Accelerate Large-Scale Iterative Computation through Asynchronous Accumulative Updates

Yanfeng Zhang, Qixin Gao, Lixin Gao, Cuirong Wang

Northeastern University, China
Univ. of Massachusetts, Amherst
Scientific Computations

Biological systems

Websites connection analysis

Pattern Recognition

Social network analysis

Data clustering
Iterative Algorithms

Pairwise clustering

PageRank

Non-negative Matrix Factorization

Adsorption

K-means
Iterative Algorithms
Challenges

Data is huge

- Single machine cannot process the huge data
- Need extremely long running time

Our work:
A distributed framework that supports Asynchronous Accumulative Updates

2.97 billion web pages (2007)

PageRank: 20-30 of iterations

Iterative app.

Distributed framework

Cloud
Our Approach

• Traditional Approach
  – Concurrent update
  – Synchronous communication
  – Synchronous computation

• Ours
  – Accumulative update
  – Asynchronous communication
  – Asynchronous computation
Traditional Approach: synchronous communication, synchronous computation, concurrent update

\[ R_j^k = d \cdot \sum_{\{i \mid (i \rightarrow j) \in E\}} \frac{R_i^{k-1}}{|N(i)|} + (1 - d) \]
PageRank with Accumulative Updates

\[
\begin{align*}
\Delta R_{j}^{k+1} &= d \cdot \sum_{\{i|(i\to j)\in E\}} \frac{\Delta R_{i}^{k}}{|N(i)|}, \\
R_{j}^{k+1} &= R_{j}^{k} + \Delta R_{j}^{k+1}
\end{align*}
\]
PageRank with Asynchronous Accumulative Updates

- Asynchronous communication
- Asynchronous computation
Asynchronous Accumulative Update

- Asynchronous computation model
  - Remove synchronization barriers
  - Support pipelining and data streaming
What kind of iterative computations can perform asynchronous accumulative update?

Condition:

\[ \Delta_1 + \Delta_2 + \Delta_3 = \Delta \]
Sufficient condition for performing asynchronous accumulative update

1. \[ v_j^k = f_j(v_1^{k-1}, v_2^{k-1}, \ldots, v_n^{k-1}) \]

2. \[ g_{i,j}(x) \] has distributive property over ‘⊕’

Iterative computation with asynchronous accumulative update converge to the same results as that with concurrent update
Algorithm Examples

- A broad class of iterative algorithms can be executed by asynchronous accumulative updates (AAU)

\[ \forall j \quad v_j^k = g_{\{1,j\}}(v_1^{k-1}) \oplus g_{\{2,j\}}(v_2^{k-1}) \oplus \cdots \oplus g_{\{n,j\}}(v_n^{k-1}) \oplus c_j \]

SSSP: \[ d_j^k = \min\{d_1^{k-1} + w(1,j), d_2^{k-1} + w(2,j), \ldots, d_n^{k-1} + w(n,j), d_j^0 \} \]

PageRank: \[ R_j^k = d \cdot \sum_{\{i|(i \rightarrow j) \in E\}} \frac{R_i^{k-1}}{|N(i)|} + (1 - d) \]

- Describe AAU by specifying a 3-tuple

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(c_j)</th>
<th>(g_{{i,j}}(x))</th>
<th>(\oplus)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSSP</td>
<td>0 ((j = s)) or (\infty ((j \neq s)))</td>
<td>(x + w(i,j))</td>
<td>(\min)</td>
</tr>
<tr>
<td>Connected Components</td>
<td>(j)</td>
<td>(x)</td>
<td>(\max)</td>
</tr>
<tr>
<td>PageRank</td>
<td>(1 - d)</td>
<td>(d \cdot \frac{x}{</td>
<td>N(i)</td>
</tr>
<tr>
<td>Adsorption</td>
<td>(p_{i,j}^{\text{inj}} \cdot I_j)</td>
<td>(p_{i}^{\text{cont}} \cdot W(i,j) \cdot x)</td>
<td>(\pm)</td>
</tr>
<tr>
<td>HITS (authority)</td>
<td>1</td>
<td>(d \cdot A^T A(i,j) \cdot x)</td>
<td>(\pm)</td>
</tr>
<tr>
<td>Katz metric</td>
<td>1 ((j = s)) or 0 ((j \neq s))</td>
<td>(\beta \cdot x)</td>
<td>(\pm)</td>
</tr>
<tr>
<td>Jacobi method</td>
<td>(\frac{b_j}{a_{jj}})</td>
<td>(-\frac{a_{ji}}{a_{jj}} \cdot x)</td>
<td>(\pm)</td>
</tr>
<tr>
<td>Rooted PageRank</td>
<td>1 ((j = s)) or 0 ((j \neq s))</td>
<td>(P(j,i) \cdot x)</td>
<td>(\pm)</td>
</tr>
</tbody>
</table>
Maiter

• Build Maiter by modifying Piccolo

  R. Power, J. Li. Piccolo: Building Fast, Distributed Programs with Partitioned Tables, in Proc. OSDI 2010

• Master – Workers Structure
  – Master: respond to user commands, control iteration termination
  – Worker: process a partition of data elements
  – Workers communicate between each other using MPI
Maiter Worker

- In-memory state table
  - k: element id
  - v and Δv: accumulated values
  - data: element related data

