MapReduce & Spark

Phil Gibbons

15-712 F15

Lecture 12

Today's Reminders

- **Emerald Therapeutics**
  - SV start-up company (co-founded by 2 CMU alums) that straddles the fields of computer science and biotechnology
  - 6 pm today in DH 2302: "Building cross platform desktop apps using web technologies: The Emerald Integrated Science Environment"
- **Discuss Project Ideas with Phil & Kevin**
  - Sign up for a slot: 11-12:30 or 3-4:20 next Friday

MapReduce: Simplified Data Processing on Large Clusters
[OSDI'04]

- **Jeffrey Dean** (Google)
  - NAE member, ACM Infosys Award, Mark Weiser Award
  - Google Translate, BigTable, Spanner, GoogleBrain

- **Sanjay Ghemawat** (Google)
  - ACM Infosys Award, Mark Weiser Award
  - GFS, BigTable, Spanner

Major Contributions

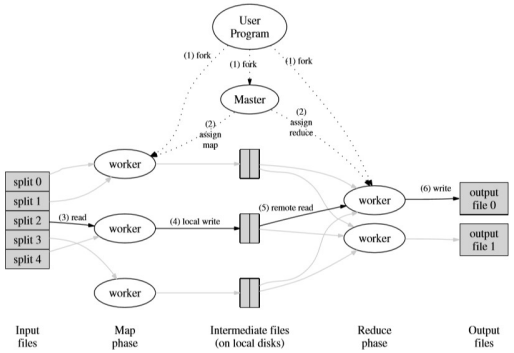
- **A simple & powerful interface that enables automatic parallelization & distribution of large-scale computations**
- **An implementation of this interface that achieves high performance on large clusters of commodity PCs**

"Programmers without any experience with parallel & distributed systems can easily [in 30 mins] utilize the resources of a large distributed system."

Programming Model

- **Map: (k1,v1) -> list(k2,v2)**
- **[Shuffle: group-by k2]**
- **Reduce: (k2,list(v2)) -> list(v2)**

Implementation



- Input split pieces typically 16-64MBs
- Buffer then write to local disk, partitioned into R regions
- Reduce-workers use RPC to remotely read from disks then sort

Fault Tolerance

- **When map-worker fails?**
  - Map-tasks re-assigned; Reduce-tasks informed
  - Completed map-tasks re-executed
- **When reduce-worker fails?**
  - Reduce-tasks re-assigned
  - Completed reduce-tasks are already in global file system
- **When master fails?**
  - Currently: abort MapReduce computation
- **Semantics on failure?**
  - When map & reduce are deterministic, semantics equivalent to sequential execution (rely on atomic file renaming)

Other Issues

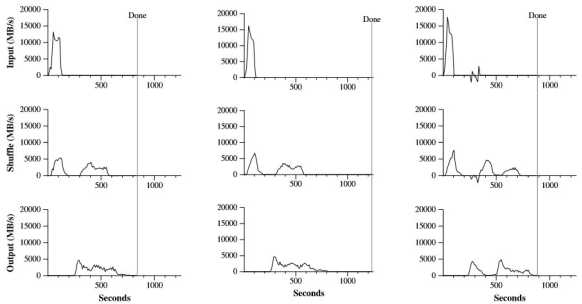
- **Locality: Schedule map task (near) where data resides**
- **Task Granularity: M & R constrained by**
  - Master makes O(M+R) scheduling decisions, keeps O(MR) state
  - R separate output files
  - Often make R a small multiple of number of machines
- **Stragglers:**
  - Causes: Error correction on bad disk, multi-tenancy, bug in initialization code that disabled processor caches
  - Solution: Fire off back-up tasks for remaining in-progress tasks
  - Why is duplicating work NOT a problem?

Refinements

- **User-defined partitioning functions**
- **User-defined combiner function for "partial reducing" in map tasks**
- **Skip bad records that cause deterministic crashes**
- **Input/output types, Side-effects, Local execution, Status info, Counters, etc**

Performance

- **Setup: 1800 machines (two 2GHz Xeons, 4GBs memory)**



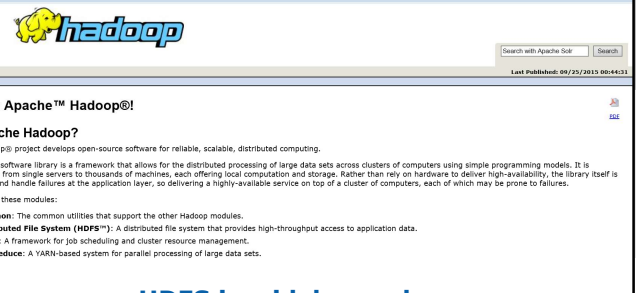
(a) Normal execution (b) No backup tasks (c) 200 tasks killed

