Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications

By Ahmad and Cheung

Presented by: Ishank Jain
Department of Computer Science

03/19/2019
Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications

CONTENT

- Background
- Research Question
- Method
- Results
- Conclusion
- Questions
BACKGROUND

- Implementations of MapReduce
- Source-to-Source Compilers
- Synthesizing Efficient Implementations
- Query Optimizers and IRs.
BACKGROUND: Implementations of MapReduce

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new
BACKGROUND: Source-to-Source Compilers

Translating Imperative Code to MapReduce

Cosmin Radoi
University of Illinois
cos@illinois.edu

Stephen J. Fink
IBM T.J. Watson Research Center
{sjfink,rabbah}@us.ibm.com

Rodric Rabbah
Samsung Research America
m.sridharan@samsung.com

Manu Sridharan

Abstract
We present an approach for automatic translation of sequential, imperative code into a parallel MapReduce framework. Automating such a translation is challenging; imperative updates must be translated into a functional MapReduce form in a manner that both preserves semantics and enables parallelism. Our approach works by first translating the input code stream MapReduce frameworks [1, 9] provide significant advantages for large-scale distributed parallel computation. In particular, MapReduce frameworks can transparently support fault-tolerance, elastic scaling, and integration with a distributed file system.

Additionally, MapReduce has attracted interest as a parallel programming model, independent of difficulties of distributed computation [2, 4]. MapReduce has been shown to be
BACKGROUND: Synthesizing Efficient Implementations

MapReduce Program Synthesis

Calvin Smith
University of Wisconsin–Madison, USA

Aws Albarghouthi
University of Wisconsin–Madison, USA

Abstract
By abstracting away the complexity of distributed systems, large-scale data processing platforms—MapReduce, Hadoop, Spark, Dryad, etc.—have provided developers with simple means for harnessing the power of the cloud. In this paper, we ask whether we can automatically synthesize MapReduce-style distributed programs from input–output examples. Our ultimate goal is to enable end users to specify large-scale data analyses through the simple interface of examples. We thus present a new algorithm and tool for complexity of distributed computing, e.g., node failures, load balancing, network topology, distributed protocols, etc.

By adding a layer of abstraction on top of distributed systems and providing developers with a restricted API, large-scale data processing platforms have become household names and indispensable tools for the modern software developer and data analyst. In this paper, we ask whether we can raise the level of abstraction even higher than what state-of-the-art platforms provide, but this time with the goal of unleashing the power of cloud computing for the average
BACKGROUND: Query Optimizers and IRs.

Tupleware: Redefining Modern Analytics

Andrew Crotty, Alex Galakatos, Kayhan Dursun, Tim Kraska, Ugur Cetintemel, Stan Zdonik
Department of Computer Science, Brown University
{crottyan, agg, kayhan, kraskat, ugur, sbz}@cs.brown.edu

Abstract

There is a fundamental discrepancy between the targeted and actual users of current analytics frameworks. Most systems are designed for the data and infrastructure of the Googles and Faces of the world—petabytes of data distributed across large cloud deployments consisting of thousands of cheap commodity machines. Yet, the vast majority of users operate clusters ranging from a few to a few dozen nodes, analyze relatively small datasets of up to several terabytes, and perform primarily compute-

Supporting the typical user, then, fundamentally changes the way we should design analytics tools. Current analytics frameworks are built around the major bottlenecks of large cloud deployments, in which data movement (disk to machine and across the network) is the primary performance bottleneck, machines are slow, and failures are the norm [19]. Conversely, with smaller clusters ranging in size from a few to a few dozen nodes, failures are the exception. Most importantly, whereas single-node performance is largely irrelevant in cloud deployments, it can no longer be ignored when targeting small clusters.
MOTIVATION

Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications
Casper is a compiler that can **automatically retarget** sequential Java programs to Big Data processing frameworks such as Spark, Hadoop or Flink.
Access Path Selection in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and retrieval. Nor does a user specify in what order joins are to be performed. The System R optimizer chooses both join order and an access path for each table in the SQL statement. Of the many possible choices, the optimizer chooses the one
MapReduce OPERATORS

- Map operator:
  - Converts a value of type $\tau$ into a multiset of key-value pairs of types $\kappa$ and $\nu$.

