Actionable Performance Analyses

Marija Selakovic
marija.selakovic@huawei.com
London, 21.01.2020
Software Performance

One of the most important aspects of software quality

- Efficiency
- Responsiveness
- Scaling
- Throughput
- User satisfaction
Software Performance

One of the most important aspects of software quality

- Efficiency
- Responsiveness
- Scaling
- Throughput
- User satisfaction

“1 second of load lag time would cost Amazon $1.6 billion in sales per year”

- Amazon

1 http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-1.6-billion-sales
Software Performance

One of the most important aspects of software quality

- Efficiency
- Responsiveness
- Scaling
- Throughput
- User satisfaction

“A 1 second of load lag time would cost Amazon $1.6 billion in sales per year”
- Amazon

“A lag time of 400ms results in a decrease of 0.44% traffic - In real terms this amounts to 440 million abandoned sessions/month and a massive loss in advertising revenue for Google”
- Google

1 http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-1-6-billion-sales
2 http://www.cedexis.com/blog/for-google-400ms-of-increased-page-load-time-results-in-044-lost-search-sessions/
Approaches for Improving Software Performance

- CPU profiling
- Compiler optimizations
- Manual tuning

Code → Optimized code
Approaches for Improving Software Performance

- CPU profiling
- Compiler optimizations
- Manual tuning

Actionable Performance Analyses

- Code → Optimized code
- Code → Actionable suggestions
Actionable Performance Analyses

- Suggest concrete code changes
- Demonstrate the impact of applying optimizations
- Optimizations that are:
  - *exploitable* - easy to understand and apply
  - *effective* - lead to significant performance improvements
  - *recurring* - applicable across multiple projects
This Talk

- Reordering opportunities [1]
- Method inlining in Big Data system [2]
This Talk

- Reordering opportunities [1]
- Method inlining in Big Data system [2]

Inefficient Order of Evaluations

expensiveAndUnlikely() && cheapAndLikely()
Inefficient Order of Evaluations

expensiveAndUnlikely() && cheapAndLikely()
Inefficient Order of Evaluations

expensiveAndUnlikely() && cheapAndLikely()

- Analysis of all conditions in in logical expressions or switch statements
- Assessment of the computational cost
- Safe to apply and beneficial optimizations
DecisionProf: An Analysis for Optimizing Orders of Evaluations

Program + input → Profiler → Optimization candidates → Program transformation

Static preprocessing
Dynamic analysis

$P_1 \ldots P_n$ → Performance evaluation → Optimization opportunities
DecisionProf: An Analysis for Optimizing Orders of Evaluations

Program + input
↓
Profiler
↓
Optimization candidates
↓
Program transformation
↓
P_1 \ldots P_n
↓
Performance evaluation

Static preprocessing
Dynamic analysis

Optimization opportunities
Dynamic Analysis

• Collecting runtime data:
  ▪ Cost - number of branching point
  ▪ Value - true/false

• Assessing the optimal order
Dynamic Analysis

- Collecting runtime data:
  - Cost - number of branching point
  - Value - true/false
- Assessing the optimal order

```javascript
var x;

a() && b()
```
Dynamic Analysis

• Collecting runtime data:
  - Cost - number of branching point
  - Value - true/false

• Assessing the optimal order

```javascript
var x;
```

<table>
<thead>
<tr>
<th>cost</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>v</td>
</tr>
<tr>
<td>a1</td>
<td>a1</td>
</tr>
<tr>
<td>a2</td>
<td>a2</td>
</tr>
<tr>
<td>a3</td>
<td>a3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cost</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>v</td>
</tr>
<tr>
<td>b1</td>
<td>b1</td>
</tr>
<tr>
<td>b2</td>
<td>b2</td>
</tr>
<tr>
<td>b3</td>
<td>b3</td>
</tr>
</tbody>
</table>
Dynamic Analysis: Example

```javascript
_.isNumber(input) && isNaN(input)
```