- Two threads
  - Receive thread
  - Update thread

- Flexible schedule in update thread
  - Round robin schedule (Maiter-RR)
  - Priority schedule (Maiter-Pri)
  - Synchronous schedule (Maiter-Sync)
Related Work

- **MapReduce**
- **Coarse-grained update**
  - Dryad (Microsoft, [EuroSys ‘07])
  - HaLoop (Univ. of Washington, [VLDB ‘10])
  - CIEL (Cambridge [NSDI ‘11])
- **Fine-grained update**
  - Pregel (Google, [SIGMOD ‘10])
  - Spark (Berkeley, [HotCloud ‘10, NSDI ‘12])
  - Piccolo (NYU, [OSDI ‘10])
  - Twister (Indiana Univ., [MAPREDUCE ‘10])
- **Our previous work:**
  - iMapReduce ([DataCloud ‘11])
  - PrIter ([SOCC ‘11])
- **GraphLab** (CMU, [UAI ‘10, VLDB ‘12])
Evaluation

- 100 EC2 medium instances
- Algorithms: SSSP, PageRank, Adsorption, Katz metric
- Data set: 100-million-node graph
- Compared frameworks:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>iMR</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>iMR-mem</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>PrIter</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Piccolo</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Maiter-Sync</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Maiter-RR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Maiter-Pri</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Convergence Time: SSSP
Convergence Time: PageRank
Convergence Time: Adsorption & Katz

**Adsorption**

- **Maiter-Sync**
- **Maiter-RR**
- **Maiter-Pri**

**Katz metric**

- **Maiter-Sync**
- **Maiter-RR**
- **Maiter-Pri**
Communication Cost

- Application: PageRank
- Metric: Transferred volume (GB)
Conclusion & Future Work

• By asynchronous accumulative update, Maiter significantly reduces running time
  – Identify a broad class of scientific computing algorithms that can be performed by asynchronous accumulative updates
  – Build Maiter to support running asynchronous accumulative updates in the cloud

• Future work:
  – Fault tolerance mechanism
  – More general framework supporting more algorithms
  – Release version of Maiter
• Thanks!
### Backup Slides: Maiter API

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$c_j$</th>
<th>$g_{i,j}(x)$</th>
<th>$\oplus$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSSP</td>
<td>0 ($j = s$) or $\infty$ ($j \neq s$)</td>
<td>$x + w(i,j)$</td>
<td>min</td>
<td>$+\infty$</td>
</tr>
<tr>
<td>Connected Components</td>
<td>$j$</td>
<td>$x$</td>
<td>max</td>
<td>$-\infty$</td>
</tr>
<tr>
<td>PageRank</td>
<td>$1 - d$</td>
<td>$d \cdot \frac{x}{N(j)}$</td>
<td>$+$</td>
<td>0</td>
</tr>
<tr>
<td>Adsorption</td>
<td>$p_j^{inj} \cdot I_j$</td>
<td>$p_i^{cont} \cdot W(i,j) \cdot x$</td>
<td>$+$</td>
<td>0</td>
</tr>
<tr>
<td>HITS (authority)</td>
<td>1</td>
<td>$d \cdot A^T A(i,j) \cdot x$</td>
<td>$+$</td>
<td>0</td>
</tr>
<tr>
<td>Katz metric</td>
<td>1 ($j = s$) or 0 ($j \neq s$)</td>
<td>$\beta \cdot x$</td>
<td>$+$</td>
<td>0</td>
</tr>
<tr>
<td>Jacobi method</td>
<td>$\frac{b_j}{\alpha_{jj}}$</td>
<td>$-\frac{\alpha_{ji}}{\alpha_{jj}} \cdot x$</td>
<td>$+$</td>
<td>0</td>
</tr>
<tr>
<td>Rooted PageRank</td>
<td>1 ($j = s$) or 0 ($j \neq s$)</td>
<td>$P(j,i) \cdot x$</td>
<td>$+$</td>
<td>0</td>
</tr>
</tbody>
</table>

```cpp
def virtual void accumulate(const V* a, const V& b) { return 0; }
def virtual void g_func(const V& delta, const D& data, list<pair<K, V> >* output) { return 0; }
```

```cpp
template <class K, class V>
struct TermChecker {
    virtual double estimate_prog(LocalTableIterator<K, V>* table_itr) { return 0; }
    virtual bool terminate(list<double> local_reports) { return 0; }
};
```
Backup Slides: Accumulative Computation Model

- Each processor:
  - Accumulate the received messages
  - Process the accumulated messages and propagate the processed messages

receive:
\[
\begin{align*}
\text{Whenever a value } m_j \text{ is received,} \\
\Delta \tilde{v}_j & \leftarrow \Delta \tilde{v}_j \oplus m_j. \\
\end{align*}
\]

update:
\[
\begin{align*}
\tilde{v}_j & \leftarrow \tilde{v}_j \oplus \Delta \tilde{v}_j; \\
\text{For any } h, \text{ if } g_{\{j,n\}}(\Delta \tilde{v}_j) \neq 0, \\
& \text{send value } g_{\{j,n\}}(\Delta \tilde{v}_j) \text{ to processor } h; \\
\Delta \tilde{v}_j & \leftarrow 0,
\end{align*}
\]

Prioritized Scheduling
Backup Slides: Efficiency of Accumulative Updates

• Progress metric:
  • SSSP: sum(d_j); smaller and smaller
  • PageRank: sum(R_j); bigger and bigger