Functional Programming for Large and Dynamic Data with Self-Adjusting Computation

Yan Chen  Umut A. Acar  Kanat Tangwongsan
MPI-SWS  CMU  Mahidol University
Big Data

• Lots of data is being generated everywhere every second
  – Social networks
  – Web pages
  – e-Commerce
  – …
## Computing with Big Data

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variety</td>
<td>Expressive</td>
</tr>
<tr>
<td>Volume</td>
<td>Parallel</td>
</tr>
<tr>
<td>Velocity</td>
<td>Incremental</td>
</tr>
</tbody>
</table>
# State of the art

<table>
<thead>
<tr>
<th>System</th>
<th>Expressive</th>
<th>Parallel</th>
<th>Incremental</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapReduce [OSDI’04]</td>
<td>✗</td>
<td>✓</td>
<td>Batch</td>
</tr>
<tr>
<td>Dryad [EuroSys’07]</td>
<td>✗</td>
<td>✓</td>
<td>Batch</td>
</tr>
<tr>
<td>Incoop [SoCC’11]</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Naiad [SOSP’13]</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Functional Language</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Why not?

- Why not use functional programming for computing big data?
  - Research shows that parallelism can be done (NESL, parallel Haskell)
  - We don’t know if we can do incremental computation in large scale
Implicit Self-Adjusting Computation

SML Program (Input) → Type-directed Translation → Self-Adjusting Program (Output)

Dependency Graph
Merge Sort — Complete Run

![Graph showing the relationship between input size and time in a merge sort algorithm. The graph plots the time (in seconds) on the y-axis against the input size (in millions) on the x-axis. The line is labeled as Self-Adjusting.](image-url)
Merge Sort — Change Propagation

![Graph showing time versus input size for self-adjusting merge sort.](image)
Merge Sort — Speedup

![Graph showing the speedup of Merge Sort with input size in millions on the x-axis and speedup on the y-axis. The line is marked as Self-Adjusting.](image-url)
Challenge: Memory Usage

Memory Usage

Input Size

Memory (GB)

Millions

Self-Adjusting

Batch

122G

1.4G
This paper

• Two techniques for controlling memory
  – Precise Dependency Tracking
  – Granularity Control --- Probabilistic chunking
How to reduce space overhead?

Idea: Time and space trade-off

- Precise Dependency Tracking
- Granularity Control
- Self-Adjusting
Type-Directed Translation

- Extend SML types with $\text{(Changeable)$/$\text{(Stable)}$
- Changeable data translates to reference cells
- Any data that depends on changeable data must be changeable
- We use a modal type system to enforce this invariant
Cause of Imprecision

• Modal type system is conservative
  – Deems more data changeable than necessary
• Runtime tracks changeable data
  – Leads to redundant dependencies
• Solution: Use an unsafe imperative type system
Our Approach

- Use the unsafe imperative language as the target language
- Generate code from pure functional programs
- Track the use of changeable precisely in the system by using labels
Theoretical Results

ML $e$ Type/Label Inference $e : \tau$

Destination Passing Translation $e' : \tau'$

Self-Adjusting Program

Correctness Efficiency

Type Soundness
Wordcount with MapReduce

With PDT, the update time is 6x faster, and the memory usage is 2.4x more efficient.
How to reduce space overhead?

Idea: Time and space trade-off

Precise Dependency Tracking

Granularity Control

Self-Adjusting

Speedup

Memory Overhead
Idea: Granularity Control

- Treat blocks of data as a single C
- Fixed block size leads to disproportionately slow update time

- We are using a probabilistic chunking scheme
- See the paper for details!
Merge Sort — Memory

![Graph showing memory usage vs input size for different block sizes.]

- **Non-blocking**: Memory usage increases linearly with input size.
- **Block Size 10**: Memory usage shows more fluctuation but remains generally lower than non-blocking.
- **Block Size 100**: Memory usage is even lower and shows less fluctuation.
- **Batch**: Memory usage is minimal and flat throughout the range.

Memory (GB) vs Millions of Input Size.
Merge Sort — Speedup

[Graph showing speedup vs input size for Block 10, Block 100, and Non-blocking]
How to reduce space overhead?

Idea: Time and space trade-off

- Precise Dependency Tracking
- Granularity Control
- Self-Adjusting

![Graph showing Speedup vs Memory Overhead with points for Precise Dependency Tracking, Granularity Control, and Self-Adjusting]
Applications & Evaluation

• Applications
  – Incremental PageRank
  – Incremental graph connectivity
  – Incremental social circles

• Prototype Implementation
  – Single machine
  – In-memory computation
  – Sequential
PageRank for Twitter Graph

- Twitter graph (30M vertices, 700M edges)
- Change Model: Randomly insert 1,000 edges
PageRank

- PageRank for different social graphs
- 300x speedup with 10x memory overhead

<table>
<thead>
<tr>
<th>Src</th>
<th>Size</th>
<th>Time (s)</th>
<th>Mem</th>
<th>Speedup</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orkut</td>
<td>V:3M E:100M</td>
<td>7</td>
<td>3G</td>
<td>333</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.021</td>
<td>36G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LiveJournal</td>
<td>V:4M E:30M</td>
<td>18</td>
<td>5G</td>
<td>783</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.023</td>
<td>61G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Twitter</td>
<td>V:30M E:700M</td>
<td>137</td>
<td>50G</td>
<td>539</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.254</td>
<td>495G</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Graph Connectivity

- Dataset: LiveJournal (1M vertices, 8M edges)
- Change Model: Randomly remove one vertex
Graph Connectivity

- Dataset: LiveJournal (1M vertices, 8M edges)
- Change Model: Randomly remove one vertex

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Mem</th>
<th>Speedup</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td>105</td>
<td>4G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incremental</td>
<td>0.531</td>
<td>140G</td>
<td>198</td>
<td>35</td>
</tr>
</tbody>
</table>
Summary

• Using functional programming for computing with large and dynamic data
  – Precise Dependency Tracking
  – Granularity Control

• See paper for details!

• Future Work
  – Parallel self-adjusting computation
Thank you!