CPSC 416 Distributed Systems

Winter 2022 Term 2 (March 23, 2023)

Tony Mason (fsgeek@cs.ubc.ca), Lecturer

Logistics

Deadlines

Project 4 Released. Late Due: April 13, 2023.

Project 5 Released Due: April 13, 2023. **No extensions.**

All project work is due April 13, 2023. Late projects are scaled to 75% of the on-time max.

Final Exam: April 20, 2023, DMP 310, 08:30-11:00. Format TBA.

Deadlines

Alternate Path 1 & 2: Review in progress

- Piazza private threads need TLC
  - **Weekly updates due each Monday @ 23:59 PT**
- Final reports due no later than Thursday April 13, 2023 @ 23:59 PT
- Optional 10 min presentation April 13, 2023, up to 10 minutes.

Instructor Office Hours:

- Zoom Office Hours (Tuesday) @ 13:00-14:00
- Discord (Casual) Office Hours (Thursday) @ 14:00-15:00

TA Office Hours:

- Eric: Friday 9-11 am (in-person and Zoom)
- Japraj: Wednesday 3-5 pm (Zoom)
- Yennis: Thursday 2-4 (Zoom), Friday 2-4 (in-person)

Readings

Required:

Recommended:

Questions?

Questions about the class?

Questions about the previous lecture?

Funny stories to share?

Today’s Failure

Github.com Outage

Event: October 21, 2018 22:52 UTC

Planned outage: goal is to replace a failing 100Gb/s optical network device.

“Connectivity between these two locations was restored in 43 seconds, but this brief outage triggered a chain of events that led to 24 hours and 11 minutes of service degradation.”

Infrastruture: MySQL with Orchestrator to manage cluster topologies.

Note: Orchestrator uses **Raft** for consensus.

Github.com Outage

Network goes out: Raft starts “leadership deselection”

Note: optical link was between two Eastern US sites.

West coast data center and East coast Orchestrator form quorum

Fail over to clusters in West coast data center: write operations begin working.

Network fixed: traffic starts going to West coast site

Note: East coast had some updates that had not propagated to west coast yet.
This **blocked** primary returning to East coast.

Github.com

Things come unraveled due to increased latency, unexpected topologies.
Decision to degrade service rather than compromise consistency.

Start restoring databases from backup.

Restoration was started October 22, 2018 00:05 UTC
Restoration completed and service restored October 22, 2018 23:03 UTC

Twenty three hours to restore from a 43 second network disruption.

Takeaway: Recovery is the hard part.

Source

Lesson Goals

Distributed Data Analytics

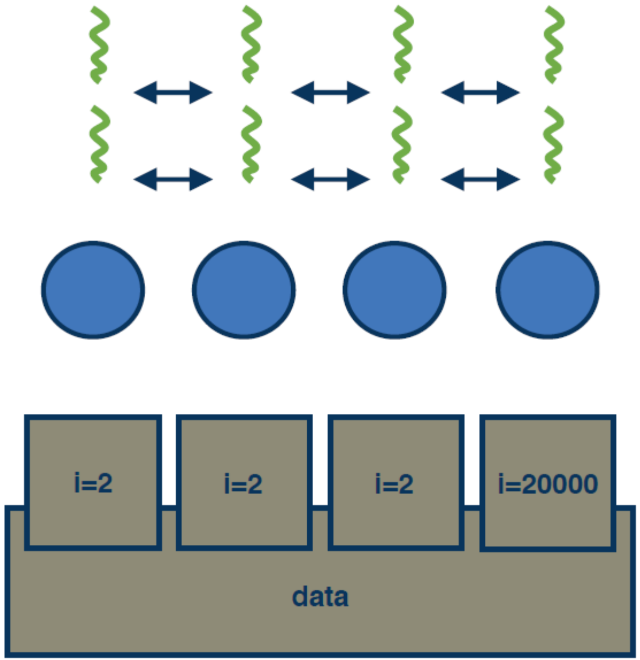
MapReduce

Spark (and RDDs)

