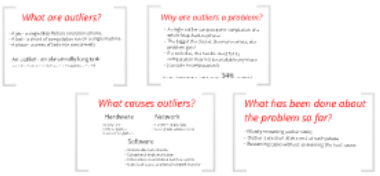
Reining in the Outliers in Map-Reduce Clusters using

Mantri

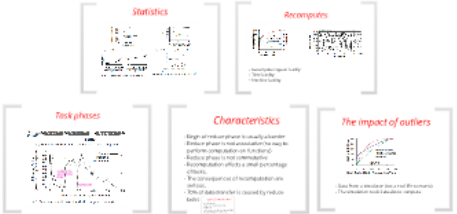
Motivation



The main idea

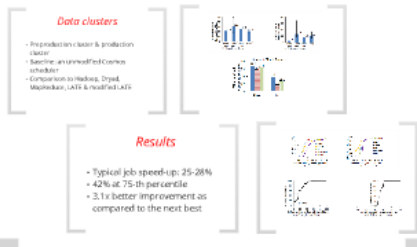
The setting

The scale of the problem



The implementation

Evaluation



Reining in the Outliers in

Map-Reduce Clusters using

Mantri

Motivation

What are outliers?

- A job - a single Map-Reduce execution scheme.
- A task - a chunk of computation run on a single machine
- A phase - a series of tasks run concurrently

An outlier - an abnormally long task

(for the purpose of this presentation: > 1.5 avg execution time)

Why are outliers a problem?

- A single outlier can postpone completion of a whole Map-Reduce phase
- The bigger the claster, the more serious the problem gets!
- If a task dies, the results used for its computation may not be available any more
- (cascade recomputation!)

Outliers inflate completion time of jobs by 34% (at median)

What causes outliers?

Hardware

- Disk failure
- CPU congestion
- Memory congestion

Network

- Transferring task data
- Moving data between racks

Software

- Undivisable data chunks
- Suboptimal implementation
- Scheduling an additional task has a price
- More task cause additional network transfer

What has been done about the problem so far?

- Mostly restarting outlier tasks
- Outlier detection at the end of each phase
- Restarting tasks without examining the root cause

What are outliers?

- A job - a single Map-Reduce execution scheme.
- A task - a chunk of computation run on a single machine
- A phase - a series of tasks run concurrently

An outlier - an abnormally long task

(for the purpose of this presentation: > 1.5 avg execution time)

What causes outliers?

Hardware

- Disk failure
- CPU congestion
- Memory congestion

Network

- Transferring task data
- Moving data between racks

Software

- Undivisable data chunks
- Suboptimal implementation
- Scheduling an additional task has a price
- More task cause additional network transfer

Why are outliers a problem?

- A single outlier can postpone completion of a whole Map-Reduce phase
- The bigger the claster, the more serious the problem gets!
- If a task dies, the results used for its computation may not be available any more
- (cascade recomputation!)

Outliers inflate completion time of jobs by **34%** (at median)

What has been done about the problem so far?

- Mostly restarting outlier tasks
- Outlier detection at the end of each phase
- Restarting tasks without examining the root cause

The main idea

Margin for improvement

- Restarting outliers is not always a good choice
- Detecting outliers early
- Use progress reports!
- Network-aware task placement
- Duplicating resource-intensive results

The Mantri approach

- Don't restart tasks that run long because of large amount of data to process
- Don't restart tasks that run long because of network congestion!*

*Unless there is potential for better network transfer

- Place tasks in a network-aware way

The estimate function

We assume that task completion time can be expressed as a function of the following arguments:

f(data size, code, machine, network location)

We will try to build an estimate of this function and we will use this estimate in our scheduler algorithm

Margin for improvement

- Restarting outliers is not always a good choice
- Detecting outliers early
- Use progress reports!
- Network-aware task placement
- Duplicating resource-intensive results

The Mantri approach

- Don't restart tasks that run long because of large amount of data to process
- Don't restart tasks that run long because of network congestion!*

  *Unless there is potential for better network transfer

- Place tasks in a network-aware way

The estimate function

We assume that task completion time can be expressed as a function of the following arguments:

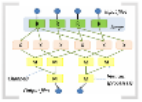
f(data size, code, machine, network location)

We will try to build an estimate of this function and we will use this estimate in our scheduler algorithm

The setting

Cosmos & Scope

- Cosmos - a commercial upgrade to Dryad
- Most of the jobs written in Scope language
  - Mash-up: SQL + user code (C#)



- Compiler transforms programs into DAGs of dependant tasks
- Compiler optimizes programs for maximal concurrency.

