Exploiting Bounded Staleness to Speed up Big Data Analytics

Henggang Cui

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

Carnegie Mellon University
Huge input data \textarrow{} Iterative program fits model \textarrow{} Model parameters (solution)
Big Data Analytics Overview

Partitioned input data

Parallel iterative program

Model parameters (solution)
Big Data Analytics Overview

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

Goal: Less sync overhead

Parameter server
Outline

• Two novel synchronization approaches
  • Arbitrarily-sized Bulk Synchronous Parallel (A-BSP)
  • Stale Synchronous Parallel (SSP)
• LazyTable architecture overview
• Taste of experimental results
Bulk Synchronous Parallel

• A barrier every **clock** (a.k.a. epoch)
  • In ML apps, often one iteration over input data

Thread progress illustration:

- **Thread 1**
  - Iterations complete, updates visible
  - Iteration 0-3, clock 2-3
  - Thread 1 blocked by barrier

- **Thread 2**
  - Iteration 1-3, clock 2-3
  - Updates not necessarily visible

- **Thread 3**
  - Iteration 2-3, clock 2-3
  - Updates not necessarily visible
Data Staleness

- In BSP, threads can see "out-of-date" values
  - May not see others' updates right away
  - Convergent apps usually tolerate that
- Allowing more staleness for 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
  - **Staleness should be tunable**
Arbitrarily-sized BSP (A-BSP)

- Work in each clock can be more than one iteration
  - Less synchronization overhead

![Diagram showing two iterations per clock and threads 1, 2, and 3 with thread 1 blocked by barrier]
Problem of (A-)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 (SSP)

- Threads are allowed to be slack clocks ahead of the slowest thread

[HotOS’13, NIPS’13]
Two Dimensional Config. Space

- Iters-per-clock and slack are both tunable
  - A-BSP is SSP with a slack of zero
  - Every SSP config. has an A-BSP counterpart with the same data staleness bound

SSP (iters-per-clock=1, slack=1):

A-BSP (iters-per-clock=2, slack=0):
LazyTable Architecture

Partitioned input data → Parallel iterative program on LazyTable

Model parameters (sharded)
Partitioned input data

Parallel iterative program on LazyTable

Model parameters (sharded)

See the paper for more details
Primary Experimental Setup

- **Hardware information**
  - 8 machines, each with 64 cores & 128GB RAM

- **Basic configuration**
  - One client & tablet server per machine
  - One computation thread per core
Application Benchmark #1

• Topic Modeling
  • Algorithm: Gibbs Sampling on LDA
  • Input: NY Times 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

More apps described and used in paper
Controlling Data Staleness

- **SSP**
  - Larger slack -> more staleness

- **A-BSP**
  - Larger iterations-per-clock -> more staleness

- The tradeoffs with increased staleness
Staleness Increases Iters/sec

Iters-per-clock is 1

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

**Iterations per sec**

- slack=0 (BSP)

**iters-per-clock is 1**

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

**Iterations per sec**

- slack=0 (BSP)
- slack=1 (SSP)

larger iters per sec with more staleness

Iters-per-clock is 1
Staleness Increases Iters/sec

Iterations per sec

- slack=0 (BSP)
- slack=1 (SSP)
- slack=3 (SSP)

Staleness Increases Iters/sec

Iters-per-clock is 1
Staleness Reduces Converge/iter

Convergence per iter

Convergence (higher is better)

0 20 40 60

Iterations done

iters-per-clock is 1

http://www.pdl.cmu.edu/
Staleness Reduces Converge/iter

Convergence per iter

- Iters-per-clock is 1
- Iterations done

Slack = 0 (BSP)
Staleness Reduces Converge/iter

Convergence per iter

more iters to converge with more staleness

Convergence (higher is better)

Iters-per-clock is 1

Iterations done

 slack=0 (BSP)

 slack=1 (SSP)

http://www.pdl.cmu.edu/
Staleness Reduces Converge/iter

Convergence per iter

Convergence (higher is better)