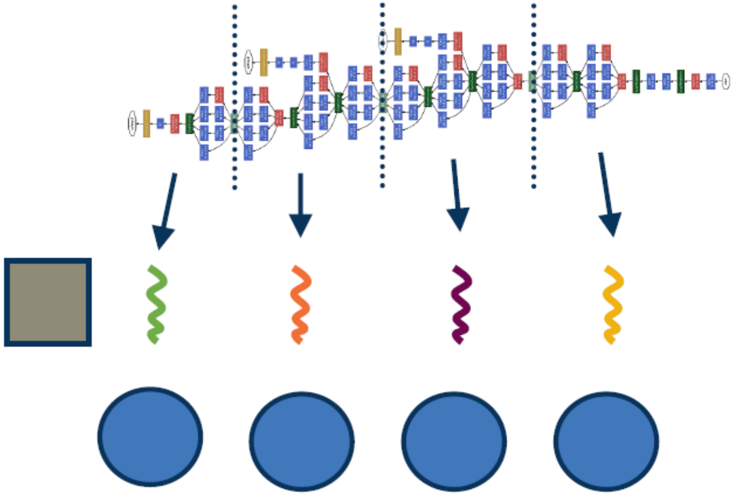
Common Techniques

Data Parallel (Divide & Conquer):

- Divide Data across nodes
- Load balancing, decomposition
- Messaging for data dependencies
- Application usage



Common Techniques



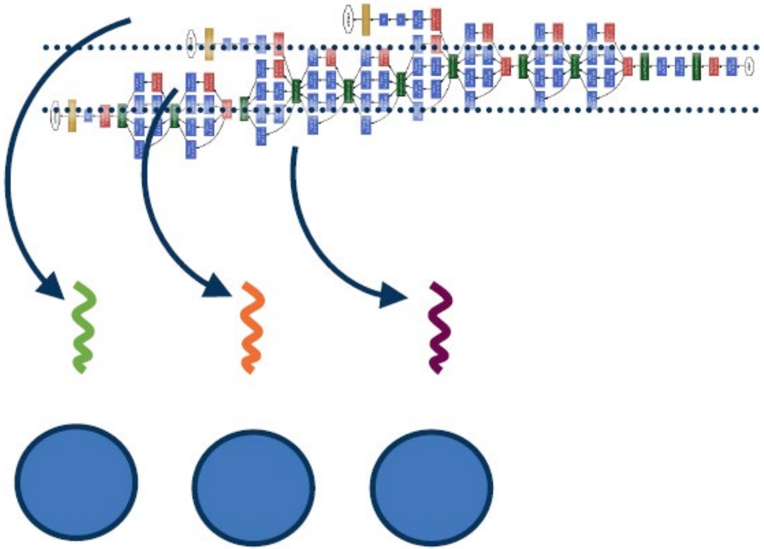
Pipelining

- Divide work into smaller tasks
  - Small number of tasks per node
  - Faster than generality
- Data streamed in chunks through task pipeline
- Increases throughput

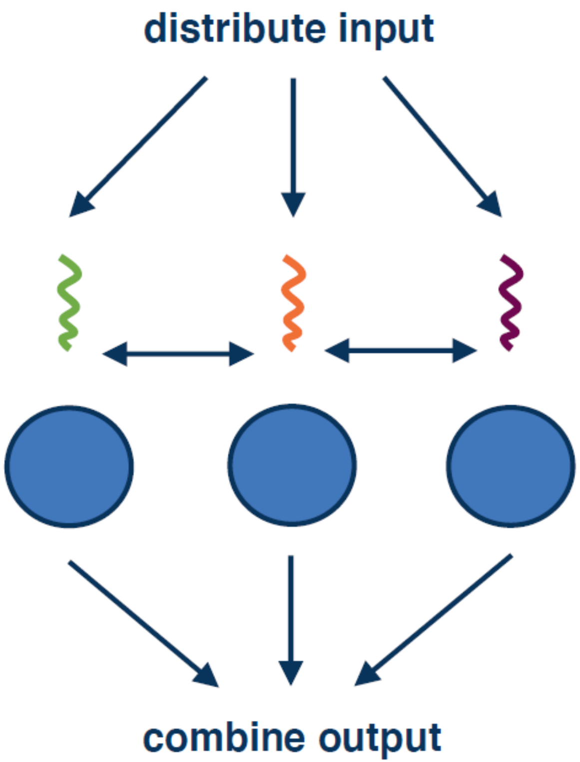
Common Techniques

Model Parallelism

- Divide state across nodes
- Less processing per node
- Input passed to all nodes
- Output combined from all nodes
- Must handle dependencies



