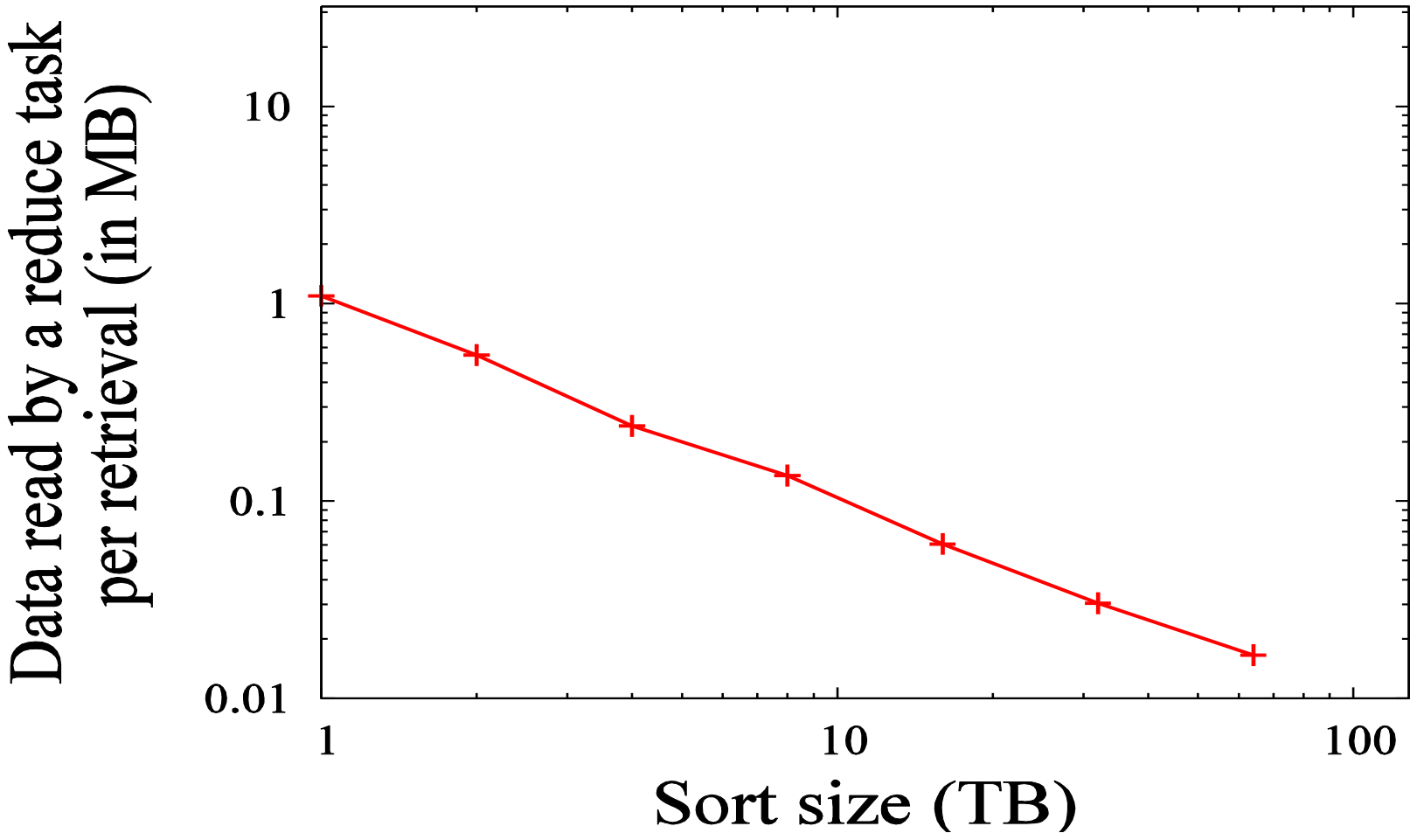
External Merge Sort @ Scale: Divide to Conquer



As the volume of data scales…

- Scale # of M's and R's
  - Sort: Want data to be read once; spilled to disk once
  - Merge: Want to do 1-pass merge of each partition
- But…
  - Since input is unsorted…any M can generate data for any R
  - This means…each R has to pull data from each M
- Distributed merge sort is known to be seek intensive
  - # of seeks α M * R => I/O's become small and random