Iterations done

Iters-per-clock is 1

slack=0 (BSP)
slack=1 (SSP)
slack=3 (SSP)

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

Convergence (higher is better)

Iterations per sec
- slack=0 (BSP)
- slack=1 (SSP)
- slack=3 (SSP)

Convergence per iter
- slack=0 (BSP)
- slack=1 (SSP)
- slack=3 (SSP)

Time (sec)
- Iterations done
- Convergence (higher is better)

Speed up with a good slack

Iterations per sec
- slack=0 (BSP)
- slack=1 (SSP)
- slack=3 (SSP)

Time (sec)
- Convergence (higher is better)

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

Henggang Cui © July 14
Key Takeaway Insight #1

The sweet spot

Convergence per iteration

Convergence per second

Iterations per second

Fresher data

Staler data
SSP vs A-BSP

- Similar performance
  - In the absence of stragglers

What about environment with stragglers?
Straggler Experiment #1

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

• Experiment setup
  • One disrupter process per machine
    – Uses 50% of CPU cycles
  • Work (disrupt) or sleep randomly for $t$ seconds
    – 10% work, 90% sleep

More straggler experiments in the paper
Straggler Results #1

![Graph showing iteration time increase (%) vs. disruption duration (sec). The graph indicates that without disruption, each iteration takes 4.2 seconds.](image-url)
Straggler Results #1

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

- **Ideal**: Ideally 5%, because 50% slow down with 10% probability.
- **Without disruption**: Each iteration takes 4.2 sec.

---

*Henggang Cui © July 14*

[http://www.pdl.cmu.edu/](http://www.pdl.cmu.edu/)
Straggler Results #1

Disruption duration (sec)

- ideal
- wpc=2, slack=0 (A-BSP)

Ideally 5%, because 50% slow down with 10% probability

w/o disrupt, each iter takes 4.2 sec
Straggler Results #1

SSP tolerates transient stragglers

w/o disrupt, each iter takes 4.2 sec

Ideally 5%, because 50% slow down with 10% probability

---

Iteration time increase (%) vs Disruption duration (sec)

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

SSP tolerates transient stragglers

http://www.pdl.cmu.edu/
Conclusion

• Staleness should be tuned
  • By iters-per-clock and/or slack

• LazyTable implements SSP and A-BSP
  • See paper for details

• Key results from experiments
  • Both SSP and A-BSP are able to exploit the staleness sweet-spot for faster convergence
  • SSP is tolerant of small transient stragglers
  • But SSP incurs more communication traffic
References


• NYTimes: http://archive.ics.uci.edu/ml/datasets/Bag+of+Words

BACK-UP
Example: Topic Modeling

Corpus of documents → Topic modeler → word-topic table ↔ doc-topic table

<table>
<thead>
<tr>
<th>Doc i</th>
<th>Topic 1: 0.8</th>
<th>Topic 2: 0.1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

- \((i, j)\) represents iteration \(i\), work \(uint j\)

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

...
A-BSP Progress and Staleness

- A-BSP, wpc = 2 iterations

Thread

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

Clock

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>barrier</th>
<th>2</th>
<th>barrier</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SSP Progress and Staleness

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

- Same staleness bound as the A-BSP one
  - But more flexible

- Data staleness for SSP with wpc and slack:
  \[ wpc \times (\text{slack} + 1) - 1 \]
SSP VS A-BSP

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

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

SSP tolerates transient stragglers

A-BSP slow down: \( \frac{d}{2} \) per iter

wpc=2, slack=0 (A-BSP)

wpc=1, slack=1 (SSP)
The Cost of Increased Flexibility

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

![Graph showing the cost of increased flexibility]

- smaller \textit{wpc}, larger slack
Key Takeaway Insight #3

• SSP incurs more traffic
  • Finer grained division of clocks
  • Avoiding barriers is still a win, if communication is not the bottleneck
Prefetching

• Conservative prefetch
  • refresh row only when it’s too stale

• Aggressive prefetch
  • refresh row every clock
Prefetching (results)

- $wpc = 1$ iter, slack = 7 clocks