Common Techniques



Model Parallelism

- Divide state across nodes
- Less processing per node
- Input passed to all nodes
- Output combined from all nodes
- Must handle dependencies

MapReduce

MapReduce: Simplified Data Processing on Large Clusters, J. Dean, OSDI 2004.

- Hadoop MapReduce
- AWS infrastructure

MapReduce

Input:

- Set of key-value pair records

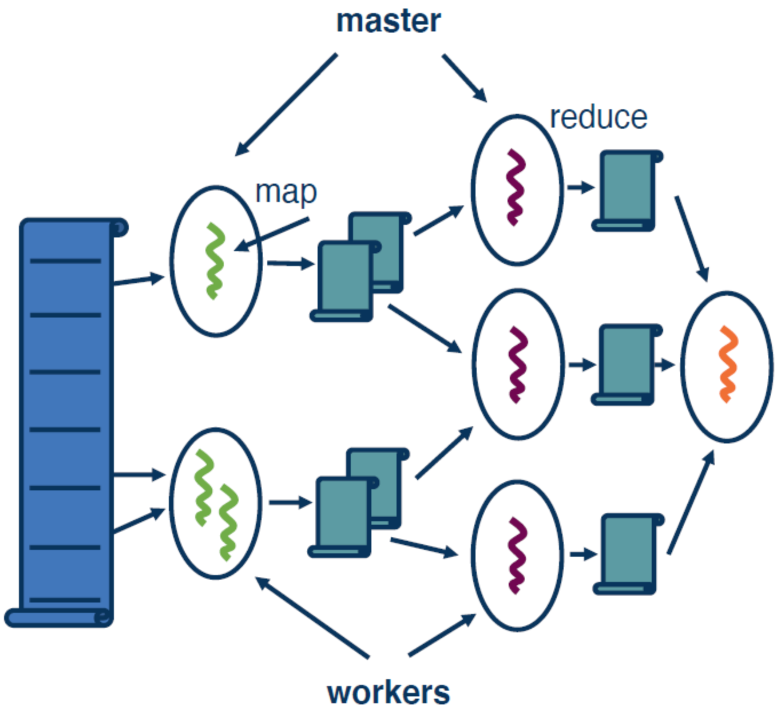
Map function

- Input: unique key-value pair
- Output: a new key-value pair

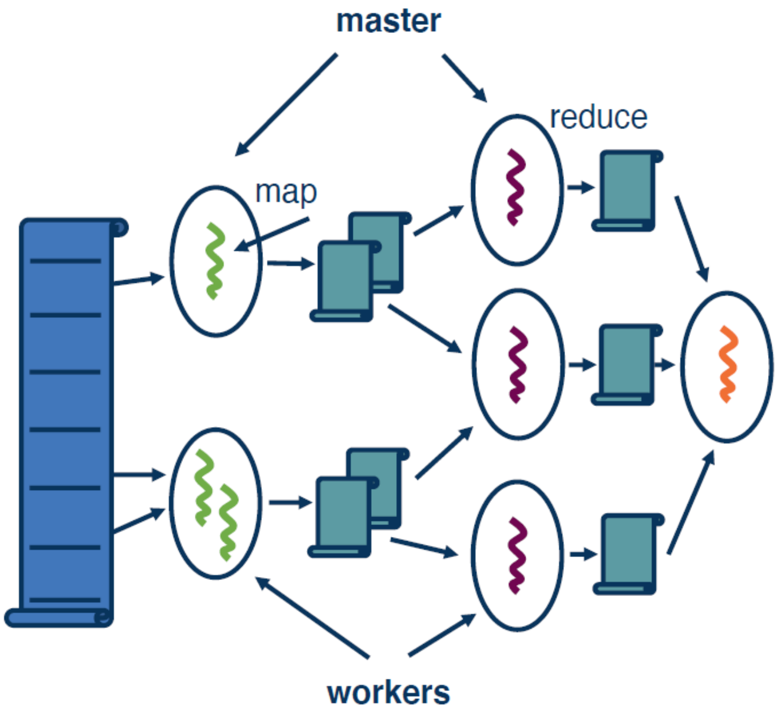
Reduce function:

