Packing Tasks with Dependencies

Robert Grandl, Srikanth Kandula,
Sriram Rao, Aditya Akella, Janardhan Kulkarni
The Cluster Scheduling Problem

Jobs

Tasks

Goal: match tasks to resources
The Cluster Scheduling Problem

Goal: match tasks to resources
The Cluster Scheduling Problem

Jobs

Goal: match tasks to resources to achieve
• High cluster utilization
• Fast job completion
• Guarantees (deadlines, fair shares)
The Cluster Scheduling Problem

**Goal:** match tasks to resources to achieve
- High cluster utilization
- Fast job completion
- Guarantees (deadlines, fair shares)

**Constraints**
- Scale ⇒ fast twitch
The Cluster Scheduling Problem

**Jobs**

- Tasks

**Goal:** match tasks to resources to achieve
  - High cluster utilization
  - Fast job completion
  - Guarantees (deadlines, fair shares)

**Constraints**
- Scale ⇒ fast twitch

- Large and high-value deployments
  - E.g., Spark, Yarn*, Mesos*, Cosmos
The Cluster Scheduling Problem

Jobs

Goal: match tasks to resources to achieve
• High cluster utilization
• Fast job completion
• Guarantees (deadlines, fair shares)

Constraints
• Scale ⇒ fast twitch

• Large and high-value deployments
  • E.g., Spark, Yarn*, Mesos*, Cosmos

• Today, schedulers are simple and (as we show) performance can improve a lot
Jobs have heterogeneous DAGs

User queries → Query optimizer → Job DAG
(Dryad, Spark-SQL, Hive,...)
Jobs have **heterogeneous DAGs**

User queries → Query optimizer → Job DAG

(Dryad, Spark-SQL, Hive,...)
Jobs have heterogeneous DAGs

User queries $\rightarrow$ Query optimizer $\rightarrow$ Job DAG
(Dryad, Spark-SQL, Hive,...)
Jobs have heterogeneous DAGs

User queries → Query optimizer → Job DAG
(Dryad, Spark-SQL, Hive,...)

• DAGs have deep and complex structures
• Task durations range from <1s to >100s
• Tasks use different amounts of resources
Challenges in scheduling heterogeneous DAGs

\[ (1 + 5\varepsilon)T \{0.6, 0.3\} \]

\[ \varepsilon T \{0.2, 0.6\} \]

\[ \varepsilon T \{0.85, \varepsilon\} \]

\[ \varepsilon T \{0.2, 0.5\} \]

\[ (1 + 2\varepsilon)T \{0.2, 0.1\} \]
Challenges in scheduling heterogeneous DAGs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Execution Order</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>$t_1 \rightarrow t_3 \rightarrow {t_4, t_0} \rightarrow {t_0, t_2, t_5}$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

Diagram:

- 0: $(1 + 5\epsilon)T \{0.6, 0.3\}$
- 1: $\epsilon T \{0.85, \epsilon\}$
- 2: $T \{0.2, 0.6\}$
- 3: $\epsilon T \{0.84, \epsilon\}$
- 4: $\epsilon T \{0.2, 0.5\}$
- 5: $(1 + 2\epsilon)T \{0.2, 0.1\}$
Challenges in scheduling heterogeneous DAGs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Execution Order</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>$t_1 \rightarrow t_3 \rightarrow {t_4, t_0} \rightarrow {t_0, t_2, t_5}$</td>
<td>$T$</td>
</tr>
<tr>
<td>CPSched</td>
<td>$t_0 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_1 \rightarrow t_2$</td>
<td>$3T$</td>
</tr>
</tbody>
</table>

Diagram:
- $0: \{0.6, 0.3\}$
- $1: \{0.85, \varepsilon\}$
- $2: \{0.2, 0.6\}$
- $3: \{0.84, \varepsilon\}$
- $4: \{0.2, 0.5\}$
- $5: \{0.2, 0.1\}$