Figure 3: Data transfer rates over time for different executions of the sort program

Experience

- **Big success: Widely used within Google**
  - Large-scale machine learning, clustering for Google News & Froogle, popular queries reports, large-scale graph computations, etc.
- **Complete rewrite of production indexing system for Google web search (20 TBs of crawled webpages)**
  - Indexing code is simpler, smaller, easier to understand
  - Keep conceptually unrelated computations separate—makes easier to change indexing process
  - Ease of elasticity

Today



HDFS is widely-used
YARN is reasonably popular

Hadoop MapReduce is performance strawman:
Large gains vs. MR not even worth noting

Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks

Isard, Budiu, Yu, Birrell, Fetterly [EuroSys'07]

E.g., Gravitational Lens Query:
Find all objects in U that have neighboring objects within 30 arc seconds (N) s.t. at least 1 neighbor has a color similar to the primary object's color

DryadLINQ [OSDI'08]: Dryad programs written using LINQ

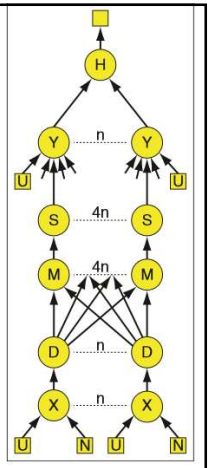
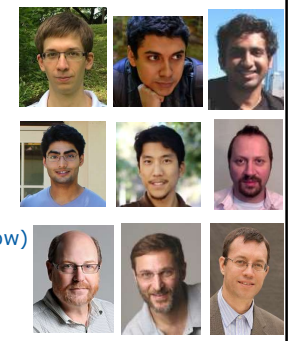


Figure 2: The communication graph for an SQL query.

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

[NSDI'12 best paper]

- **Matei Zaharia** (MIT)
- **Mosharaf Chowdhury** (to Michigan)
- **Tathagata Das** (Databricks)
- **Ankur Dave** (UC Berkeley)
- **Justin Ma** (UC Berkeley)
- **Murphy McCauley** (UC Berkeley)
- **Mike Franklin** (UC Berkeley, ACM Fellow)
- **Scott Shenker** (UC Berkeley, NAE)
- **Ion Stoica** (UC Berkeley, ACM Fellow, PhD@CMU)



Spark: Key Idea

Features:
- In-memory speed w/fault tolerance via lineage tracking
- Bulk Synchronous

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
- A restricted form of shared memory, based on coarse-grained deterministic transformations rather than fine-grained updates to shared state: expressive, efficient and fault tolerant

[Slides from NSDI'12 Talk]

Resilient Distributed Datasets

A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, Ion Stoica

UC Berkeley

Motivation

MapReduce greatly simplified "big data" analysis on large, unreliable clusters

But as soon as it got popular, users wanted more:

- » More **complex,** multi-stage applications (e.g. iterative machine learning & graph processing)
- » More **interactive** ad-hoc queries

Response: *specialized* frameworks for some of these apps (e.g. Pregel for graph processing)

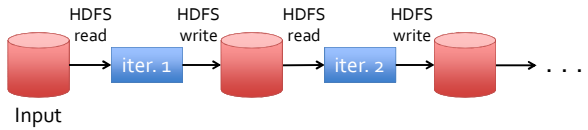
Motivation

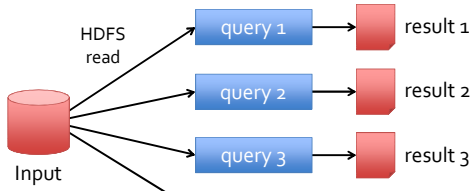
Complex apps and interactive queries both need one thing that MapReduce lacks:

Efficient primitives for **data sharing**

In MapReduce, the only way to share data across jobs is stable storage ➔ slow!

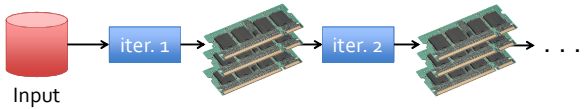
Examples

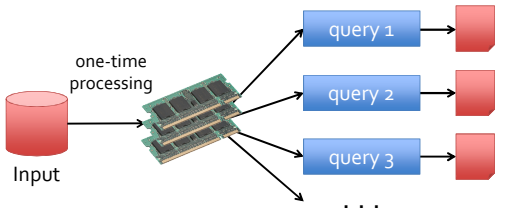




Slow due to replication and disk I/O, but necessary for fault tolerance

Goal: In-Memory Data Sharing





10-100× faster than network/disk, but how to get FT?

Challenge

How to design a distributed memory abstraction that is both **fault-tolerant** and **efficient**?