- Input: output from map function
- Output: final result

Master: orchestrates workers, I/O, failure management



MapReduce



Wordcount example:

- Input: Collection of files

Map function:

- Input: File, key=filename, content=value
- Output: file with key=word, value=list of counts

Reduce function:

- Input: file with key=word, value=list of counts
- Output: list of words with total counts

Other examples:

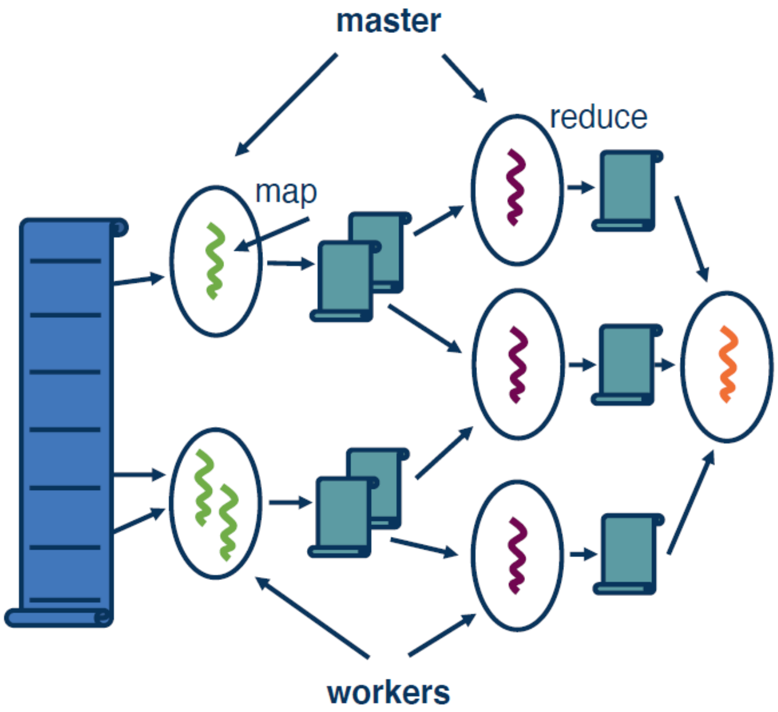
- URL access frequency, page rank, inverted word index

MapReduce

Combining Techniques:

- Data parallel: chunks to mappers
- Pipelining: mapper to reducer
- Model parallelism: reducers process parts of key space, combine

Dataflow model means flow of data determines execution



Map Reduce: Design Decisions

Master data structures:
- Tracking

Locality:
- Scheduling, placement of intermediate data

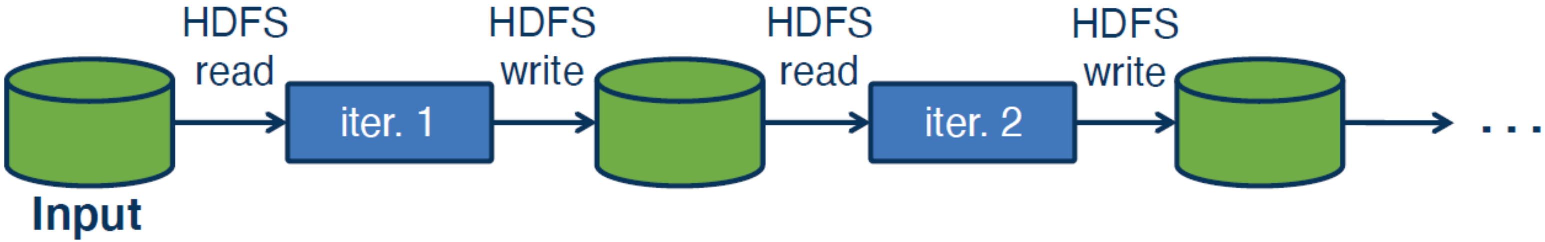
Task granularity:
- Finer granularity: more flexibility, management operation execution time
- Coarse granularity: lower management overhead

