MapReduce and Spark

Oct 8, 2015

MapReduce

Parallel computing platform built at Google

Still runs millions of jobs / day

"Functional" API with deterministic recomputation for fault tolerance

Key Ideas in MapReduce

Recomputation for fault tolerance

Parallel recovery: lost work is spread out

Straggler mitigation through backup tasks

Dynamic scheduling

Key Design Elements

Centralized master

“Pull” based communication model

- Reduce tasks fetch files from mappers
- Provides cheaper fault recovery and room for dynamic scheduling of tasks

Real-World MR Use Cases

Extract, Transform and Load (ETL)

SQL-like queries (Tenzing, Hive)

Complex analytics with non-SQL code

Spark

Generalizes MapReduce while retaining its scheduling and fault tolerance benefits

Main addition: efficient data sharing

Enables more applications

- Iterative algorithms
- Interactive queries
- Stream processing

Resilient Distributed Datasets (RDDs)

Restricted form of shared memory

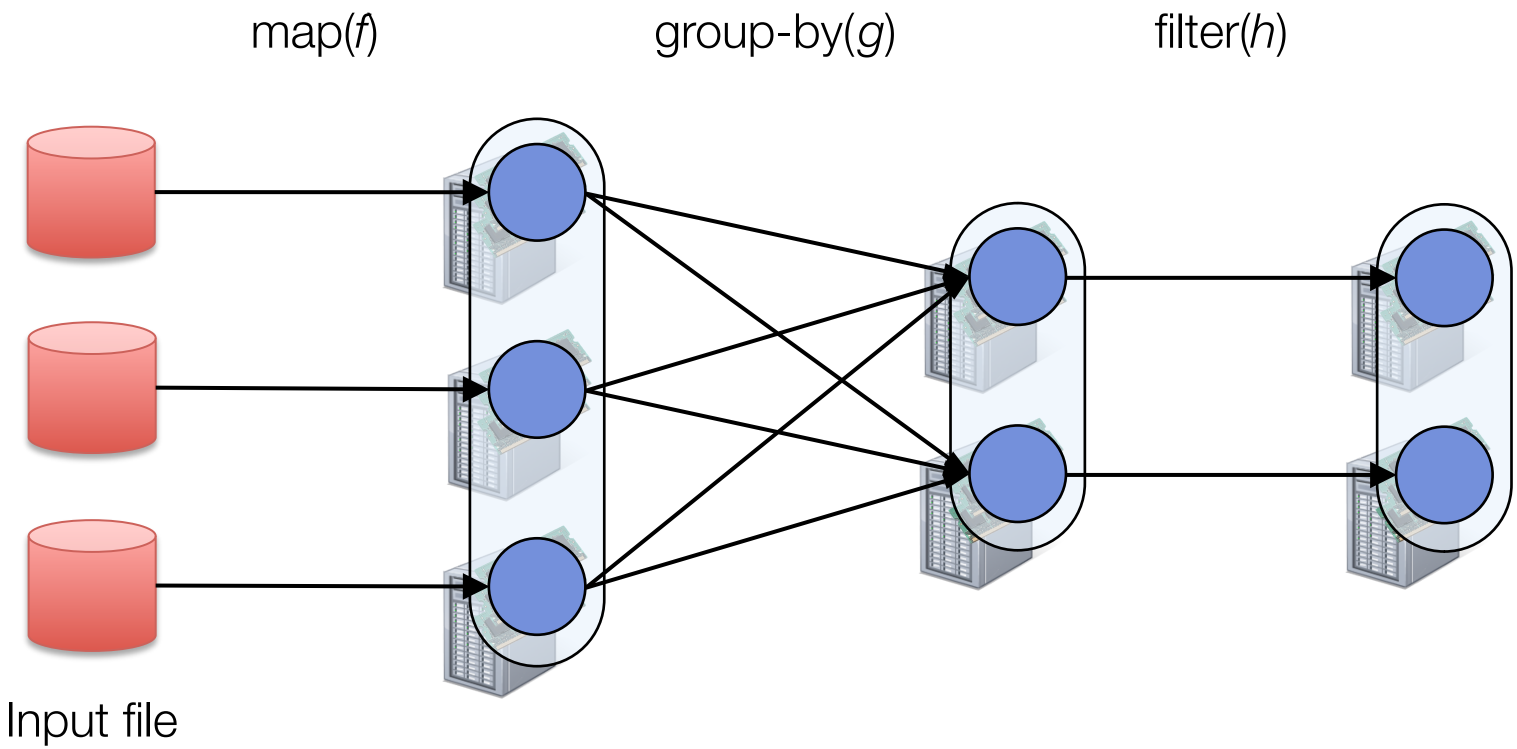
- Immutable, partitioned sets of records
- Can only be built through coarse-grained, deterministic operations (map, filter, join, …)