Equations:
- $(1 + 5\varepsilon)T$
- $0.6, 0.3$
- $0.85, \varepsilon$
- $0.2, 0.6$
- $0.84, \varepsilon$
- $0.2, 0.5$
- $(1 + 2\varepsilon)T$
- $0.2, 0.1$
Challenges in scheduling heterogeneous DAGs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Execution Order</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>( t_1 \rightarrow t_3 \rightarrow { t_4, t_0 } \rightarrow { t_0, t_2, t_5 } )</td>
<td>( T )</td>
</tr>
<tr>
<td>CPSched</td>
<td>( t_0 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_1 \rightarrow t_2 )</td>
<td>( 3T )</td>
</tr>
</tbody>
</table>

CPSched cannot overlap tasks with complementary demands
Challenges in scheduling heterogeneous DAGs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Execution Order</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>$t_1 \rightarrow t_3 \rightarrow {t_4, t_0} \rightarrow {t_0, t_2, t_5}$</td>
<td>$T$</td>
</tr>
<tr>
<td>CPSched</td>
<td>$t_0 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_1 \rightarrow t_2$</td>
<td>$3T$</td>
</tr>
<tr>
<td>Packers$^1$</td>
<td>$t_0 \rightarrow t_1 \rightarrow t_3 \rightarrow t_2 \rightarrow t_4 \rightarrow t_5$</td>
<td>$3T$</td>
</tr>
</tbody>
</table>

CPSched cannot overlap tasks with complementary demands

Challenges in scheduling heterogeneous DAGs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Execution Order</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>$t_1 \rightarrow t_3 \rightarrow {t_4, t_0} \rightarrow {t_0, t_2, t_5}$</td>
<td>$T$</td>
</tr>
<tr>
<td>CPSched</td>
<td>$t_0 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_1 \rightarrow t_2$</td>
<td>$3T$</td>
</tr>
<tr>
<td>Packers$^1$</td>
<td>$t_0 \rightarrow t_1 \rightarrow t_3 \rightarrow t_2 \rightarrow t_4 \rightarrow t_5$</td>
<td>$3T$</td>
</tr>
</tbody>
</table>

CPSched cannot overlap tasks with complementary demands
Packers do not handle dependencies

Challenges in scheduling heterogeneous DAGs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Execution Order</th>
<th>Time</th>
<th>Worst-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>$t_1 \rightarrow t_3 \rightarrow {t_4, t_0} \rightarrow {t_0, t_2, t_5}$</td>
<td>$T$</td>
<td>–</td>
</tr>
<tr>
<td>CPSched</td>
<td>$t_0 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_1 \rightarrow t_2$</td>
<td>$3T$</td>
<td>$O(n) \times \text{OPT}$</td>
</tr>
<tr>
<td>Packers$^1$</td>
<td>$t_0 \rightarrow t_1 \rightarrow t_3 \rightarrow t_2 \rightarrow t_4 \rightarrow t_5$</td>
<td>$3T$</td>
<td>$O(d) \times \text{OPT}$</td>
</tr>
</tbody>
</table>

CPSched cannot overlap tasks with complementary demands
Packers do not handle dependencies

Challenges in scheduling heterogeneous DAGs ...
Challenges in scheduling heterogeneous DAGs ...

1. Simple heuristics lead to poor schedules
Challenges in scheduling heterogeneous DAGs ...

1. Simple heuristics lead to poor schedules
2. Production DAGs are roughly 50% slower than lower bounds
Challenges in scheduling heterogeneous DAGs ...

1. Simple heuristics lead to poor schedules
2. Production DAGs are roughly 50% slower than lower bounds
3. Simple variants of “Packing dependent tasks” are NP-hard problems
Challenges in scheduling heterogeneous DAGs ...

1. Simple heuristics lead to poor schedules
2. Production DAGs are roughly 50% slower than lower bounds
3. Simple variants of “Packing dependent tasks” are NP-hard problems
4. Prior analytical solutions miss some practical concerns
Challenges in scheduling heterogeneous DAGs ...