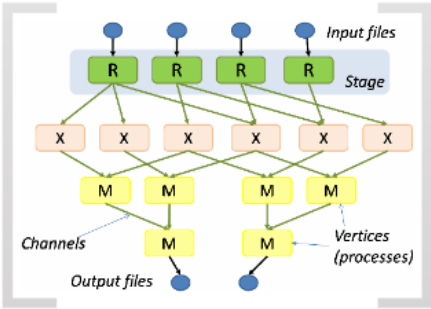
The Bing cluster

- Reading through network or from local disk storage
- Writing (always!) to local disk storage
- Data stored on the same machines that perform the computation
- By default Cosmos scheduler assigns tasks to machines where the data is available (data locality)
- Cross-rack traffic is costly
- Sum of data storage of the entire rack is smaller than the outgoing bandwidth

The input data

- Log files from Cosmos scheduler
- Log files from Scope compiler
- Begin and end times for each task
- Input / output data sizes for each task
- Task workflow graph

Cosmos & Scope

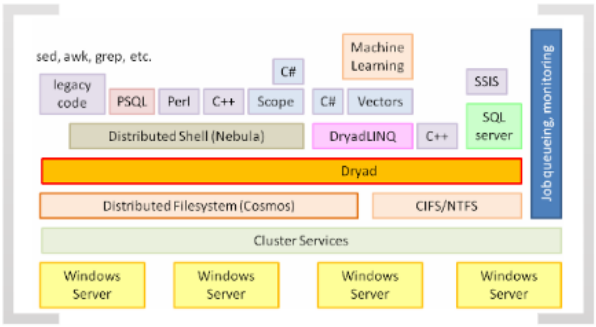


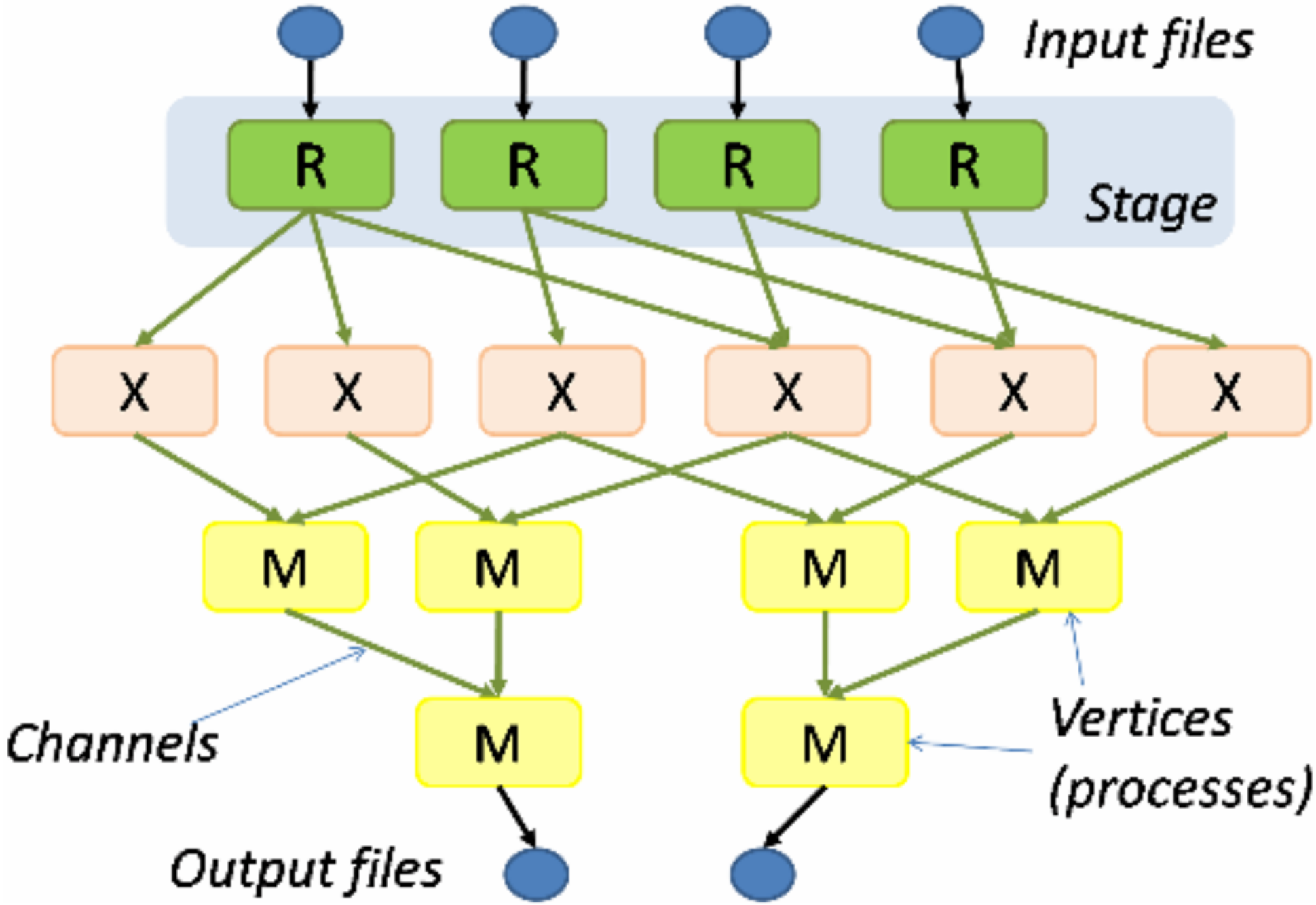
- Cosmos - a commercial upgrade to Dryad
- Most of the jobs written in Scope language
  - Mash-up: SQL + user code (C#)