- Reduce operator:
  - Combines two values of type $\nu$ to produce a final value.
  - Shuffling.

\[
\begin{align*}
\text{map} : (\text{mset}[\tau], \lambda_m) & \rightarrow \text{mset}[(\kappa, \nu)] \\
\lambda_m : \tau & \rightarrow \text{mset}[(\kappa, \nu)] \\
\text{reduce} : (\text{mset}[(\kappa, \nu)], \lambda_r) & \rightarrow \text{mset}[(\kappa, \nu)] \\
\lambda_r : (\nu, \nu) & \rightarrow \nu
\end{align*}
\]
The program summary, a high-level intermediate representation (IR), describes how the output of the code fragment (i.e., m) can be computed using a series of map and reduce stages from the input data (i.e., mat).

```java
@Summary(
    m = map(reduce(map(mat, \lambda m1, \lambda r), \lambda m2))
    \lambda m1 : (i, j, v) \rightarrow \{(i, v)\}
    \lambda r : (v1, v2) \rightarrow v1 + v2
    \lambda m2 : (k, v) \rightarrow \{(k, v/cols)\}
)
SYSTEM ARCHITECTURE

- Program analyzer:
  - search space description
  - Verification condition
- Summary generator.
- Code generator.
PROGRAM SUMMARIES

- High level IR:
  - To express summaries that are translatable into the target API.
  - Let the synthesizer efficiently search for summaries that are equivalent to the input program.

- Limited number of operations.

Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications

\[
\begin{align*}
PS & := \forall v. \ v = MR \mid \forall v. \ v = MR[v_{id}] \\
MR & := map(MR, \lambda_m) \mid reduce(MR, \lambda_r) \mid join(MR, MR) \mid data \\
\lambda_m & := f : (val) \rightarrow \{\text{Emit}\} \\
\lambda_r & := f : (val_1, val_2) \rightarrow \text{Expr} \\
\text{Emit} & := \text{emit(Expr, Expr)} \mid \text{if(Expr) emit(Expr, Expr)} \mid \\
& \quad \text{if(Expr) emit(Expr, Expr) else Emit} \\
\text{Expr} & := \text{Expr op Expr} \mid \text{op Expr} \mid f(Expr, Expr, \ldots) \mid \\
& \quad n \mid \text{var} \mid (\text{Expr, Expr})
\end{align*}
\]

- \( v \in \text{Output Variables} \quad v_{id} \in \text{Variable ID} \)
- \( op \in \text{Operators} \quad f \in \text{Library Methods} \)
SEARCH SPACE

- To generate the search space grammar, Casper analyzes the input.

- Code analyzer:
  - Dataflow analysis
  - Scanning function

\[
\begin{align*}
PS & : = \forall v. \ v = MR \ | \ \forall v. \ v = MR[v_{id}] \\
MR & : = \text{map}(MR, \lambda_m) \ | \ \text{reduce}(MR, \lambda_r) \ | \ \text{join}(MR, \text{data}) \\
\lambda_m & : = f : (\text{val}) \to \{\text{Emit}\} \\
\lambda_r & : = f : (\text{val}_1, \text{val}_2) \to \text{Expr} \\
\text{Emit} & : = \text{emit}(\text{Expr}, \text{Expr}) \ | \ \text{if}(\text{Expr}) \ \text{emit}(\text{Expr}, \text{Expr}) \ | \\
& \quad \quad \quad \quad \quad \quad \text{if}(\text{Expr}) \ \text{emit}(\text{Expr}, \text{Expr}) \ \text{else} \ \text{Emit} \\
\text{Expr} & : = \text{Expr} \ \text{op} \ \text{Expr} \ | \ \text{op} \ \text{Expr} \ | \ f(\text{Expr}, \text{Expr}, \ldots) \ | \\
& \quad \quad \quad \quad \quad \quad n \ | \ \text{var} \ | \ (\text{Expr}, \text{Expr})
\end{align*}
\]

\[v \in \text{Output Variables, } v_{id} \in \text{Variable ID, } f \in \text{Library Methods}\]
SEARCH SPACE

<table>
<thead>
<tr>
<th>Property</th>
<th>G₁</th>
<th>G₂</th>
<th>G₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map/Reduce Sequence</td>
<td>m → r</td>
<td>m → r → m</td>
<td>m → r → m</td>