1. Simple heuristics lead to poor schedules
2. Production DAGs are roughly 50% slower than lower bounds
3. Simple variants of “Packing dependent tasks” are NP-hard problems
4. Prior analytical solutions miss some practical concerns
   • Multiple resources
   • Complex dependencies
   • Machine-level fragmentation
   • Scale; Online; ...
Given an annotated DAG and available resources, compute a good schedule + practical model
Main ideas for one DAG

Existing schedulers:

A task is schedulable *after* all its parents have *finished*
Main ideas for one DAG

Existing schedulers:

A task is schedulable after all its parents have finished.
Main ideas for one DAG

Existing schedulers:
A task is schedulable after all its parents have finished

Graphene:
Identifies troublesome tasks and places them first
Main ideas for one DAG

Existing schedulers:
A task is schedulable after all its parents have finished

Graphene:
Identifies troublesome tasks and places them first
Main ideas for one DAG

Existing schedulers:
A task is schedulable *after* all its parents have *finished*.

Graphene:
Identifies *troublesome tasks* and places them *first*
Place other tasks around trouble.
Does placing troublesome tasks first help?

Revisit the example

0: \((1 + 5\varepsilon)T\) 
{0.6, 0.3}

1: \(\varepsilon T\) 
{0.85, \varepsilon}

2: \(T\) 
{0.2, 0.6}

3: \(\varepsilon T\) 
{0.84, \varepsilon}

4: \(\varepsilon T\) 
{0.2, 0.5}

5: \((1 + 2\varepsilon)T\) 
{0.2, 0.1}
Does placing troublesome tasks first help?

Revisit the example

If troublesome tasks $\supseteq$ long-running tasks, Graphene $\equiv$ OPT
Does placing troublesome tasks first help?

Revisit the example

If troublesome tasks $\supseteq$ long-running tasks, Graphene $\equiv$ OPT
Does placing troublesome tasks first help?

Revisit the example

If troublesome tasks $\supseteq$ long-running tasks, Graphene $\equiv$ OPT
How to choose troublesome tasks $T$?

$\text{frag} \geq f$
How to choose troublesome tasks $T$?

Optimal choice is intractable (recall: NP-Hard)

$$\text{frag} \geq f$$
How to choose troublesome tasks $T$?

Optimal choice is **intractable** (recall: NP-Hard)

\[
\text{frag} \geq f
\]
How to choose troublesome tasks $T$?

Optimal choice is intractable (recall: NP-Hard)

Graphene:

$\text{BuildSchedule}(T)$

Stage fragmentation score

Task duration
How to choose troublesome tasks $T$?

**ff**

Optimal choice is intractable (recall: NP-Hard)

Graphene: \[ \text{BuildSchedule}(T) \]

Vary $l, f$

Pick the most compact schedule

Stage fragmentation score

Task duration
How to choose troublesome tasks $T$?

Optimal choice is intractable (recall: NP-Hard)

Graphene:

$\text{BuildSchedule}(T)$

Pick the most compact schedule

Extensions

1) Explore different choices of $T$ in parallel
2) Recurse
3) Memoize ...
Schedule dead-ends
Schedule dead-ends
1) Since some parents and children of $\bigcirc$ are already placed with $T$, may not be able to place $\bigcirc$.
1) Since some parents and children of \( \bullet \) are already placed with \( T \), may not be able to place \( T \). 

\[ T \leftarrow \text{TransitiveClosure}(T) \]
Schedule dead-ends

1) Since some parents and children of are already placed with \( T \), may not be able to place

\[ T \leftarrow \text{TransitiveClosure}(T) \]
1) Since some parents and children of $\bigcirc$ are already placed with $T$, may not be able to place $\bigcirc$

$T \leftarrow \text{TransitiveClosure}(T)$
Schedule dead-ends

1) Since some parents and children of \( \bullet \) are already placed with \( T \), may not be able to place \( \bullet \)

\[ T \leftarrow \text{TransitiveClosure}(T) \]

2) When placing tasks in \( P \), have to go \textit{backwards} (place task after all children are placed)
Schedule dead-ends