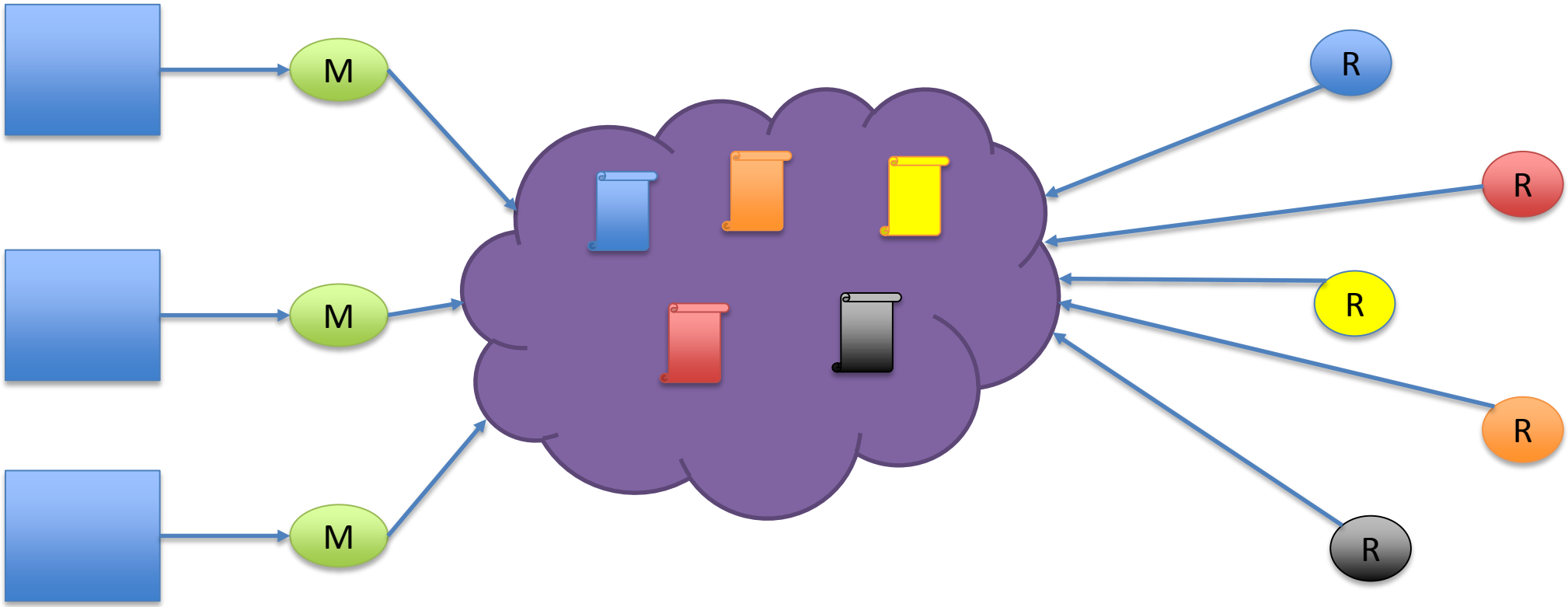
Sorting @ Scale: I/O's become small…tuning is hard…



