Adaptive MapReduce using Situation-Aware Mappers

Rares Vernica[1] (HP Labs),
Andrey Balmin, Kevin S. Beyer, Vuk Ercegovac (IBM Research)

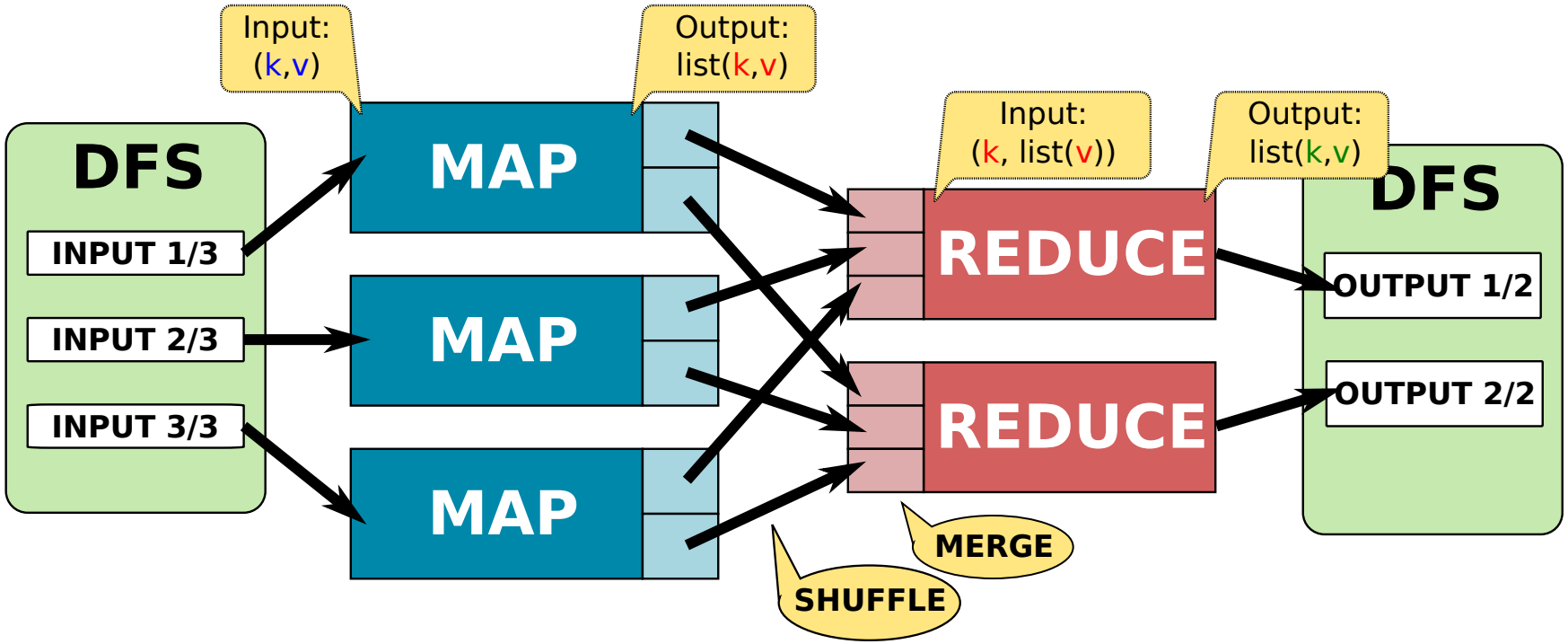
[1]Work done at IBM Research.

15th International Conference on Extending Database Technology,
March 26-30 2012

Outline

MapReduce Review

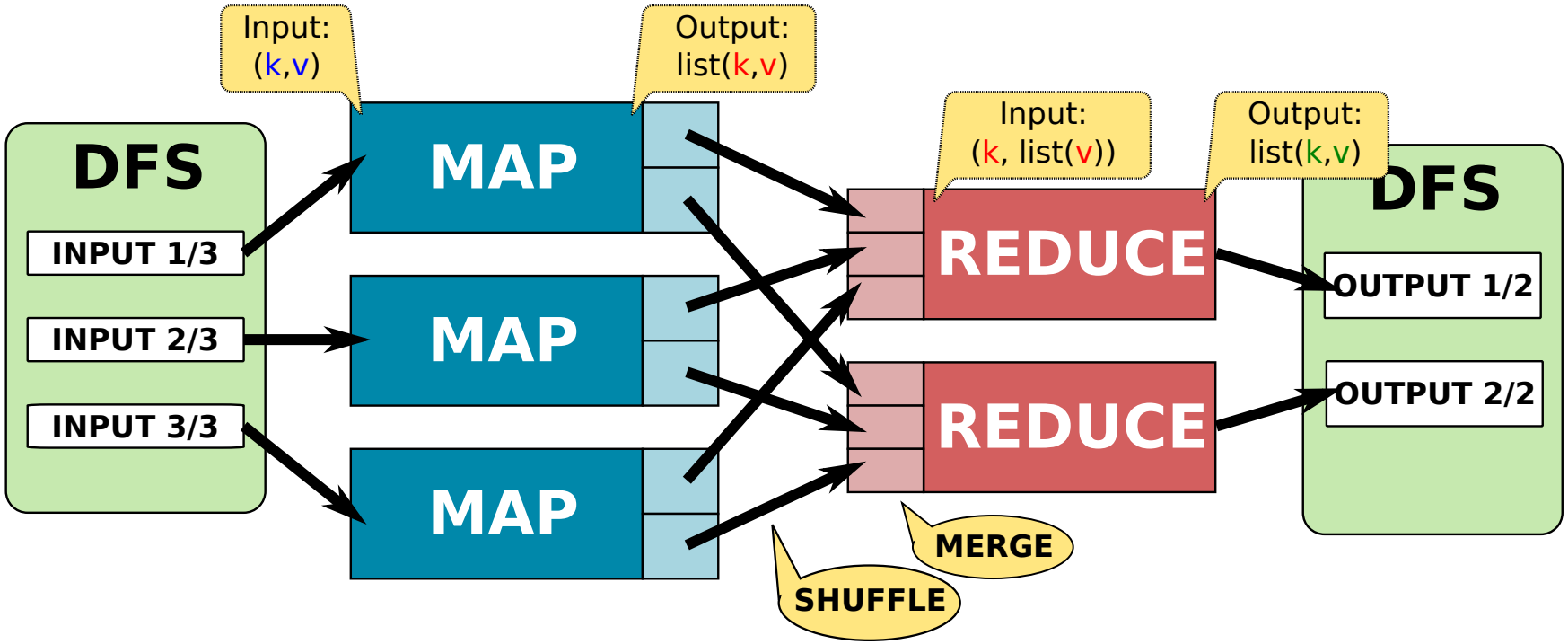
```
map    (k,v)          → list(k,v);
reduce (k,list(v))    → list(k,v).
```




```
combine (k,list(v)) → list(k,v).
```

MapReduce Review

```
map    (k,v)          → list(k,v);
reduce (k,list(v))    → list(k,v).
```




```
combine (k,list(v)) → list(k,v).
```

Motivation: MapReduce Issues

MapReduce

- Parallel data-processing framework
- Open-source implementation (Hadoop)
- Simple programming environment

MapReduce: "simplicity over performance"

Limited choice of execution strategies:

- Mappers checkpoint after every split
- Map outputs are sorted and written to file
- Reducer read statically predetermined partitions

Solutions to MapReduce Issues

MapReduce-inspired alternatives

- Dryad (Microsoft)
- Spark (UC Berkeley)
- Hyracks (UC Irvine)
- Nephele (TU Berlin)

Have more choices in runtime execution

Our Solution: Adaptive MapReduce

Make MapReduce (Hadoop) more flexible

- Leverage existing investment in:
  - Framework (Hadoop)
  - Query processing systems (Jaql, Pig, Hive)
- Techniques for:
  - Dynamic checkpoint intervals (Map)
  - Best-effort hash-based aggregation (Combine)
  - Dynamic, sample-based, partitioning (Reduce)
- Performance tuning:
  - ~~Cardinality and cost estimation~~ (due to UDFs)
  - **Adaptive** to runtime environment

Problem Statement: Adaptive MapReduce

Goals

Improve MapReduce (Hadoop) performance by:

- New runtime options
- Adaptive to runtime environment

Preserve Hadoop's

- Fault-tolerance
- Scalability
- Programability

Outline

Situation-Aware Mappers

Main idea

- Make MapReduce more dynamic

Situation-Aware Mappers

Main idea

- Make MapReduce more dynamic
- Mappers:
  - Aware of the global state of the job

Situation-Aware Mappers

Main idea

- Make MapReduce more dynamic
- Mappers:
  - Aware of the global state of the job
  - Communicate through a distributed meta-data store

Situation-Aware Mappers

Main idea

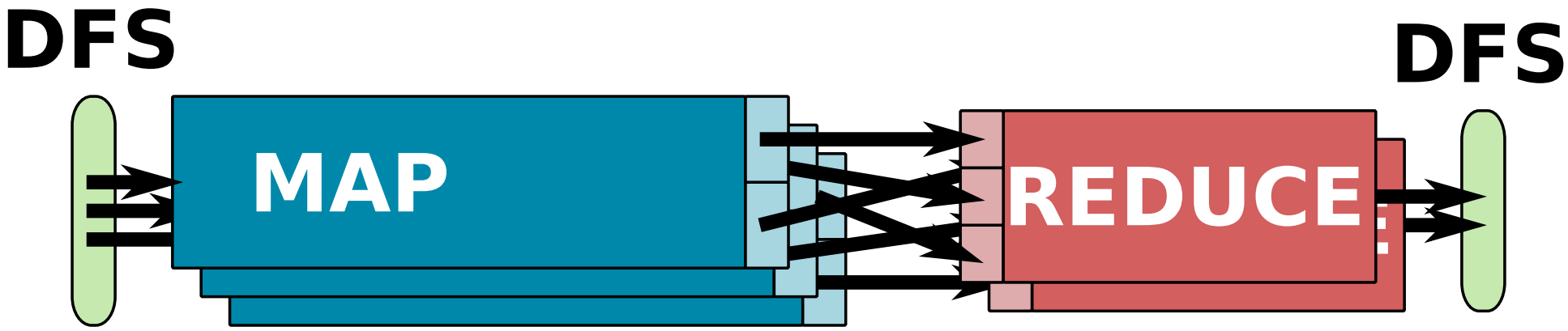
- Make MapReduce more dynamic
- Mappers:
  - Aware of the global state of the job
  - Communicate through a distributed meta-data store
  - **Break** assumption: isolation

Situation-Aware Mappers

Main idea

- Make MapReduce more dynamic
- Mappers:
  - Aware of the global state of the job
  - Communicate through a distributed meta-data store
  - **Break** assumption: isolation
- **Situation-Aware Mappers**