<table>
<thead>
<tr>
<th>Execution</th>
<th>cost</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution 1</td>
<td>3</td>
<td>true</td>
</tr>
<tr>
<td>Execution 2</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>Execution 3</td>
<td>4</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Execution</th>
<th>cost</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>true</td>
</tr>
</tbody>
</table>
Dynamic Analysis: Example

\[ _.isNumber(input) \land isNaN(input) \]

<table>
<thead>
<tr>
<th>Execution</th>
<th>cost</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution 1</td>
<td>3</td>
<td>true</td>
</tr>
<tr>
<td>Execution 2</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>Execution 3</td>
<td>4</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cost</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
</tr>
</tbody>
</table>

Overall cost = 12
Dynamic Analysis: Example

```plaintext
_.isNumber(input) && isNaN(input)
```

<table>
<thead>
<tr>
<th>Execution</th>
<th>cost</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution 1</td>
<td>3</td>
<td>true</td>
</tr>
<tr>
<td>Execution 2</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>Execution 3</td>
<td>4</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cost</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
</tr>
</tbody>
</table>

Overall cost = 12
Dynamic Analysis: Example

$$\text{isNaN}(\text{input}) \&\& \_\_\_.\text{isNumber}(\text{input})$$

<table>
<thead>
<tr>
<th>Execution 1</th>
<th>Cost</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>Execution 2</td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>Execution 3</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>true</td>
</tr>
</tbody>
</table>
Dynamic Analysis: Example

\[ \text{isNaN(input)} && \_._\text{isNumber(input)} \]

<table>
<thead>
<tr>
<th>Execution 1</th>
<th>Cost</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>Execution 2</td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>Execution 3</td>
<td>1</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Execution 3</th>
<th>Cost</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>true</td>
</tr>
</tbody>
</table>

Overall cost = 7

1 See pull request #2496 of Underscore.js
Pruning Non-Commutative Conditions

Non-commutative conditions: change program semantics

Two approaches:

**Static**: known patterns

**Dynamic**: write to the same memory location
Performance Evaluation

- Program transformation for each optimization candidate
- Methodology by Georges et al. [1]

Original program + input

Optimized program + input

VM instances

Warm up

Measure

t-test

DecisonProf: Evaluation

• Subject programs:
  ▪ 9 JavaScript libraries and test suites
  ▪ 34 benchmarks from JetStream suite

• Results:
  ▪ 23 opportunities across libraries
  ▪ 29 opportunities across benchmarks
  ▪ Performance improvements: 2.5% - 59% (function level)
    and 2.5% - 6.5% (application level)
Examples of Reordering Opportunities

Cheerio library:

//code before
isTag (elem) && elems.indexOf(elem) === -1

//code after
elems.indexOf(elem) === -1 && isTag (elem)

tests: 26%, 34%

Gbemu benchmark:

//code before
numberType != "float32" && GameBoyWindow.opera
&& this.checkForOperaMathBug ()

//code after
GameBoyWindow.opera && numberType != "float32"
&& this.checkForOperaMathBug ()

application: 5.8%
DecisionProf: Summary

- The first profiler to detect inefficient orders of evaluations
- Simple and easy to exploit optimizations
- Suggests program refactorings
- Guaranteed improvements for given inputs
Cross-language Optimizations in Big Data Systems

- **SCOPE** - Structured Computations Optimized for Parallel Executions
- Relational (SQL) + non-relational (C#)
- SCOPE job is DAG where:
  - Vertices - processes
  - Edges - data flows

```
Stream
```
```
V1
```
```
V2
V3
```
```
V4
```
```
Output
```
Performance Problem in SCOPE

- Cross runtime interaction
- Intrinsics vs non-intrinsics

```sql
data =
    SELECT *
    FROM inputStream;
```
Performance Problem in SCOPE

- Cross runtime interaction
- **Intrinsics** vs non-intrinsics

```csharp
data =
    SELECT *
    FROM inputStream;
```

```csharp
data =
    SELECT *
    FROM inputStream
    WHERE !String.IsNullOrEmpty(A);
```

- Intrinsics (has C++ impl.)
Performance Problem in SCOPE

- Cross runtime interaction
- Intrinsics vs non-intrinsics

```sql
data =
SELECT *
FROM inputStream
WHERE A.Equals('ColumnA');
```