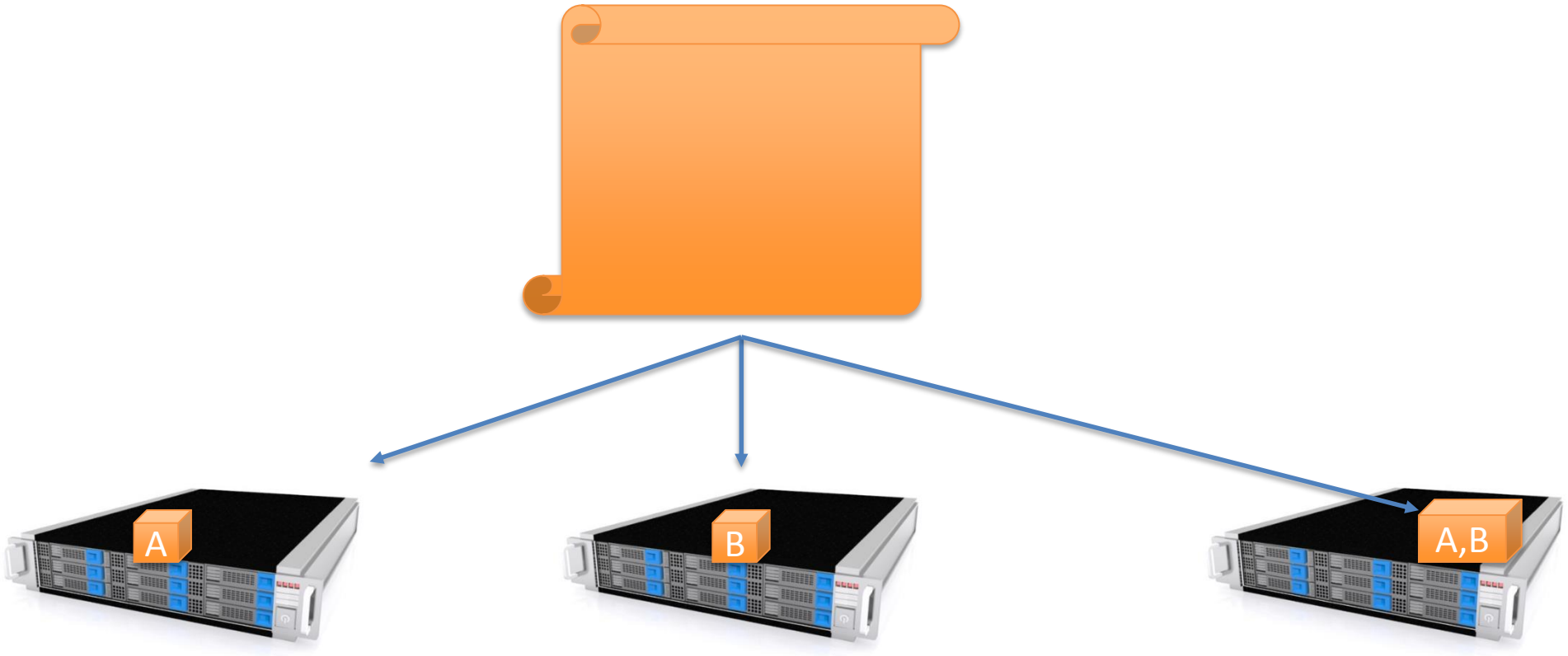
Scaling External Merge-Sort: Divide *and* Conquer

- Solving a seek problem: do "batch commit"

What if we did network-wide "batch commit"?



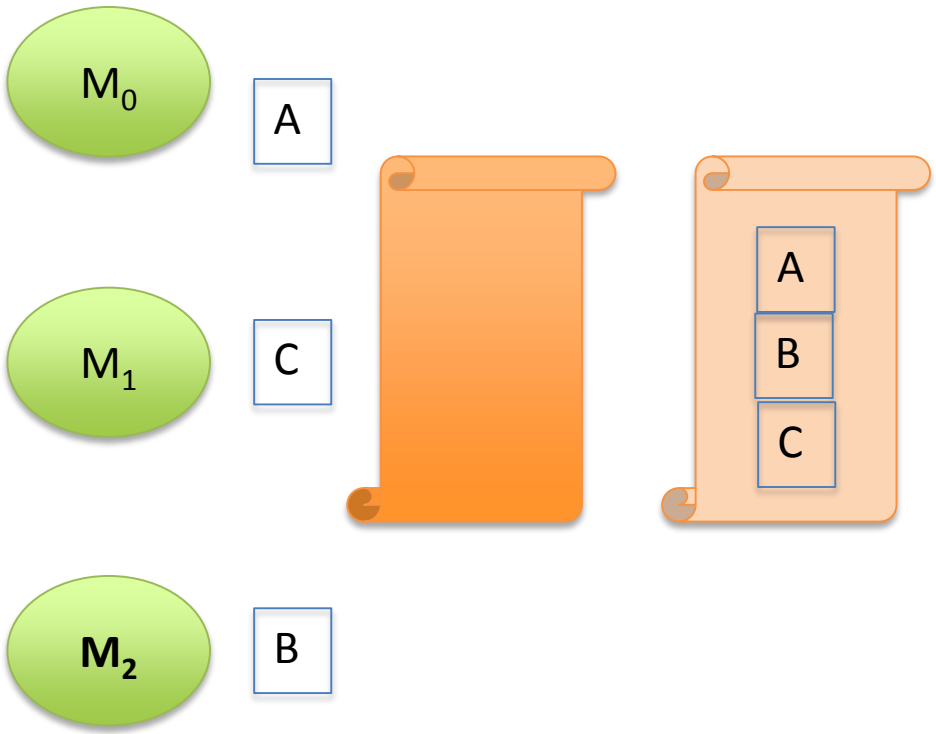
Scalable Merge-Sort Using KFS: I-files



- I-files (like any other KFS file) are striped across nodes
- I-files store sorted runs which must be subsequently merged