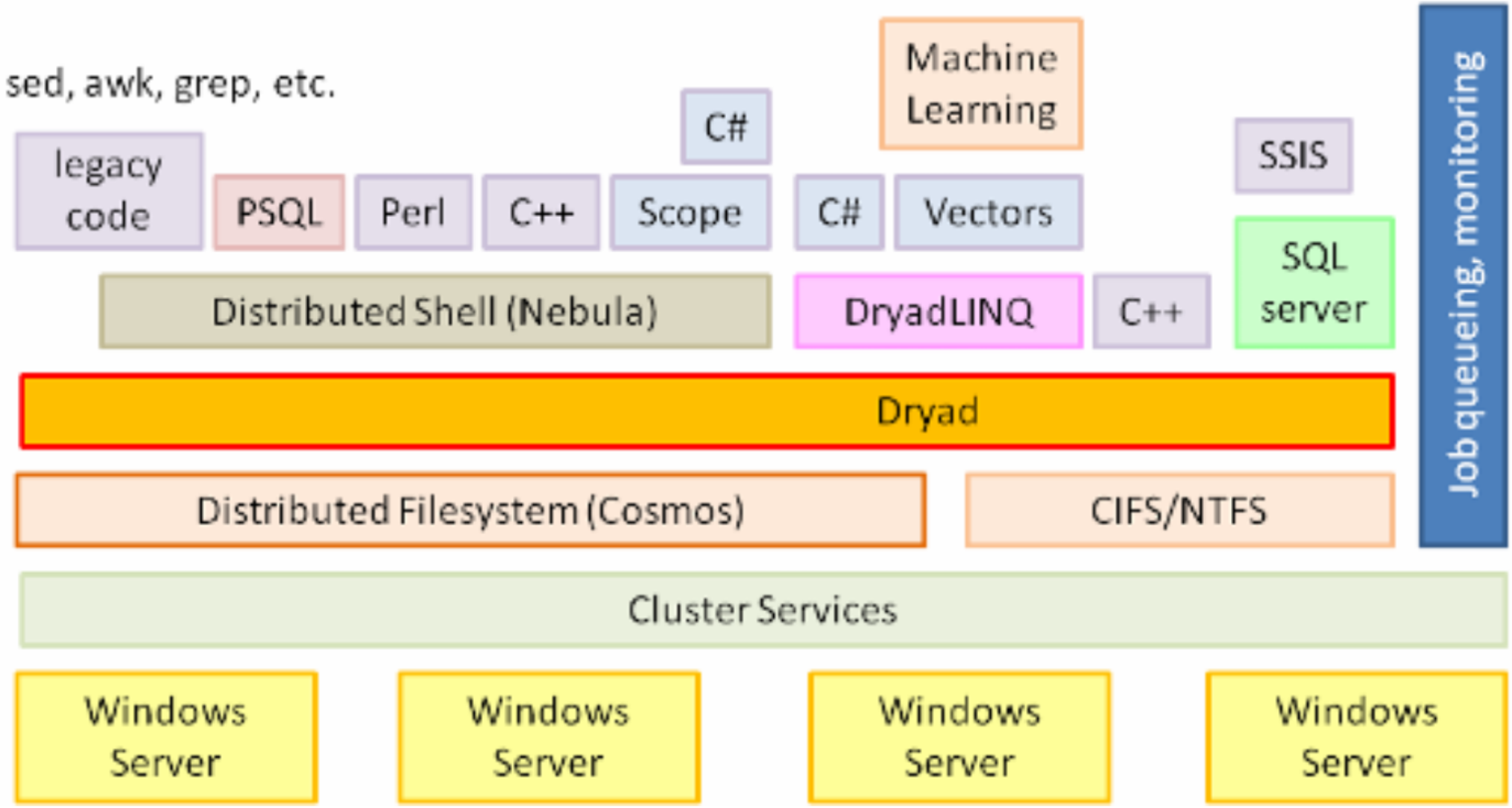
```
SELECT query, COUNT(*) AS count
FROM "search.log" USING LogExtractor
GROUP BY query
HAVING count > 1000
ORDER BY count DESC;
OUTPUT TO "qcount.result";
```

```
e = EXTRACT query
    FROM "search.log"
    USING LogExtractor;

s1 = SELECT query, COUNT(*) as count
     FROM e
     GROUP BY query;

s2 = SELECT query, count
     FROM s1
     WHERE count > 1000;

s3 = SELECT query, count
     FROM s2
     ORDER BY count DESC;

OUTPUT s3 TO "qcount.result"
```

- Compiler transforms programs into DAGs of dependant tasks
- Compiler optimizes programs for maximal concurrency.