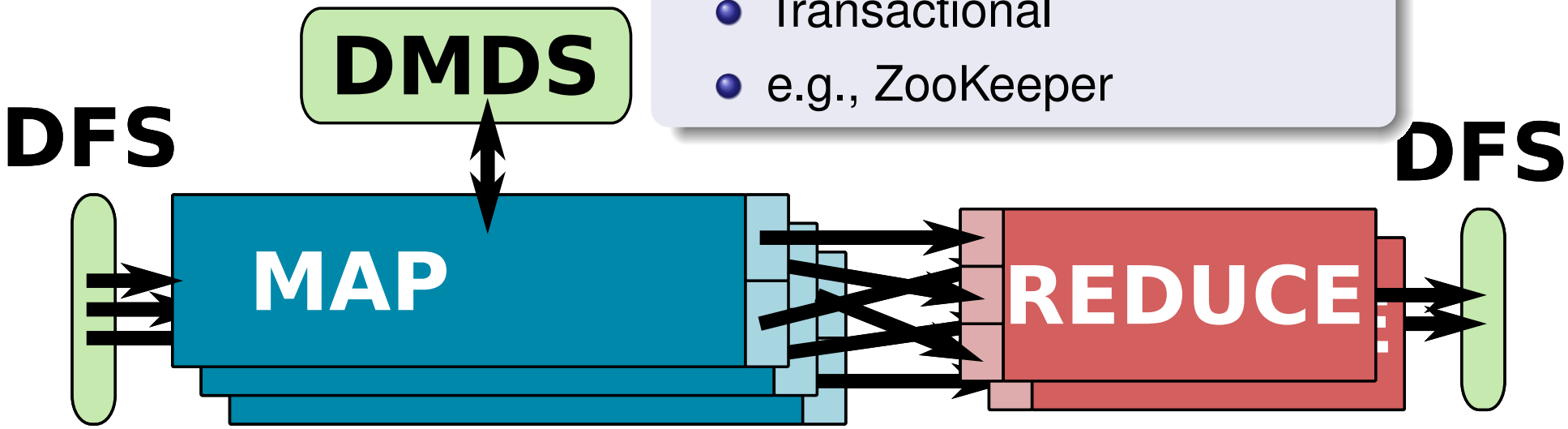
Adaptive MapReduce



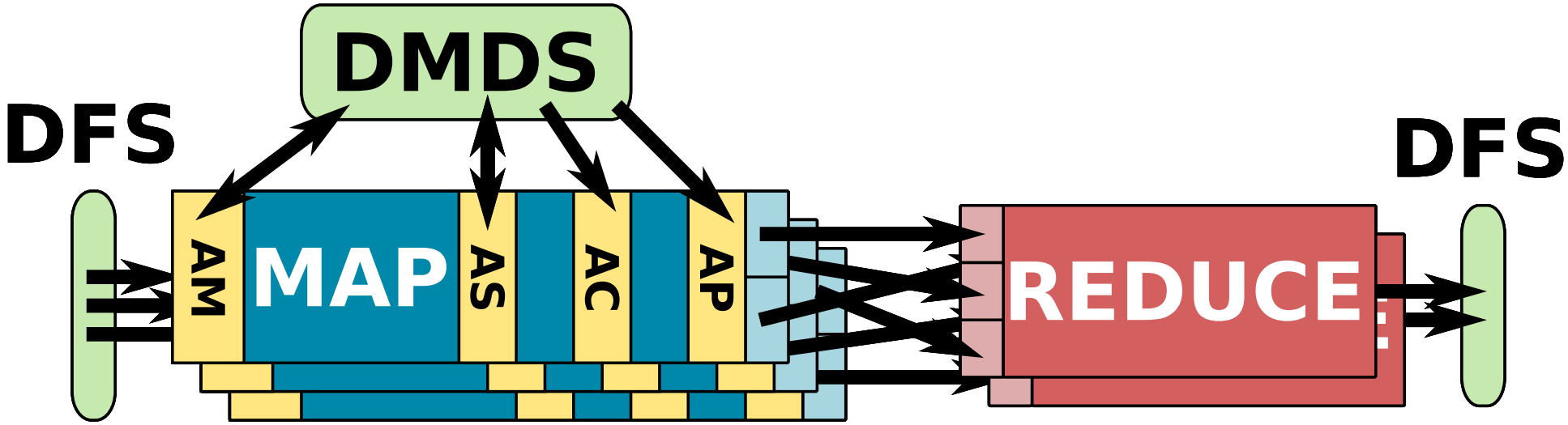
Adaptive MapReduce

Distributed Meta-Data Store

- Distributed read/write
- Transactional
- e.g., ZooKeeper



Adaptive MapReduce

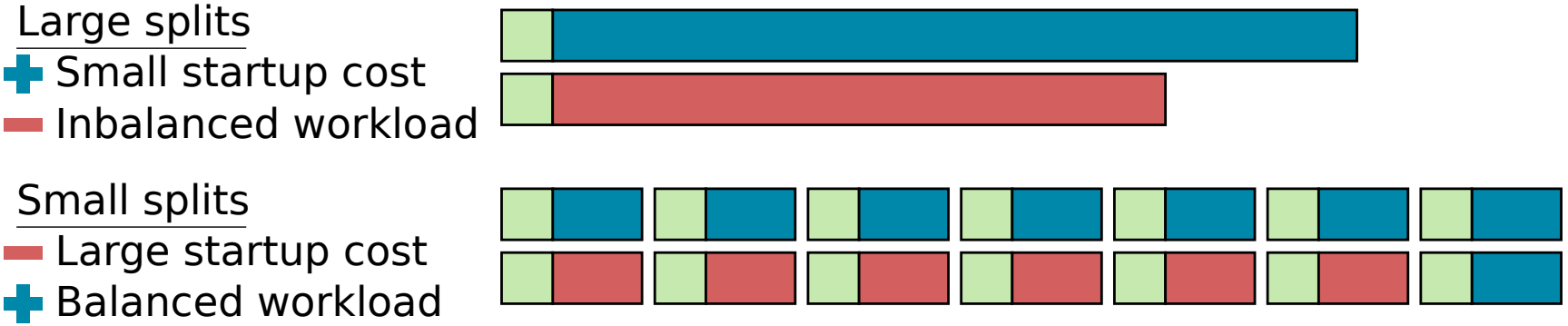


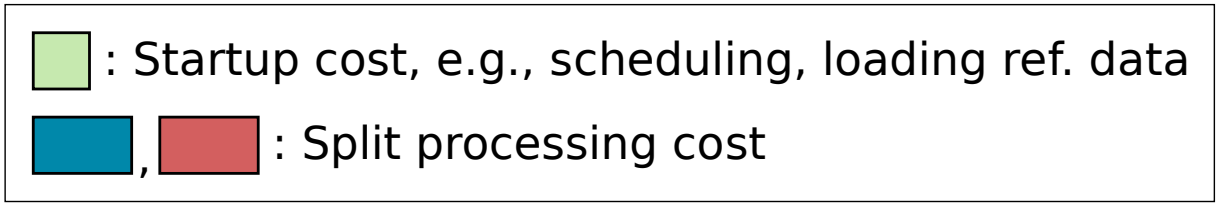
Adaptive Techniques

- AM: Adaptive Mappers
- AC: Adaptive Combiners
- AS: Adaptive Sampling
- AP: Adaptive Partitioning

Adaptive Mappers Motivation

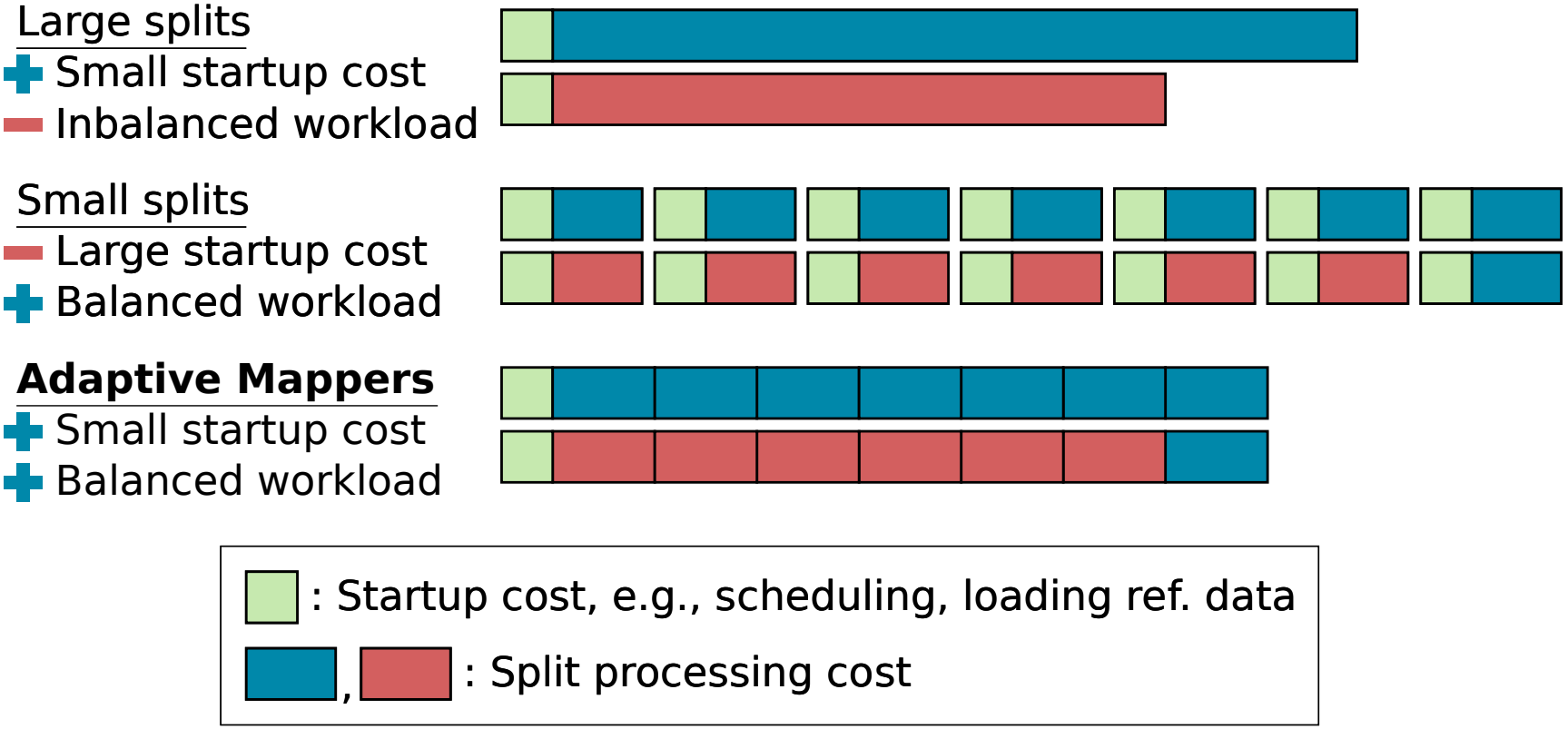
- Input data is divided into splits
- One-to-one correspondence of mappers and splits
- AM **decouple** # splits from # mappers



