CRYSTALPERF: Learning to Characterize the Performance of Dataflow Computation through Code Analysis

Huangshi Tian, Minchen Yu, Wei Wang
Dataflow computation is *prevailing* and *diverse*.
The Troubles of Data Analyst Jane

1. Jane is developing a TensorFlow model for her medical project.
2. She gets confused when the program cannot scale even with multiple GPUs.

1. Based on a real question from Stack Overflow: https://bit.ly/2kiT2dD
The Troubles of Data Analyst Jane

3. She tries to visualize the model, but the static graph does not tell much about the execution.

4. The built-in profiler gives overwhelmingly much low-level information.

Dataflow toolchain is lagging behind.
Resource Problems

Performance Debugging
examine malfunctioning resources

Performance Reasoning
what if more resources are allocated

Program Diagnosis
detect bottleneck and inefficiency

Existing Solutions

Framework-Specific
Starfish (VLDB’11)
Ernest (NSDI’16)

Requires Instrumentation
Monotasks (SOSP’17)
SnailTrail (NSDI’18)

Objective: General Performance Characterization without Instrumentation
Overview: Finding Resource-Time Relationship

**Instrumentation-Free**
Collecting runtime information *non-intrusively*.

**Framework-Independent**
Exploiting resource information in source code.

★ Resource-Time Breakdown

Source Code

ML Model

Resource Label

CPU

Disk

Network
Outline

- Background and Overview
  - Resource Classifiers
    how we infer resource usage from source code
  - Execution Profile
    how we represent job execution and debug performance
  - Resource Models
    how we model resource behavior and predict performance
- Evaluation Highlights
Resource Classifiers

Resource Vector
- each component is the probability of using certain resources
- in the form of $(p_{cpu}, p_{disk}, p_{net})$

Classification Models
- two separate models for code and documentation
- model details in the paper
Model Training

Open-Source Projects

Labeled Code

Labeled Doc

Labeled Dataset

Classification Models

extract & manually label

train models

code classifier
doc classifier
Why Classifiers Work?

Explanatory Framework: LIME

• explains why a model makes certain prediction

Procedure

• Collect a manually verified dataset.
• Train two classifiers.
• Explain their predictions.

Table 1: Top 5 words that cause the classifier predict a certain resource.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>engine, entry, stream, key, certificate</td>
</tr>
<tr>
<td>Disk</td>
<td>file, error, tar, info, name</td>
</tr>
<tr>
<td>Network</td>
<td>socket, sock, send, result, address</td>
</tr>
</tbody>
</table>
### Execution Profile

#### Key Information
- *runtime* of the operator from logs
- *resource vector* inferred by the classifiers

#### Performance Debugging
- estimate resource-time from resource vector

#### Execution Profiles

<table>
<thead>
<tr>
<th>op</th>
<th>runtime</th>
<th>res. vec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ</td>
<td>25</td>
<td>[0.6, 0.2, 0.2]</td>
</tr>
<tr>
<td>Σ</td>
<td>100</td>
<td>[0.8, 0.1, 0.1]</td>
</tr>
<tr>
<td>π</td>
<td>15</td>
<td>[0.6, 0.1, 0.3]</td>
</tr>
<tr>
<td>⊗</td>
<td>200</td>
<td>[0.7, 0.1, 0.2]</td>
</tr>
</tbody>
</table>

*bottleneck: compute*
Performance Prediction

Key Question
• How runtime would change given a resource variation?

Resource Models
• CPU: simulated rescheduling with warmup effect
• Memory: reverse-roofline model
• I/O: buffered transmission model
• detailed models in the paper

<table>
<thead>
<tr>
<th>op</th>
<th>runtime</th>
<th>res. vec.</th>
<th>pred. (2x CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ</td>
<td>25</td>
<td>[0.6,0.2,0.2]</td>
<td>-7.5</td>
</tr>
<tr>
<td>Σ</td>
<td>100</td>
<td>[0.8,0.1,0.1]</td>
<td>-40</td>
</tr>
<tr>
<td>π</td>
<td>15</td>
<td>[0.6,0.1,0.3]</td>
<td>-4.5</td>
</tr>
<tr>
<td>∇</td>
<td>200</td>
<td>[0.7,0.1,0.2]</td>
<td>-70</td>
</tr>
</tbody>
</table>

Approach implemented as a CLI tool.
## Evaluation: Prediction Accuracy

<table>
<thead>
<tr>
<th>Framework</th>
<th>Workload</th>
<th>Varied Resources</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark v2.4.3</td>
<td>two queries from TPC-H with scale factor 100</td>
<td># CPU cores / memory / network bandwidth</td>
<td>13.49% ± 8.57%</td>
</tr>
<tr>
<td>Flink v1.7.2</td>
<td>Yahoo Streaming Benchmark and Dhalion Benchmark</td>
<td>CPU share / network bandwidth</td>
<td>12.70% ± 10.11%</td>
</tr>
<tr>
<td>TensorFlow v1.13</td>
<td>ResNet and VGG on a flower image dataset</td>
<td>computing devices / # CPU cores / memory</td>
<td>14.22% ± 11.77%</td>
</tr>
</tbody>
</table>

### Graphs:

- **ResNet Workload**: Iteration time comparison between actual and predicted results across different configurations.
- **VGG Workload**: Iteration time comparison between actual and predicted results across different configurations.
Case Study: Identifying Bottleneck

- In Jane’s case, CrystalPerf identifies the bottleneck as I/O.
- We enable NCCL, an optimized library for inter-GPU I/O, and find the scalability improved.

More Results in the Paper

- Robustness against labeling inaccuracy.
- Generalizability to other frameworks.
- More real-world case studies.
Thank You for Your Attention!

For more Q&A, please contact
Huangshi Tian, htianna@cse.ust.hk