1) Since some parents and children of \( \bullet \) are already placed with \( T \), may not be able to place \( \bullet \)

\[
T \leftarrow \text{TransitiveClosure}(T)
\]

2) When placing tasks in \( \text{TransitiveClosure}(T) \), \( P \), have to go backwards (place task after all children are placed)
Schedule dead-ends

1) Since some parents and children of \( \bullet \) are already placed with \( T \), may not be able to place \( \bullet \)

\[ T \leftarrow \text{TransitiveClosure}(T) \]

2) When placing tasks in \( P \), have to go backwards (place task after all children are placed)
Schedule dead-ends

1) Since some parents and children of \(\bullet\) are already placed with \(T\), may not be able to place \(\bullet\).

\[ T \leftarrow \text{TransitiveClosure}(T) \]

2) When placing tasks in \(P\), have to go *backwards* (place task after all children are placed)

Which of these orders are legit?

\[
\begin{align*}
T_{fb} & P_{fb} S_f C_f \\
T_{fb} & P C S_f \\
T_{fb} & S P C_f
\end{align*}
\]
Schedule dead-ends

1) Since some parents and children of T are already placed with T, may not be able to place T

\[ T \leftarrow \text{TransitiveClosure}(T) \]

2) When placing tasks in P, have to go backwards (place task after all children are placed)
Schedule dead-ends

1) Since some parents and children of a task are already placed with $T$, may not be able to place another task.

$$T \leftarrow \text{TransitiveClosure}(T)$$

2) When placing tasks in $P$, have to go backwards (place task after all children are placed)
Main ideas for one DAG

1. Identify *troublesome tasks* and place them *first*
2. Systematically place tasks to avoid dead-ends
Computed offline schedule for One DAG → Production clusters have Multiple DAGs
Convert offline schedule to priority order on tasks
Convert offline schedule to priority order on tasks

0: 

1: \( (1 + 5\varepsilon)T \) \{0.6, 0.3\}

2: \( \varepsilon T \) \{0.2, 0.6\}

3: \( \varepsilon T \) \{0.84, \varepsilon\}

4: \( \varepsilon T \) \{0.2, 0.5\}

5: \( (1 + 2\varepsilon)T \) \{0.2, 0.1\}
Convert offline schedule to priority order on tasks

0: $(1 + 5\varepsilon)T \{0.6, 0.3\}$

1: $\varepsilon T \{0.85, \varepsilon\}$

2: $T \{0.2, 0.6\}$

3: $\varepsilon T \{0.84, \varepsilon\}$

4: $\varepsilon T \{0.2, 0.5\}$

5: $(1 + 2\varepsilon)T \{0.2, 0.1\}$

Job1:
1: 2
2: 0
3: 1
4: 3
5: 4
Convert offline schedule to priority order on tasks

\[ (1 + 5\varepsilon)T, \{0.6, 0.3\} \]

\[ T, \{0.2, 0.6\} \]

\[ (1 + 2\varepsilon)T, \{0.2, 0.1\} \]

\[ \varepsilon T, \{0.85, \varepsilon\} \]

\[ \varepsilon T, \{0.84, \varepsilon\} \]

\[ \varepsilon T, \{0.2, 0.5\} \]

Job1

2
0
1
3
4
5

\[ t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\} \]
Convert offline schedule to priority order on tasks

\[(1 + 5\varepsilon)T\{0.6, 0.3\}\]

\[\varepsilon T\{0.85, \varepsilon\}\]

\[T\{0.2, 0.6\}\]

\[(1 + 2\varepsilon)T\{0.2, 0.1\}\]

\[\varepsilon T\{0.84, \varepsilon\}\]

\[\varepsilon T\{0.2, 0.5\}\]

