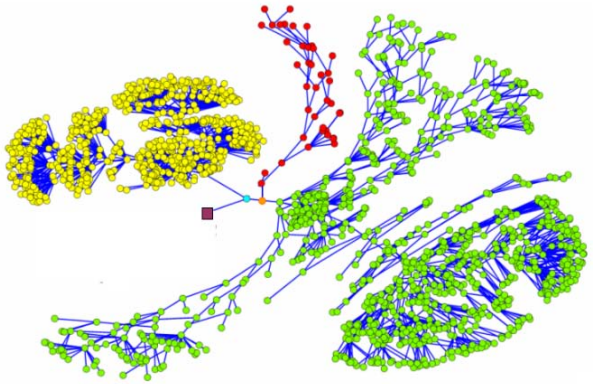
Accelerate Large-Scale Iterative Computation through Asynchronous Accumulative Updates

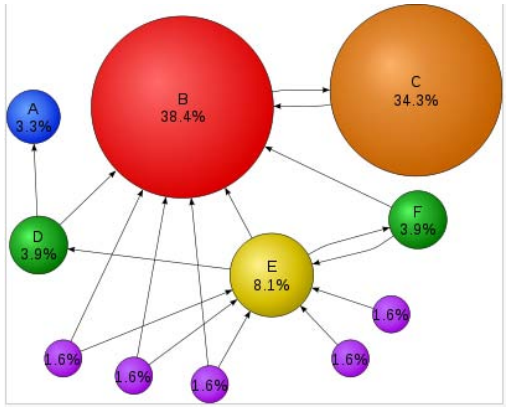
Yanfeng Zhang, QixinGao, Lixin Gao, Cuirong Wang