I-file Design

- I-file use case: store data that is input to a sort
  - Sort operates on records
- Sort defines the order in which records are consumed
  - Order in which records are written doesn't matter
- Requirements:
  - I-files need a record oriented interface
  - Records will be appended to I-file



(Scale Out) Atomic Record Append

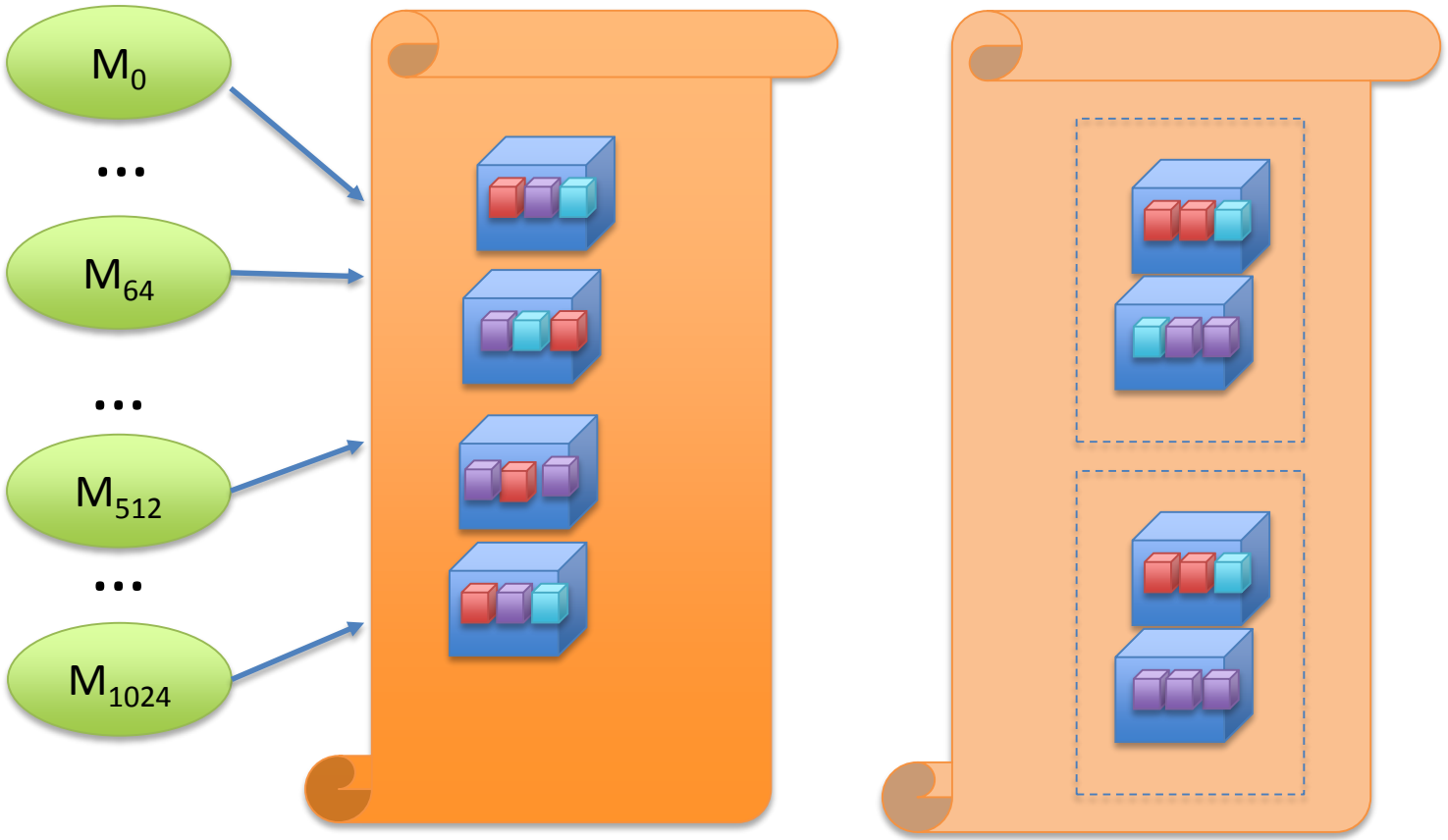
Challenge

- Large sorts involve 1000's of tasks, 1024 I-files
- Example: 1TB sort
  - # of mappers: 1024
    - Each mapper generates 1GB of data
  - # of reducers: 1024
    - Each reducer consumes 1GB of data that was generated by 1024 appenders

Approach

- Scale out!
  - Divide to conquer ☺
- Extend I-file API to allow multiple chunks to be concurrently appended to
- Example: I-file with 1024 appenders…
  - 64 mappers append to a chunk
  - 16 chunks appended to in parallel

Using I-files for Sorting



Unsorted I-file

File With Sorted Runs

- Sort chunks when they are stable
  - Overlap computation with sorting
- Create long sort runs by “batching”
  - Sort multiple stable chunks at a time

Summary: KFS I-files Characteristics

- I-files provide a record-oriented interface
- Append-only
  - Clients append records to an I-file : *record_append(fd, <key, value>)*
    - Append on a file translates to append on a chunk
    - At least once semantics => Filtering duplicates is application responsibility
    - Records do not span chunk boundaries
  - Metaserver controls the number of appenders to a chunk
  - Chunkserver does “group commit”