Offline

Runtime

\[t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\}\]
Convert offline schedule to priority order on tasks

\[
(1 + 5\varepsilon)T \{0.6, 0.3\} \\
\varepsilon T \{0.85, \varepsilon\} \\
T \{0.2, 0.6\} \\
(1 + 2\varepsilon)T \{0.2, 0.1\}
\]

\[
1: \quad (1 + 5\varepsilon)T \{0.6, 0.3\} \\
2: \quad T \{0.2, 0.6\} \\
3: \quad \varepsilon T \{0.84, \varepsilon\} \\
4: \quad T \{0.2, 0.5\} \\
5: \quad (1 + 2\varepsilon)T \{0.2, 0.1\}
\]

Offline

Runtime

\[
t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\}
\]
Convert offline schedule to priority order on tasks

\[
(1 + 5\varepsilon)T \{0.6, 0.3\} \\
\varepsilon T \{0.85, \varepsilon\} \\
T \{0.2, 0.6\} \\
\varepsilon T \{0.84, \varepsilon\} \\
(1 + 2\varepsilon)T \{0.2, 0.1\}
\]

\[
\begin{align*}
t_1 & \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\}
\end{align*}
\]
Convert offline schedule to priority order on tasks

\[
\begin{align*}
(1 + 5\epsilon)T & \{0.6, 0.3\} \\
T & \{0.2, 0.6\} \\
(1 + 2\epsilon)T & \{0.2, 0.1\}
\end{align*}
\]

Offline

\[
\begin{align*}
\epsilon T & \{0.84, \epsilon\} \\
\epsilon T & \{0.84, \epsilon\} \\
\epsilon T & \{0.84, \epsilon\}
\end{align*}
\]

Runtime

Job1

\[
\begin{align*}
2 \\
0 \\
5
\end{align*}
\]

Job2

\[
\begin{align*}
2' \\
0' \\
5'
\end{align*}
\]

[t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\}]

\[
\begin{align*}
1' & \rightarrow 3' \rightarrow 4 \\
1 & \rightarrow 3 \rightarrow 4
\end{align*}
\]
Convert offline schedule to priority order on tasks

\[(1 + 5\varepsilon)T \{0.6, 0.3\}\]
\[T \{0.2, 0.6\}\]
\[(1 + 2\varepsilon)T \{0.2, 0.1\}\]
\[\varepsilon T \{0.84, \varepsilon\}\]
\[\varepsilon T \{0.2, 0.5\}\]

Job1:
- Task 2
- Task 0
- Task 5

Job2:
- Task 2'
- Task 0'
- Task 5'

Sequence:
- \(t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\}\)
Convert offline schedule to priority order on tasks

\[ (1 + 5\varepsilon)T \quad \{0.6, 0.3\} \]

\[ \varepsilon T \quad \{0.85, \varepsilon\} \]

\[ T \quad \{0.2, 0.6\} \]

\[ (1 + 2\varepsilon)T \quad \{0.2, 0.1\} \]

\[ t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\} \]
Convert offline schedule to priority order on tasks

\[
\begin{align*}
0: & \quad (1 + 5\varepsilon)T \\
& \quad \{0.6, 0.3\} \\
1: & \quad \varepsilon T \\
& \quad \{0.85, \varepsilon\} \\
2: & \quad T \\
& \quad \{0.2, 0.6\} \\
3: & \quad \varepsilon T \\
& \quad \{0.84, \varepsilon\} \\
4: & \quad (1 + 2\varepsilon)T \\
& \quad \{0.2, 0.5\} \\
5: & \quad (1 + 2\varepsilon)T \\
& \quad \{0.2, 0.1\}
\end{align*}
\]

Time

Offline

2

Runtime

\[
t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\}
\]

Job1

\[
\begin{array}{c}
2 \\
0 \\
5
\end{array}
\]

Job2

\[
\begin{array}{c}
2' \\
0' \\
5'
\end{array}
\]

Time

2T
Convert offline schedule to priority order on tasks

$\varepsilon T$ \{0.85, $\varepsilon$\} 
\(\{0.6, 0.3\}\) 
\(\{0.2, 0.6\}\) 
\((1 + 2 \varepsilon) T\) \{0.2, 0.1\} 

