Exploiting Bounded Staleness to Speed up Big Data Analytics

Henggang Cui

James Cipar, Qirong Ho, Jin Kyu Kim, Abhimanu Kumar, Seunghak Lee, Greg Ganger, Phil Gibbons (Intel), Garth Gibson, Eric Xing

PARALLEL DATA LABORATORY
Carnegie Mellon University
Big Data Analytics

Huge input data → Iterative program fits model → Model parameters (solution)

Design a model

Iterative program fits model
Big Data Analytics

Design a model

Sequential?!?!?

Huge input data
Iterative program fits model
Model parameters (solution)
Big Data Analytics

Partitioned input data

Parallel iterative program

Model parameters (solution)
Big Data Analytics

Partitioned input data → Parallel iterative program → Model parameters (solution)

Parameter server
Example: Topic Modeling

Corpus of documents

Topic modeler

| word-topic table |
| doc-topic table |
Outline

• Stale Synchronous Parallel
• LazyTable design
• Experiment results
Bulk Synchronous Parallel

- Bulk Synchronous Parallel (BSP)
  - Common approach to parallelizing
  - A barrier every a fixed amount of work
    - Define this fixed amount of work as one **clock**
    - In ML apps, often one iteration over input data
    - Clock represents work, not time
  - Threads compute on the states from the last clock
    - Updates during current clock not visible
About Data Staleness

• In BSP, threads can see "out-of-date" values
  • May not see others' updates right away
• Why allow it? Speed!
  • Less synchronizing among threads
  • More using cached values
  • More delaying and batching of updates
• But, too much staleness hurts convergence
  • Important to have staleness bound
Arbitrarily-sized BSP

- Arbitrarily-sized BSP (A-BSP)
  - A clock is not necessarily an iteration
  - Work-per-clock (WPC) controls data staleness
    - Can be tuned to maximize performance
  - Call it “A-BSP" to contrast with normal ML practice
Problem of BSP: Stragglers

• A-BSP still has the straggler problem
  • A slow thread will slow down all
  • Stragglers are common in large systems

• Many reasons for stragglers
  • Hardware: lost packets, SSD cleaning, disk resets
  • Software: garbage collection, virtualization
  • Algorithmic: Calculating objectives and stopping conditions
Stale Synchronous Parallel

- Stale Synchronous Parallel (SSP)
  - Updates are propagated at each clock
  - Threads are allowed to be some number of clocks ahead of the slowest thread
    - The “slack” parameter
  - Another way to explicitly control staleness
  - Tolerant of transient stragglers
Data Staleness

Design a model

Huge input data  Iterative program fits model  Model parameters (solution)

http://www.pdl.cmu.edu/
Data Staleness

Partitioned input data
Parallel iterative program
Model parameters (solution)
• \((i, j)\) -> iteration \(i\), work \(j\)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Clock</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>barrier</td>
<td>2</td>
<td>barrier</td>
<td>3</td>
<td>barrier</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>(2,a)</td>
<td>(2,b)</td>
<td>(3,a)</td>
<td>(3,b)</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>(2,c)</td>
<td>(2,d)</td>
<td>(3,c)</td>
<td>(3,d)</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>(2,e)</td>
<td>(2,f)</td>
<td>(3,e)</td>
<td>(3,f)</td>
</tr>
</tbody>
</table>
BSP Progress and Staleness

- \((i, j) \rightarrow \text{iteration } i, \text{ work } j\)

```
<table>
<thead>
<tr>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
```

**Barriers:**
- Between iterations
- Between works

Carnegie Mellon University
Parallel Data Laboratory

http://www.pdl.cmu.edu/
A-BSP Progress and Staleness

- A-BSP, $wpc = 2$ iterations, slack = 0

<table>
<thead>
<tr>
<th>Clock</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>... (2,a) (2,b) (3,a) (3,b) (4,a) (4,b)</td>
<td>... (2,c) (2,d) (3,c) (3,d) (4,c) (4,d)</td>
<td>... (2,e) (2,f) (3,e) (3,f) (4,e) (4,f)</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 barrier

2 barrier
SSP Progress and Staleness

- SSP, wpc = 1 iteration, slack = 1 clock

<table>
<thead>
<tr>
<th>Thread</th>
<th>Slack of 1 clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>... (2,a)</td>
</tr>
<tr>
<td>2</td>
<td>... (2,c)</td>
</tr>
<tr>
<td>3</td>
<td>... (2,e)</td>
</tr>
</tbody>
</table>

- Same staleness bound as the A-BSP one
- Data staleness for SSP with wpc and slack:
  \[ wpc \times (slack + 1) - 1 = 11 \]
SSP VS A-BSP

• A-BSP is SSP with a slack of zero
• Data staleness bound
  • SSP \{wpc, slack\} == A-BSP \{wpc \times (slack + 1), 0\}
• SSP is a “pipelined” version of A-BSP
  • Tolerates transient stragglers
Outline