Fault recovery using *lineage*

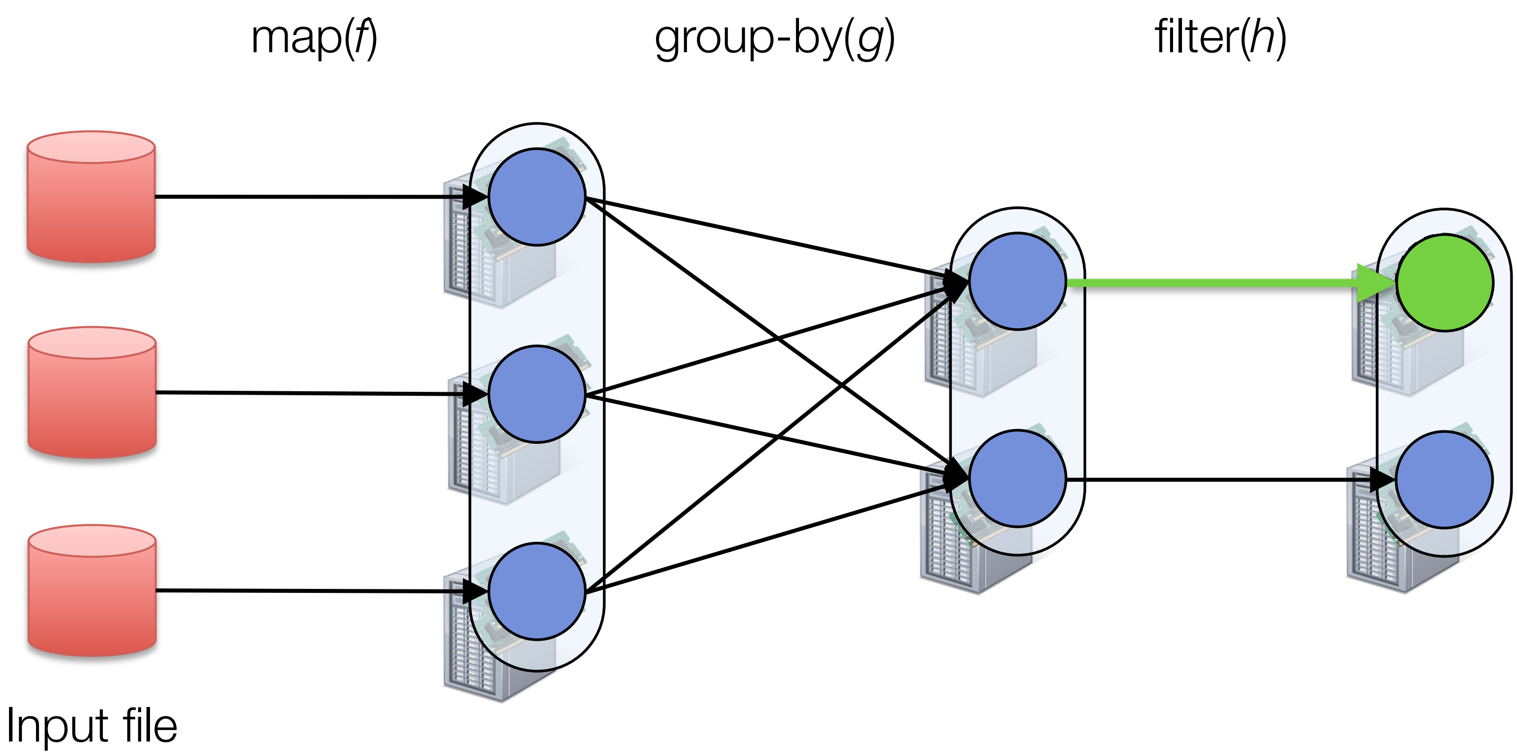
- Log one operation to apply to many elements
- Recompute lost partitions on failure

[NSDI 2012]

