A Testing Engine for High-Performance and Cost-Effective Workflow Execution in the Cloud

V. K. Pallipuram\textsuperscript{1}, T. Estrada\textsuperscript{2}, M. Taufer\textsuperscript{3}

University of the Pacific\textsuperscript{1}
University of New Mexico\textsuperscript{2}
University of Delaware\textsuperscript{3}
What if ...

... we want to run our (HPC) applications on the Cloud but:

• The workflow looks like a DAG with inter-task dependencies
• Our knowledge of the host platform’s physical characteristics is only imperfect
• Our knowledge of the task execution times is even more imperfect
What if ...

... we want to run our (HPC) applications on the Cloud but:

• The workflow looks like a DAG with inter-task dependencies
• Our knowledge of the host platform’s physical characteristics is only imperfect
• Our knowledge of the task execution times is even more imperfect

Challenges: What scheduling policy? What resource instances? How many?

LU-decomposition DAG
Our contributions

• Static task schedules can compete with dynamic task schedules on the Cloud
• Performance-cost trade-off is an effective metric to select number of resource instances and instances’ vCPU
• We demonstrate our claims by:
  ▪ Working with a diverse dataset of DAG-like workflows
  ▪ Emulating static and dynamic task schedules on a testing engine fed by real Cloud traces
  ▪ Assessing the performance-cost trade-off of optimal instance configurations
We would like to ... 

globally 

maximize the number of eligible tasks on the DAG at every step of the computation despite the imperfect knowledge.

Given the DAG $G$ and the schedule $\Sigma$:

$E_{\Sigma}(t) \triangleq$ number of jobs that are eligible at step $t$ of $\Sigma$'s execution of DAG $G$

$(\forall t \in [1,N_G]) \ E_{\Sigma}(t) = \max \Sigma$ a schedule for $G \ \{E_{\Sigma}(t)\}$
IC-optimal schedule

- Internet-based computing (IC)-optimal schedule globally maximize the number of eligible tasks on the DAG at every step of the computation despite the imperfect knowledge.

Given the DAG $G$ and the schedule $\Sigma$:

\[
E_{\Sigma}(t) \triangleq \text{number of jobs that are eligible at step } t \text{ of } \Sigma \text{’s execution of DAG } G \\
(\forall t \in [1, N_G]) E_{\Sigma^*}(t) = \max \{ E_{\Sigma}(t) \}
\]

- Outperform a variety of common scheduling heuristics on simulated executions of DAGs

- Not applicable to several (too many) DAGs [Malewicz, Rosenberg, Yurkewich, 2005]
We can relax the constraints and ...

*maximize the average number of eligible tasks at every step on DAG of the computation*

Given the DAG $\hat{G}$ and the schedule $\Sigma$:

- $E_\Sigma(t) \triangleq$ number of jobs eligible at step $t$ of $\Sigma$’s
- $\text{AREA}(\Sigma) \triangleq E_\Sigma(0) + E_\Sigma(1) + \ldots + E_\Sigma(N_{\hat{G}})$
- $E(\Sigma) \triangleq \text{AREA}(\Sigma) \div N_{\hat{G}}$ Average number of nodes that are ELIGIBLE when $\Sigma$ executes $\hat{G}$
Area-maximizing (AM) schedule

- Area-maximizing (AM) schedule \(\rightarrow\) maximize the average number of eligible tasks at every step on DAG of the computation

Given the DAG \(\mathcal{G}\) and the schedule \(\Sigma\):

\[
E_{\Sigma}(t) \triangleq \text{number of jobs eligible at step } t \text{ of } \Sigma' \text{'s}
\]

\[
\text{AREA}(\Sigma) \triangleq E_{\Sigma}(0) + E_{\Sigma}(1) + \ldots + E_{\Sigma}(N_{\mathcal{G}})
\]

\[
E(\Sigma) \triangleq \frac{\text{AREA}(\Sigma)}{N_{\mathcal{G}}} \quad \text{Average number of nodes that are ELIGIBLE when } \Sigma \text{ executes } \mathcal{G}
\]

