DON’T CRY OVER SPILLED RECORDS
Memory elasticity of data-parallel applications and its application to cluster scheduling

Călin Iorgulescu (EPFL), Florin Dinu (EPFL), Aunn Raza (NUST Pakistan), Wajih Ul Hassan (UIUC), Willy Zwaenepoel (EPFL)
Cluster operators care about resource utilization

✓ Best bang for your buck!
✓ Maximize performance of data-parallel applications

• Idea: Efficient resource utilization through under-provisioning
Cluster memory is under-utilized!

Avg. mem. utilization: **78%**
Cluster memory is under-utilized!

Avg. mem. utilization: 78%

Leverage this idle memory!
Impact of memory constraining applications

• Conventional wisdom: do not touch memory!

• Risks:
  – crashes
  – severe performance degradation (e.g., thrashing)

Can we safely, deterministically, and with modest impact constrain memory?
Context: batch jobs and their memory usage
Context: batch jobs and their memory usage
Context: batch jobs and their memory usage

Input

Ideal memory
Context: batch jobs and their memory usage

Input

Ideal memory
Context: batch jobs and their memory usage
Context: batch jobs and their memory usage

![Diagram showing batch jobs and memory usage](image)
Context: batch jobs and their memory usage

- Input
- Ideal memory
- Disk I/O
- CPU
- Ideal duration
- Read
- Compute
- Write
Context: batch jobs and their memory usage

- **Input**
- **Ideal memory**
- **Ideal duration**
- **Read**
- **Compute**
- **Write**
- **Non-ideal memory**

Legend:
- **Disk I/O**
- **CPU**
Context: batch jobs and their memory usage

- Input
- Ideal memory
- Non-ideal memory
- Ideal duration
- Disk I/O
- CPU
- Read
- Compute
- Write
Context: batch jobs and their memory usage

- **Input**
- **Ideal memory**
- **Ideal duration**
- **Read**
- **Compute**
- **Write**

- **Non-ideal memory**

- **Disk I/O**
- **CPU**
Context: batch jobs and their memory usage

Input

Ideal memory

Ideal duration

Read  Compute  Write

Non-ideal memory

Disk I/O  CPU
Context: batch jobs and their memory usage

Input

Ideal memory

Ideal duration

Read  Compute  Write

Non-ideal memory

Disk I/O

CPU
Context: batch jobs and their memory usage

Ideal memory

Ideal duration

Input

Read

Compute

Write

Disk I/O

CPU

Non-ideal memory

Read

Spill

Read

Spill

Read

Compute

Merge

Write

7/12/17

École Polytechnique Fédérale de Lausanne
Context: batch jobs and their memory usage

Input

Ideal memory

Ideal duration

Penalty

Disk I/O

CPU

Ideal memory

Non-ideal memory

Read

Compute

Write

Spill

Spill

Read

Compute

Merge

Write

Read

Write

Disk I/O

CPU

Penalty
Batch jobs handle memory under-provisioning → intermediate results spilled to disk
MEMORY ELASTICITY
What is Memory Elasticity?

Non-ideal memory

Disk I/O
CPU
What is Memory Elasticity?

✓ **Safely** constrain memory
What is Memory Elasticity?

✓ Safely constrain memory
✓ Moderate penalties
What is Memory Elasticity?

✓ **Safely** constrain memory
✓ **Moderate** penalties
✓ **Highly** predictable
What is Memory Elasticity?

✓ Safely constrain memory
✓ Highly predictable
✓ Moderate penalties
✓ Ubiquitous for most data-parallel apps
An empirical study of Memory Elasticity

• Analysis of 18 jobs across 8 different applications

• Constrain tasks’ memory $\rightarrow$ measure **penalty**

• Bypass disk buffer cache (to not mask impact of spilling to disk)
Questions about Memory Elasticity
Questions about Memory Elasticity

- Are the penalties large?
Questions about Memory Elasticity

• Are the penalties large?

• Do penalties vary considerably w.r.t given memory?
Questions about Memory Elasticity

- Are the penalties large?
- Do penalties vary considerably w.r.t given memory?
- Does the additional I/O cause disk contention?
Questions about Memory Elasticity

- Are the penalties large?
- Do penalties vary considerably w.r.t given memory?
- Does the additional I/O cause disk contention?