• Stale Synchronous Parallel
• LazyTable design
• Experiment results
LazyTable

• A parameter server that supports SSP
  • And, A-BSP by setting slack to 0

• Two primary components
  • A cluster of tablet server processes
    – Each keeps one shard of shared data
    – No replication (currently)
  • A client library
    – For threads accessing shared data
LazyTable Architecture

Partitioned input data

Parallel iterative program on LazyTable

Model parameters (sharded)
Data Model

• A (sparse) collection of “rows”
  • Indexed by key (table_id, row_id)
  • Atomic single-row access
  • Application defined row type
    – Usually STL vector or map
    – Natural representation for vectors in ML app
Primary API functions

• `read(table_id, row_id, slack)`
  • Atomically retrieves a row
  • Data cannot be staler than `slack`, otherwise block

• `update(table_id, row_id, delta)`
  • Atomically updates a row by delta

• `clock()`
  • Signals completion of one clock of work
  • Not a barrier!
    – Synchronize by blocks in read, when necessary
Outline

• Stale Synchronous Parallel
• LazyTable design
• Experiment results
Experiment Setup

• Hardware information
  • Susitna cluster
    – 8 machines, each with 64 cores & 128GB RAM
    – For most experiments

• Basic configuration
  • One client & tablet server per machine
  • One computation thread per core
Application Benchmark

• Topic Modeling
  • Algorithm: Gibbs Sampling on LDA
  • Input: *Nytimes* dataset
    – 300k docs, 100m words, 100k vocabulary
  • Solution quality criterion: Loglikelihood
    – How likely the model generates observed data
    – Becomes higher as the algorithm converges
    – A larger value indicates better quality
Controlling Data Staleness

• SSP
  • Larger slack -> more staleness

• A-BSP
  • Larger wpc -> more staleness

• The tradeoffs with increased staleness
Staleness Increases iters/sec

- Iterations per sec
- Work done (iterations)
- Time (sec)

Henggang Cui © October 13

http://www.pdl.cmu.edu/
Staleness Increases iters/sec

Iterations per sec

Work done (iterations)

Time (sec)

---

Henggang Cui © October 13

http://www.pdl.cmu.edu/
Staleness Increases iter/sec

Iterations per sec

- wpc=1, slack=0
- wpc=1, slack=1

Work done (iterations)

Time (sec)

wpc=1, inc slack
Staleness Increases iters/sec

- **wpc=1, slack=0**
- **wpc=1, slack=1**
- **wpc=1, slack=3**

**Iterations per sec**

**Work done (iterations)**

**Time (sec)**

wpc=1, inc slack
Staleness Increases iter/sec

![Graph showing iterations per sec vs time with different values for wpc and slack](image)

- **wpc=1, slack=0**
- **wpc=1, slack=1**
- **wpc=1, slack=3**
- **wpc=1, slack=inf**

**Time (sec)**

**Work done (iterations)**

**Iterations per sec**

Henggang Cui © October 13
Staleness Reduces converge/iter

Convergence per iter

Loglikelihood (higher is better)

Work done (iterations)
Staleness Reduces convergence/iter

Convergence per iter

Loglikelihood (higher is better)

Work done (iterations)

-1.04 \times 10^9
-1.02 \times 10^9
-1.00 \times 10^9
-9.80 \times 10^8
-9.60 \times 10^8
-9.40 \times 10^8
Staleness Reduces convergence/iter

Convergence per iter

Loglikelihood (higher is better)

Work done (iterations)

-1.04 \times 10^9
-1.02 \times 10^9
-1.00 \times 10^9
-9.80 \times 10^8
-9.60 \times 10^8
-9.40 \times 10^8
-9.00 \times 10^8
-8.60 \times 10^8
-8.20 \times 10^8
-1.04 \times 10^9
-1.02 \times 10^9
-1.00 \times 10^9
-9.80 \times 10^8
-9.60 \times 10^8
-9.40 \times 10^8
-9.00 \times 10^8
-8.60 \times 10^8
-8.20 \times 10^8

- \text{loglikelihood (higher is better)}

- \text{Work done (iterations)}

- \text{Convergence per iter}

- \text{wpc=1, stale=0}

- \text{wpc=1, slack=1}

Staleness Reduces convergence/iter

Henggang Cui © October 13
Staleness Reduces convergence/iter

Convergence per iter

-9.40x10^8
-9.60x10^8
-9.80x10^8
-1.00x10^9
-1.02x10^9
-1.04x10^9

0 20 40 60

Work done (iterations)

- Loglikelihood (higher is better)

- wpc=1, stale=0
- wpc=1, slack=1
- wpc=1, slack=3
Staleness Reduces convergence/iter

Convergence per iter

Loglikelihood (higher is better)

Work done (iterations)

- $wpc=1$, stale=0
- $wpc=1$, slack=1
- $wpc=1$, slack=3
- $wpc=1$, slack=inf

Carnegie Mellon
Parallel Data Laboratory

http://www.pdl.cmu.edu/
Sweet Spot Balances the Two

Convergence per sec

Loglikelihood (higher is better)

Time (sec)

wpc=1, stale=0
wpc=1, slack=1
wpc=1, slack=3
wpc=1, slack=inf

Convergence per iter

Loglikelihood (higher is better)

Work done (iterations)

wpc=1, inc slack

wpc=1, stale=0
wpc=1, slack=1
wpc=1, slack=3
wpc=1, slack=inf

Speed up with best slack

Iterations per sec

Work done (iterations)

Time (sec)
Understanding the Tradeoff

Convergence per iteration

Convergence per second

Iterations per second

The sweet spot

Fresher data

Staler data
Staleness Tradeoff: A-BSP

- **Iterations per sec**
  - Work done (iterations) vs Time (sec)
  - Graphs show convergence per iteration for different wpc values (1, 2, 4, 8) with slack=0.

