D-Goldilocks: Automatic Redistribution of Remote Functionalities for Performance and Efficiency

KIJIN AN AND ELI TILEVICH
SOFTWARE INNOVATIONS LAB
Distribution

• Why Distributed Computing?
  • Take advantage of remote computing resources
  • Improve performance and/or efficiency

• Distribution Benefits
  • Access superior remote resources
  • Share the computational load

• Distribution Costs
  • Communication Overhead
  • Partial failure
  • Security
Distribution
Distribution
Distribution
Distribution
Distribution
Distribution (Granularity of Remote Service)

Too Crude!
(Granularity of Remote Service)
Distribution (Granularity of Remote Service)

Too Fine!
(Granularity of Remote Service)

(Asynchronous Executions)
Too Much Remote Execution not always beneficial: Nano Service Anti-pattern [Mohar 2012 et. al]

Slower Speed | Low Overhead | Bulk
---|---|---
Faster Speed | High Overhead | Nano

Too Much Remote Execution not always beneficial: Nano Service Anti-pattern [Mohar 2012 et. al]

"Performance" | "Efficiency"
Motivating Real-world’s Example: Bookworm

Client-Side

Server-Side
/api/ladydog

Executed Sequentially

GetSentence →
getUniqueVocabulary →
GetDialog → getColon →
g getSemiColon

Initial distribution
Motivating Real-world’s Example: Bookworm

Client-Side

Server-Side

/api/ladydog

Executed Sequentially

GetSentence →
getUniqueVocabulary →
GetDialog → getColon →
getSemiColon

“No Results Until All tasks to complete!”

Initial distribution
Motivating Real-world's Example: Bookworm

Client-Side

Server-Side
/api/ladydog

GetSentence
getUniqueVocabulary
GetDialog
color
getSemicolon

Independent of each other
Motivating Real-world’s Example: Bookworm

Client-Side

Server-Side
/api/ladydog

GetSentence
getUniqueVocabulary
GetDialog
color
getSemiColon

Splitting into smallest Units to invoke Remotely
Motivating Real-world’s Example: Bookworm

Client-Side

Server-Side

/api/ladydog

GetSentence

getUnique

Vocabulary

GetDialog

getColor

getSemiColon

Too much Distribution

“It’s Faster Execution!”
By invoking all together, “Asynchronously”
Goldilocks Principle

Too Crude

Too Fine
Goldilocks Principle

Too Crude

Just the right one!

Too Fine
D-Goldilocks

Client-Side

Server-Side (/api/ladydog)

Too Small Distribution

getVoc GetC
GetSen GetDial
getCol

Too Much Distribution

Commotion Overheads

getVoc GetC
GetSen GetDial
getCol

“Too Crude” (Long Latency)

“Too Fine” (Too much Overheads)

Level of granularity
D-Goldilocks

Client-Side

Server-Side (/api/ladydog)

Too Small Distribution

Too Much Distribution

Commutation Overheads

Level of granularity

Goldilocks Principle

“Too Crude” (Long Latency)

“Too Fine” (Too much Overheads)

GetVoc GetC
GetSen GetDial
getCol

“Right” Redistribution

(Right boundary)
Problem Formulation

- Determine which functional distribution from the client’s standpoint would minimize the cost of distributed execution.

\[ C_{\text{Dist}_\text{Exec}}(r) = \alpha \cdot \text{latency}(r) + (1-\alpha) \cdot \Sigma \text{resource}(r) \]

- Execution Time (Performance)
- Consumed Resource (Efficiency)
- Normalizing Parameter
Problem Solution Outline

• Redistribution operations:
  • Partition
    • \([r_1, ..., r_k] = \text{partition}(r)\)
  • Batch
    • \(r_h = \text{batch}([r_1, ..., r_n])\)
How to restructure Remote Services?

- Client Insourcing Refactoring [WWW '20]
  - Undoing Distribution
How to restructure Remote Services?

- Client Insourcing Refactoring [WWW ’20]
  - Undoing Distribution
How to restructure Remote Services?

- Client Insourcing Refactoring [WWW ’20]
  - Undoing Distribution
How to restructure Remote Services?

- **Client Insourcing Refactoring [WWW ’20]**

1. Identifying Entry/Exit Points of Remote Functionality
2. Constraints Solving & Code Transformation
How to restructure Remote Services?