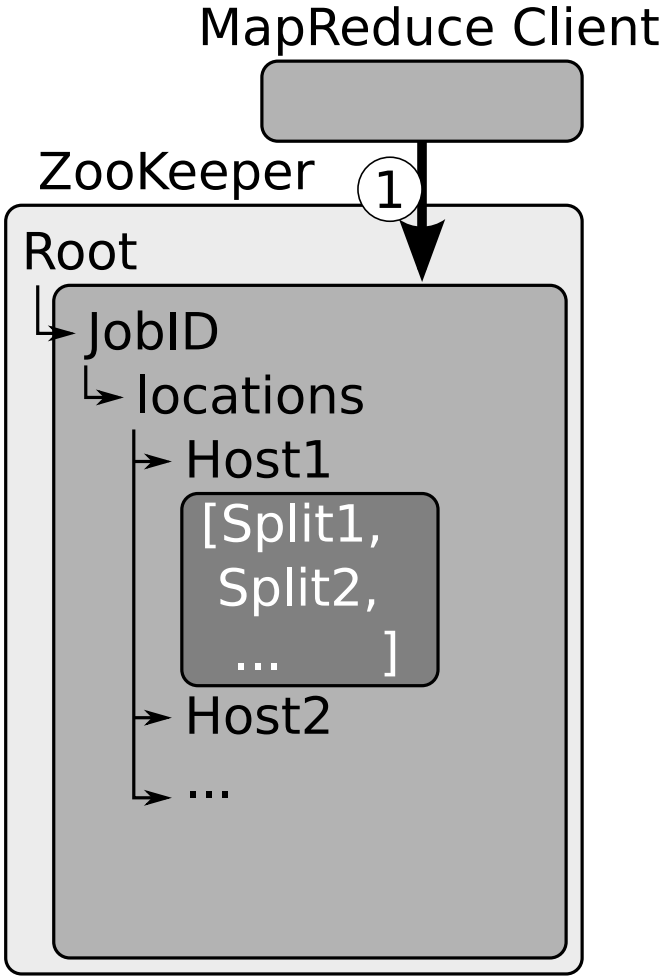


Adaptive Mappers Motivation

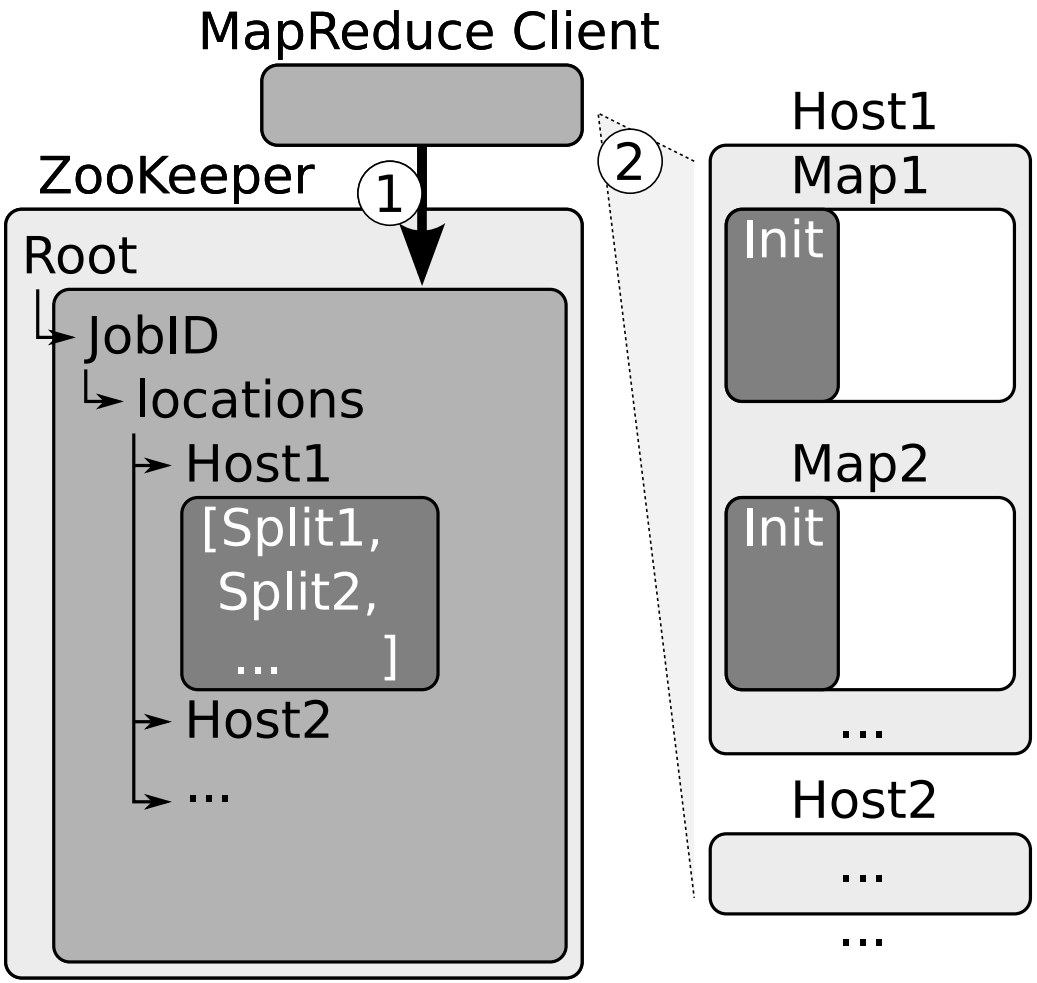
- Input data is divided into splits
- One-to-one correspondence of mappers and splits
- AM **decouple** # splits from # mappers