Northeastern University, China
Univ. of Masschusetts, Amherst

Scientific Computations

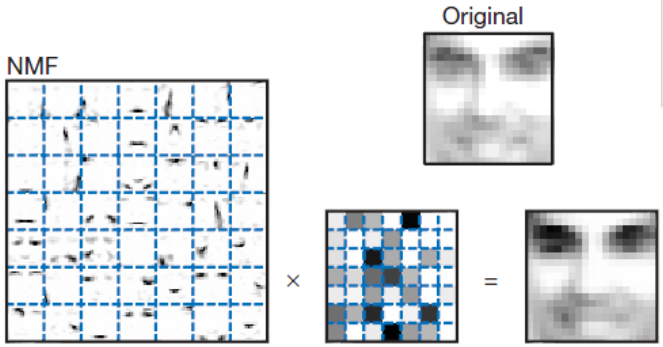


Biological systems

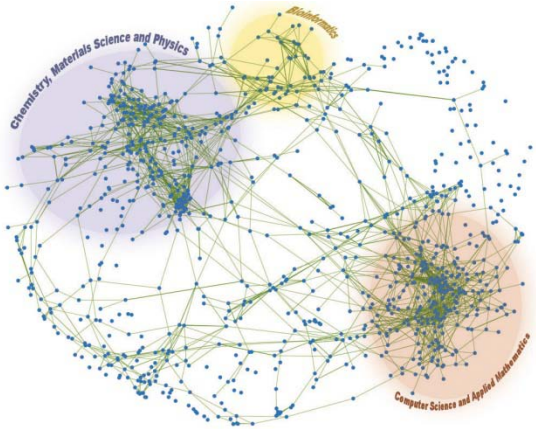


Websites connection analysis

Social network analysis

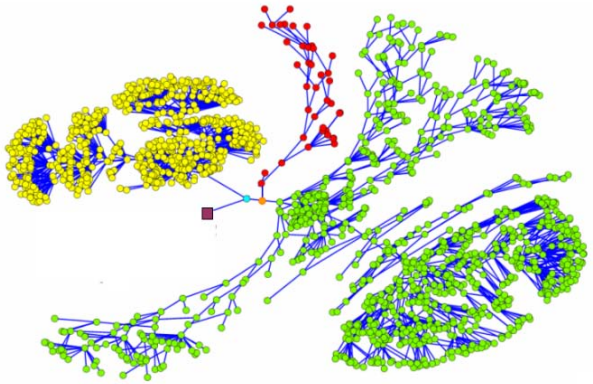


Pattern Recognition

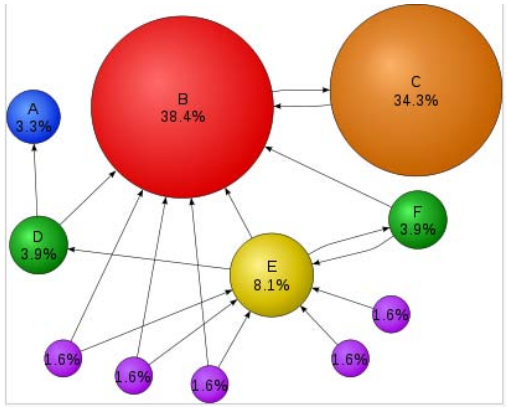


Data clustering

Iterative Algorithms

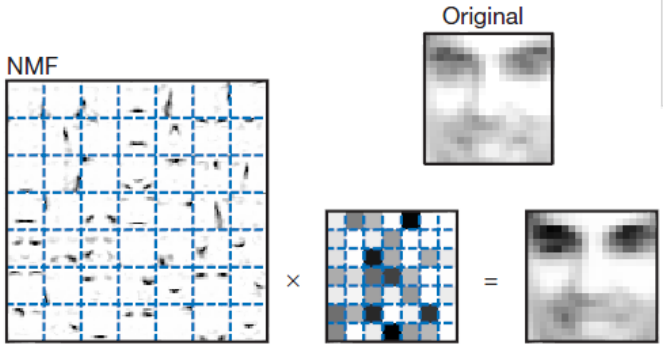


Pairwise clustering

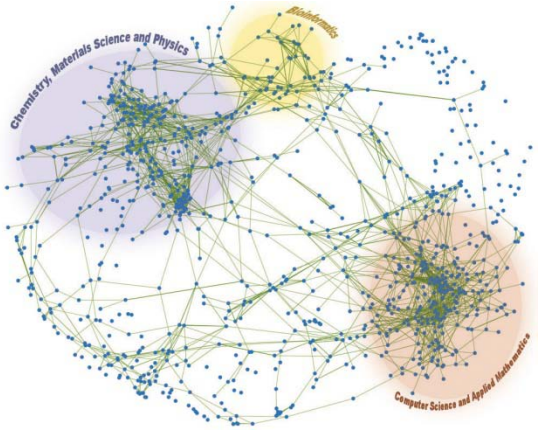


PageRank

Adsorption

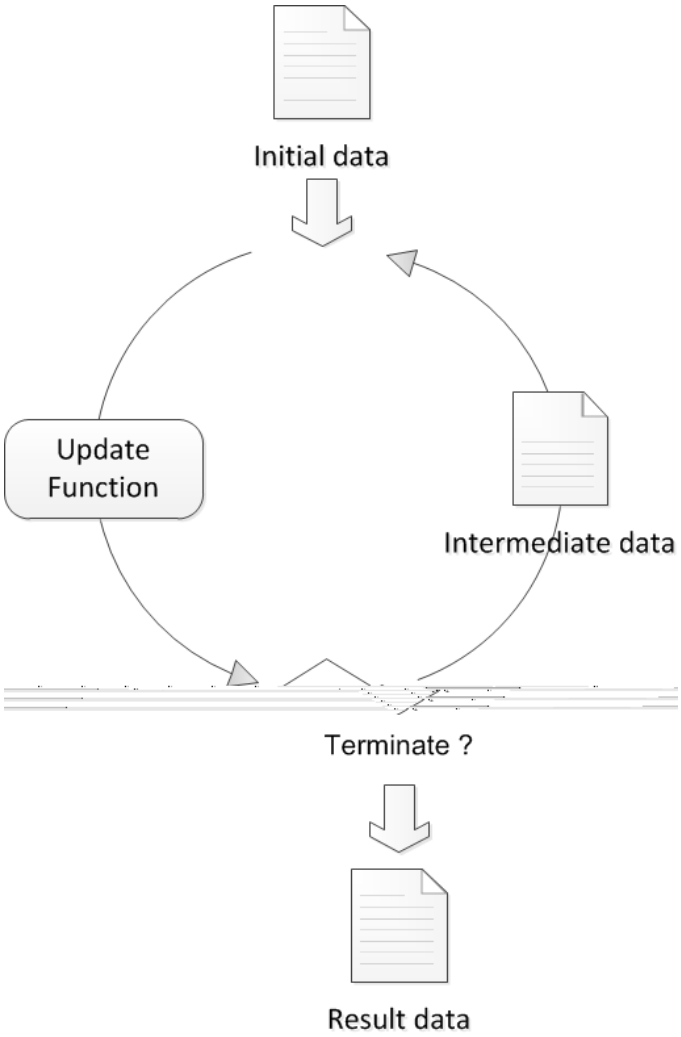


Non-negative Matrix Factorization



K-means

Iterative Algorithms

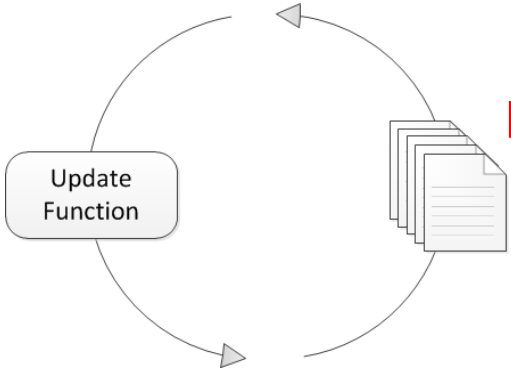


Challenges

2.97 billion web pages (2007)

Data is huge

+

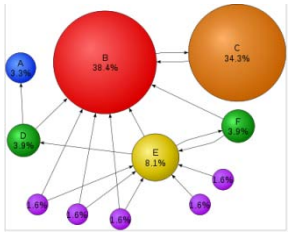


PageRank: 20-30 of iterations

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

Our work:

A distributed framework that supports Asynchronous Accumulative Updates



Iterative app.