- Client Insourcing Refactoring as re-distribution framework

(Original Distribution) \[ \text{r} \rightarrow \text{r\_local} \]

(Re-Distribution) \[ \text{r} \rightarrow \text{r\_local}' \]

- [Any Refactoring] For Centralized Apps
- [Any Distributing Frameworks]

- [Kwon ICDCS'13, EXTREMEJS '12,..]

- [Client Insourcing Refactoring] [WWW '20]
Restructuring: Partition

(Original Distribution)

Remote

Local

Client
Insourcing
Refactoring

r

r_local

(Equivalent Variant)

(Independent sets of functions)

r_1

r_2

r_{k-1}

r_k

Partitioning
Restructuring: Partition

(Original Distribution)

(Re-Distribution)

Maximum # of Distributions

(Equivalent Variant)

(Independent sets of functions)

Remote

Local

Client

Insourcing

Refactoring

Partitioning

Distribution

28
Restructuring: Partition

(Original Distribution)

Remote

Local

Client Insourcing Refactoring

r

$\text{partition}$

(Re-Distribution)

$r_1$

$r_2$

$r_{k-1}$

$r_k$

Partioning \texttt{r\_local} into independent sets of functions

- Initial Candidates: \texttt{ALL function decls} in \texttt{r\_local}
- Find partitions that are independent each other by using \texttt{Dependency analysis} for "Control flows" and "global variables" between \texttt{function decls}
Restructuring: Partition & Batch

(Original Distribution)

Remote

Local

Client Insourcing

(Re-Distribution)

Partitioning

Batching

(Re-Distribution)

Partitioning

Batching

(Equivalent Variant)

(r) → partition → \( r_1 \) \( \ldots \) \( r_{k-1} \) \( r_k \) → batch* → (Re-Distribution)

r_local

\( r_{local1} \) \( \ldots \) \( r_{local,k-1} \) \( r_{localK} \)
Restructuring: Partition & Batch

(Original Distribution)

Remote

Local

Client Insourcing

(Equivalent Variant)

(Reduced Function Refactoring)

Restructuring: Partition & Batch

How to distribute?
(local->remote)
Restructuring: Partition & Batch

(Original Distribution)

Remote

\( r \)

\[ \rightarrow \]

**partition**

\[ \rightarrow \]

\( r_1 \)

\( r_2 \)

\( r_{k-1} \)

\( r_k \)

\[ \rightarrow \]

**batch**

(Re-Distribution)

Remote Facade

Remote Facade

\( r_1 \)

\( r_{k-1} \)

\( r_k \)

\( r_2 \)

(Inline Function Refactoring)

Partitioning

Batching Remote Invocation

Remote

Local

Client

Insourcing
Batching Remote Invocation (Batch)

- **Distributing Programming Pattern** [Fowler '02, Ibrahim et.al ECOOP '09]

The **Client DTO** stub accumulates the fine-grained service invocations then, transfers them (parameters) in bulk.

The **remote Façade** function sequentially invokes the bundled services. Then, it combines their execution results and returns in bulk.
Process for D-GOLDILOCKS

Initial Distribution (Full-Stack JS app)

JS

Client Insourcing (z3 Solver)

Centralized Variant function

Partitioning

Batching
Remote Invocation

Batching Parameter Sets

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[0, 1, 2, 3, 4, 5, 6, 8, 9]

[0, 1, 2, 3, 4, 5, 7, 8]

[0, 1, 2, 3, 4, 5, 6, 7, 8]

Façade

Distribution Middleware (multi-core)

BRI patterns

DTO

templates

(Headless Browser Testing framework)

new Distributions

Measuring Cost Functions
Process for D-GOLDILOCKS

Initial Distribution (Full-Stack JS app)

Client Insourcing (z3 Solver)

Centralized Variant function

Partitioning

Batching Remot Invocation

Distribution Middleware (multi-core)

BRI patterns

Facade

DTO

templates

new Distributions

(Headless Browser Testing framework)

Measuring Cost Functions

Batching Parameter Sets

[0, 1, 6, 8, 2, 3, 5, 7],
[0, 1, 2, 3, 5, 7, 6, 8],
[2, 3, 6, 8, 0, 1, 5, 7],
[0, 3, 6, 8, 1, 2, 5, 7],
[0, 1, 2, 6, 8, 3, 5, 7],
[0, 1, 2, 3, 6, 5, 7, 8],
[0, 1, 6, 7, 2, 3, 5, 8],
[0, 1, 6, 7, 2, 3, 5, 8],
[0, 1, 6, 7, 2, 3, 5, 8]
Process for D-GOLDILOCKS