NOT SO MUCH!
Elasticity of Hadoop workloads: Reducers

Normalized task execution time

- Nutch Indexing
- Pagerank 1
- Pagerank 2
- TPC-DS Q40
- TPC-DS Q7
- Conn. Comp. 1
- Word Count
- Terasort
- Recomm. 1
- Recomm. 2

10% 50% 90%
Elasticity of Hadoop workloads: **Reducers**

**Surprise!** The median penalty is **<1.6x**!
Why are the penalties so modest?

- Data buffer – most of app. memory
  - Memory pressure absorbed by data buffer

- Sequential disk access
  - Spilling records to disk is faster than OS paging

- Logarithmic external merge algorithms
  - Merge steps required << disk spills
Elasticity of Hadoop workloads: Reducers

Normalized task execution time

- Nutch Indexing
- Pagerank 1
- Pagerank 2
- TPC-DS Q40
- TPC-DS Q7
- Conn. Comp. 1
- Word Count
- Terasort
- Recomm. 1
- Recomm. 2

10% 50% 90%

3.3x
Elasticity of Hadoop workloads: **Reducers**

**Surprise!** For 10%, 50%, and 90% memory, penalties vary by at most **0.25x!**
Why do penalties vary so little w/ memory?

• Static spilling threshold $\rightarrow$ comparable data spilling for 90% and 10% of memory
Why do penalties vary so little w/ memory?

• Static spilling threshold → comparable data spilling for 90% and 10% of memory
Why do penalties vary so little w/ memory?

- Static spilling threshold → comparable data spilling for 90% and 10% of memory
Why do penalties vary so little w/ memory?

- Static spilling threshold → comparable data spilling for 90% and 10% of memory

Input | 2.1 GB
Buffer | 2 GB
Spills | 1 x 2 GB

Input | 2.1 GB
Why do penalties vary so little with memory?

- Static spilling threshold $\rightarrow$ comparable data spilling for 90% and 10% of memory

### Diagram

#### Case 1
- **Input**: 2.1 GB
- **Buffer**: 2 GB
- **Spills**: 1 x 2 GB

#### Case 2
- **Input**: 2.1 GB
- **Buffer**: 200 MB

Why do penalties vary so little w/ memory?

- Static spilling threshold $\rightarrow$ comparable data spilling for 90% and 10% of memory
Why do penalties vary so little w/ memory?

- Static spilling threshold → comparable data spilling for 90% and 10% of memory

**Total data spilled in both cases: 2GB**
Elasticity of Hadoop workloads: Mappers

<table>
<thead>
<tr>
<th>Normalized task execution time</th>
<th>WordCount w/ combiner</th>
<th>Conn. Comp. 1</th>
<th>Pagerank 2</th>
<th>Pagerank 1</th>
<th>WordCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>1.05</td>
<td>1.1</td>
<td>1.15</td>
<td>1.15</td>
<td>1.15</td>
</tr>
<tr>
<td>50%</td>
<td>1.10</td>
<td>1.2</td>
<td>1.25</td>
<td>1.25</td>
<td>1.25</td>
</tr>
<tr>
<td>90%</td>
<td>1.15</td>
<td>1.3</td>
<td>1.35</td>
<td>1.35</td>
<td>1.35</td>
</tr>
</tbody>
</table>
Elasticity of Hadoop workloads: Mappers

Penalties are even lower! Median penalty is ~1.2x!
Elasticity of Hadoop workloads: **Mappers**

Penalties for 10%, 50%, and 90% memory vary by at most **0.05x**!

Penalties are even lower! Median penalty is **~1.2x**!
Elasticity of Spark and Tez workloads

- Spark Terasort
- Spark WordCount
- Tez SortMergeJoin
- Tez WordCount

Normalized task execution time

- 10%
- 50%
- 90%
Elasticity of Spark and Tez workloads

Median penalty is \(~1.75\times!\)
Elasticity of Spark and Tez workloads

Median penalty is ~1.75x!

Penalties for 10%, 50%, and 90% memory vary by at most 0.3x!
Does the additional I/O cause disk contention?

- Measure slowdown of elastic tasks on same machine spilling to the same disk
Does the additional I/O cause disk contention?

- Measure slowdown of elastic tasks on same machine spilling to the same disk

Cluster operators usually provision $\geq 1$ disk / 2 cores* $\implies <10\%$ slowdown!

* Facebook (2010) and Nutanix
Does the additional I/O cause disk contention?

- Measure slowdown of elastic tasks on same machine spilling to the same disk

Cluster operators usually provision ≥ 1 disk / 2 cores*  \[\rightarrow\] <10% slowdown!

