Leveraging Parallel Data Processing Frameworks with Verified Lifting

Maaz Ahmad

Alvin Cheung
Motivation

Data Collection Tool ➔ Data ➔ Data Analytics Application (Sequential Java)
Motivation

Data Collection Tool → Data → Data Analytics Application (Sequential Java)
Motivation

Data Collection Tool

Data

Data Analytics Application (Sequential Java)
Motivation
Motivation

Data Collection Tool → Data → Data Analytics Application (Sequential Java)
Motivation

Data Collection Tool → Data → Data Analytics Application (Sequential Java)
Motivation

Data Collection Tool → Data → Data Analytics Application (Sequential Java)
Motivation

I need something faster.

Data Collection Tool → Data → Data Analytics Application (Sequential Java)
Parallel Processing Frameworks

- Spark
- GraphLab
- Hadoop
- Apache Storm™
- OpenMP
- Flink
Parallel Processing Frameworks

Which one is right for me?
Parallel Processing Frameworks

Which one is right for me?

How do I program in this?
Parallel Processing Frameworks

Which one is right for me?
How do I program in this?
I will have to re-write my application!
Parallel Processing Frameworks

Which one is right for me?

How do I program in this?

I will have to re-write my application!

Re-write might introduce bugs.
How can we make life easier?
How can we make life easier?

Java To Spark Compiler
Syntax Directed Rules

Hard to come up with rules
Brittle to code pattern changes
Syntax Directed Rules

Hard to come up with rules
Brittle to code pattern changes

Java

```java
for(int i = 0; i < data; i++){
    ...
}
```

Hadoop

```java
mapper(key, data){
    ...
}
reducer(key, values){
    ...
}
```
How do we do this?

- Program analysis
- Synthesis
- Theorem prover

Syntax Directed Rules

```
for(int i = 0; i < data; i++) {
    ...
}
```

Verified Lifting

```
f_m(val) \rightarrow ...
f_r(val1, val2) \rightarrow ...
output = reduce(map(data, f_m), f_r);
```

Syntax Directed Rules

```
mapper(key, data) {
    ...
}
reducer(key, values) {
    ...
}
```

Hard to come up with rules
Brittle to code pattern changes
Introducing CASPER

- Re-targts sequential Java code fragments to Hadoop/Spark frameworks.

- **Input:** Unannotated sequential Java application source code.

- **Output:** Translated application source code that runs on top of Hadoop/Spark to leverage its parallel execution.
MapReduce Overview
MapReduce Overview

Input Data

Data Split → Mapper

Data Split → Mapper

Data Split → Mapper

Diagram showing the flow of data through MapReduce, starting with input data being split and processed by mappers.
Verified Lifting

• Infer code semantics (summary) in a high level specification

• A summary describes the effect of code on the output variables

Java Code Fragment

```java
data_sqr = 0;
for(int i = 0; i < data.size(); i++){
    data_sqr += data[i] * data[i];
}
```

Summary

\[ data_sqr \equiv \sum_{i = 0}^{i = \text{data.size() - 1}} data[i]^2 \]
Verified Lifting

• Infer code semantics (summary) in a high level specification

• A summary describes the effect of code on the output variables

```java
data_sqr = 0;
for(int i = 0; i < data.size(); i++) {
    data_sqr += data[i] * data[i];
}
```

Post-condition

Summary

\[
data_sqr \equiv \sum_{i = 0}^{\text{data.size()} - 1} data[i]^2
\]
Verified Lifting

- Infer code semantics (summary) in a high level specification
- A summary describes the effect of code on the output variables

**Java Code Fragment**

```java
data_sqr = 0;
for(int i = 0; i < data.size(); i++){
    data_sqr += data[i] * data[i];
}
```

**Post-condition**

\[ data_sqr \equiv \sum_{i = 0}^{i = \text{data.size() - 1}} data[i]^2 \]
Verified Lifting

• Infer code semantics (summary) in a high level specification

• A summary describes the effect of code on the output variables

Java Code Fragment

```java
data_sqr = 0;
for(int i = 0; i < data.size(); i++){
    data_sqr += data[i] * data[i];
}
```

Summary

\[ data_sqr \equiv \sum_{i=0}^{i=data.size()-1} data[i]^2 \]

• Specifications must be trivial to translate.
• Program specification exhibits good parallelism.
Code Summaries in Casper
Code Summaries in Casper

\[ \forall v \in \text{outputVariables}. \]
∀\(v \in \text{outputVariables}\). \(v \equiv f_{\text{reduce}}(v_0, \text{reduce}(\text{map}\,(\text{data}, f_{\text{map}}), f_{\text{reduce}}))\)
∀ v ∈ outputVariables. v ≡ f_{reduce}(v_0, reduce(map(data, f_{map}), f_{reduce}))

Where,

map and f_{reduce} are synthesized for each code fragment.

f_{map} map p map and f_{reduce} are synthesized for each code fragment.
Restricting Search Space

• Use Syntax-Guided Synthesis (SyGuS) to generate $f_{map}$ and $f_{reduce}$.

• Use a grammar to specify a set of candidate summaries.

• Grammar is dynamically generated for each code fragment.
Grammar Generation: $f_{map}$

- The body of $f_{map}$ is just a sequence of emits.
  - Begin with number of emits equal to number of output variables.
  - Incrementally add emits statements up to a user-defined bound.

```
Map  →  Map Map | Emit

Emit →  emit(Key, Value); | if (Condition) emit(Key, Value);

Key  →  IntExp | StringExp | BoolExp | ... 

Value →  IntExp | StringExp | BoolExp | ... 
```
Grammar Generation: $f_{map}$

- The key and value for each emit are generated using expression grammars.

```
data_sqr = 0;
for(int i = 0; i < data.size(); i++) {
    data_sqr += data[i] * data[i];
}
```

**Java Code Fragment**

**Integer Expression Grammar**

\[
\text{IntExp} \rightarrow \text{IntExp} + \text{IntExp} | \text{IntExp} \ast \text{IntExp} | \text{data[IntExp]} | \text{IntVal}
\]

\[
\text{IntVal} \rightarrow \text{data_sqr} | i | \text{literal}
\]
Grammar Generation: $f_{\text{reduce}}$

- The body of $f_{\text{reduce}}$ implements a fold operation.