- **Instrumenting Time**: a couple of MINs ~ a couple of HOURS for each subject (DELL-OPTIPLEX5050)
- Depending on possible Combinations to batch them
Evaluation: Research Questions

- **RQ1:** Value: How much **programmer effort** is **saved** by D-GOLDILOCKS’s automatic redistribution operations?
- **RQ2:** Cost Model Correctness: How applying the partition and batch operations affect the distributed execution’s “latency” and “consumed resources” attributes?
- **RQ3:** Utility of Cost Model for Redistribution: How useful is the **cost function** for guiding redistribution decisions?
- **RQ4:** Energy Consumption: What is the effect of redistribution on the amount of **energy consumed** by the client?
## Subject Full-stack JavaScript Apps: Original Performance and Efficiency

<table>
<thead>
<tr>
<th>Remote Services</th>
<th>L(ms)</th>
<th>$\Sigma$TCP</th>
<th>$f_{CL}^{LOC}$</th>
<th>$f_{decl}$</th>
<th>$f_{sub}^{ind}$</th>
<th>IDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/ladypet</td>
<td>77.83</td>
<td>337</td>
<td>394</td>
<td>9</td>
<td>8</td>
<td>1.6M</td>
</tr>
<tr>
<td>/api/thedea</td>
<td>164.62</td>
<td>695</td>
<td>394</td>
<td>9</td>
<td>8</td>
<td>1.6M</td>
</tr>
<tr>
<td>/api/bigtrip</td>
<td>42.11</td>
<td>304</td>
<td>394</td>
<td>9</td>
<td>8</td>
<td>1.6M</td>
</tr>
<tr>
<td>/string-fasta</td>
<td>29.85</td>
<td>328</td>
<td>38</td>
<td>5</td>
<td>2</td>
<td>76</td>
</tr>
<tr>
<td>/cflow-rec</td>
<td>35.43</td>
<td>326</td>
<td>49</td>
<td>4</td>
<td>3</td>
<td>245</td>
</tr>
<tr>
<td>/prprty/brokers</td>
<td>20.64</td>
<td>323</td>
<td>379</td>
<td>3</td>
<td>3</td>
<td>1.5K</td>
</tr>
</tbody>
</table>

... (Total 12 Subjects from 4 Full-Stack Apps)
### Subject Full-stack JavaScript Apps

How many Lines of Code ($f^{LOC}_{CI}$) and Independent sub-functions are in the original remote functionality (**Centralized Variant**)?

<table>
<thead>
<tr>
<th>Remote Services</th>
<th>L(ms)</th>
<th>$\sum$TCPU</th>
<th>$f^{LOC}_{CI}$</th>
<th>$f_{decl}$</th>
<th>$f^{ind}_{sub}$</th>
<th>IDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/ladypet</td>
<td>77.83</td>
<td>337</td>
<td>394</td>
<td>9</td>
<td>8</td>
<td>1.6M</td>
</tr>
<tr>
<td>/api/thedea</td>
<td>164.62</td>
<td>695</td>
<td>394</td>
<td>9</td>
<td>8</td>
<td>1.6M</td>
</tr>
<tr>
<td>/api/bigtrip</td>
<td>42.11</td>
<td>304</td>
<td>394</td>
<td>9</td>
<td>8</td>
<td>1.6M</td>
</tr>
</tbody>
</table>

... (Total 12 Subjects from 4 Full-Stack Apps)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>$f^{LOC}_{CI}$</th>
<th>$f_{decl}$</th>
<th>$f^{ind}_{sub}$</th>
<th>IDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>/string-fasta</td>
<td>29.85</td>
<td>328</td>
<td>38</td>
<td>5</td>
<td>2</td>
<td>76</td>
</tr>
<tr>
<td>/cflow-rec</td>
<td>35.43</td>
<td>326</td>
<td>49</td>
<td>4</td>
<td>3</td>
<td>245</td>
</tr>
<tr>
<td>/prprty/brokers</td>
<td>20.64</td>
<td>323</td>
<td>379</td>
<td>3</td>
<td>3</td>
<td>1.5K</td>
</tr>
</tbody>
</table>
Subject Full-stack JavaScript Apps

RQ1: Value: How much programmer effort is saved by D-GOLDILOCKS’s automatic redistribution operations?