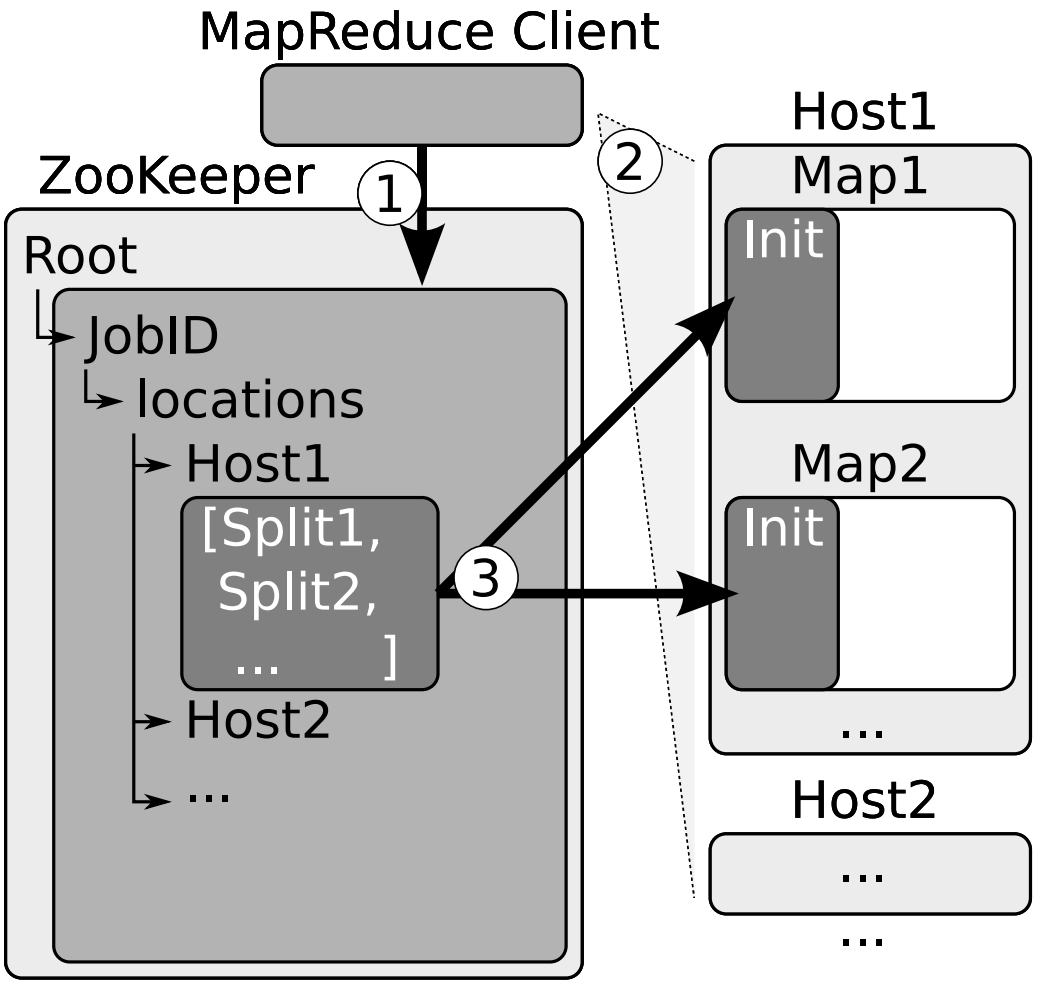
Adaptive Mappers Algorithm



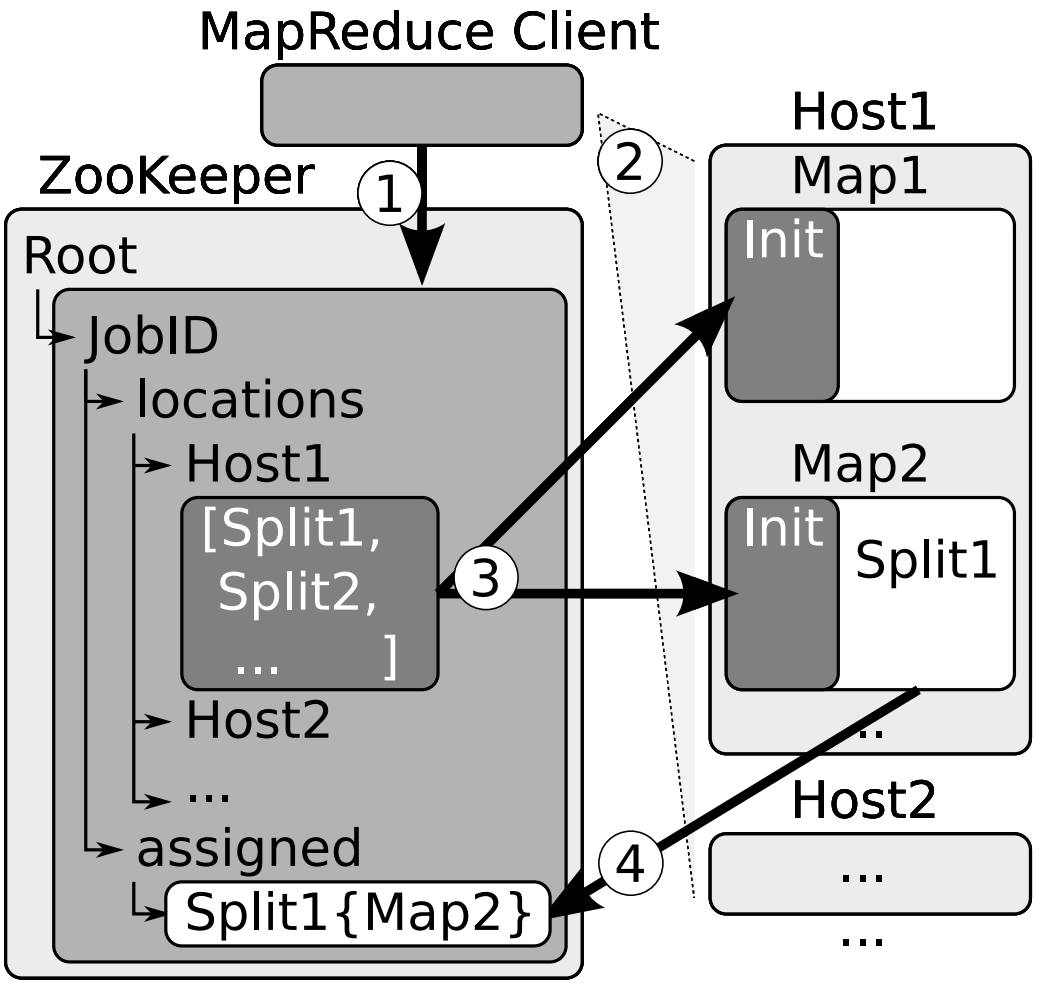
Adaptive Mappers Algorithm



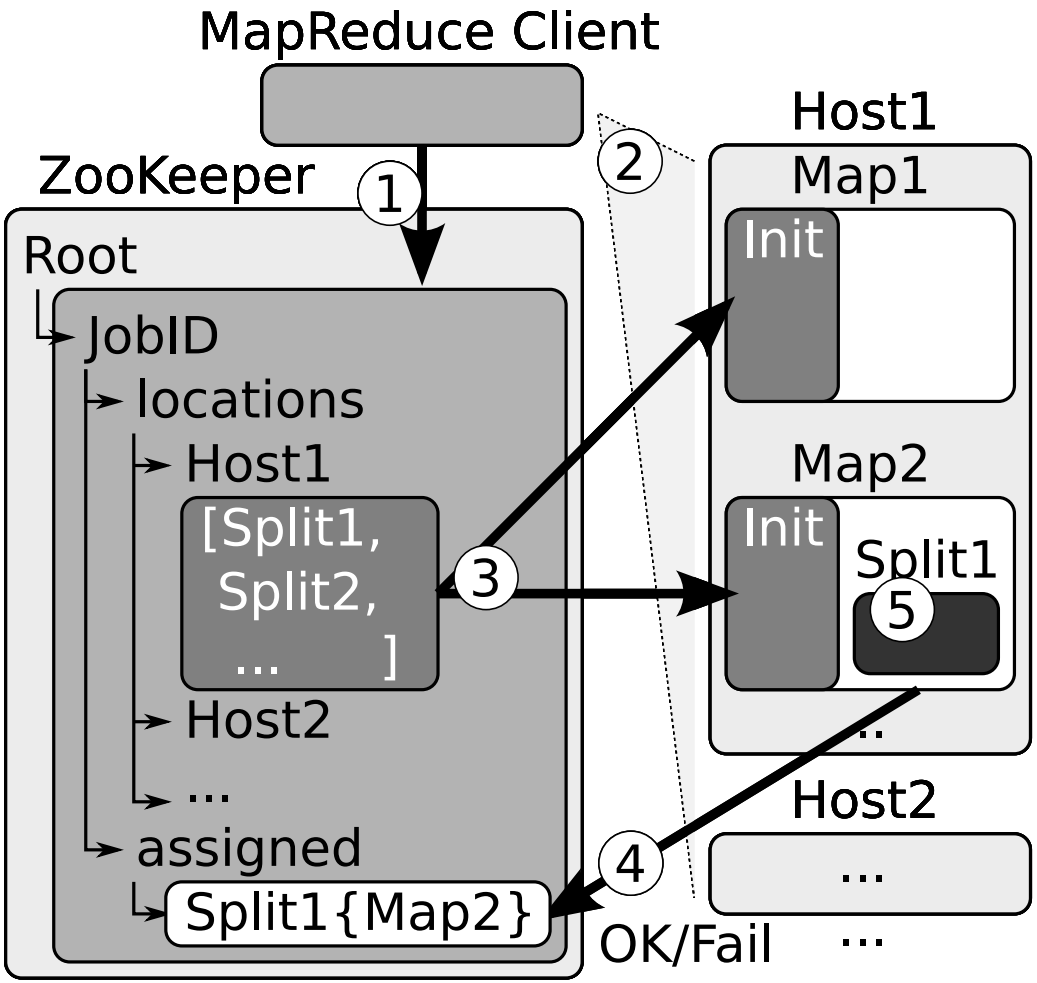
Adaptive Mappers Algorithm



Adaptive Mappers Algorithm



Adaptive Mappers Algorithm



- Store meta-data in ZooKeeper
- Implemented as a new InputFormat

Adaptive Mappers Algorithm

Additional Features

- Process local splits first, then remote splits
- Fault tolerance
  - Restated task unlocks splits
  - Split reprocessing is shared
- Scheduler aware (FIFO, FAIR, and FLEX)

Experimental Setting

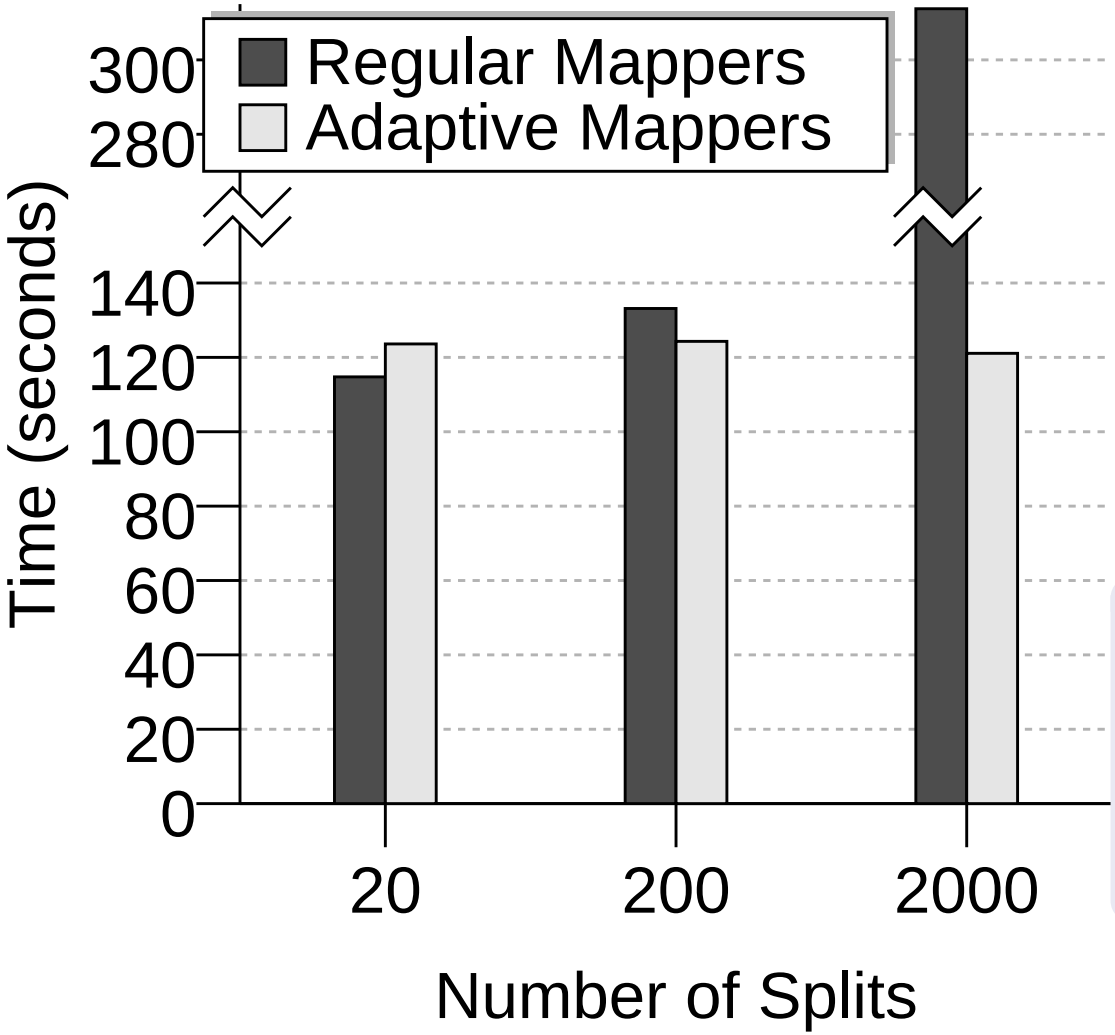
Hardware

- 40-node IBM Systemx iDataPlex dx340
- Two quad-core Intel Xeon E5540 64-bit 2.83GHz
- 32GB RAM
- Four SATA disks
- 160 map and 160 reduce slots

Software

- Ubuntu Linux, kernel 2.6.32-24 64-bit server edition
- Java 1.6 64-bit server edition
- Hadoop 0.20.2
- ZooKeeper 3.3.1