Input files
R
R
R
R
Stage
X
X
X
X
X
X
M
M
M
M
M
M
Channels
Vertices
(processes)
Output files


sed, awk, grep, etc.
legacy code
PSQL
Perl
C++
C#
Scope
Machine Learning
C#
Vectors
SSIS
SQL server
Job queueing, monitoring
Distributed Shell (Nebula)
DryadLINQ
C++
Dryad
Distributed Filesystem (Cosmos)
CIFS/NTFS
Cluster Services
Windows Server
Windows Server
Windows Server
Windows Server

```
SELECT query, COUNT(*) AS count
FROM "search.log" USING LogExtractor
GROUP BY query
HAVING count > 1000
ORDER BY count DESC;
OUTPUT TO "qcount.result";
```

```
e = EXTRACT query
    FROM “search.log"
    USING LogExtractor;

s1 = SELECT query, COUNT(*) as count
     FROM e
     GROUP BY query;

s2 = SELECT query, count
     FROM s1
     WHERE count > 1000;

s3 = SELECT query, count
     FROM s2
     ORDER BY count DESC;

OUTPUT s3 TO “qcount.result"
```

The input data

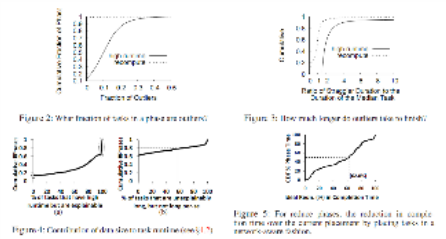
- Log files from Cosmos scheduler
- Log files from Scope compiler
- Begin and end times for each task
- Input / output data sizes for each task
- Task workflow graph

The Bing cluster

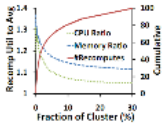
- Reading through network or from local disk storage
- Writing (always!) to local disk storage
- Data stored on the same machines that perform the computation
- By default Cosmos scheduler assigns tasks to machines where the data is available (data locality)
- Cross-rack traffic is costly
- Sum of data storage of the entire rack is smaller than the outgoing bandwidth

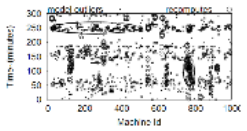
The scale of the problem

Statistics



Recomputes

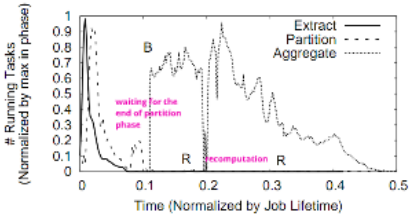




- Recomputes appear locally
- Time locality
- Machine locality

Task phases

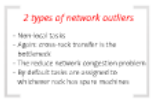
(a) Partial workflow with the number of tasks in each phase



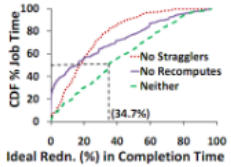
(b) Time lapse of task execution (R–Recomputes, B–Barrier).

Characteristics

- Begin of reduce phase is usually a border
- Reduce phase is not associative (no way to perform computation on functions)
- Reduce phase is not commutative
- Recomputation affects a small percentage of tasks.
- The consequences of recomputation are serious.
- 70% of data transfer is caused by reduce tasks



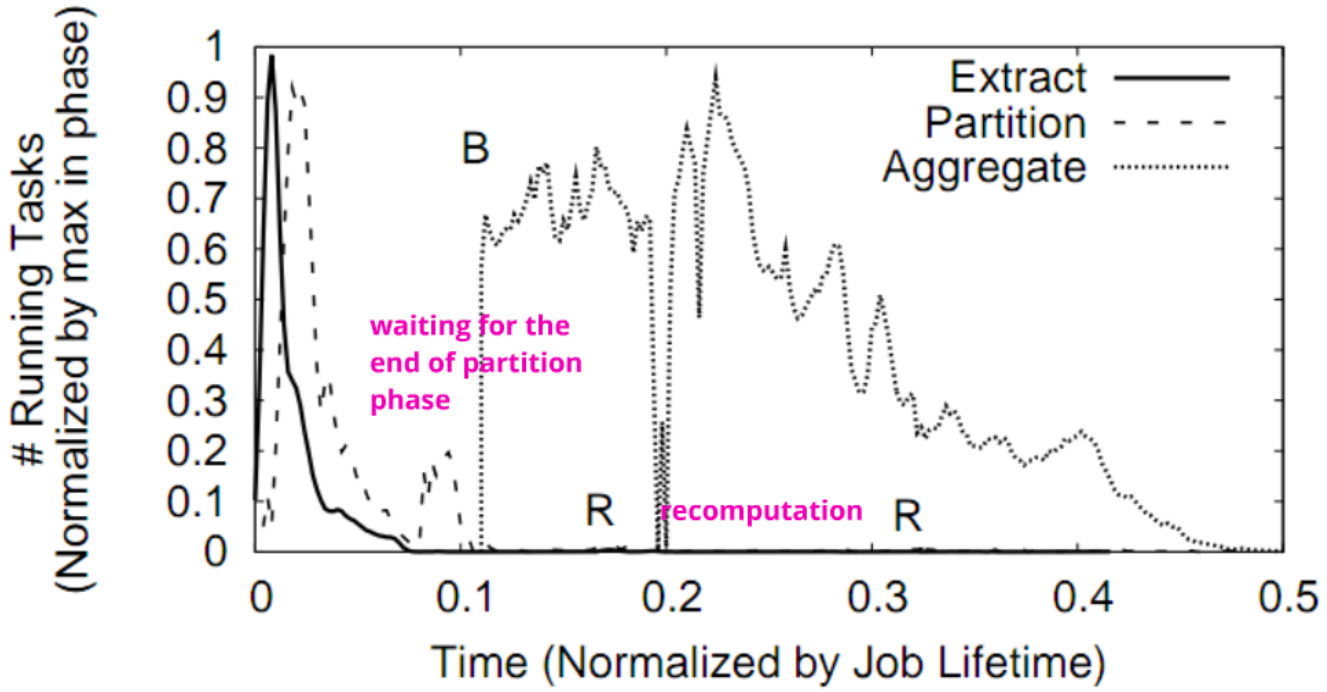
The impact of outliers



- Data from a simulator (not a real life scenario)
- The simulation took 2 weeks to compute