Degradation <25% for up to 1 disk / 5 cores!

* Facebook (2010) and Nutanix
Summary: Memory Elasticity of real workloads

✓ Modest performance penalties (<1.6x median)

✓ Similar penalties for 10% and 90% of ideal memory

✓ Disk contention negligible for existing clusters’ setup (<10%)
MODELING MEMORY ELASTICITY
Modeling Memory Elasticity

- How does penalty vary for a task?
Modeling Memory Elasticity

• How does penalty vary for a task?

Penalty = f (disk speed, input size, framework configuration)
Modeling Memory Elasticity

- Penalties vary little between percentages $\rightarrow$ **step** model
Modeling Memory Elasticity

- Penalties vary little between percentages → step model
Modeling Memory Elasticity

- Requires 2 profiling runs $\rightarrow$ infers all other points
Modeling Memory Elasticity

- Requires 2 profiling runs → infers all other points
Modeling Memory Elasticity

• Requires 2 profiling runs → infers all other points

The step model represents our baseline.
More complex models are possible.
Accuracy of our reducer model

Real duration normalized to model

- Pagerank 1
- Pagerank 2
- TPC-DS Q7
- TPC-DS Q40
- Conn. Comp. 1
- Terasort
- Recomm. 1
- Recomm. 2
- Word Count
- Nutch
- Spark Terasort
- Tez SMJ
- Tez Word Count

Avg. offset
Most reducers are off at most by +/- 5%!
Summary: Modeling memory elasticity

✓ Step model is adequate (more complex models available)

✓ Only 2 profiling points → full model

✓ Models are very robust (+/- 5% error for most reducers)
LEVERAGING MEMORY ELASTICITY IN CLUSTER SCHEDULING
How can a scheduler reason about Memory Elasticity?

Trade-off between

↓ task queueing time  
↑ task execution time

Elastic allocation
YARN-ME: Decision process

- Make an elastic allocation *iff* it does not exceed the expected job completion time.
YARN-ME: Design and components

Components

- **Timeline generator** – computes expected JCTs
- **Profiler** – generates the model metadata
Memory utilization analysis for YARN-ME

- 50 node cluster
- Homogeneous trace: 5x Pagerank jobs
Memory utilization analysis for YARN-ME

- 50 node cluster
- Homogeneous trace: 5x Pagerank jobs

Memory utilization increased to 95%
What gains can YARN-ME achieve for heterogeneous workloads?

- 50 node cluster
- Mixed trace – 14 jobs
  - 3x PageRank
  - 3x Recommender
  - 8x Wordcount
What gains can YARN-ME achieve for heterogeneous workloads?

- 50 node cluster
- Mixed trace – 14 jobs
  - 3x PageRank
  - 3x Recommender
  - 8x Wordcount

Up to 65% improvement for JCT and makespan
What gains can YARN-ME achieve for **homogeneous** workloads?

- 50 node cluster
- Pagerank
  - concurrent runs
What gains can YARN-ME achieve for **homogeneous** workloads?

- Up to **40% improvement** for JCT and makespan
  - 50 node cluster
  - Pagerank
    - **concurrent runs**
Trace-driven simulation of YARN-ME

- We built DSS (the Discrete Scheduler Simulator)
  - and it is open-source!

<table>
<thead>
<tr>
<th>Trace parameter sweep</th>
<th>Robustness analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>• &gt; 8,000 traces</td>
<td>• &gt; 20,000 traces</td>
</tr>
<tr>
<td>• up to 3,000 nodes</td>
<td>• YARN-ME is robust to model mis-estimations</td>
</tr>
<tr>
<td>• results comparable to real workloads</td>
<td></td>
</tr>
</tbody>
</table>
Related work

• Efficient packing $\rightarrow$ better resource utilization
  – Tetris [SIGCOMM ‘14], GRAPHENE [OSDI ‘16]

• Collocate batch-jobs with latency-critical services
  – Heracles [ISCA ‘15]

• Resource over-committing
  – Apollo [OSDI ‘14], Borg [EuroSys ‘15]

• Suspend tasks under memory pressure
  – ITask [SOSP ‘15]
Conclusion: Don’t cry over spilled records!

✓ Memory Elasticity → highly **predictable**, **low penalty**

✓ Memory Elasticity in scheduling → trade task **queueing-time** for **running-time**

✓ YARN-ME → up to 60% improvement in average JCT

✓ DSS code available: [https://github.com/epfl-labos/dss](https://github.com/epfl-labos/dss)