Start-up Cost vs. ZooKeeper Overhead



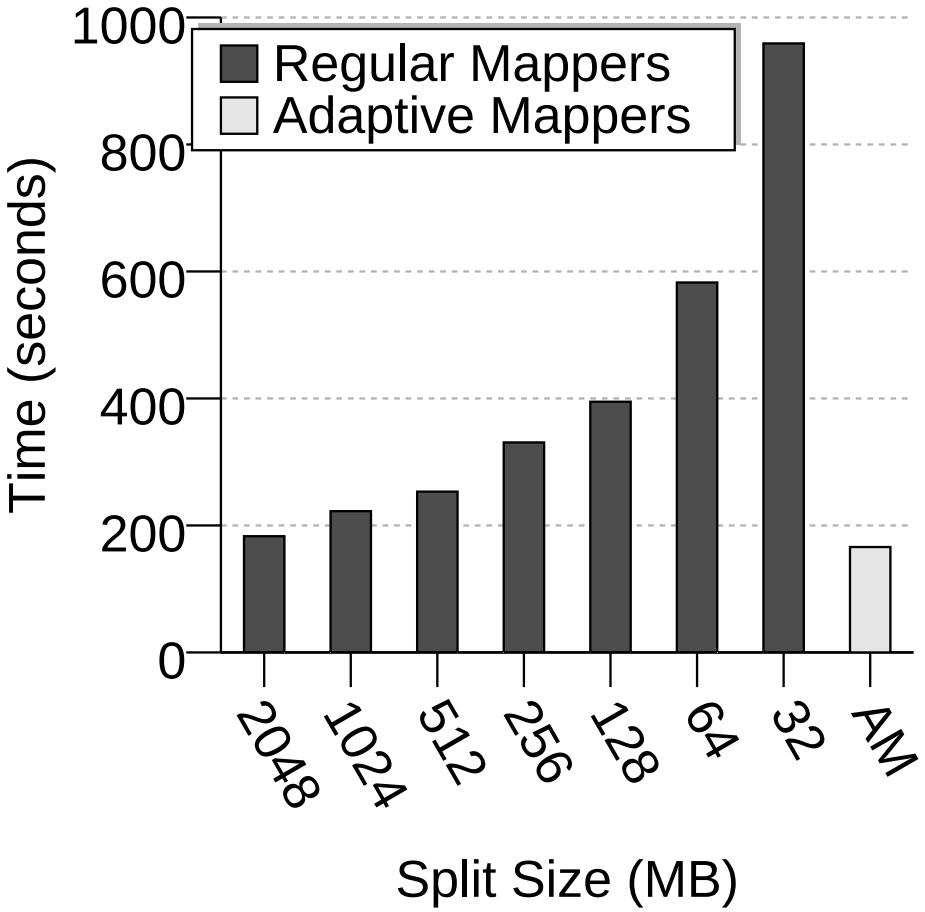
- 2000 1-byte records
- Sleep 1s/record
- 5 nodes, 20 map slots
- 20-2000 Reg. Mappers
- 20 Adaptive Mappers

- **Small ZooKeeper overhead**
- **Large Map startup cost ∼2s/map**

Adaptive Mappers Workloads

1. Set-Similarity Join [Vernica et al., 2010]
   - Publication datasets
   - DBLP: 1.2M records, 310MB
   - CITESEERX: 1.3M records, 1,750MB
   - Increased to $\times 10$ and $\times 100$
2. JOIN
   - Single dataset ("fact" table), Sort Benchmark data generator
   - Fan-out coefficient ("dimension" table)
   - average join fan-out $1 : 30$
   - TERASORT: 1B records, 93GB

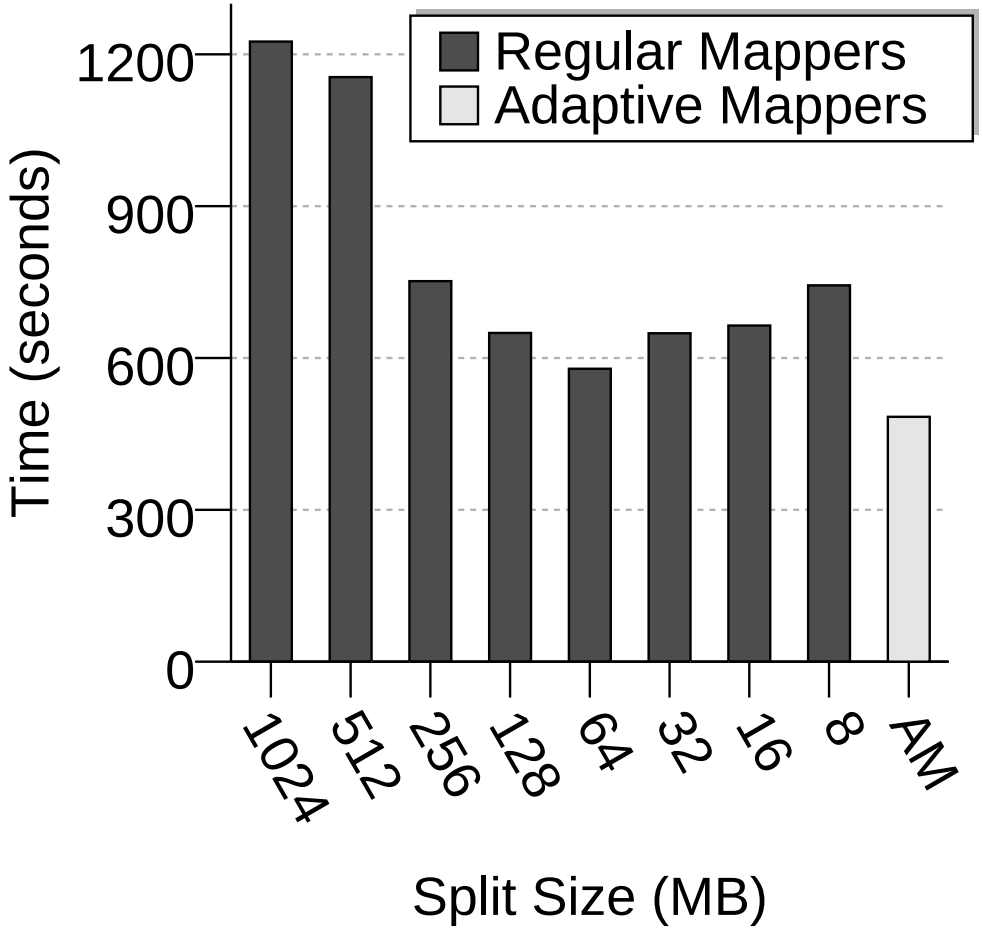
Adaptive Mappers Experiments - Set-Similarity Join



- Stage 3:
  One-Phase Record Join
- Broadcast join equivalent
- DBLP and CITESEERX $\times 10$
- Single wave of AM

- **$\times 3$ speedup over default Hadoop split size (64MB)**
- **Optimal with no tuning**

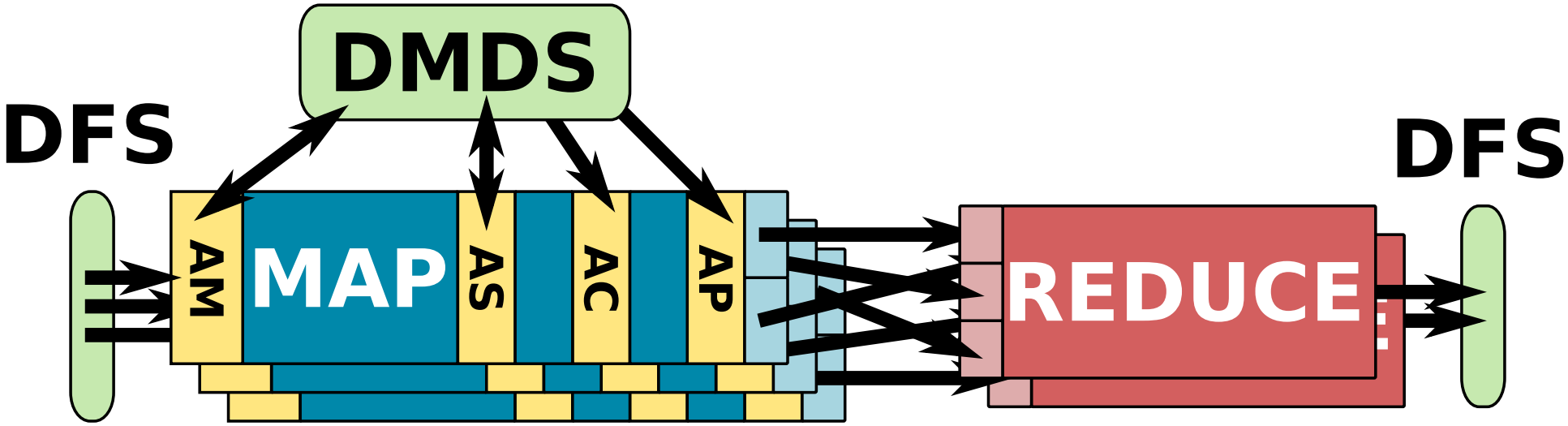
Adaptive Mappers Experiments - JOIN



- Map-only job
- 1B TERASORT records
- Models a skewed join
- Single wave of AM

- Regular Mappers:
  - **Large split: data skew**
  - **Small split: scheduling and start-up overhead**
- **Optimal with no tuning**

Adaptive MapReduce



Adaptive Techniques

- AM: Adaptive Mappers
- **AC: Adaptive Combiners**
- AS: Adaptive Sampling
- AP: Adaptive Partitioning