$\varepsilon T$ \{0.84, $\varepsilon$\} 

$\varepsilon T$ \{0.2, 0.5\} 
\((1 + 2 \varepsilon) T\) \{0.2, 0.6\} 

Offline 
Runtime 

Job1 
1 \ 3 \ 4 \ 5 

Job2 
1' \ 3' \ 4' \ 5' 

$t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\}$ 

CPSched 
0 \ 0' \ 3' \ 4 \ 5' 

2T 
3T 
4T
Main ideas for multiple DAGs

1) Convert offline schedule to priority order on tasks
Main ideas for multiple DAGs

1) Convert offline schedule to priority order on tasks

2) Online, enforce schedule priority along with heuristics for
   (a) Multi-resource packing
   (b) “SRPT” to lower average job completion time
   (c) Bounded amount of unfairness
   (d) Overbooking ...
Main ideas for multiple DAGs

1) Convert **offline schedule** to **priority order** on tasks

2) **Online**, enforce schedule priority along with **heuristics** for
   (a) Multi-resource packing
   (b) “SRPT” to lower average job completion time
   (c) Bounded amount of unfairness
   (d) Overbooking ...
Main ideas for multiple DAGs

1) Convert offline schedule to priority order on tasks

2) Online, enforce schedule priority along with heuristics for
   (a) Multi-resource packing
   (b) “SRPT” to lower average job completion time
   (c) Bounded amount of unfairness
   (d) Overbooking ...

Trade-offs:

- Packing Efficiency: may delay job completion, ...
- Schedule priority: best for one DAG; overall?
- Job Completion Time: may lose packing efficiency, ...
- Fairness: may counteract all others
Main ideas for multiple DAGs

1) Convert offline schedule to priority order on tasks

2) Online, enforce schedule priority along with heuristics for
   (a) Multi-resource packing
   (b) “SRPT” to lower average job completion time
   (c) Bounded amount of unfairness
   (d) Overbooking ...

Trade-offs:

We show that:
{best “perf” | bounded unfairness} ~ best “perf”
Graphene summary & implementation

1) Offline, schedule each DAG by placing troublesome tasks first
2) Online, enforce priority over tasks along with other heuristics
Graphene summary & implementation

1) Offline, schedule each DAG by placing troublesome tasks first
2) Online, enforce priority over tasks along with other heuristics
Graphene summary & implementation

1) Offline, schedule each DAG by placing troublesome tasks first
2) Online, enforce priority over tasks along with other heuristics
Graphene summary & implementation

1) Offline, schedule each DAG by placing troublesome tasks first
2) Online, enforce priority over tasks along with other heuristics
Implementation details

- DAG annotations
- Bundling: improve schedule quality w/o killing scheduling latency
- Co-existence with (many) other scheduler features
Evaluation

• **Prototype**
  • 200 server multi-core cluster
  • TPC-DS, TPC-H, ..., GridMix to replay traces
  • Jobs arrive online

• **Simulations**
  • Traces from production Microsoft Cosmos and Yarn clusters
  • Compare with many alternatives
Results - 1

[20K DAGs from Cosmos]

- Lower bound: 
  - [20K DAGs from Cosmos]
Results - 1

[20K DAGs from Cosmos]

Fraction of DAGs

Random → Packing → Packing + Deps. → Lower bound

Reduction in job duration [%]

0 20 40 60 80 100

-20 0 20 40 60 80 100

1.5X

2X
Results - 2

[200 jobs from TPC-DS, 200 server cluster]
Results - 2

[200 jobs from TPC-DS, 200 server cluster]
Scheduling heterogeneous DAGs well requires an online solution that handles multiple resources and dependencies.
Scheduling heterogeneous DAGs well requires an online solution that handles multiple resources and dependencies

Graphene
- Offline, construct per-DAG schedule by placing troublesome tasks first
- Online, enforce schedule priority along with other heuristics
- New lower bound shows nearly optimal for half of the DAGs
Scheduling heterogeneous DAGs well requires an online solution that handles multiple resources and dependencies.

