Large-scale computation without sacrificing expressiveness

Sangjin Han     Sylvia Ratnasamy

UC Berkeley
Review: MapReduce and Friends

Input

Computation

map
filter
group by
reduce
join
...

Output
Observation 1: \textit{Bulk transformation of immutable data (no fine-grained updates)}
Example 1: Sparse Operations

• k-hop reachability with iterative MapReduce
Example 1: Sparse Operations

- k-hop reachability with iterative MapReduce
Example 1: Sparse Operations

- k-hop reachability with iterative MapReduce
Example 1: Sparse Operations

- k-hop reachability with iterative MapReduce
Example 1: Sparse Operations

- k-hop reachability with iterative MapReduce
Example 1: Sparse operations

- k-hop reachability with iterative MapReduce

Internet router topology graph (1.7M nodes, 22.2M edges)
Review: MapReduce and Friends (cont’d)
Review: MapReduce and Friends (cont’d)
Review: MapReduce and Friends (cont’d)

Observation 2: Static dataflow
(no data-dependent control flow)
Example 2: Irregular parallelism

• Parallel SAT solver

\[ E = (p \lor \neg q) \land (\neg p \lor r \lor s) \land (q \lor \neg s \lor \neg t) \land (\neg p \lor s) \land \cdots \]
Example 2: Irregular parallelism

• Parallel SAT solver

\[ E = (p \lor \neg q) \land (\neg p \lor r \lor s) \land (q \lor \neg s \lor \neg t) \land (\neg p \lor s) \land \ldots \]
Example 2: Irregular parallelism

- Parallel SAT solver

\[ E = (p \lor \neg q) \land (\neg p \lor r \lor s) \land (q \lor \neg s \lor \neg t) \land (\neg p \lor s) \land \ldots \]
Example 2: Irregular parallelism

- Parallel SAT solver

\[ E = (p \lor \neg q) \land (\neg p \lor r \lor s) \land (q \lor \neg s \lor \neg t) \land (\neg p \lor s) \land \ldots \]
MapReduce-like frameworks assume:

1. Bulk transformation of immutable data

2. Static dataflow
Existing frameworks assume:

Our work:

1. **Bulk transformation of immutable data**
   Fine-grained operations on mutable data

2. **Static dataflow**
   Dynamic, data-dependent control flow

Yet we still want **elastic scalability and fault tolerance**
Spinning a small twist to Linda

CELIAS PROGRAMMING MODEL
Programming model =
  data model + computation model
Data Models for Mutable Shared Memory
Data Models for Mutable Shared Memory

Global address space: UPC, X10, Fortress...

Too low level
Data Models for Mutable Shared Memory

Global address space: UPC, X10, Fortress...

Too low level

Key-value tables: RAMCloud, Dynamo, Piccolo...

Limited lookup ability

Consistency concerns
Data Models for Mutable Shared Memory

Global address space: UPC, X10, Fortress...

Key-value tables: RAMCloud, Dynamo, Piccolo...

Tuplespace: Linda

Flexible lookup with any attributes
Individual tuples are immutable

Too low level
Limited lookup ability
Consistency concerns

('employee', 'John', 29)
('todo', 'walk')
('todo', 'shopping')
Programming model =
  data model + computation model

Linda =
  Tuplespace + Linda processes
Linda Processes

in(…)
... out(…)
... in(…)
...
...

Process A

... out(…)
... out(…)
... in(…)
...
...

Process B

... in(…)
... in(…)
... in(…)
...
...

Process C
Linda Processes

_process A_
in(...)  
...  
out(...)  
...

_process B_
...  
out(...)  
...  
out(...)  
...

_process C_
...  
in(...)  
...  
in(...)  
...

 toaster { # No automatic scaling  # No fault tolerance}
Programming model =
  
data model + computation model

Linda =
  
  Tuplespace + Linda processes

Celias =
  
  Tuplespace + microtasks
Microtasks

(‘hello’, 5) (‘world’, 2)
(‘hello’, 7) ...

Function wordcount()

<table>
<thead>
<tr>
<th>Signature</th>
<th>(?word, ?cnt1), (?word, ?cnt2)</th>
</tr>
</thead>
</table>
| Code      | sum := cnt1 + cnt2
          | emit (word, sum) |
Microtasks

<table>
<thead>
<tr>
<th>Function</th>
<th>wordcount()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>(?word, ?cnt1), (?word, ?cnt2)</td>
</tr>
<tr>
<td>Code</td>
<td>sum := cnt1 + cnt2&lt;br&gt;emit (word, sum)</td>
</tr>
</tbody>
</table>

When a signature matches:

1. microtask launch

word = ‘hello’
cnt1 = 5
cnt2 = 7
Microtasks

Function wordcount()

<table>
<thead>
<tr>
<th>Signature</th>
<th>(?word, ?cnt1), (?word, ?cnt2)</th>
</tr>
</thead>
</table>
| Code      | sum := cnt1 + cnt2  
emit (word, sum) |

When a signature matches:

1. microtask launch
2. code execution

5 + 7 = ??
Microtasks

Function `wordcount()`

<table>
<thead>
<tr>
<th>Signature</th>
<th>(?word, ?cnt1), (?word, ?cnt2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td><code>sum := cnt1 + cnt2</code></td>
</tr>
<tr>
<td></td>
<td><code>emit (word, sum)</code></td>
</tr>
</tbody>
</table>

When a signature matches:

1. microtask launch
2. code execution
3. atomic replacement
Two functions: add() and multiply()

\[(A + B) \times (C + D)\]
Two functions: add() and multiply()

\[(A + B) \times (C + D)\]
Two functions: `add()` and `multiply()`

 символь

 笑 Automatic scaling
Two functions: add() and multiply()

😊 Automatic scaling
Two functions: add() and multiply()

😊 Automatic scaling
Two functions: add() and multiply()

😊 Automatic scaling

😊 Fault tolerance
More Examples in the Paper...

• MapReduce
  – Celias is Turing-complete MapReduce-complete!
  – without any artificial sync. barriers

• Single-source shortest path
  – Pregel-style graph processing

• Quicksort
  – Recursive control flow
Summary

• MapReduce-like frameworks are not suitable for algorithms with:
  – Sparse/incremental/fine-grained computation
  – Dynamic dataflow

• Celias comes to our rescue, yet it is also
  – automatically scalable
  – fault tolerant
Open Questions

• Microtask abstraction: good enough? went too far?

• Feasibility of an efficient implementation
  – Reliable tuplespace
  – Signature matching
  – Microtask transactions

• ... what is a killer app of Celias?

• <Your questions here>