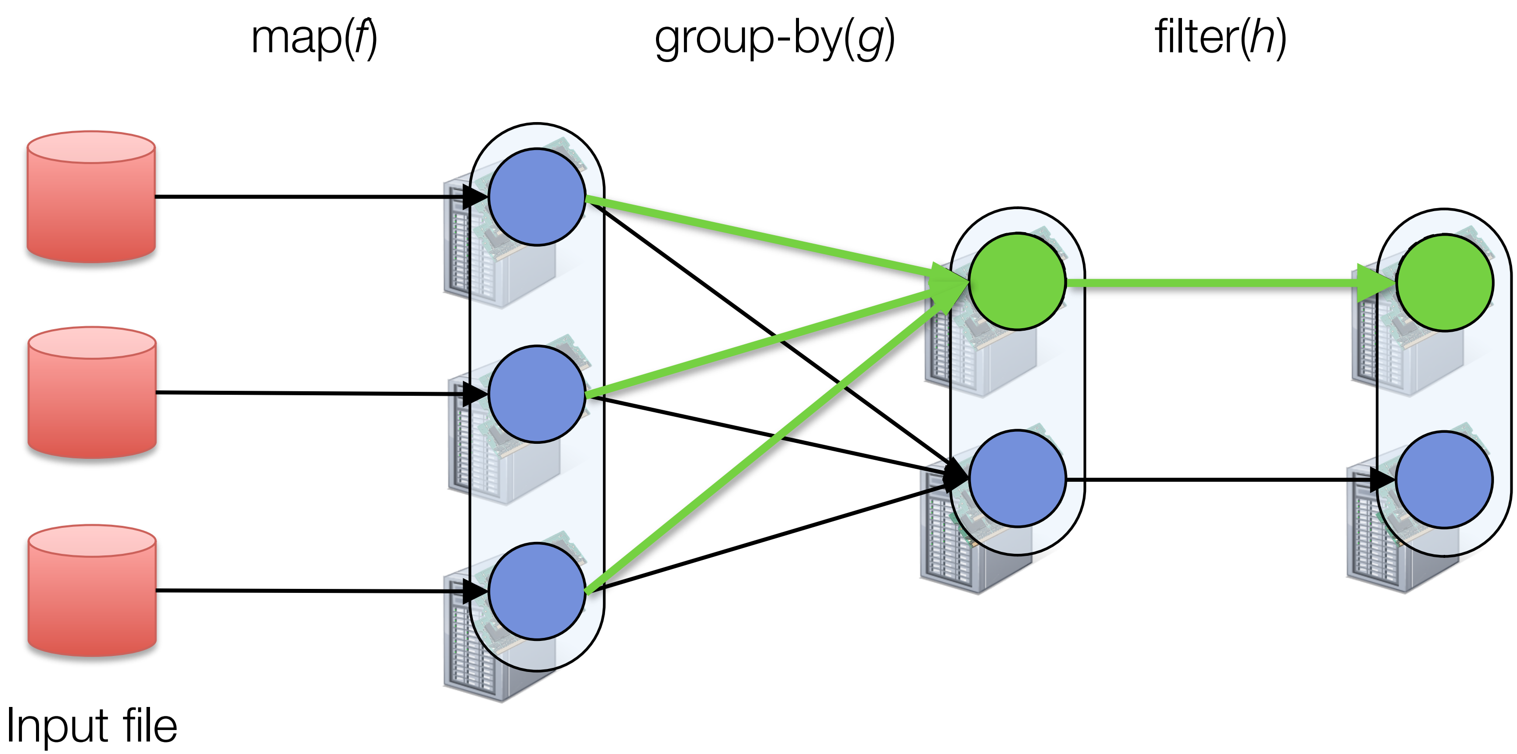
RDD Recovery



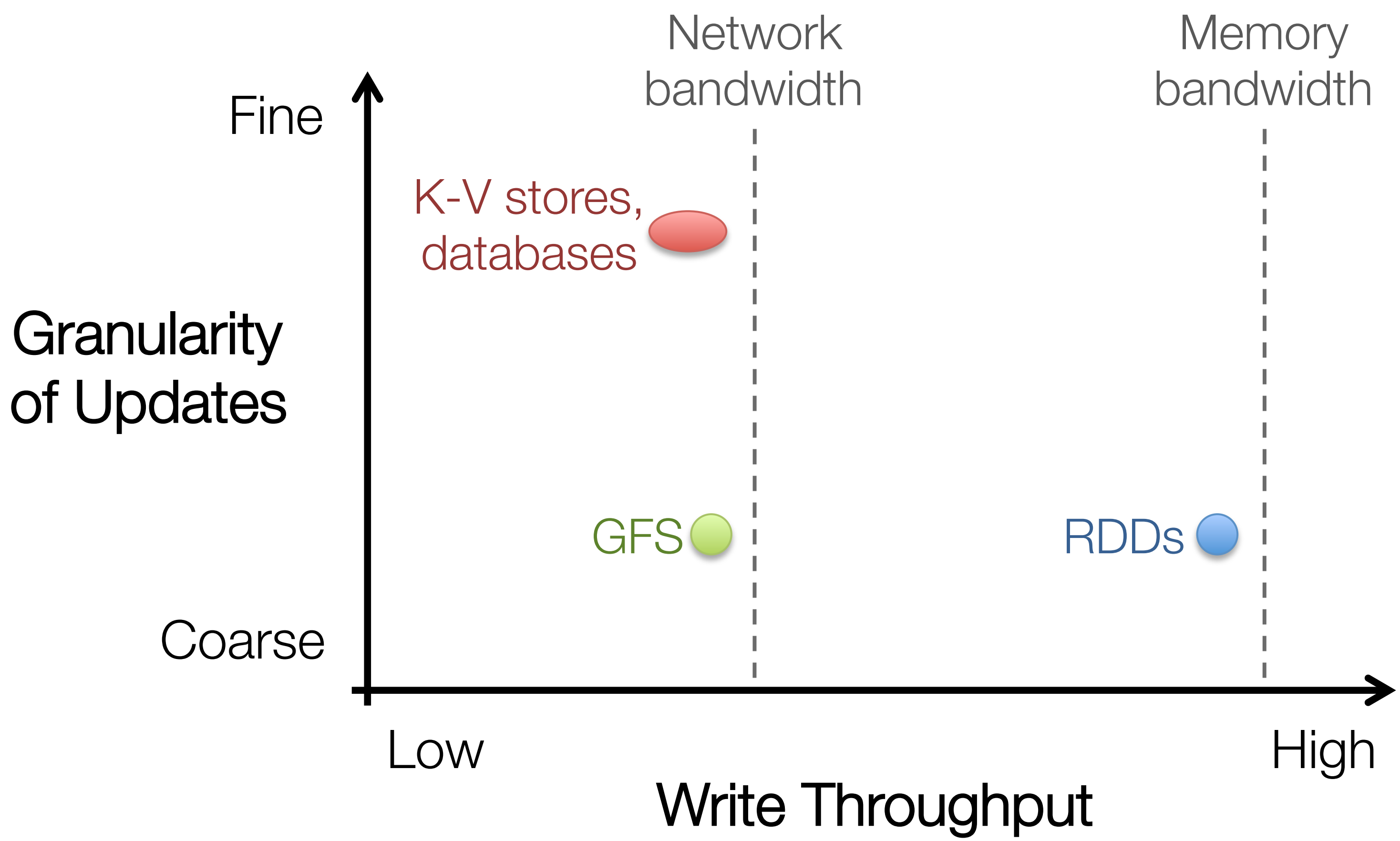
RDD Recovery



RDD Recovery



Tradeoff Space



RDDs vs Distributed Shared Mem.

Aspect	RDDs	Dist. Shared Mem. (including key-value stores, etc)
Writes	Coarse-grained	Fine-grained
Reads	Fine-grained	Fine-grained
Consistency	Trivial (immutable)	Expensive
Fault recovery	Fine-grained & low-cost using lineage	Replication or checkpoint/rollback
Straggler recovery	Possible using speculation	Difficult

Other Differences from MR

1. Explicit partitioning, partitioning-aware ops
   – E.g. a 3x speedup in PageRank

2. More complex DAGs of tasks
   – Better performance even if data is not reused

RDD API

Operation	Meaning
partitions()	Return a list of Partition objects
preferredLocations(p)	List nodes where partition p can be accessed faster due to data locality
dependencies()	Return a list of dependencies
iterator(p, parentIters)	Compute the elements of partition p given iterators for its parent partitions
partitioner()	Return metadata specifying how RDD records are partitioned across nodes

Supported Applications

Iterative MapReduce (e.g. machine learning)