Adaptive Combiners

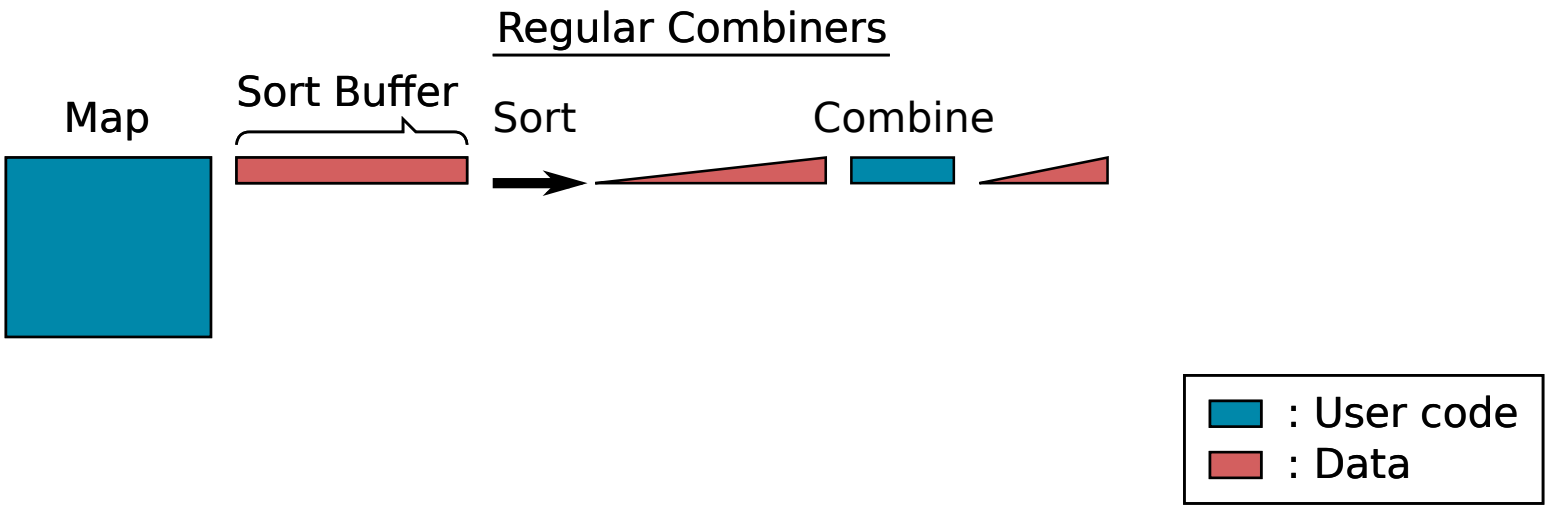
Main idea

- Replace sort with hashing
- Reduce serialization, sort, and IO

Adaptive Combiners

Main idea

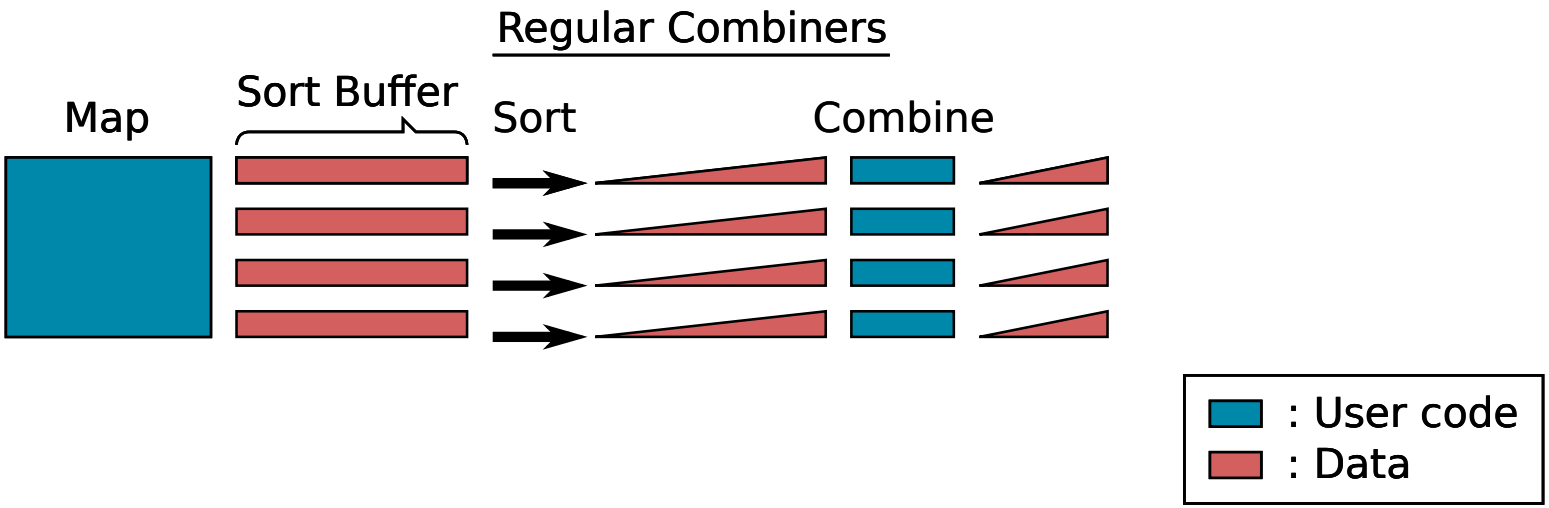
- Replace sort with hashing
- Reduce serialization, sort, and IO



Adaptive Combiners

Main idea

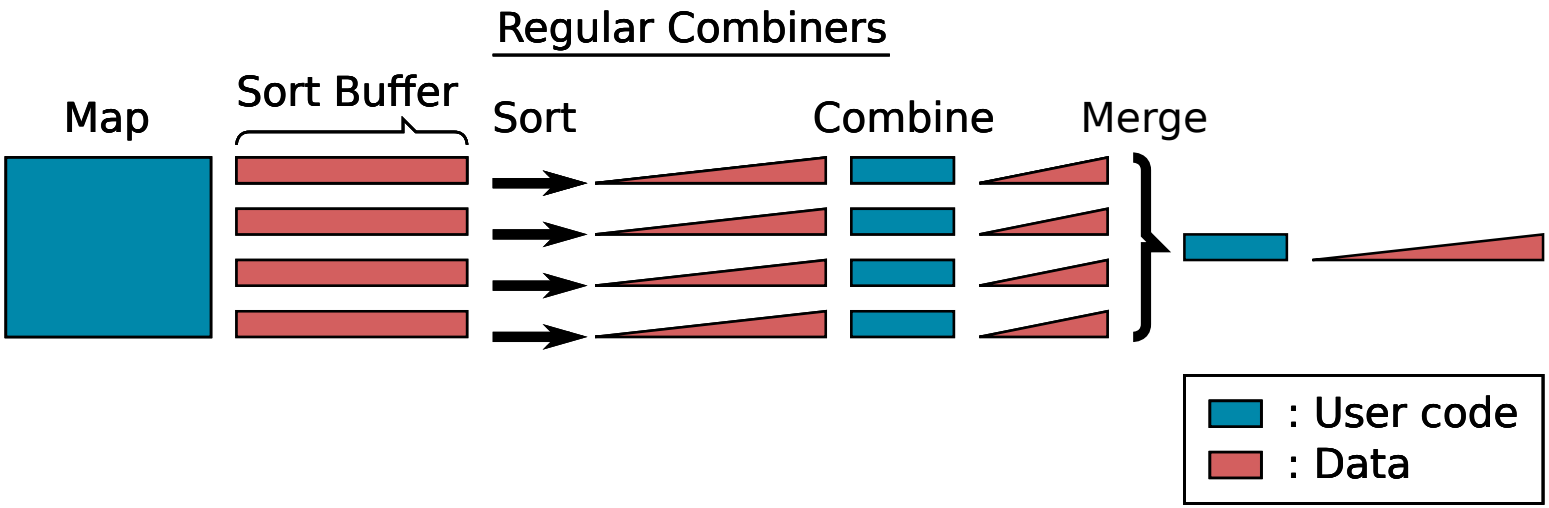
- Replace sort with hashing
- Reduce serialization, sort, and IO



Adaptive Combiners

Main idea

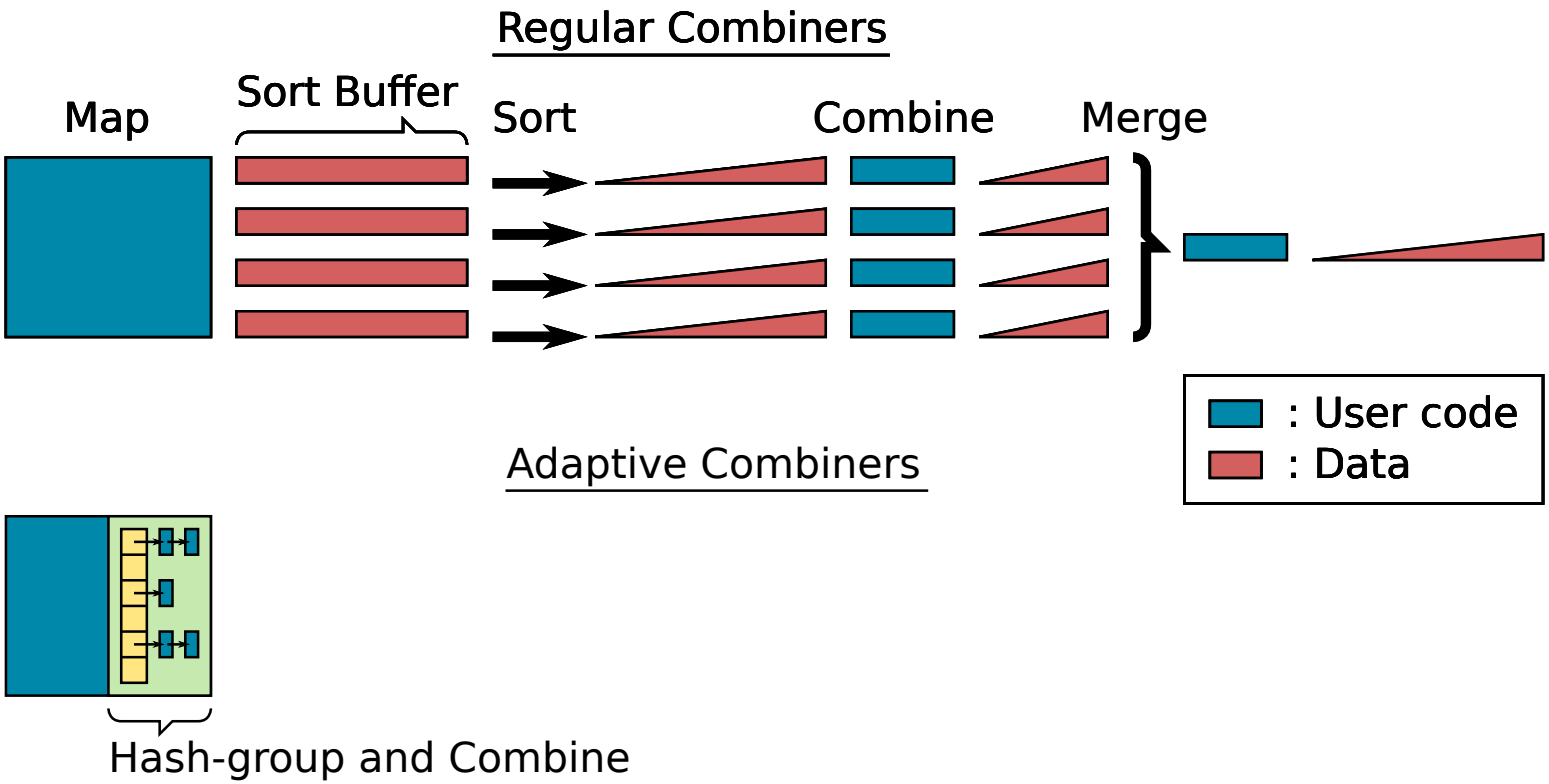
- Replace sort with hashing
- Reduce serialization, sort, and IO