Challenge

Existing storage abstractions have interfaces based on *fine-grained* updates to mutable state
- » RAMCloud, databases, distributed mem, Piccolo

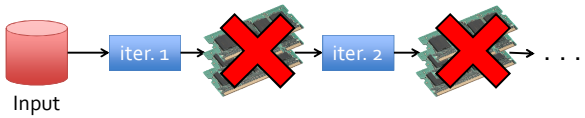
Requires replicating data or logs across nodes for fault tolerance
- » Costly for data-intensive apps
- » 10-100x slower than memory write

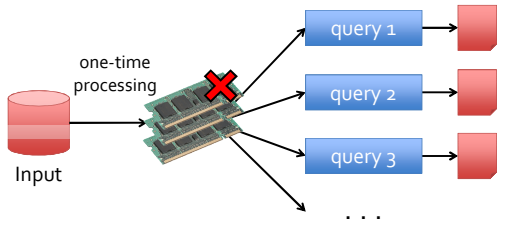
Solution: Resilient Distributed Datasets (RDDs)

Restricted form of distributed shared memory
- » Immutable, partitioned collections of records
- » Can only be built through *coarse-grained* deterministic transformations (map, filter, join, …)

Efficient fault recovery using *lineage*
- » Log one operation to apply to many elements
- » Recompute lost partitions on failure
- » No cost if nothing fails

RDD Recovery





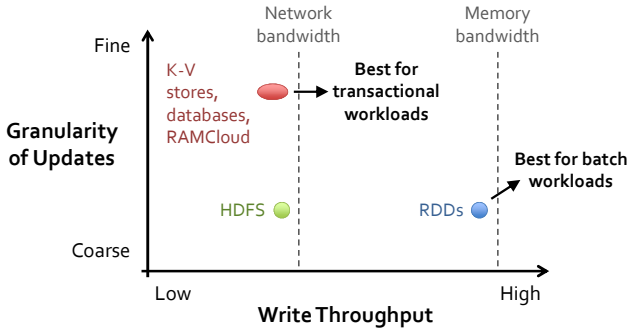
Generality of RDDs

Despite their restrictions, RDDs can express surprisingly many parallel algorithms
- » These naturally *apply the same operation to many items*

Unify many current programming models
- » *Data flow models:* MapReduce, Dryad, SQL, …
- » *Specialized models* for iterative apps: BSP (Pregel), iterative MapReduce (Haloop), bulk incremental, …

Support *new apps* that these models don't

Tradeoff Space



Spark Programming Interface

DryadLINQ-like API in the Scala language

Usable interactively from Scala interpreter

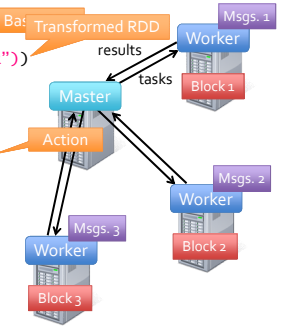
Provides:
- » Resilient distributed datasets (RDDs)
- » Operations on RDDs: *transformations* (build new RDDs), *actions* (compute and output results)
- » Control of each RDD's *partitioning* (layout across nodes) and *persistence* (storage in RAM, on disk, etc)

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
messages.persist()

messages.filter(_.contains("foo")).count
messages.filter(_.contains("bar")).count
```

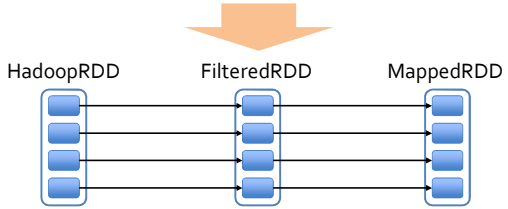


Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)

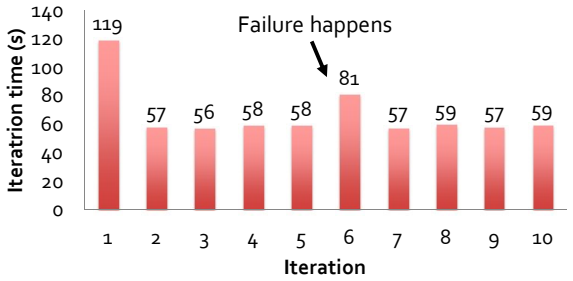
Fault Recovery

RDDs track the graph of transformations that built them (their *lineage*) to rebuild lost data

E.g.: `messages = textFile(...).filter(_.contains("error"))`
`.map(_.split('\t')(2))`



Fault Recovery Results



Example: PageRank

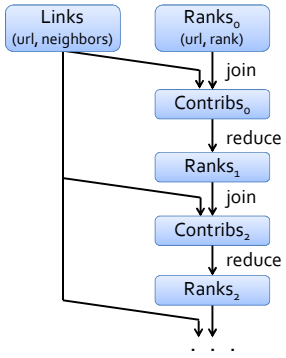
1. Start each page with a rank of 1
2. On each iteration, update each page's rank to

$$\Sigma_{i \in neighbors} rank_i / |neighbors_i|$$