Task phases

(a) Partial workflow with the number of tasks in each phase



(b) Time lapse of task execution (R=Recomputes, B=Barrier).

Characteristics

- Begin of reduce phase is usually a border
- Reduce phase is not associative (no way to perform computation on functions)
- Reduce phase is not commutative
- Recomputation affects a small percentage of tasks.
- The consequences of recomputation are serious.
- 70% of data transfer is caused by reduce tasks

2 types of network outliers

- Non-local tasks
- Again: cross-rack transfer is the bottleneck
- The reduce network congestion problem
- By default tasks are assigned to whichever rack has spare machines

2 types of network outliers

- Non-local tasks
- Again: cross-rack transfer is the bottleneck
- The reduce network congestion problem
- By default tasks are assigned to whichever rack has spare machines

Statistics

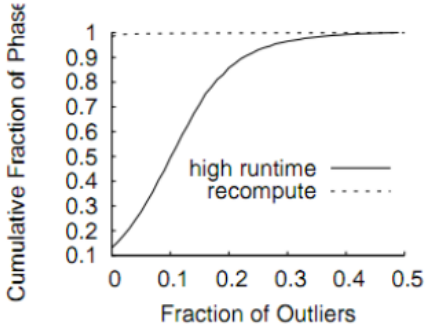


Figure 2: What fraction of tasks in a phase are outliers?

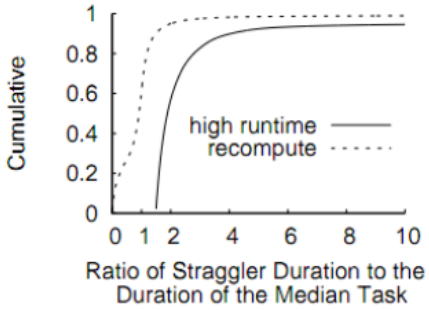


Figure 3: How much longer do outliers take to finish?

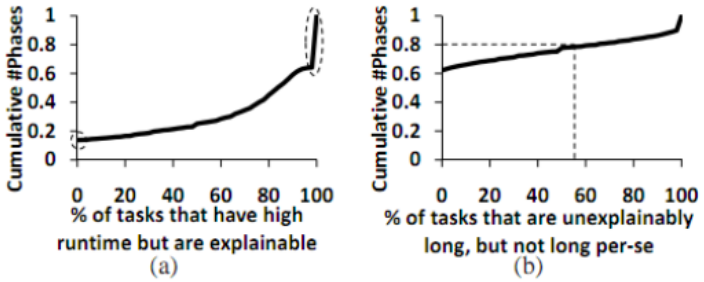


Figure 4: Contribution of data size to task runtime (see §4.2)

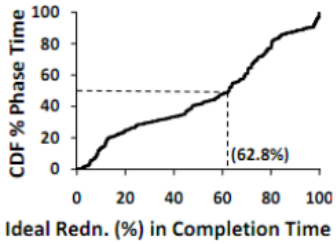
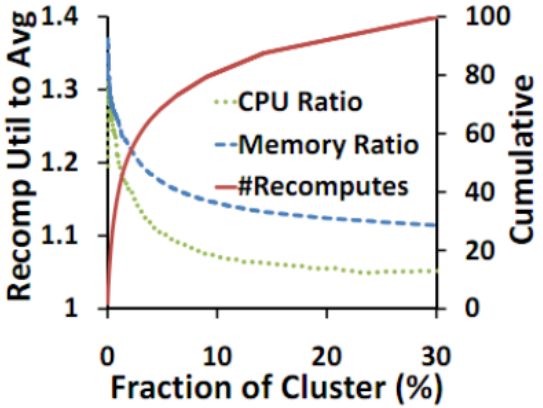
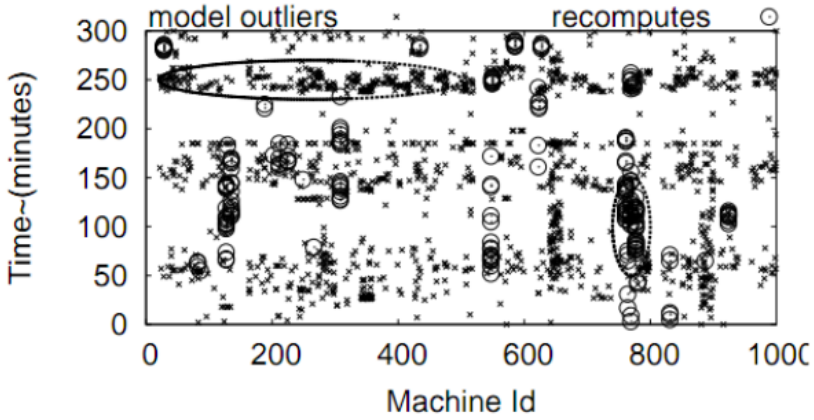


Figure 5: For reduce phases, the reduction in completion time over the current placement by placing tasks in a network-aware fashion.