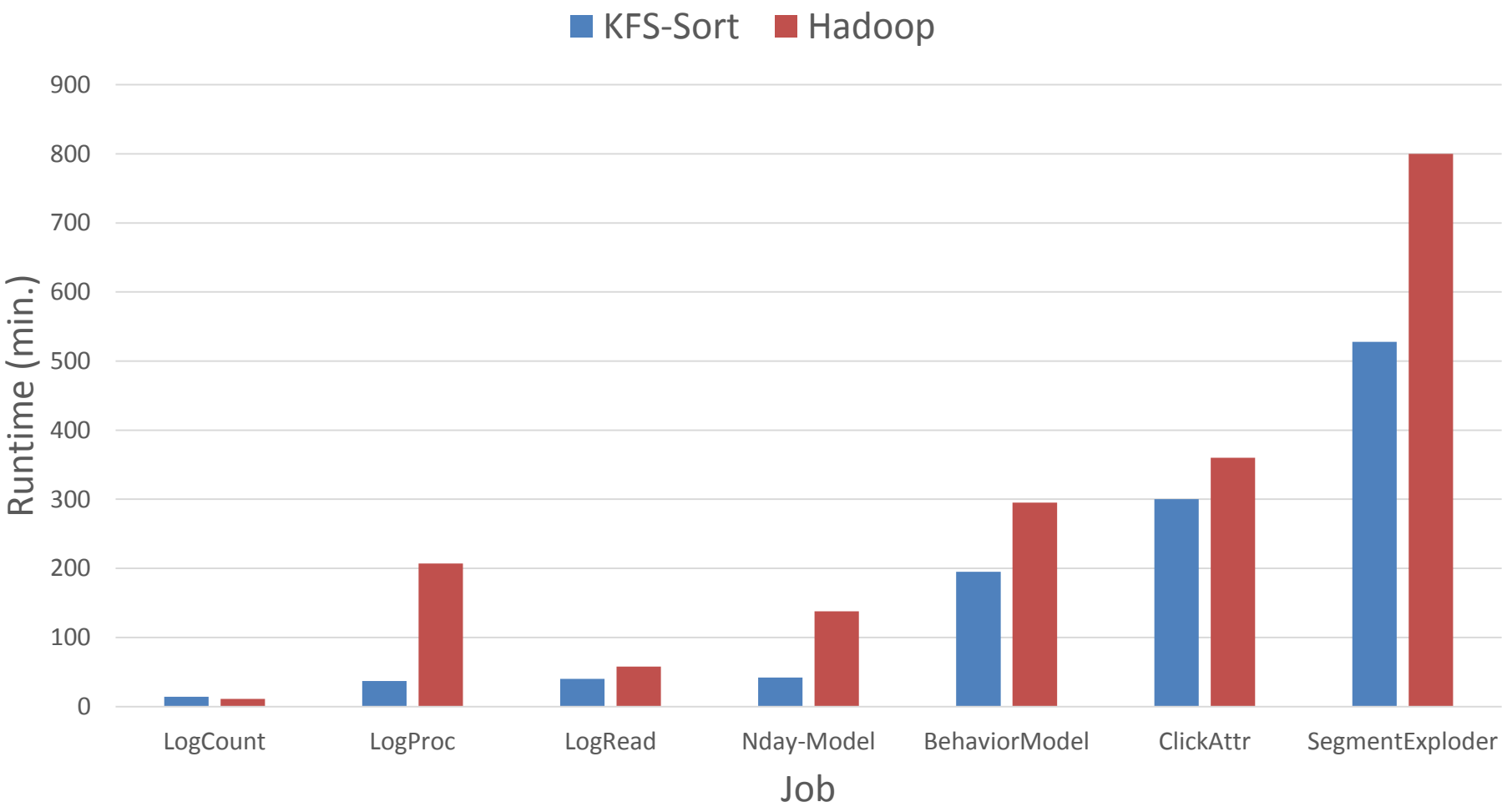
Other Details…

- Chunks of a I-file are replicated 2x
  - Each chunk has a “chunk master” that orders appends and replica follows the same
  - Loss of both chunks => job fails
- Sorted chunks of I-files were initially replicated 2x
  - (Since 2012)…replaced with 1.33x Reed-Solomon coding
- Length of sorted run: More the merrier ☺
  - RAM for sorting is a key constraint
    - 64GB RAM in a box; 12 tasks each with 4G; 6 sorters with 1G apiece
  - Started with 3 chunks; translated to 200MB buffer; compression…

Taking KFS-sort to production

- 4 months of implementation work
- Deploy on 16 machines…run 1TB sort
  - A weekend…"lock myself in room and debug"…Sort successful: 30 mins
  - CPU/NIC pegged ☺
- Show me a 10TB sort. We will give you 128 machines…
  - Two weeks later…done: ~40 mins ☺
- Show me a 100TB sort (actually, did a 140TB sort due to misconfig)
  - Six weeks later…**done**. **6:20** ☺
- Let us go to production…

How Did We Do…


KFS-Sort
Hadoop
900
800
700
600
500
400
300
200
100
0
Runtime (min.)
LogCount
LogProc
LogRead
Nday-Model
BehaviorModel
ClickAttr
SegmentExploder
Job

KFS-Sort in Production

- Deployed on the multi-tenanted cluster with 2 person devops
  - Self-validation:
    - map output records == reduce input records; Can't lose records
    - Records stored in compressed binary format; Can't corrupt records
- Went from working…to…working
  - For "flighting" without downtime, switch back to older sort via configuration changes
- Very supportive management, (even) more supportive users
  - Jobs fail; automagically switch to (slower) old one if needed
  - Cluster "detonated" a few times

KFS-Sort in Production

- 8 months of debugging in production…sort volume went from 100TB/day to 600TB/day
  - Purely software improvements on the same hardware
- Individual jobs sorting 100TB became typical
- Few other improvements to KFS-sort:
  - Distributed erasure coding
  - Outer-track optimization
    - Write sort spills to disk's outer track

And finally…

- In 2012, Quantcast…
  - (Re) Released their extensions to KFS as QFS
  - QFS replaced HDFS as their default filesystem
- Sailfish paper published at SoCC'12
  - Sailfish released as open source project
    - http://code.google.com/p/sailfish
- QFS paper published at VLDB'13
  - Paper describes distributed erasure coding implementation and results/experience