</tr>
<tr>
<td># Emits in λₘ</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Key/Value Type</td>
<td>int</td>
<td>int</td>
<td>int or Tuple&lt;int,int&gt;</td>
</tr>
</tbody>
</table>

G₁ = map(mat, λₘ)

λₘ =

\[
\begin{align*}
(i,j,v) & \rightarrow [(i,f)] \\
(i,j,v) & \rightarrow [(i,v)] \\
(i,j,v) & \rightarrow [(j,v+i)] \\
(i,j,v) & \rightarrow [(i+j,v)] \\
\end{align*}
\]

G₂ = reduce(map(mat, λₘ), λₚ)

λₚ =

\[
\begin{align*}
(i,j,v) & \rightarrow [(i,v)] \\
(i,j,v) & \rightarrow [(j,v+1)] \\
(i,j,v) & \rightarrow [(i,j),(v,1)] \\
\end{align*}
\]

G₃ = map(reduce(map(mat, λₘ₁), λₚ), λₘ₂)

λₘ₁ =

\[
\begin{align*}
(i,j,v) & \rightarrow [(i,v)] \\
(i,j,v) & \rightarrow [(i,v,1)] \\
(i,j,v) & \rightarrow [(i+1,j,v),(i,v)] \\
\end{align*}
\]

λₚ :=

\[
\begin{align*}
(v₁,v₂) & \rightarrow v₁ \\
v₁,v₂ & \rightarrow v₁ + v₂ \\
(v₁,1,v₂) & \rightarrow (v₁,1,v₂,2) \\
\end{align*}
\]

λₘ₂ :=

\[
\begin{align*}
(k,v) & \rightarrow [(k,v),(v,k)] \\
k,v & \rightarrow [(v,1,k),(v,2)] \\
(k,v) & \rightarrow [(k,v-cols)] \\
k,v & \rightarrow if(v>0)[(k,v)] \\
\end{align*}
\]
VERIFYING SUMMARIES

- Verification conditions:
  - Hoare logic
  - Predicate logic

\[
\text{invariant}(m, i) \equiv 0 \leq i \leq \text{rows} \land
m = \text{map}(\text{reduce}(\text{map}(\text{mat}[0..i], \lambda_{m1}), \lambda_r), \lambda_{m2})
\]

(a) Outer loop invariant

\[
\text{Initiation} \quad (i = 0) \rightarrow \text{Inv}(m, i)
\]

\[
\text{Continuation} \quad \text{Inv}(m, i) \land (i < \text{rows}) \rightarrow
\text{Inv}(m[i \mapsto \text{sum(}\text{mat}[i])/\text{cols}]) \land i + 1
\]

\[
\text{Termination} \quad \text{Inv}(m, i) \land \neg(i < \text{rows}) \rightarrow \text{PS}(m, i)
\]
SEARCH STRATEGY

- Input:
  - a set of candidate summaries and invariants encoded as a grammar,
  - The correctness specification for the summary in the form of verification conditions.

- CEGIS Algorithm

function synthesize (G, VC):
    \( \Phi = \{\} \) // set of random program states
    while true do
        ps, inv_{1..n} = generateCandidate(G, VC, \Phi)
        if ps is null then return null // search space exhausted
        \( \phi = \text{boundedVerify}(ps, \text{inv}_{1..n}, \text{VC}) \)
        if \( \phi \) is null then return (ps, inv_{1..n}) // summary found
        else \( \Phi = \Phi \cup \phi \) // counter-example found

function findSummary (A, VC):
    G = generateGrammar(A)
    \( \Gamma = \text{generateClasses}(G) \)
    \( \Omega = \{\} \) // summaries that failed verification
    \( \Delta = \{\} \) // summaries that passed verification
    for \( y \in \Gamma \) do
        while true do
            c = synthesize(y - \( \Omega \) - \( \Delta \), VC)
            if c is null and \( \Delta \) is null then
                break // move to next grammar class
            else if c is null then
                return \( \Delta \) // search complete
            else if fullVerify(c, VC) then \( \Delta = \Delta \cup c \)
            else \( \Omega = \Omega \cup c \)
        return null // no solution found
IMPROVISATION

- Verifier failures:
  - Casper must first prevent summaries that failed the theorem prover from being regenerated by the synthesizer.

- Incremental grammar generation:
  - Helps find summaries quicker and is more syntactically expressive.