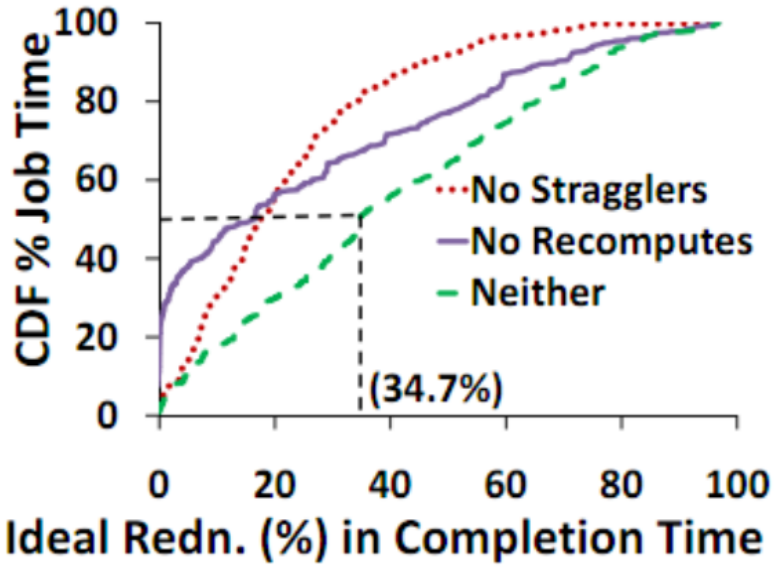
Recomputes





- Recomputes appear locally
- Time locality
- Machine locality

The impact of outliers



- Data from a simulator (not a real life scenario)
- The simulation took 2 weeks to compute

The implementation

The concept

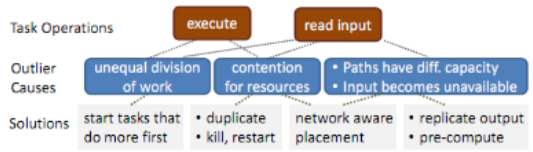


Figure 9: The Outlier Problem: Causes and Solutions

Network aware task placement

- Attempting to overcome the reduce network congestion problem
- Analyzing every assignment of reduce tasks to racks
- Required transfer bandwidths are known ahead of the time
- Restarts are network aware as well

Resource-aware restart

1: let Δ = period of progress reports
2: let c = number of copies of a task
3: periodically, for each running task, kill all but the fastest c copies after Δ time has passed since begin
4: **while** slots are available **do**
5: if tasks are waiting for slots **then**
6: kill, restart task if $t_{new} > E(t_{new}) + \Delta$, stop at γ restarts
7: duplicate if $P(t_{new} > t_{s,c=\frac{c}{c+1}}) > \delta$
8: start the waiting task that has the largest data to read
9: **else**
10: duplicate iff $\mathbb{E}(t_{new} - t_{rem}) > \rho\Delta$
11: **end if**
12: **end while**

- **Typically Delta equals 10 sec**
- **Reschedule requires additional time**
- **Don't duplicate recently duplicated tasks**
- **No response, assume no progress**

Pseudocode 1: Algorithm for Resource-aware restarts (simplified).

- Consider kill & restart or duplication
- Keep number of running copies constant
- Duplicate if the expected amount of resources needed to complete the task would decrease
- Kill tasks when duplicates exceed budget cap

Task result duplication

- Replicate results if they are small
- Replicate when the result is costly to recompute
- Replicate the tasks that take the longest to recompute
- Budget cap on the amount of replicated data
- Replication trottled through a system of tokens