- Theoretically area-maximal schedules exist for every DAGs
- But efficient generators of AM schedules are known only for well-structured DAGs [Cordasco et al. IPDPS 2010]
The good DAGs

- Expansion-reduction computation DAGs
- Convolutional DAGs
- And combinations

[Cordasco et al. JPDC 2010]
Compose Serial/Parallel (SP) DAGs

\[ G' \rightarrow G'' \]

Series composition

\[ t = t'' \]
\[ t' = s'' \]
\[ s = s' \]

Parallel composition

\[ t = t' = t'' \]
\[ s = s' = s'' \]

[Cardasco et al. JPDC 2010]
Multiphase static heuristics

- DAG schedules with large AREA are computational intractability
- Handle computational intractability with polynomial-time static heuristics:
  - Local-Optimal (L-OPT) heuristic
  - Area-oriented (AO)-scheduling heuristic
L-OPT heuristic

- Organize eligible tasks in a list structure that is (partially) ordered by tasks’ yields (with ties broken randomly)

- Yield of task \( v \) at step \( t \) \( \uparrow \) number of non-eligible tasks that would be rendered eligible if \( v \) were executed at this step

- Select maximal yield task at each step (local decision)

- Yield of a task changes the yields of several tasks

\[
greedy = \sum (a; b; f; c; d; e; g; h; i; j; k; l; m; n)
\]
Area-oriented (AO)-scheduling heuristic

- Find G’s transitive skeleton $G^T$ by removing all shortcut arcs
  - $G^T$ is a smallest sub-DAG of G that shares G’s task-set and transitive closure
- Convert $G^T$ to an SP-DAG $\sigma(G^T)$ by invoke an SP-ization algorithm to convert $G^T$ to $(G^T)$
  - Maintain inter-task dependencies, retain the degree of parallelism, add extra tasks, and operates within time $O(n^2)$
- Find an AREA-max schedule $\sum^T$ for $\sigma(G^T)$
  - Use [Hall et al. 2007]
- Remove the extra tasks added by the SP-ization algorithm
Example: LU-decomposition DAG

\[ \text{ao} = \sum (a; b; c; f; d; e; i; g; j; h; l; k; m; n) \]

- \( \Sigma = \{a, b, c, f, d, e, i, g, j, h, l, k, m, n\} \)
- AO-schedule
- Filtering
- SP-ization
- \( \sigma(G') \)
- \( \Sigma' = \{a, b, c, d, e, o, i, j, g, h, p, l, k, m, n\} \)
- an A-M schedule for \( \sigma(G') \)
Testing engine

- **AO**
  - Scheduling Policy
    - Task Feeder
    - Task priority list
    - DAG workflow
  - Resource Selection Heuristic
    - Instance candidates <t,n>
    - DAG workflow
  - Discrete Event Simulator
    - CPU Performance, S3 Performance, etc.
    - Real EC2 Traces
- **L-OPT**

Results (wall-clock time, cost)
Resource selection heuristic

- Identify suitable number of instances \((n)\) for each instance type available in the Cloud containing \(t\) virtual CPUs (vCPUs)
- Select optimal instance candidates before execution and exclusively based on DAG structure
  - Levelize DAG, i.e., define all tasks at a given level as concurrent tasks
  - Compute histogram of DAG level-widths and histogram median
  - Select of instance candidates that minimize number of instances and maximize instance size to match median
  - Return selected instance candidates \((n, t)\) in \(Z\) to the discrete-event simulator for workflow execution
EC2 traces: CPU performance

• Model performance variability of the vCPUs in instances

Weibull distributions for the Xeon and Opteron UBench CPU performance

Data courtesy of Prof. Jens Dittrich and Joerg Schad from Saarland University
EC2 Traces: S3 storage upload and download

- Model data transfers to/from the AWS S3

Weibull distributions for the AWS S3 storage upload and download bandwidth performance
LEGO-DAG dataset

- Build 160 LEGO-DAGs by composing a series of *basic building blocks* (BBBs) randomly
  - Numbers of tasks in the range of 1000–4000 per DAGs