- **Convergence per iter**
  - Loglikelihood vs Work done (iterations) for different wpc values (1, 2, 4, 8) with slack=0.

- **Convergence per sec**
  - Loglikelihood vs Time (sec) for different wpc values (1, 2, 4, 8) with slack=0.

- **Speed up with best wpc**
  - Graphs indicate that increasing wpc improves convergence per second and iterations per second.
Key Takeaway Insight #1

• Both SSP and A-BSP explicitly control data staleness
  • More data staleness -> iterations faster
  • More data staleness -> iterations less effective
  • Sweet spot balances the two

• Similar performance
  • Because no stragglers
Introducing Stragglers

• Experiments on two types of stragglers
  • Stragglers caused by delays
  • Stragglers caused by background activities

• Compare the behavior of SSP VS A-BSP
  • With the same data staleness bound
Stragglers: Delay

• Delaying some threads
  • Artificially introduce stragglers to the system
  • Have some threads sleep() for a time

• Experiment setup
  • Threads sleep “d” seconds in turn
    – Threads of machine “i” sleep at iteration “i”
  • Compare influence of different “d”
Stragglers: Delay (Results)
Stragglers: Delay (Results)

Ideal slow down: \( \frac{d}{8} \) per iter on 8 machines

Delay \( d \) (sec)

Time per iteration (sec)

Ideal
Stragglers: Delay (Results)

Ideal slow down: $d/8$ per iter on 8 machines

A-BSP slow down: $d/2$ per iter

Time per iteration (sec)

Delay $d$ (sec)

Ideal

$wpc=2$, $slack=0$ (A-BSP)
Stragglers: Delay (Results)

- Ideal slow down: $d/8$ per iter on 8 machines
- SSP tolerates transient stragglers
- A-BSP slow down: $d/2$ per iter

Graph showing:
- Time per iteration vs. Delay $d$ (sec)
- Lines for ideal, A-BSP ($wpc=2$, slack=0), and SSP ($wpc=1$, slack=1)

Stragglers: Delay (Results)
Stragglers: Bg. Work

• Stragglers caused by background disrupting
  • Fairly common in large, shared clusters

• Experiment setup
  • One disrupter process per machine
    – Same number of threads as the client process
    – Should slow down the computation by 50%
  • Work or sleep randomly at each time slot
    – 10% work, 90% sleep
    – Length of each time slot is “t” seconds
  • Experiment on 32 vCloud machines
    – Each with 8 cores & 23GB RAM

Henggang Cui © October 13
http://www.pdl.cmu.edu/
Stragglers: Bg. Work (Results)

![Graph showing the relationship between Disruption duration (sec) and Iteration time increase (%)]
Stragglers: Bg. Work (Results)

Ideally 5%, because 50% slow down with 10% probability.
Stragglers: Bg. Work (Results)

![Graph showing iteration time increase vs. disruption duration](image)

- **Ideal**
  - Disruption duration: 100
  - Iteration time increase: 5%

- wpc=2, slack=0 (A-BSP)
  - Disruption duration: 1-64 sec
  - Iteration time increase: 10%

*Ideally 5%, because 50% slow down with 10% probability*
Stragglers: Bg. Work (Results)

Disruption duration (sec)
Iteration time increase (%)

- Ideal
- wpc=1, slack=1 (SSP)
- wpc=2, slack=0 (A-BSP)

SSP tolerates transient stragglers

Ideally 5%, because 50% slow down with 10% probability
Key Takeaway Insight #2

- SSP is more tolerant of transient stragglers
  - A-BSP has barriers
    - A straggler slows down all
  - SSP gives us more flexibility
The Cost of Increased Flexibility

• Comparing \{wpc=X, \ldots\} with \{wpc=2X, \ldots\}
  • Bytes sent doubled (send update twice as often)
  • Bytes received almost doubled

![Graph showing data comparison between different wpc and slack values.](http://www.pdl.cmu.edu/)

**smaller wpc, larger slack**
Key Takeaway Insight #3

- SSP uses more communication
  - Finer grained division of clocks
Conclusion

• Explicitly control data staleness for parallel ML app.
  • Reduces synchronization and comm. overhead
  • Both SSP and A-BSP

• SSP is tolerant of transient stragglers

• Continuing work
  • Enhance SSP with more efficient communications
  • SSP + Work stealing => no straggler problem?
References


• Nytimes: http://archive.ics.uci.edu/ml/datasets/Bag+of+Words
Related Work References