Metaserver Performance: QFS vs HDFS

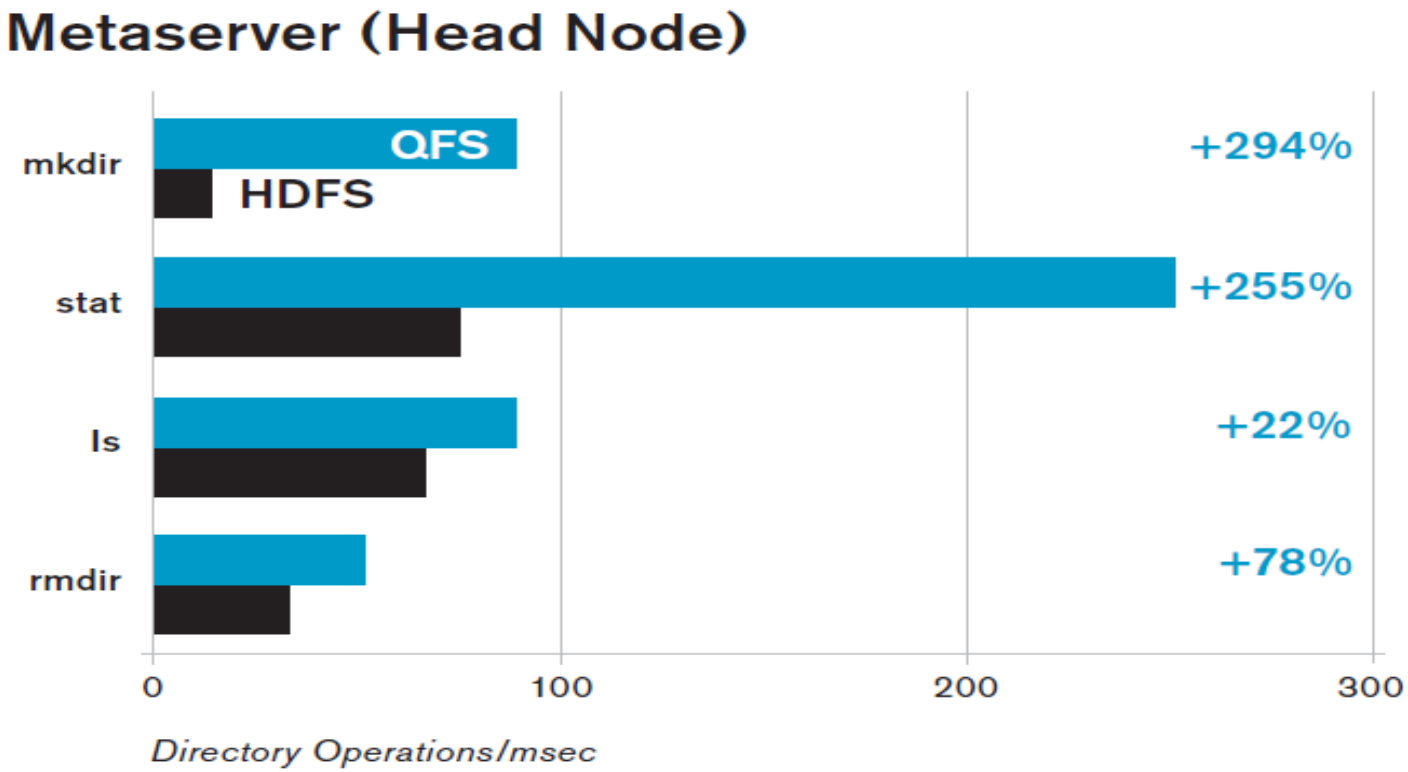


Figure 1: Comparison of directory management on QFS metaserver and HDFS head node, based on total throughput achieved by 512 clients building, inspecting, or deconstructing a balanced tree totaling 75.7 million directories. Dual E5-2670, 64GB RAM, HDFS version 2.0.0-cdh4.0.0 (Cloudera 4.0).

Reed-Solomon Error Correction
Leveraging high-speed modern networks

HDFS optimizes toward data locality for older networks.

10Gbps networks are now common, making disk I/O a more critical bottleneck.