Fault tolerance:
- Master: standby replication
- Worker: detect failures or stragglers and re-execute

Failure semantics:
- Importance of Consistency and complete results

Backup tasks:
- Inevitable failures: speculative task backup

Map Reduce: Limitations



Failure inevitable: cannot re-execute entire operation

Fault-tolerant mechanism: requires intermediate data availability

- Serialiation to/from persistent storage
- Remote access and data movement

Data amplification:

- Intermediate data may be much larger than input
- Executions are iterative
- Storage level replication

System scale: cannot assume best-in-class storage devices

Spark

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

Faster analytics (10x) versus Hadoop

- Workloads: graph, streaming, SQL, Machine Learning, etc.
- Languages: Java, Python, Scala, etc.
- Platforms: AWS, Kubernetes, etc.

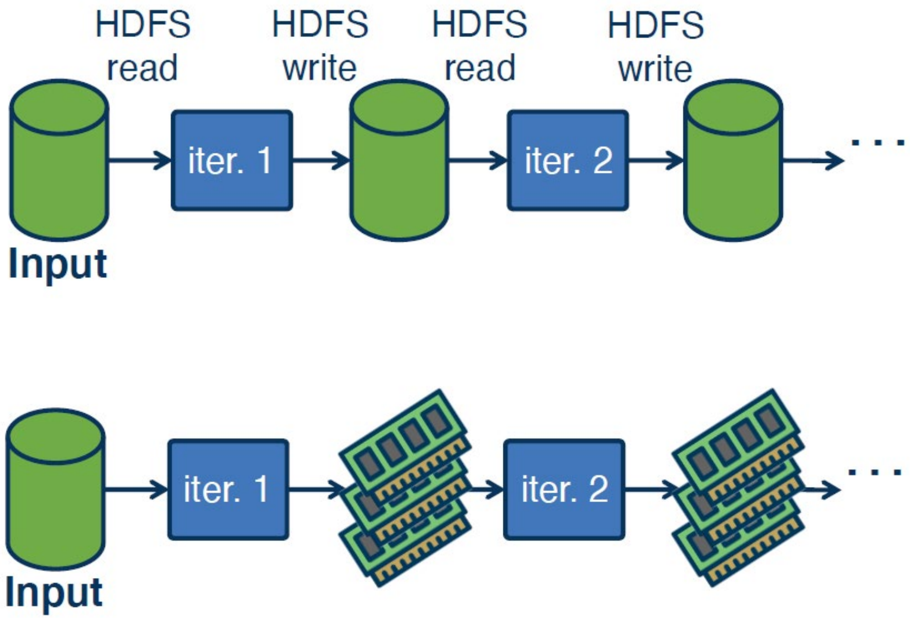
[Apache Spark]