<table>
<thead>
<tr>
<th>Distribution</th>
<th>JavaScript Template Functions</th>
<th>New Distributions</th>
<th>IDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0], [1], [6, 8], [2, 3], [5, 7]</td>
<td>[0], [1], [2], [3], [5, 7], [6, 8]</td>
<td>[0, 1], [2], [6], [8], [3, 5, 7]</td>
<td>1.6M</td>
</tr>
<tr>
<td>[0], [1], [2], [3], [5, 7], [6, 8]</td>
<td>[0], [1], [2], [3], [6], [5, 7, 8]</td>
<td>[0], [1], [2], [3], [6], [5, 7, 8]</td>
<td>1.6M</td>
</tr>
<tr>
<td>[2], [3], [6], [8], [0, 1], [5, 7]</td>
<td>[0], [3], [6], [8], [1, 2], [5, 7]</td>
<td>[0], [1], [2], [6], [8], [3, 5, 7]</td>
<td>1.6M</td>
</tr>
<tr>
<td>[0], [1], [2], [6], [8], [3, 5, 7]</td>
<td>[0], [1], [2], [3], [6], [5, 7, 8]</td>
<td>[0], [1], [2], [3], [6], [5, 7, 8]</td>
<td>1.6M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distribution</th>
<th>JavaScript Template Functions</th>
<th>New Distributions</th>
<th>IDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 1]</td>
<td>[2, 3]</td>
<td>[5, 8]</td>
<td>[6, 7]</td>
</tr>
<tr>
<td>[0, 1]</td>
<td>[2, 3]</td>
<td>[5, 8]</td>
<td>76</td>
</tr>
<tr>
<td>[0, 1]</td>
<td>[2, 3]</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>[0, 1]</td>
<td>76</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Subject Full-stack JavaScript Apps: How much programmer effort is saved by D-GOLDILOCKS’s automatic redistribution operations?
Subject Full-stack JavaScript Apps

**RQ1 Value:** How much **Programmer Effort** is **Saved** by D-GOLDILOCKS’s automatic redistribution operations?

**Initial Distribution**

- **Templates**
- **Batching** Remote Invocation
- **Client Insourcing & Partitioning**

(Headless Browser Testing framework)

- **LOC**
- **f_{CI}**
- **f_{decl}**
- **f_{sub}**
- **IDI**

<table>
<thead>
<tr>
<th>LOC</th>
<th>decl</th>
<th>sub</th>
<th>IDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>394</td>
<td>9</td>
<td>8</td>
<td>1.6M</td>
</tr>
<tr>
<td>394</td>
<td>9</td>
<td>8</td>
<td>1.6M</td>
</tr>
<tr>
<td>394</td>
<td>9</td>
<td>8</td>
<td>1.6M</td>
</tr>
<tr>
<td>38</td>
<td>5</td>
<td>2</td>
<td>76</td>
</tr>
<tr>
<td>49</td>
<td>4</td>
<td>3</td>
<td>245</td>
</tr>
<tr>
<td>379</td>
<td>3</td>
<td>3</td>
<td>1.5K</td>
</tr>
</tbody>
</table>

- **All combinations of Batching of** $f_{sub}$

394 $\times$ 4139 $\sim= 1.6 \times 10^6$ ULOCs
**RQ2: Model Correctness (latency)**

- The larger the number of new remote functionalities, the smaller is the aggregate average latency
  \[ \text{latency}(r) = \frac{1}{n} \sum_{i=1}^{n} T(r_i) \]

- Splitting a single long-running remote function into a small number of asynchronously invoked parts

**Latency**

<table>
<thead>
<tr>
<th># of Remote Executions</th>
<th>Latency[ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RQ2: Model Correctness (Resources)

- We measured total CPU Utilization to invoke a remote service \( r \): \( \text{resource}(r) = \sum_{i=1}^{n} CPU(r_i) \)

- Consuming Client’s Resource a lot to invoke multiple remote executions, propositionally to # of invocations
Cost Function