```plaintext
function synthesize(G, VC):
    \( \Phi = \{\} \) // set of random program states
    while true do
        ps, inv1..n = generateCandidate(G, VC, \( \Phi \))
        if ps is null then return null // search space exhausted
        \( \phi = \text{boundedVerify}(ps, \text{inv1..n}, VC) \)
        if \( \phi \) is null then return (ps, \text{inv1..n}) // summary found
        else \( \Phi = \Phi \cup \phi \) // counter-example found

function findSummary(A, VC):
    G = generateGrammar(A)
    \( \Gamma = \{\} \) // summaries that failed verification
    \( \Delta = \{\} \) // summaries that passed verification
    for \( \gamma \in \Gamma \) do
        while true do
            \( c = \text{synthesize}(\gamma - \Gamma - \Delta, VC) \)
            if \( c \) is null and \( \Delta \) is null then
                break // move to next grammar class
            else if \( c \) is null then
                return \( \Delta \) // search complete
            else if fullVerify(c, VC) then \( \Delta = \Delta \cup c \)
            else \( \Omega = \Omega \cup c \)
        return null // no solution found
```
IMPROVISATION

- Search Algorithm for summaries:
  - Each synthesized summary (correct or not) is eliminated from the search space, forcing the synthesizer to generate a new summary each time.
  - When the grammar is exhausted, Casper returns the set of correct summaries $\Delta$ if it is non-empty

```python
function synthesize ($G$, $VC$):
    $\Phi = \{}$ // set of random program states
    while true do
        $ps$, $inv_{1..n} = generateCandidate(G$, $VC$, $\Phi$)
        if $ps$ is null then return null // search space exhausted
        $\phi = boundedVerify(ps$, $inv_{1..n}$, $VC$)
        if $\phi$ is null then return $(ps$, $inv_{1..n})$ // summary found
        else $\Phi = \Phi \cup \phi$ // counter-example found
    