Adaptive Combiners

Main idea

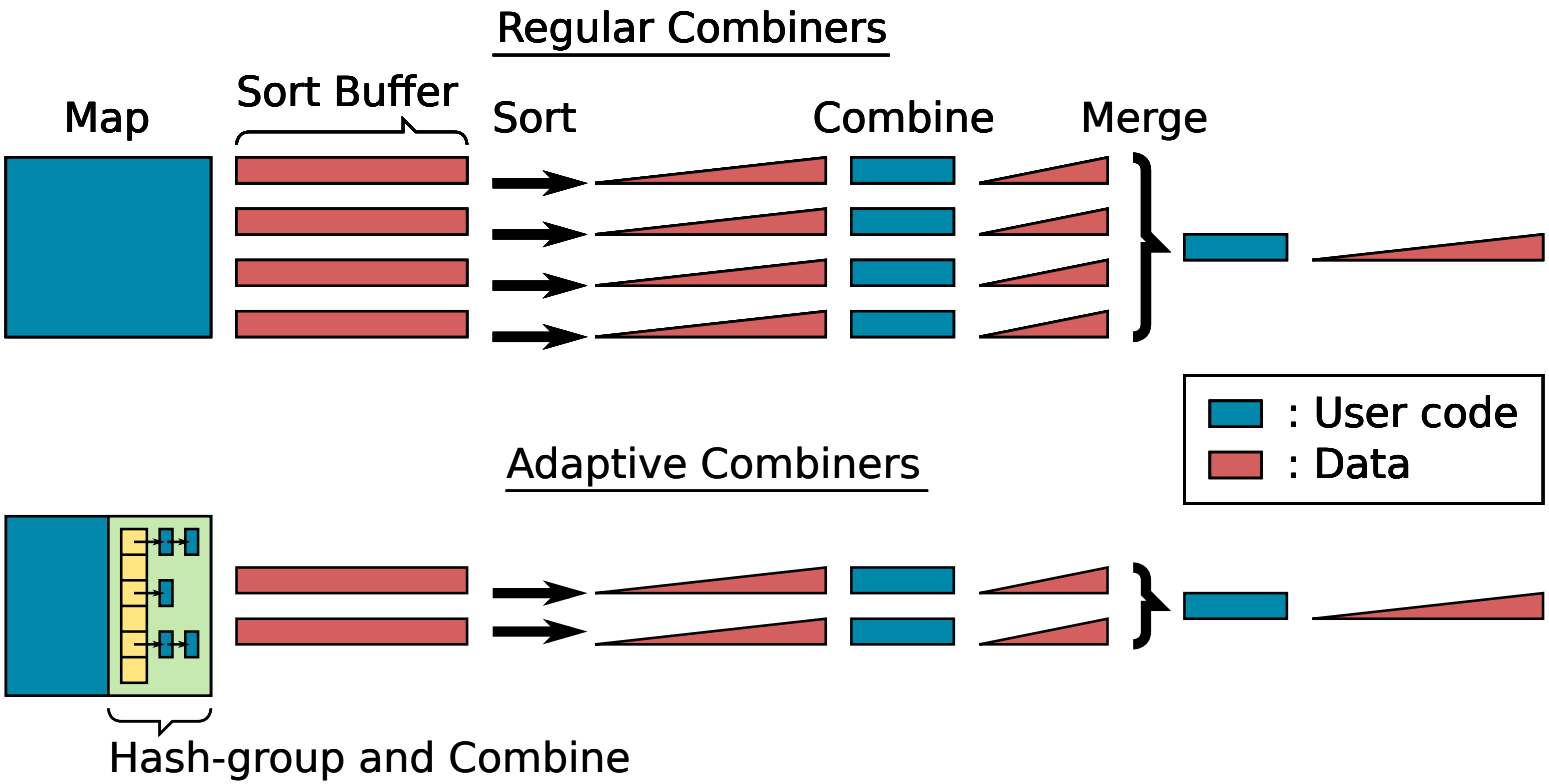
- Replace sort with hashing
- Reduce serialization, sort, and IO



Adaptive Combiners

Main idea

- Replace sort with hashing
- Reduce serialization, sort, and IO



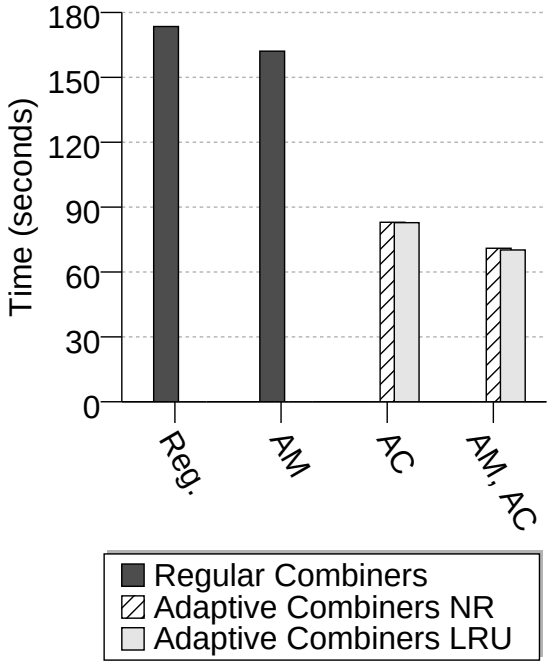
Adaptive Combiners Details

- “Best-effort” aggregation
- Never spill to disk
- Hash-table replacement policies:
  - No-Replacement (NR)
  - Least-Recently-Used (LRU)
- Implemented as:
  - Library for Hadoop
  - Optimization choice for Jaql

Adaptive Combiners Experiments

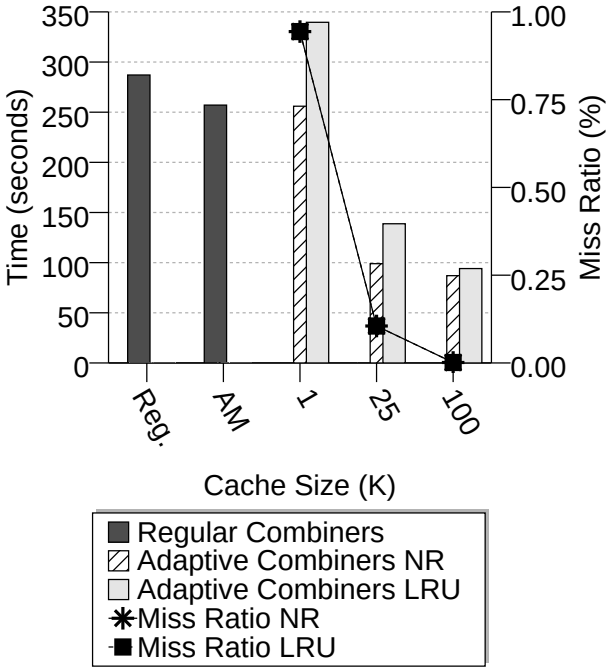
GROUP-BY

- Synthetic dataset with 3 dimensions (A1, A2, and A3) and 1 fact
- Group records and apply aggregation function
- TWL: 10B records, 120GB