Spark: Goals

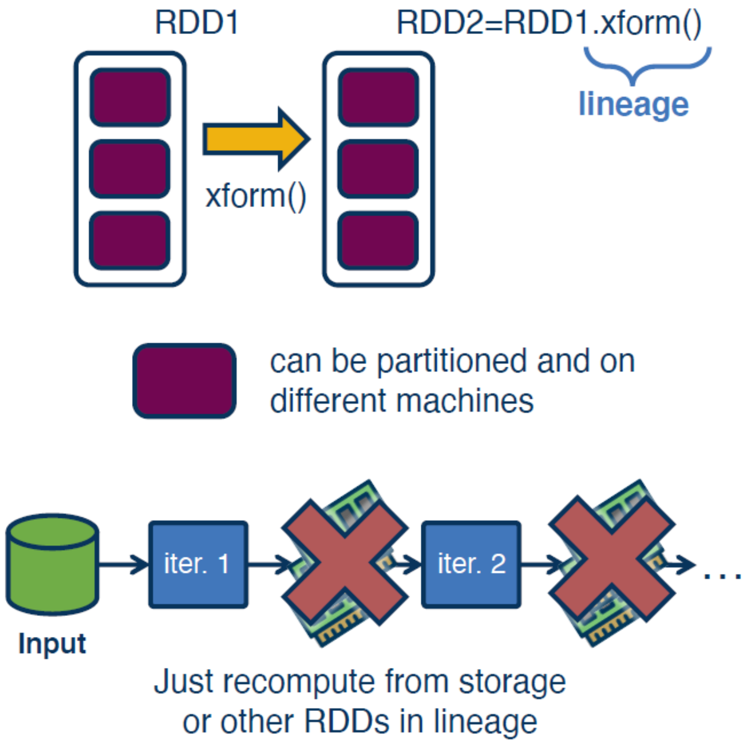
Allow in-memory data sharing

- Fast DRAM versus slow hard disk
- No serialization cost

Fault-tolerant



Resilient Distributed Datasets: Introduction



Immutable partitioned record collection

Created using transformations

- Operations on data in stable storage
- Map/join/filter on other RDDs

Used via actions (count, collect, save)