Sample BBBs generated with random sizes and structures:

A sample LEGO-DAG built from six BBBs (All arcs go upward)
Classifying DAG-like workflows

- Depending length: length of inherently sequential critical path
- Degree of concurrency: weighted median of concurrent tasks
Sampling across DAG dataset

![Graph showing the relationship between degree of concurrency and number of tasks per group across different dependency lengths. The graph is color-coded with 16 DAG groups, ranging from 1000 to 4000 tasks per group. The x-axis represents short to long dependency length, while the y-axis represents low to high degree of concurrency.]
Sampling across DAG dataset

![Diagram showing the relationship between dependency length and degree of concurrency with different numbers of tasks per group. The diagram includes 16 DAG groups with varying numbers of tasks per group, ranging from 1000 to 4000. The x-axis represents dependency length, with short, medium, and long categories, and the y-axis represents the degree of concurrency, ranging from low to high. The data points are color-coded based on the number of tasks per group.]
Sampling across DAG dataset

The graph shows the relationship between dependency length, degree of concurrency, and the number of tasks per group across 16 DAG groups. The x-axis represents the short, medium, and long dependency length categories, while the y-axis represents the degree of concurrency from low to high. Each point on the graph corresponds to a specific group with a different number of tasks per group, indicated by the colors ranging from 1000 to 4000.
Dynamic resource allocation
• **Loose** enforcement of priority queue

Static **L-OPT** resource allocation
• **Strict** enforcement of priority queue
**Dynamic resource allocation**

- **Loose** enforcement of priority queue

  - Loose enforcement of priority queue
  - Strict enforcement of priority queue

**Static AO resource allocation**

- **Strict** enforcement of priority queue

### Graph Details

- **X-axis**: Execution time (hours)
- **Y-axis**: Execution time (hours)

#### Graph Legend

- 2
- 4
- 8
- 16
- 32
Optimal performance-cost trade-off

• When using the AO schedule, what is the “best” type and number of resource instances?
Optimal performance-cost trade-off

• When using the AO schedule, what is the “best” type and number of resource instances?

<table>
<thead>
<tr>
<th>Instance</th>
<th>vCPUs #</th>
<th>ECUs #</th>
<th>On-demand cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3.large</td>
<td>2</td>
<td>7</td>
<td>$0.105</td>
</tr>
<tr>
<td>c3.xlarge</td>
<td>4</td>
<td>14</td>
<td>$0.21</td>
</tr>
<tr>
<td>c3.2xlarge</td>
<td>8</td>
<td>28</td>
<td>$0.42</td>
</tr>
<tr>
<td>c3.4xlarge</td>
<td>16</td>
<td>55</td>
<td>$0.84</td>
</tr>
<tr>
<td>c3.8xlarge</td>
<td>32</td>
<td>108</td>
<td>$1.68</td>
</tr>
</tbody>
</table>
Performance and cost variations

Performance variation

Cost variation

Wall-clock time (hours)

Cost ($)

type of instances (vCPUs)

2 4 8 16 32

2 4 8 16 32

type of instances (vCPUs)
Lessons learned

• We study the effectiveness of static scheduling policies for DAG-like workflows when executed on Cloud resources

• Simulated scenarios of diverse DAG-like workflows support the static AO schedule for the Cloud
  ▪ Exhibit comparable performance than a dynamic schedule while allowing a priori allocations of resources

• Our technique allows for identifying optimal number of resource instances and instance vCPUs before the execution
  ▪ Prevent over- or under-allocation of resources
  ▪ Allow reliable time and cost estimates
Acknowledgments

• Trilce Estrada, Univ. New Mexico
• Vivek K. Pallipuram, Univ. of Delaware
• Arnold L. Rosenberg, Northeastern University
• Julian Uran Martinez, Univ. of Delaware
• Gennaro Cordasco, Univ. Naples 2
• Rosario De Chiara, Poste Italiane Res. Ctr.
• Rajmohan Rajaraman, Northeastern Univ.
• Scott T. Roche, Northeastern Univ.
• NSF – CNF #1217812