```
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }.reduceByKey(_ + _)
}
```

Optimizing Placement



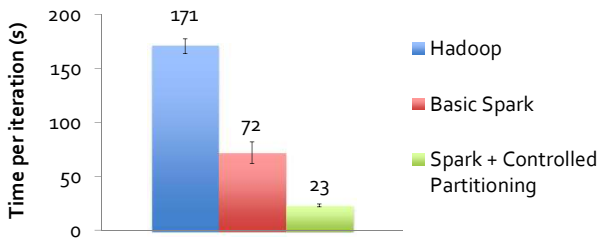
`links` & `ranks` repeatedly joined

Can *co-partition* them (e.g. hash both on URL) to avoid shuffles

Can also use app knowledge, e.g., hash on DNS name

```
links = links.partitionBy(
          new URLPartitioner())
```

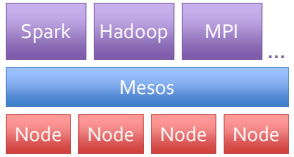
PageRank Performance



Implementation

Runs on Mesos [NSDI 11] to share clusters w/ Hadoop

Can read from any Hadoop input source (HDFS, S_3, ...)



No changes to Scala language or compiler
- » Reflection + bytecode analysis to correctly ship code

www.spark-project.org

Programming Models Implemented on Spark

RDDs can express many existing parallel models
- » **MapReduce, DryadLINQ**
- » **Pregel** graph processing [200 LOC]
- » **Iterative MapReduce** [200 LOC]
- » **SQL**: Hive on Spark (Shark) [in progress]

All are based on coarse-grained operations

Enables apps to efficiently *intermix* these models

Open Source Community

15 contributors, **5+** companies using Spark, **3+** applications projects at Berkeley

User applications:
- » Data mining 40x faster than Hadoop (Conviva)
- » Exploratory log analysis (Foursquare)
- » Traffic prediction via EM (Mobile Millennium)
- » Twitter spam classification (Monarch)
- » DNA sequence analysis (SNAP)
- » . . .

Related Work

RAMCloud, Piccolo, GraphLab, parallel DBs
» Fine-grained writes requiring replication for resilience

Pregel, iterative MapReduce
» Specialized models; can't run arbitrary / ad-hoc queries

DryadLINQ, FlumeJava
» Language-integrated "distributed dataset" API, but cannot share datasets efficiently *across* queries

Nectar [OSDI 10]
» Automatic expression caching, but over distributed FS

PacMan [NSDI 12]
» Memory cache for HDFS, but writes still go to network/disk

Conclusion

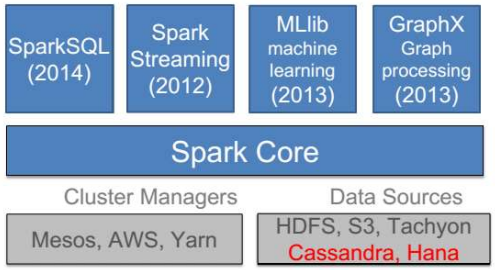
RDDs offer a simple and efficient programming model for a broad range of applications

Leverage the coarse-grained nature of many parallel algorithms for low-overhead recovery

Try it out at www.spark-project.org

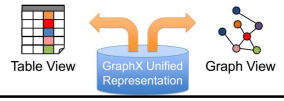
[End of slides from NSDI'12 Talk]

Spark Stack continued innovations



Tachyon: Memory-speed data sharing among jobs in different frameworks (e.g., Spark & Hadoop). Keeps in-memory data safe even when job crashes

GraphX: Tables & Graphs are views of same physical data, exploit semantics of view for efficient operation



A Brave New World

Spark Timeline

- Research idea in 2009
- Open source release in 2011
- Into Apache Incubator in 2013
- In all major Hadoop releases in 2014
- 1000+ companies using Spark in 2015

cloudera IBM Hortonworks MAPR ORACLE Pivotal SAP

- Pipeline of research breakthroughs (publications in best conferences) fuel continued leadership & uptake
- Start-up (Databricks), Open Source Developers, and Industry partners (IBM, Intel) make code commercial-grade

Fast Path for impact via Open Source:
Pipeline of Research Breakthroughs that make it into widespread commercial use in under 2 years

Monday's Paper

Naiad: A Timely Dataflow System
Derek Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, Martin Abadi

SOSP'13 best paper