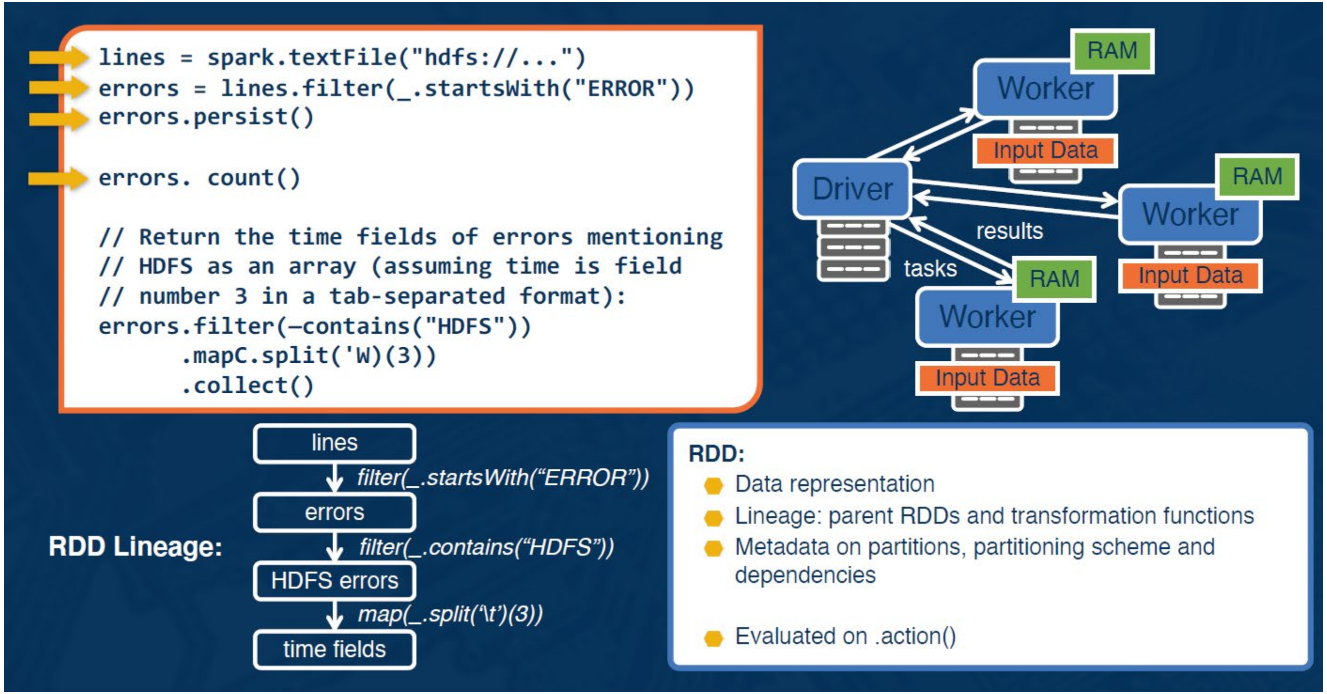
RDDs map back to source

- Compute partitions from data in stable storage

Users control persistence and partitioning

Resilient Distributed Datasets: Example

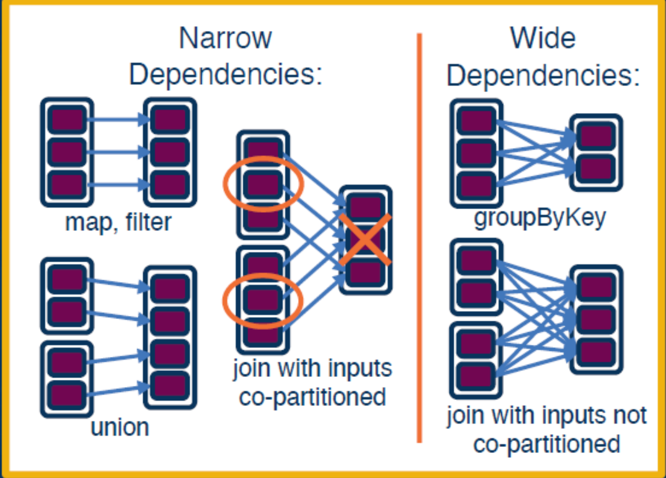
Console log mining example



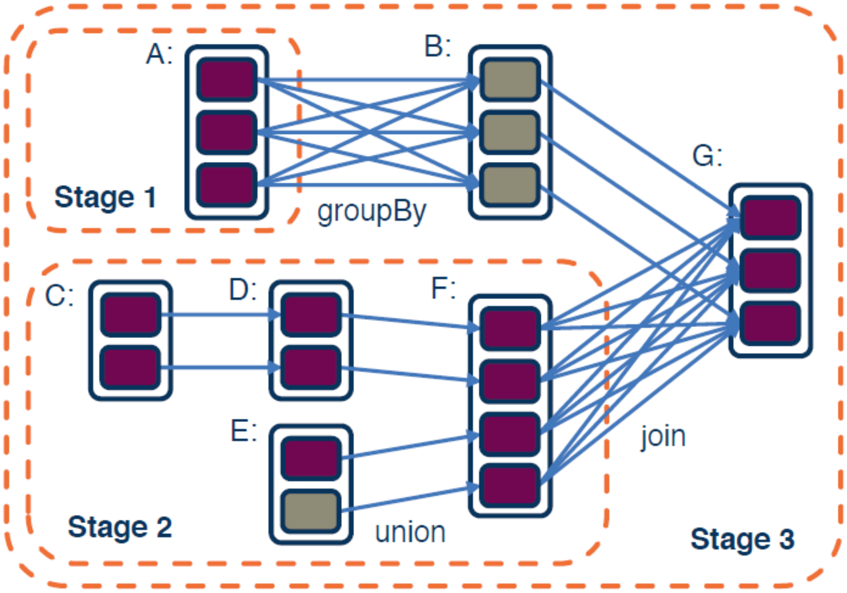
Resilient Distributed Datasets: Transformations & Actions

Transformations	*map*(*f* : T ⇒ U)	: RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	: RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	: RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	: RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	: RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	: RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	: (RDD[T], RDD[T]) ⇒ RDD[T]
	join()	: (RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	: (RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	: (RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	: RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	: RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	: RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	: RDD[T] ⇒ Long
	collect()	: RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	: RDD[T] ⇒ T
	lookup(*k* : K)	: RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	: Outputs RDD to a storage system, *e.g.*, HDFS

Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.