function findSummary ($A$, $VC$):
    $G = generateGrammar(A)$
    $\Gamma = generateClasses(G)$
    $\Omega = \{}$ // summaries that failed verification
    $\Delta = \{}$ // summaries that passed verification
    for $y \in \Gamma$ do
        $c = synthesize(y - \Omega - \Delta$, $VC$)
        if $c$ is null and $\Delta$ is null then
            break // move to next grammar class
        else if $c$ is null then
            return $\Delta$ // search complete
        else if fullVerify($c$, $VC$) then $\Delta = \Delta \cup c$
        else $\Omega = \Omega \cup c$
    return null // no solution found
```
COST MODEL

- Dynamic cost estimation:
  - It counts the number of unique data values that are emitted as keys.

\[
\begin{align*}
\text{cost}_m(\lambda_m, N, W_m) &= W_m \times N \times \sum_{i=1}^{\lambda_m} \text{sizeOf}(\text{emit}_i) \times p_i \\
\text{cost}_r(\lambda_r, N, W_r) &= W_r \times N \times \text{sizeOf}(\lambda_r) \times \epsilon(\lambda_r) \\
\text{cost}_j(N_1, N_2, W_j) &= W_j \times N_1 \times N_2 \times \text{sizeOf}(\text{emit}_j) \times p_j
\end{align*}
\]
The IR does not currently model the full range of operators across different MapReduce implementations.

Biasing the search towards smaller grammars likely produces program summaries that run more efficiently. Although this is not sufficient to guarantee optimality of generated summaries. It’s a tradeoff between efficient solution and time spent to generate the grammar.

Casper can currently do this for basic Java statements, conditionals, functions, user-defined types, and loops.

Recursive methods and methods with side-effects are not currently supported.
### EVALUATION

Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications

<table>
<thead>
<tr>
<th>Suite</th>
<th># Translated</th>
<th>Mean Speedup</th>
<th>Max Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoenix</td>
<td>7 / 11</td>
<td>14.8x</td>
<td>32x</td>
</tr>
<tr>
<td>Ariths</td>
<td>11 / 11</td>
<td>12.6x</td>
<td>18.1x</td>
</tr>
<tr>
<td>Stats</td>
<td>18 / 19</td>
<td>18.2x</td>
<td>28.9x</td>
</tr>
<tr>
<td>Bigλ</td>
<td>6 / 8</td>
<td>21.5x</td>
<td>32.2x</td>
</tr>
<tr>
<td>Fiji</td>
<td>23 / 35</td>
<td>18.1x</td>
<td>24.3x</td>
</tr>
<tr>
<td>TPC-H</td>
<td>10 / 10</td>
<td>31.8x</td>
<td>48.2x</td>
</tr>
<tr>
<td>Iterative</td>
<td>7 / 7</td>
<td>18.4x</td>
<td>28.8x</td>
</tr>
</tbody>
</table>
(a) CASPER achieves speedup competitive with manual translations
EVALUATION

(b) TPC-H benchmarks

(c) Iterative algorithms
## Evaluation

### Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications

<table>
<thead>
<tr>
<th>Source</th>
<th>Mean Time (s)</th>
<th>Mean LOC</th>
<th>Mean # Op</th>
<th>Mean TP Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoenix</td>
<td>944</td>
<td>13.8 (13.1)</td>
<td>2.3 (2.1)</td>
<td>0.35</td>
</tr>
<tr>
<td>Ariths</td>
<td>223</td>
<td>9.4 (7.6)</td>
<td>1.6 (1.2)</td>
<td>4</td>
</tr>
<tr>
<td>Stats</td>
<td>351</td>
<td>7.6 (5.8)</td>
<td>1.8 (1.8)</td>
<td>0.6</td>
</tr>
<tr>
<td>Bigλ</td>
<td>112</td>
<td>13.6 (10)</td>
<td>1.8 (2.0)</td>
<td>0.4</td>
</tr>
<tr>
<td>Fiji</td>
<td>1294</td>
<td>7.2 (7.4)</td>
<td>1.4 (1.6)</td>
<td>0.1</td>
</tr>
<tr>
<td>TPC-H</td>
<td>476</td>
<td>5.9 (n/a)</td>
<td>7.25 (n/a)</td>
<td>0</td>
</tr>
<tr>
<td>Iterative</td>
<td>788</td>
<td>3.3 (3.7)</td>
<td>4.5 (3.5)</td>
<td>2</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------</td>
<td>------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WordCount</td>
<td>2</td>
<td>827</td>
<td></td>
<td></td>
</tr>
<tr>
<td>StringMatch</td>
<td>24</td>
<td>416</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear Regression</td>
<td>1</td>
<td>94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D Histogram</td>
<td>5</td>
<td>118</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YelpKids</td>
<td>1</td>
<td>286</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wikipedia PageCount</td>
<td>1</td>
<td>568</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Covariance</td>
<td>5</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hadamard Product</td>
<td>1</td>
<td>484</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database Select</td>
<td>1</td>
<td>397</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anscombe Transform</td>
<td>2</td>
<td>78</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Casper covers limited set of operations and doesn’t perform well on ML related and Scientific images dataset. Does this make it usable only for beginner programmers?

“Summaries are restricted to only those expressible using the IR, which lacks many features (e.g., pointers) that a general purpose language would have”. Does this restrict the scope of finding a better target code?

Certain methods such as recursive methods are not supported (reason: they don’t gain any speedup). Is the paper not addressing issues that are essential part of general purpose coding?

NOTE: The paper wanted to reduce complexity for user to learn multiple DSL.