The concept

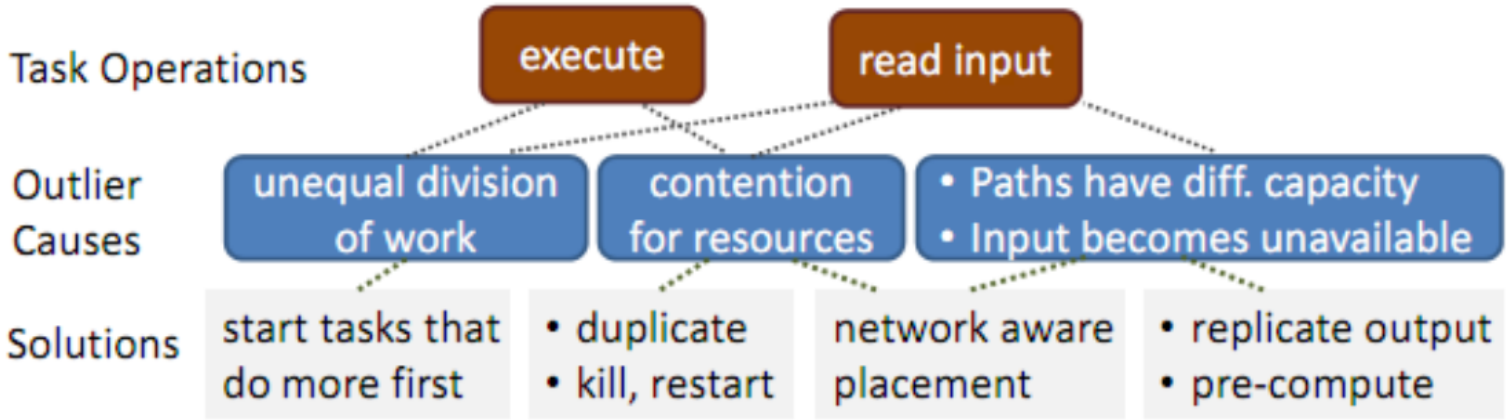


Figure 9: The Outlier Problem: Causes and Solutions

Resource-aware restart

1: let Δ = period of progress reports
2: let c = number of copies of a task
3: periodically, for each running task, kill all but the fastest α copies after Δ time has passed since begin
4: **while** slots are available **do**
5: **if** tasks are waiting for slots **then**
6: kill, restart task if $t_{rem} > \mathbb{E}(t_{new}) + \Delta$, stop at γ restarts
7: duplicate if $\mathbb{P}(t_{rem} > t_{new}\frac{c+1}{c}) > \delta$
8: start the waiting task that has the largest data to read
9: **else**
10: duplicate iff $\mathbb{E}(t_{new} - t_{rem}) > \rho\Delta$
11: **end if**
12: **end while**

- **Typically Delta equals 10 sec**
- **Reschedule requires additional time**
- **Don't duplicate recently duplicated tasks**
- **No response, assume no progress**

▷ all tasks have begun

Pseudocode 1: Algorithm for Resource-aware restarts (simplified).

- Consider kill & restart or duplication
- Keep number of running copies constant
- Duplicate if the expected amount of resources needed to complete the task would decrease
- Kill tasks when duplicates exceed budget cap

Network aware task placement

- Attempting to overcome the reduce network congestion problem
- Analyzing every assignment of reduce tasks to racks
- Required transfer bandwidths are known ahead of the time
- Restarts are network aware as well

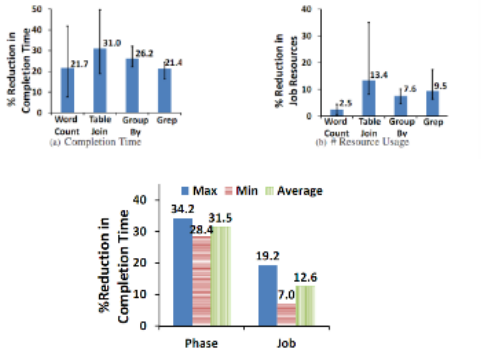
Task result duplication

- Replicate results if they are small
- Replicate when the result is costly to recompute
- Replicate the tasks that take the longest to recompute
- Budget cap on the amount of replicated data
- Replication trottled through a system of tokens

Evaluation

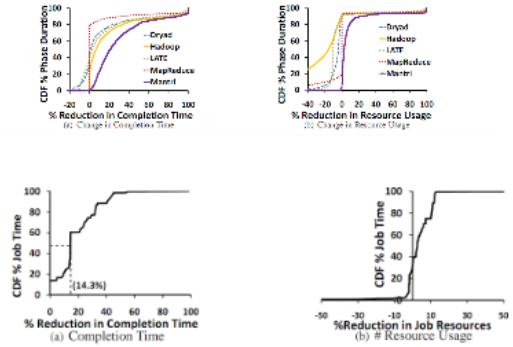
Data clusters

- Preproduction cluster & production cluster
- Baseline: an unmodified Cosmos scheduler
- Comparison to Hadoop, Dryad, MapReduce, LATE & modified LATE



Results

- Typical job speed-up: 25-28%
- 42% at 75-th percentile
- 3.1x better improvement as compared to the next best

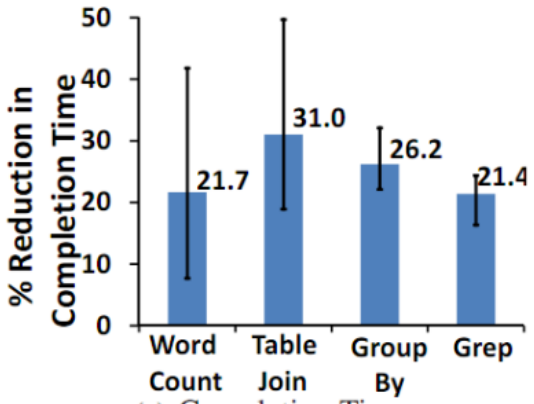


Data clusters

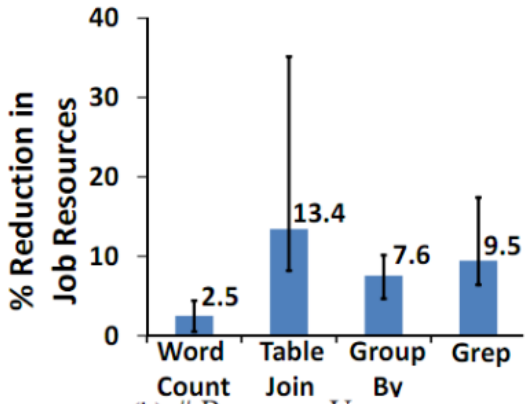
- Preproduction cluster & production cluster
- Baseline: an unmodified Cosmos scheduler
- Comparison to Hadoop, Dryad, MapReduce, LATE & modified LATE