![Diagram](C#)

- data
  - deserialize
    - vertex
      - serialize
        - data
Method Inlining: Example

- Replacing function call with the body of a function

```c#
data =
    SELECT *
    FROM inputStream
    WHERE filter(JobID);

#CS

bool filter (string s) {
    return (!string.IsNullOrEmpty(s)
        && s.StartsWith("07"));
}

#ENDCS
```
Method Inlining: Example

- Replacing function call with the body of a function

```c++
data =
    SELECT *  
    FROM inputStream  
    WHERE !string.IsNullOrEmpty(JobID)  
    && JobID.StartsWith("07");
```
Method Inlining: Example

- Replacing function call with the body of a function

```c++
data =
    SELECT *
    FROM inputStream
    WHERE !string.IsNullOrEmpty(JobID)
    && JobID.StartsWith("07");
```

~ 4X faster
Static Analysis

Goal: C# to C++ translation for a vertex

Inlineable methods:

- Only calls to intrinsics
- No loops
- No try-catch blocks
- No new and cast operations
- No arguments passed by reference
Static Analysis: Evaluation

Optimizable vertices:

![Bar chart showing proportions of optimizable vertices across data centers.](image)
Static Analysis: Evaluation

Optimizable vertices:

![Bar chart showing proportions of optimizable vertices across data centers.](image-url)
Static Analysis: Evaluation

Optimizable vertices:

0.13% ~ 40,000 hours
Static Analysis: Evaluation

Optimizable vertices:

Case studies:

<table>
<thead>
<tr>
<th>Job</th>
<th>CPU Time</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>23%</td>
<td>30%</td>
</tr>
<tr>
<td>B</td>
<td>no change</td>
<td>no change</td>
</tr>
<tr>
<td>C</td>
<td>25%</td>
<td>38%</td>
</tr>
<tr>
<td>E</td>
<td>4.7%</td>
<td>5%</td>
</tr>
<tr>
<td>F</td>
<td>no change</td>
<td>115%</td>
</tr>
</tbody>
</table>

0.13% ~ 40,000 hours
Static Analysis: Evaluation

Optimizable vertices:

- Cosmos8
- Cosmos9
- Cosmos11
- Cosmos14
- Cosmos15

Proportion (%)

Data center

Cosmos8  Cosmos9  Cosmos11  Cosmos14  Total
Static Analysis: Evaluation

Optimizable vertices:

![Bar chart showing proportions of optimizable vertices for different data centers]

- Cosmos8
- Cosmos9
- Cosmos11
- Cosmos14
- Cosmos15

Data center

Proportion (%)

0.05 0.1 0.15 0.2 0.25 0.3

Total

0.13
Optimizable vertices:

Data center

0.13% ~ 40,000 hours
Static Analysis: Evaluation

Optimizable vertices:

Case studies:

<table>
<thead>
<tr>
<th>Job</th>
<th>CPU Time</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>23%</td>
<td>30%</td>
</tr>
<tr>
<td>B</td>
<td>no change</td>
<td>no change</td>
</tr>
<tr>
<td>C</td>
<td>25%</td>
<td>38%</td>
</tr>
<tr>
<td>E</td>
<td>4.7%</td>
<td>5%</td>
</tr>
<tr>
<td>F</td>
<td>no change</td>
<td>115%</td>
</tr>
</tbody>
</table>

0.13% ~ 40,000 hours
Case Study in SCOPE: Example

//optimization opportunity

```sql
SELECT url_id, url, t_url_id,
    CommonMethod.GetTargetUrl(url,t_url) AS t_url
FROM SSTREAM @VLPMapSS;
```

//optimized code

```sql
SELECT url_id, url, t_url_id,
    string.IsNullOrEmpty(t_url)?url:t_url AS t_url
FROM SSTREAM @VLPMapSS;
```

Vertex level improvement: 42%
Job level improvement: 25%
Cross-language Optimizations in Big Data Systems: Summary