Graphene

- Offline, construct per-DAG schedule by placing troublesome tasks first
- Online, enforce schedule priority along with other heuristics
- New lower bound shows nearly optimal for half of the DAGs

Experiments show gains in job completion time, makespan, ...
Scheduling heterogeneous DAGs well requires an online solution that handles multiple resources and dependencies.

Graphene
- Offline, construct per-DAG schedule by placing troublesome tasks first
- Online, enforce schedule priority along with other heuristics
- New lower bound shows nearly optimal for half of the DAGs

Experiments show gains in job completion time, makespan, ...

Graphene generalizes to DAGs in other settings
When scheduler works with erroneous task profiles

-0.75, -0.50, -0.50, -0.25, +0.25, +0.50, +0.50, +0.75

CDF over DAGs

Fractional change in JCT

Amount of error

[-0.75, -0.50]  
[-0.50, -0.25]  
[+0.25, +0.50]  
[+0.50, +0.75]  

CPSched
When scheduler works with erroneous task profiles

Fractional change in JCT

Amount of error

CDF over DAGs

[−0.75,−0.50]
[−0.50,−0.25]
[+0.25,+0.50]
[+0.50,+0.75]
When scheduler works with erroneous task profiles

CDF over DAGs

Fractional change in JCT

Amount of error

Fractional change in JCT

[−0.75,−0.50]

[−0.50,−0.25]

[+0.25,+0.50]

[+0.50,+0.75]
When scheduler works with erroneous task profiles

- Fractional change in JCT

Amount of error:
- $[-0.75, -0.50]$
- $[-0.50, -0.25]$
- $[+0.25, +0.50]$
- $[+0.50, +0.75]$
DAG annotations

G uses per-stage **average duration and demands** of \{cpu, mem, net. disk\}

1) Almost all frameworks have user’s **annotate** cpu and mem
2) Recurring jobs\(^1\) have predictable profiles (correcting for input size)
3) Ongoing work on building **profiles for ad-hoc jobs**
   • Sample and project\(^2\)
   • Program analysis\(^3\)

[1] RoPE, NSDI’12; ...
[2] Perforator, SOCC’16; ...
[3] SPEED, POPL’09; ...
Using Graphene to schedule other DAGs

(a) Distributed Build Systems: (b) Request-response workflows: Compilation time Query latency
Characterizing DAGs in Cosmos clusters
Characterizing DAGs in Cosmos clusters – 2

Fraction of Tasks with Value < x

In degree
Out degree

Value
Runtime of production DAGs

Fraction of DAGs with Gap < x

Gap = 1 - (Measure / DAG runtime)

- Gap from NewLB
- Gap from TWork
- Gap from CPLength
Job completion times on different workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>50^{th} percentile</th>
<th>75^{th} percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>G</td>
<td>T+C</td>
</tr>
<tr>
<td>TPC-DS</td>
<td>27.8</td>
<td>4.1</td>
</tr>
<tr>
<td>TPC-H</td>
<td>30.5</td>
<td>3.8</td>
</tr>
<tr>
<td>BigBench</td>
<td>25.0</td>
<td>6.4</td>
</tr>
<tr>
<td>E-Hive</td>
<td>19.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

G stands for **GRAPHENE**. T+C and T+T denote Tez + CP and Tez + Tetris respectively (see §7.1). The improvements are relative to Tez.
<table>
<thead>
<tr>
<th>Workload</th>
<th>Makespan</th>
<th>Tez+CP</th>
<th>Tez+Tetris</th>
<th>GRAPHENE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-DS</td>
<td>+2.1%</td>
<td></td>
<td>+8.2%</td>
<td>+30.9%</td>
</tr>
<tr>
<td>TPC-H</td>
<td>+4.3%</td>
<td></td>
<td>+9.6%</td>
<td>+27.5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Workload</th>
<th>Scheme</th>
<th>2Q vs. 1Q Perf. Gap</th>
<th>Jain’s fairness index</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-DS</td>
<td>Tez</td>
<td>−13%</td>
<td>0.82, 0.86, 0.88</td>
</tr>
<tr>
<td></td>
<td>Tez+DRF</td>
<td>−12%</td>
<td>0.85, 0.89, 0.90</td>
</tr>
<tr>
<td></td>
<td>Tez+Tetris</td>
<td>−10%</td>
<td>0.77, 0.81, 0.92</td>
</tr>
<tr>
<td></td>
<td>GRAPHENE</td>
<td>+2%</td>
<td>0.72, 0.83, 0.89</td>
</tr>
</tbody>
</table>
Comparison with other alternatives