Resilient Distributed Datasets: Scheduling Action Execution



Program defines dependencies

Actions:

- Directed acyclic graph (DAG)
- Minimize dependencies
- Optimize parallelism
- Limit I/O contention

Tasks assigned based on data locality

Spark: Goals

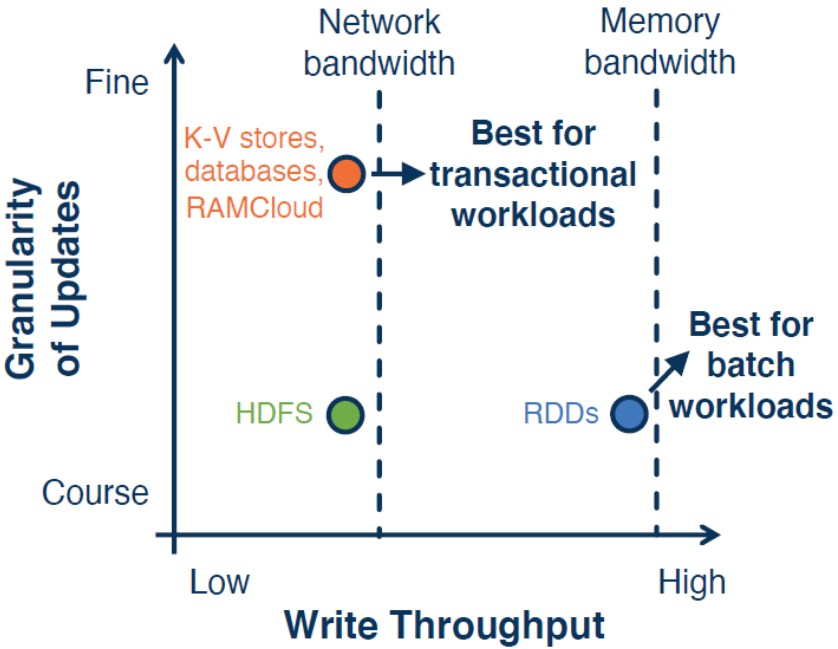
Data in memory?

- Distributed shared memory like runtime
  - Log updates
  - Persist lineage

Log coarse grained operations applied to all items in RDD elements

- ➕ less data to persist in execution critical path
- ➕ read data as low as once, less slow storage I/O
- ➕ more control on locality
- ➖ recovery time

Spark: Goals



Data in memory?

- Distributed shared memory like runtime
  - Log updates
  - Persist lineage

Log coarse grained operations applied to all items in RDD elements

Spark: Evaluation

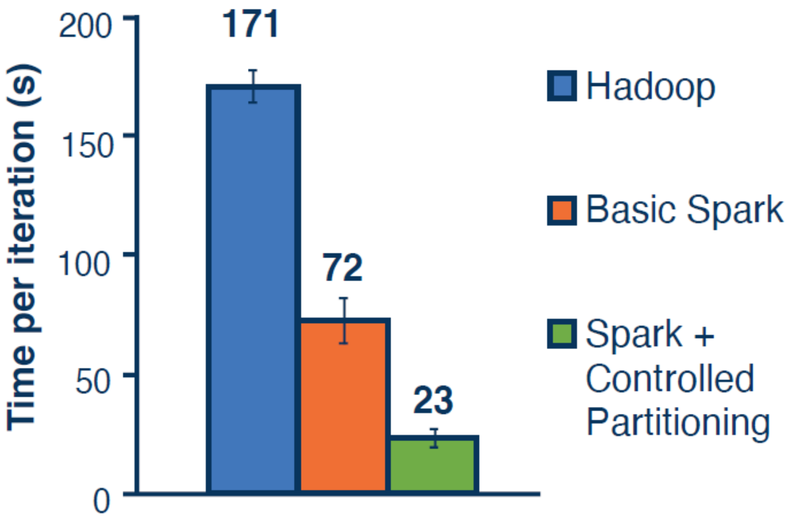
Up to 20x better than Hadoop

- Iterative
- Machine learning
- Graph applications

Analytics report generation 40x

Rapid failure recovery

1TB dataset queries with 5-7 second latencies



Lesson Review

Distributed Data Analytics

Systems for scalable data processing

MapReduce

Spark

Questions?


UBC
THE UNIVERSITY OF BRITISH COLUMBIA
THE UNIVERSITY OF BRITISH