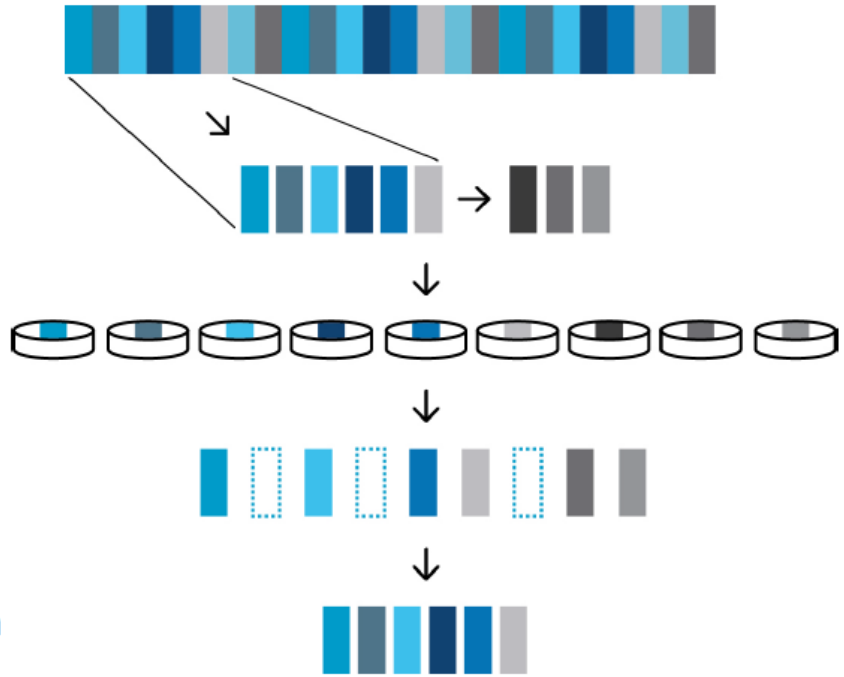
QFS leverages faster networks to achieve better parallelism and encoding efficiency.

Result: higher error tolerance, faster performance, with half the disk space.

1. Break original data into 64K stripes.
2. Reed-Solomon generates three parity stripes for every six data strips
3. Write those to nine different drives.
4. Up to three stripes can become unreadable...
5. …yet the original data can still be recovered

Every write parallelized across 9 drives, every read across 6

Reed-Solomon Parallel Data I/O

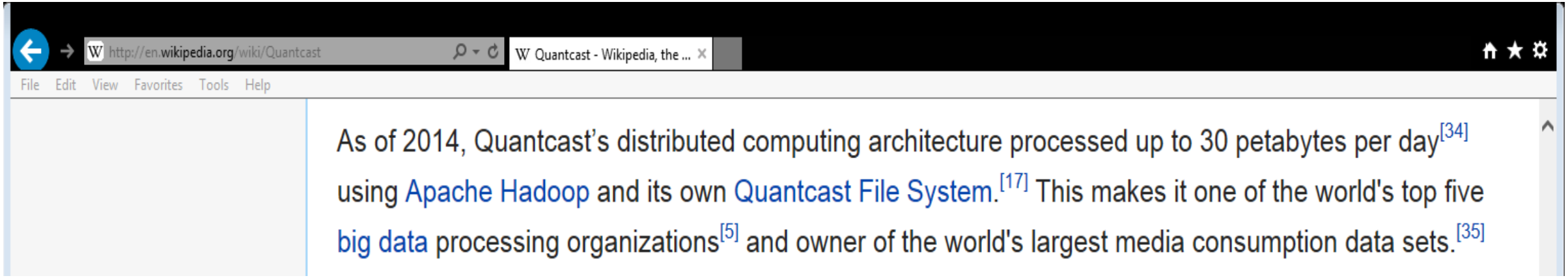


QFS Metaserver Stability (From QFS Paper)

Table 4: Some of the QFS instances currently in production at Quantcast. The third column shows the effective replication factor.

					Metaserver			
Purpose	Size	Repl.	Files	Blocks	Clients	RAM	Uptime	Last Restart
MapReduce input & output	3.6 PB	1.506	$31 * 10^6$	$240 * 10^6$	40,000	45 GB	197 days	HW upgrade
Logs	3.1 PB	1.500	$46 * 10^6$	$398 * 10^6$	10,000	74 GB	111 days	Increase RAM
Backups	406 TB	1.530	$8 * 10^6$	$45 * 10^6$	400	9 GB	156 days	Can't remember
Column store	21 TB	1.509	$2 * 10^6$	$11 * 10^6$	115,000	10 GB	197 days	Can't remember
Sort spill file system	0-400 TB	N/A	$0.5 * 10^6$	$2.7 * 10^6$	25,000	6 GB	50 days	SW Optimization

What is QFS upto…



Not bad… ☺

Concluding Remarks

- Software developed in the Big data space is mostly OSS
  - Embrace/extend-and-contribute is key
- Building software in OSS is fun
  - Once you have a community, you get “free” maintenance
- Key lessons from building software to run a service…
  - Setup a flighting environment
  - Iterate quickly…