- Static analysis for method inlining opportunities
- Only optimizations that reduce cross-runtime interactions
- Large scale evaluation
Conclusions

• Actionable performance analyses
• Easy to exploit classes of optimizations
• Future work: automatically inferring optimization patterns across and within *different domains*
Huawei Dresden Research Center

Research topics:

- Design and development of embedded systems
- Program analysis and formal verification
- Software testing: fuzzing and automated test case generation
Huawei Dresden Research Center

Research topics:

- Design and development of embedded systems
- Program analysis and formal verification
- Software testing: fuzzing and automated test case generation

We are hiring!
Questions?
DecisionProf: Static Preprocessing

\[
\text{startDecision;}
\]

\[
\text{startCheck: a(); endCheck();}
\]
\[
\text{startCheck: b(); endCheck();}
\]
\[
\text{startCheck: c(); endCheck();}
\]

\[
\text{endDecision;}
\]

\[
\text{if (a() && b() && c()) { }
\]
\[
\text{...}
\]

- Hoists leaf expressions
- Beginning and end of each decision
- Beginning and end of each checks
DecisionProf: Safe Check Evaluation

Collect and undo all writes to variables and object properties that may affect code after check evaluation

```javascript
var x = 0;

function a() {
    x++;
    var y = 1;
    .......
}

startCheck: a();
startCheck: b();

// reset all side effects
if (a() && b()) ...
```
Collect and undo all writes to variables and object properties that may affect code after check evaluation

```javascript
var x = 0;

function a() {
    x++;
    var y = 1;
    ......  
}

startCheck: a();
startCheck: b();

// reset all side effects
if (a() && b()) ...
Collect and undo all writes to variables and object properties that may affect code after check evaluation.

```javascript
var x = 0;

function a () {
    x++;
    var y=1;
    ........
}

startCheck: a();
startCheck: b();

//reset all side effects
if (a () && b()) ...
```

- write to x affects program state
- program state is changed outside normal execution
Collect and undo all writes to variables and object properties that may affect code after check evaluation.

```javascript
var x = 0;

function a () {
  x++;
  var y = 1;
  .......
}

startCheck: a();
startCheck: b();

//reset all side effects
if (a () && b()) ...
```

- Write to `x` affects program state.
- Program state is changed outside normal execution.
- Dynamically execute `x = 0;`
DecisionProf: Finding Optimal Order

<table>
<thead>
<tr>
<th>Cost</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>false</td>
</tr>
<tr>
<td>3</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
</tr>
</tbody>
</table>
DecisionProf: Actual vs Estimated Cost

![Graph showing the comparison between actual and estimated costs. The x-axis represents the estimated cost, while the y-axis represents the execution time in milliseconds. The data points are plotted and show a trend where execution time increases with estimated cost.]
Job Artifacts

- Job Algebra
  - Vertices - processes
  - Edges - data flow
- Runtime statistics
- Script source code
- Generated C# and C++ code
Profiling Infrastructure

Job algebra

V1 → V2 → V3

Analysis of C++ code

Class names

Analysis of C# code

- .NET framework methods
- User-written methods
- Processors and reduces

Sources of C# code + Inlinealbe methods
## Native vs Non-Native Time

The chart below illustrates the proportions of time relative to data center time for different data centers.

<table>
<thead>
<tr>
<th>Data Center</th>
<th>Cosmos8</th>
<th>Cosmos9</th>
<th>Cosmos11</th>
<th>Cosmos14</th>
<th>Cosmos15</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Non-Native</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Grey</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

- **Cosmos8**: 60% Native, 40% Non-Native, 0% Grey
- **Cosmos9**: 60% Native, 40% Non-Native, 0% Grey
- **Cosmos11**: 60% Native, 40% Non-Native, 0% Grey
- **Cosmos14**: 60% Native, 40% Non-Native, 0% Grey
- **Cosmos15**: 60% Native, 40% Non-Native, 0% Grey
- **Total**: 60% Native, 40% Non-Native, 0% Grey

*Note: The chart provides a visual representation of the time proportions for each data center, with Native, Non-Native, and Grey sections indicated.*