- Scaling Factor $\alpha$

  - We empirically determined the required normalizing factor for the latency(millisecond) and sum of CPU usages terms by scaling the observed latency/CPU usage ratios across all measurements

$$\alpha = \frac{L}{\Sigma T_{CPU}} = 0.9281$$

$$C_{Dist\_Exec}(r) = \alpha \cdot \text{latency}(r) + (1-\alpha) \cdot \Sigma \text{resource}(r)$$

Execution Time (Performance)  
Consumed Resource (Efficiency)
RQ3:—Utility of Cost Model
RQ3: Utility of Cost Model

Cost Function

Too Small Distribution
Optimal Cost
Too Much Distribution

/api/thereadroom
/api/thebigtrip
/api/ladypet
/api/thegift
/api/the_d
/api/thecask
/api/wallpaper
/api/offshore
**RQ4:** What is the effect of redistribution on the amount of energy consumed by the client?

- We natively build the subject app *(BookWorm)* by using *Apache Cordova*
- *PowerTutor* [L Zhang et.al]: a model-based energy profiler for mobile apps
- *Energy Consumptions* *(EC)* for *Original, Worse, and Best*
Energy Consumption (Original)

- Cost Function Versus. Energy Consumption

<table>
<thead>
<tr>
<th>EC_{original_dist}</th>
<th>Original</th>
<th>8.4mJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_{best_dist}</td>
<td>MIN Cost</td>
<td>13.4mJ</td>
</tr>
<tr>
<td>EC_{worst_dist}</td>
<td>MAX Cost</td>
<td>47.4mJ</td>
</tr>
</tbody>
</table>

Lowest Energy Consume But Highest Latency!

/api/thereadroom

getSentence
getDialog
getVocabulary
getPeriods
getQuestions
getCommas
getColons
getSemiColons
## Energy Consumption (Worst)

- Cost Function Versus. Energy Consumption

<table>
<thead>
<tr>
<th>EC_{original_dist}</th>
<th>Original</th>
<th>8.4mJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_{best_dist}</td>
<td>MIN Cost</td>
<td>13.4mJ</td>
</tr>
<tr>
<td>EC_{worst_dist}</td>
<td>MAX Cost</td>
<td>47.4mJ</td>
</tr>
</tbody>
</table>

600% More Energy Consumption!

/api/thereadroom

getSentence
getDialog
getCommas
getColons
getQuestions
getPeriods
getSemiColons

getVocabulary
Energy Consumption (Best Dist. by D-Goldilocks)

- Cost Function Versus. Energy Consumption

<table>
<thead>
<tr>
<th>EC_{original_dist}</th>
<th>Original</th>
<th>8.4mJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_{best_dist}</td>
<td>MIN Cost</td>
<td>13.4mJ</td>
</tr>
<tr>
<td>EC_{worst_dist}</td>
<td>MAX Cost</td>
<td>47.4mJ</td>
</tr>
</tbody>
</table>

50% More Energy Consumption

- getSentence
- getDialog
- getVocabulary
- getQuestions
- getCommas
- getColons
- getPeriods
- getSemiColons

53
Conclusion

- A set of domain-specific automatic refactoring for reshaping and redistribution.

- A cost function-based heuristic for identifying how to improve the performance and efficiency of distributed apps by reshaping the original distribution, which was too crude.

- A systematic evaluation of our approach’s value, utility, and efficiency for our reference implementation “D-Golilocks”
Future Work

- Problem Formulation & Solution for different **Capacity** of **Clients** and **Servers** including **Network condition**

- Adaptation to **Edge Computing** for addressing their resource constraints and execution volatility

- Other types of Software **Evolution Scenarios**
Thank you!
**Appendix-1**: Can multiple Executions reduce the aggregate average latency non-linearly?:\[ \text{latency}(r) = \frac{1}{n} \sum_{i=1}^{n} T(r_i) \]

- **Latency_1**
  \[
  = \frac{1}{5} (1000+1000+1000+1000+1000) = 1000 \text{ (ms)}
  \]

- **Latency_2**
  \[
  = \frac{1}{5} (950+150+150+150+150) = 310 \text{ (ms)}
  \]

 Executed 5 sub-tasks
 Sequentially for 1000ms

 Executed 4 sub-tasks
 Sequentially for 150ms

 Executed a heavy processing function for 950ms
Appendix-2: Code-Base Example

Client Insourcing

Batching fine-grained service invocations