GROUP-BY on A1

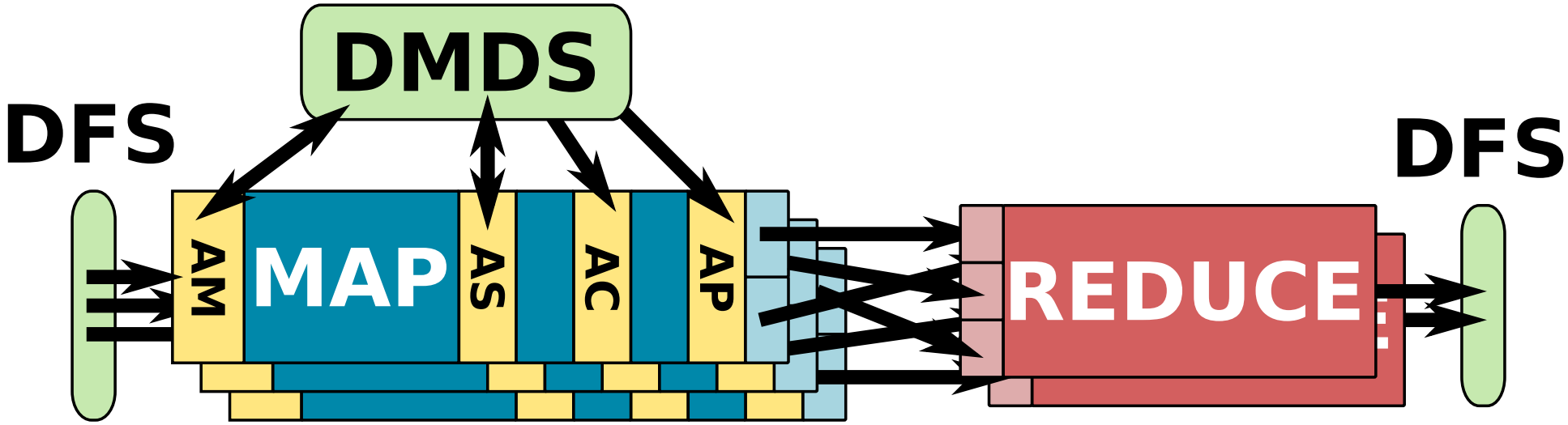
×2.5 speedup



GROUP-BY on A1 and A2

×3 speedup

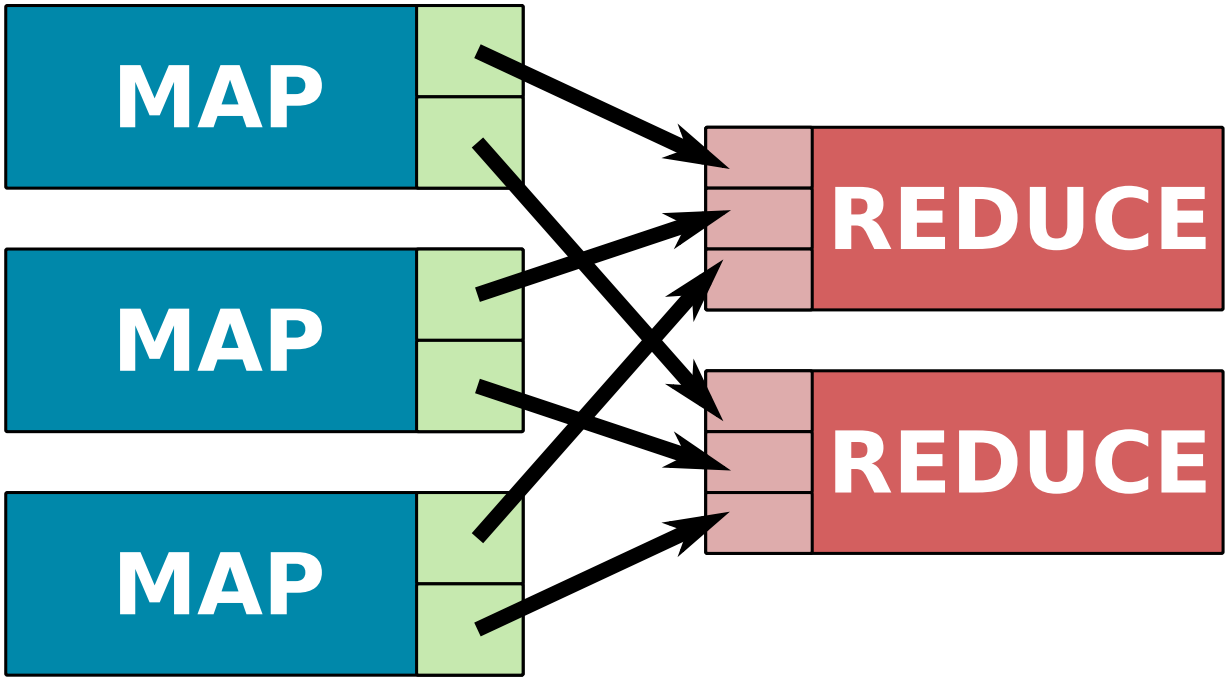
Adaptive MapReduce



Adaptive Techniques

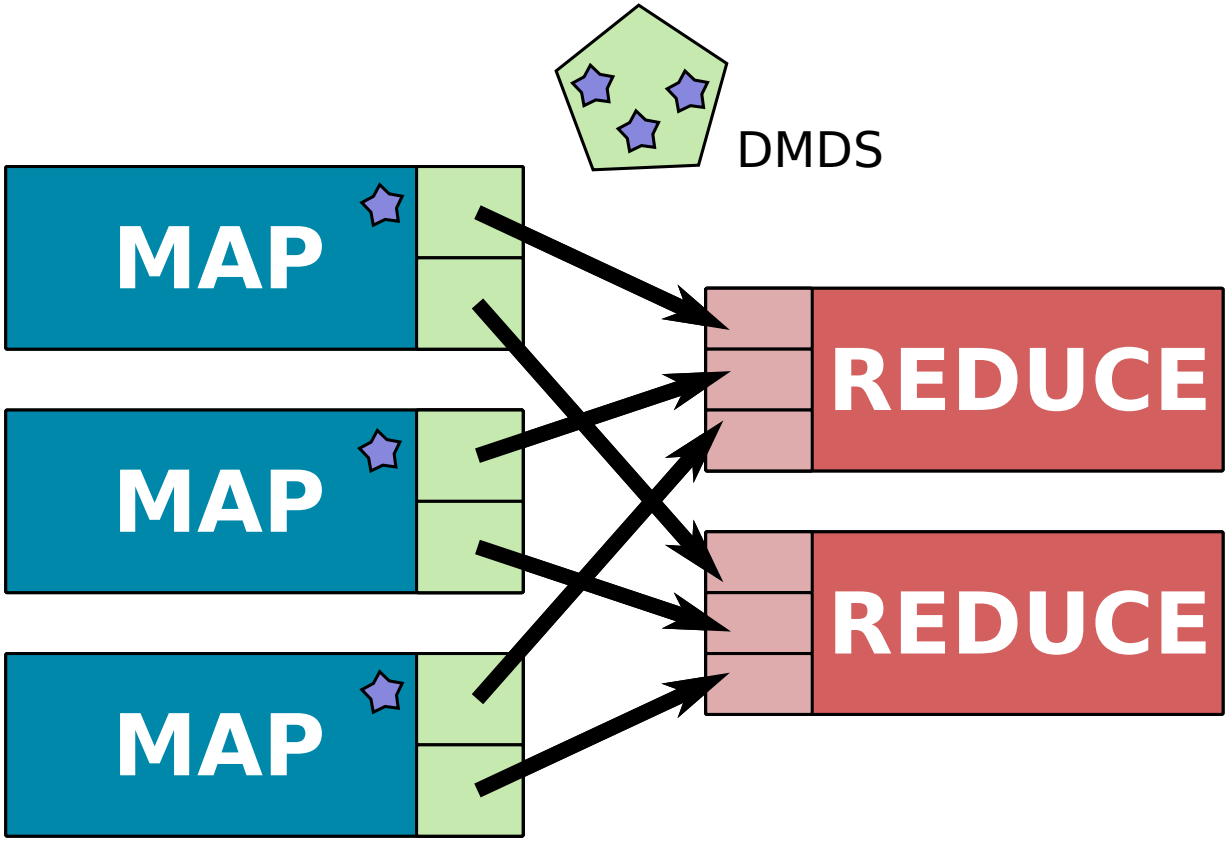
- AM: Adaptive Mappers
- AC: Adaptive Combiners
- **AS: Adaptive Sampling**
- **AP: Adaptive Partitioning**

Adaptive Sampling and Partitioning



Adaptive Sampling and Partitioning

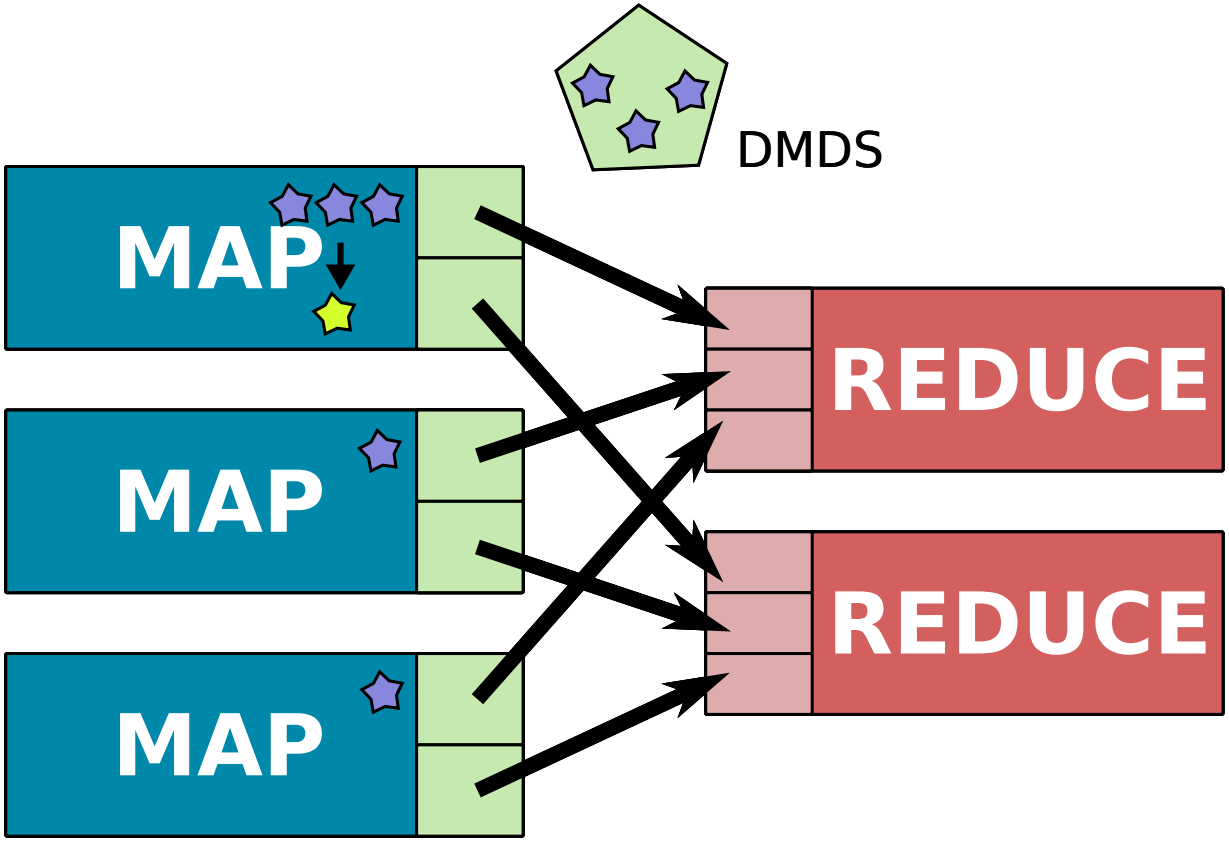
Step 1 Compute and publish local histogram



Adaptive Sampling and Partitioning

Step 1 Compute and publish local histogram

Step 2 Collect local histograms and compute partitioning function

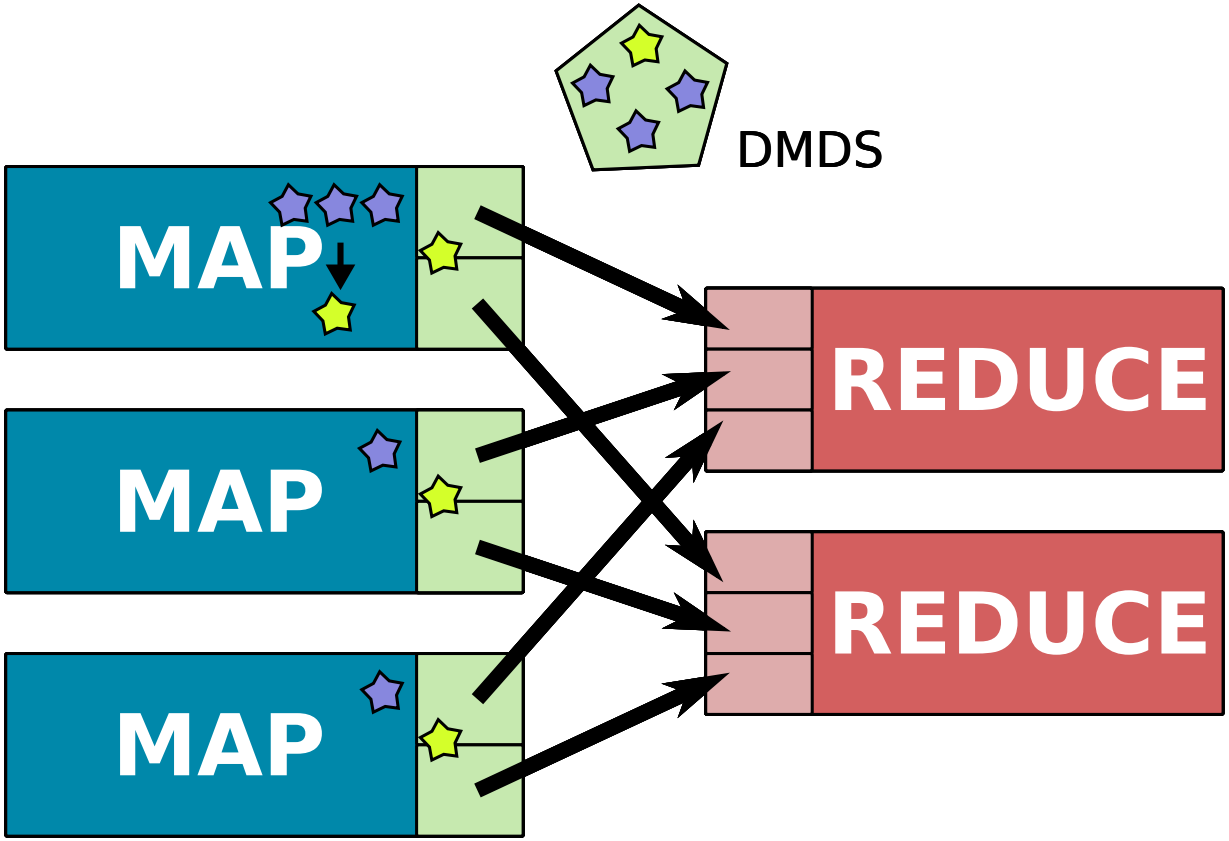


Adaptive Sampling and Partitioning

Step 1 Compute and publish local histogram

Step 2 Collect local histograms and compute partitioning function

Step 3 Broadcast partitioning function



Summary

- Adaptive runtime techniques for MapReduce
- Situation-Aware Mappers
- Make MapReduce more dynamic

- Up to $\times 3$ speedup for well-tuned jobs
- Orders of magnitude speedup for badly tuned jobs
- Never hurt performance
- Configure themselves
- Part of IBM InfoSphere BigInsights

Vernica, R., Carey, M., and Li, C. (2010).
Efficient parallel set-similarity joins using MapReduce.
In SIGMOD Conference.