Java Code Fragment

```java
int data_sqr = 0;
for(int i = 0; i < data.size(); i++){
    data_sqr += data[i] * data[i];
}
```

Fold Expression Grammar

\[
\begin{align*}
\text{Reduce} & \rightarrow \text{int } \text{res} = \text{literal}; \text{ for} (\text{value} : \text{values})\{ \text{res} = \text{FoldExp}; \}\text{ emit}(\text{key, res}); \\
\text{FoldExp} & \rightarrow \text{FoldExp} + \text{FoldExp} | \text{FoldExp} * \text{FoldExp} | \text{IntVal} \\
\text{IntVal} & \rightarrow \text{res} | \text{val} | \text{key} | \text{literal}
\end{align*}
\]
Verifying Equivalence

• CASPER uses Hoare-style verification conditions.

• Verification conditions are the *weakest pre-conditions* for the post-condition (code summary) to hold.

• Proving post-conditions for code fragments containing loops requires loop-invariants.
Verifying Equivalence Pt. 2

data_sqr = 0;
for(int i = 0; i < data.size(); i++){
    data_sqr += data[i] * data[i];
}

Verifying Equivalence Pt. 2

```java
data_sqr = 0;
for(int i = 0; i < data.size(); i++){
    data_sqr += data[i] * data[i];
}
```

\[
\text{preCondition} \equiv \text{data}._\text{sqr} = 0
\]
Verifying Equivalence Pt. 2

```c
data_sqr = 0;
for(int i = 0; i < data.size(); i++){
    data_sqr += data[i] * data[i];
}
```

preCondition ≡ data_sqr = 0

postCondition ≡ data_sqr = reduce(map(data, f_map), f_reduce)
Verifying Equivalence Pt. 2

data_sqr = 0;
for(int i = 0; i < data.size(); i++){
    data_sqr += data[i] * data[i];
}

\textit{preCondition} \equiv data\_sqr = 0

\textit{postCondition} \equiv data\_sqr = \text{reduce(map(data, f_{map}), f_{reduce})}

\textit{loopInvariant} \equiv data\_sqr = \text{reduce(map(data[0..i], f_{map}), f_{reduce})}
\quad \land 0 \leq i \leq data\_length
Formal Verification

• We have modelled the MapReduce library in Dafny.

• The generated summary is compiled down to Dafny code.

• Code annotations are automatically generated. These include:
  • Verification conditions
  • Proof lemmas
Lemma Example

```
lemma InductiveStep (data: seq<int>, i: int, data_sqr: int)
  requires invariant(data, i, data_sqr) && i < |data|
  ensures invariant(data, i + 1, data_sqr + (data[i] * data[i]));
{
  assert map (data, i+1) == f_map(data, i) + map(data, i);
  assert f_reduce(f_map(data, i), 0) == data[i] * data[i];
  ...
}
```
CASPER Architecture Diagram
CASPER Architecture Diagram
CASPER Architecture Diagram

Original Source Code

Program Analyzer

Grammar

Failed

Failed

Input Examples (Random)

Candidate Solution Generator

Candidate Summary

Failed

Counter-example

Failed

Bounded Model Checker

Candidate Summary

Failed

Verified Summary

Theorem Prover

Verified

Verified

Code Generator

Hadoop / Spark Code

19
CASPER Architecture Diagram

Original Source Code

Program Analyzer

Grammar

Failed

Failed

Polyglot

Input Examples (Random)

Candidate Solution Generator

Candidate Summary

Counter-example

SKETCH

Candidate Solution Generator

Bounded Model Checker

Candidate Summary

Failed

Dafny

Theorem Prover

Candidate Summary

Failed

Verified Summary

Hadoop / Spark Code

Code Generator

Polyglot
Evaluation

- Compilation performance
- Run-time performance
- Five benchmarks:
  - Summation
  - Word Count
  - String Search (Grep)
  - Linear Regression
  - 3D Histogram
## Compilation Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Program Analysis</th>
<th>Synthesis and BMC</th>
<th># of grammar Iterations</th>
<th>Formal Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summation</td>
<td>&lt; 1s</td>
<td>13s</td>
<td>1</td>
<td>2.8s</td>
</tr>
<tr>
<td>Word Count</td>
<td>&lt; 1s</td>
<td>44s</td>
<td>1</td>
<td>3.4s</td>
</tr>
<tr>
<td>String Match</td>
<td>&lt; 1s</td>
<td>1406s</td>
<td>2</td>
<td>3.3s</td>
</tr>
<tr>
<td>3D Histogram</td>
<td>&lt; 1s</td>
<td>2355s</td>
<td>2</td>
<td>4.2s</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>&lt; 1s</td>
<td>1801s</td>
<td>2</td>
<td>4.8s</td>
</tr>
</tbody>
</table>
Runtime Performance

• Configuration:
  10 node cluster
  8 vCPU, 15GB Memory
  HDFS for data storage
  Hadoop 2.7.2 and Spark 1.6.1

• Average Speedup:
  6.1x on Spark
  3.3x on Hadoop

Benchmark: String Matching (Grep)
Demo!
Summary

Web-page: http://tinyurl.com/casper-homepage
Mailing-list: http://tinyurl.com/casper-subscribe