Results

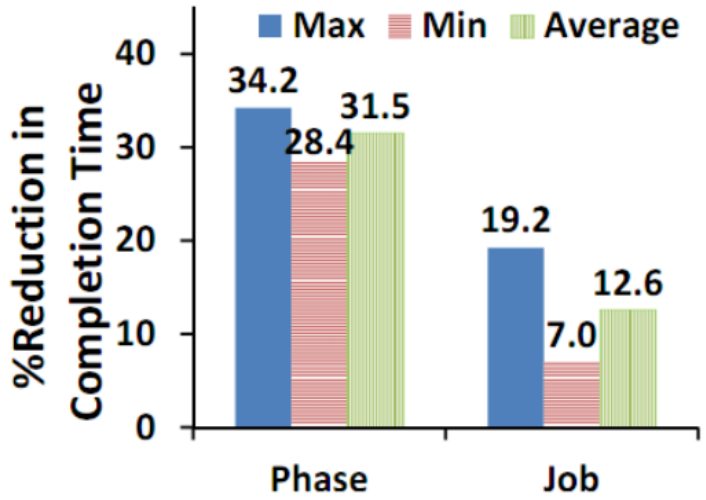
- Typical job speed-up: 25-28%
- 42% at 75-th percentile
- 3.1x better improvement as compared to the next best

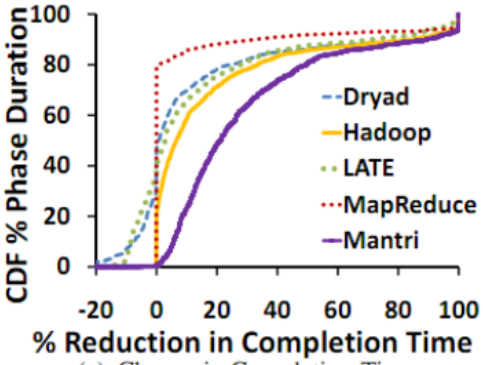


(a) Completion Time

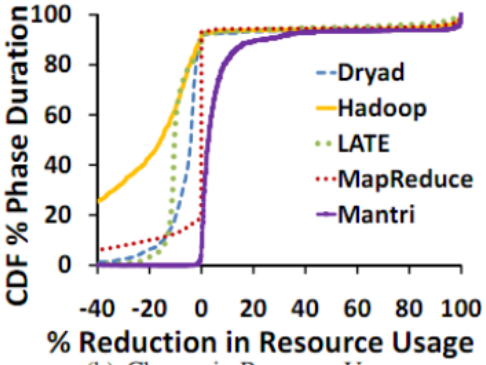


(b) # Resource Usage

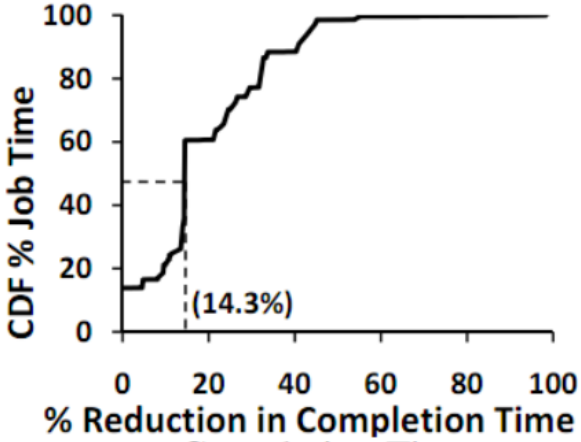




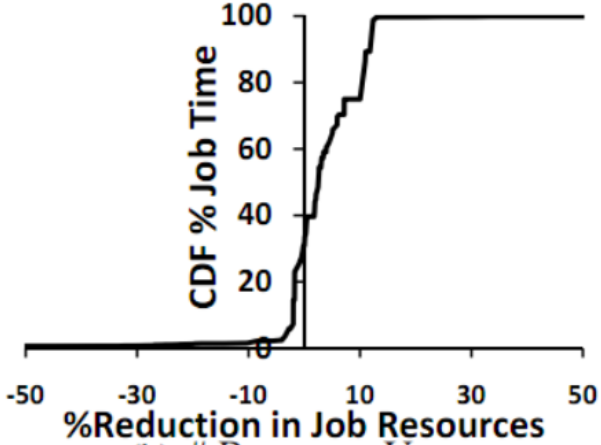
(a) Change in Completion Time



(b) Change in Resource Usage



(a) Completion Time



(b) # Resource Usage

Reining in the Outliers in Map-Reduce Clusters using Mantri