<table>
<thead>
<tr>
<th></th>
<th>25&lt;sup&gt;th&lt;/sup&gt;</th>
<th>50&lt;sup&gt;th&lt;/sup&gt;</th>
<th>75&lt;sup&gt;th&lt;/sup&gt;</th>
<th>90&lt;sup&gt;th&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphene</strong></td>
<td>7</td>
<td>25</td>
<td>57</td>
<td>74</td>
</tr>
<tr>
<td>Random</td>
<td>−2</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Crit.Path</td>
<td>−2</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>Tetris</td>
<td>0</td>
<td>7</td>
<td>29</td>
<td>42</td>
</tr>
<tr>
<td>Strip Part.</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>27</td>
</tr>
<tr>
<td>Coffman-Graham.</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>−2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
**Online Pseudocode**

```
Func: FindAppropriateTasksForMachine:
Input: \( m \): vector of available resources at machine; \( \mathcal{J} \): set of jobs with task details \( \text{\{duration, demands, priScore\}} \); deficit: counters for fairness;
Parameters: \( \kappa \): unfairness bound; \( rp \): remote penalty
Output: \( S \), the set of tasks to be allocated on the machine
\( S \leftarrow \emptyset \)
while true do
  foreach task \( t \) do
    \{pScore, oScore\} \leftarrow \{0, 0\}
    rPenalty \leftarrow t \text{ is locality sensitive? } rp \text{ : 1}
    if \( t_{\text{demands}} \leq m \text{ // fits? then} \)
      pScore \leftarrow (m \cdot t_{\text{demands}}) \cdot rPenalty \text{ // dot product}
    else
      // what-if analysis: "overbook or wait".
      \forall \text{tasks} \( t' \) affected by \( t \) running at \( m \), let before\( (t') \),
      after\( (t') \) be expected completion times before and
      after placing \( t \) at \( m \)
      benefit = nextSchedOpp + t_{\text{duration}} \cdot \text{after}(t)
      cost = \text{\sum aff. tasks } t' (after(t') - before(t'))
      if benefit > cost then oScore \leftarrow benefit - cost;
    end
    job \( j \geq t \), srpt \leftarrow \text{\sum pending } u_{j} \cdot t_{\text{duration}} \cdot |u_{\text{demands}}|
    perfScore \leftarrow t_{\text{priScore}} \cdot \{pScore, oScore\} - \eta \cdot srpt
  \end{forall}
  \( t^\text{best} \leftarrow \arg \max \{\text{perfScore}, |t|\} \text{// task with highest perf score}
  \text{if } t^\text{best} = \emptyset \text{ then break} \text{// no new task can be scheduled on this machine;}
  \forall \text{jobgroup with highest deficit counter}
  \text{if deficit}_{g'} \geq \kappa \text{ then } t^\text{best} \leftarrow \arg \max \{\text{perfScore}, |t| \in g'\}
  \text{S} \leftarrow \text{S} \cup t^\text{best}
  m \leftarrow [m - t^\text{best}_{\text{demands}}]_{+}
  \text{deficit}_{g} \leftarrow \text{deficit}_{g} +\text{factor}(t^\text{best}_{\text{demands}}) \cdot \{\text{fairShare}_{g} - 1 \text{ if } t \in \text{jobgroup } g\}
  \text{fairShare}_{g}
  \text{otherwise}
```