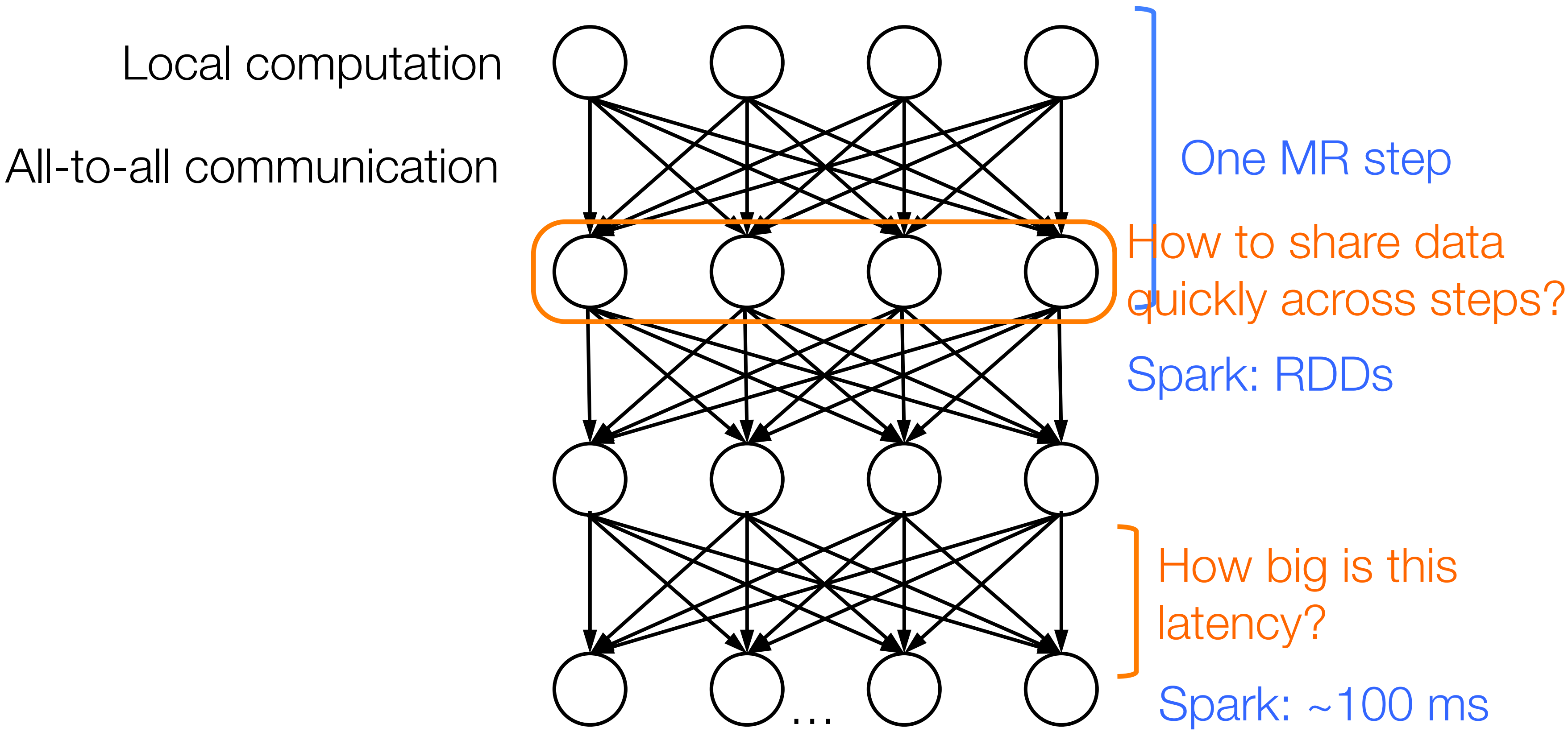
Pregel-like graph processing

Interactive ad-hoc queries

More were built later (e.g. SQL, streaming)

How General is Spark?

MapReduce + data sharing can emulate any distributed system!



Push vs Pull-Based Systems

"Push" = senders write to receivers (e.g. parallel DB)
"Pull" = senders write locally, receivers fetch (e.g. MR)

Aspect	Push	Pull
Latency	Lower	Higher
Throughput	Similar	Similar
Fault recovery	Expensive (rerun all senders)	Cheap
Straggler recovery	Difficult	Easy (backup tasks)
Elasticity / multitenancy	Difficult	Easy