Distributed framework

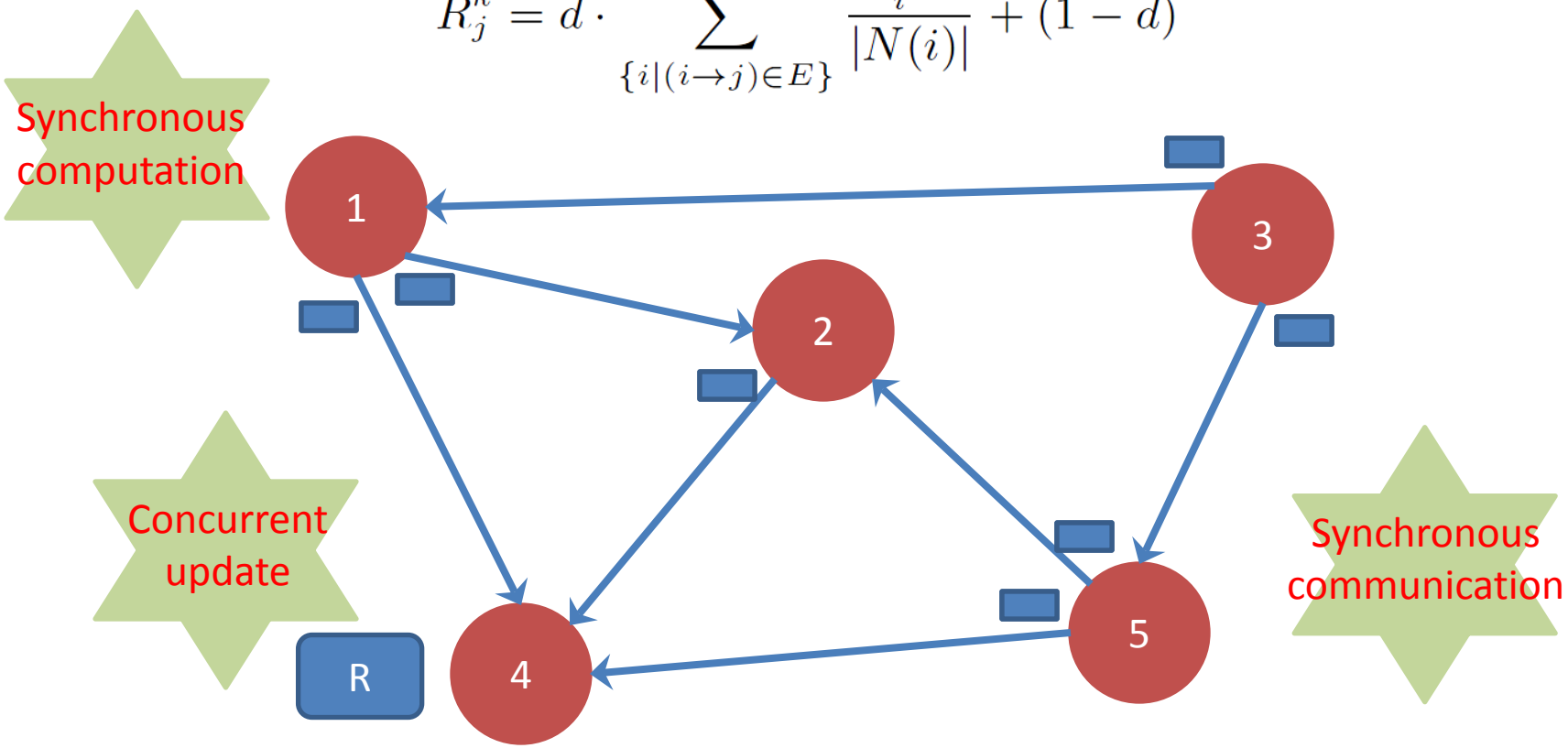
Cloud

Our Approach

- Traditional Approach
  - Concurrent update
  - Synchronous communication
  - Synchronous computation
- Ours
  - Accumulative update
  - Asynchronouscommunication
  - Asynchronous computation

PageRank with Traditional Method

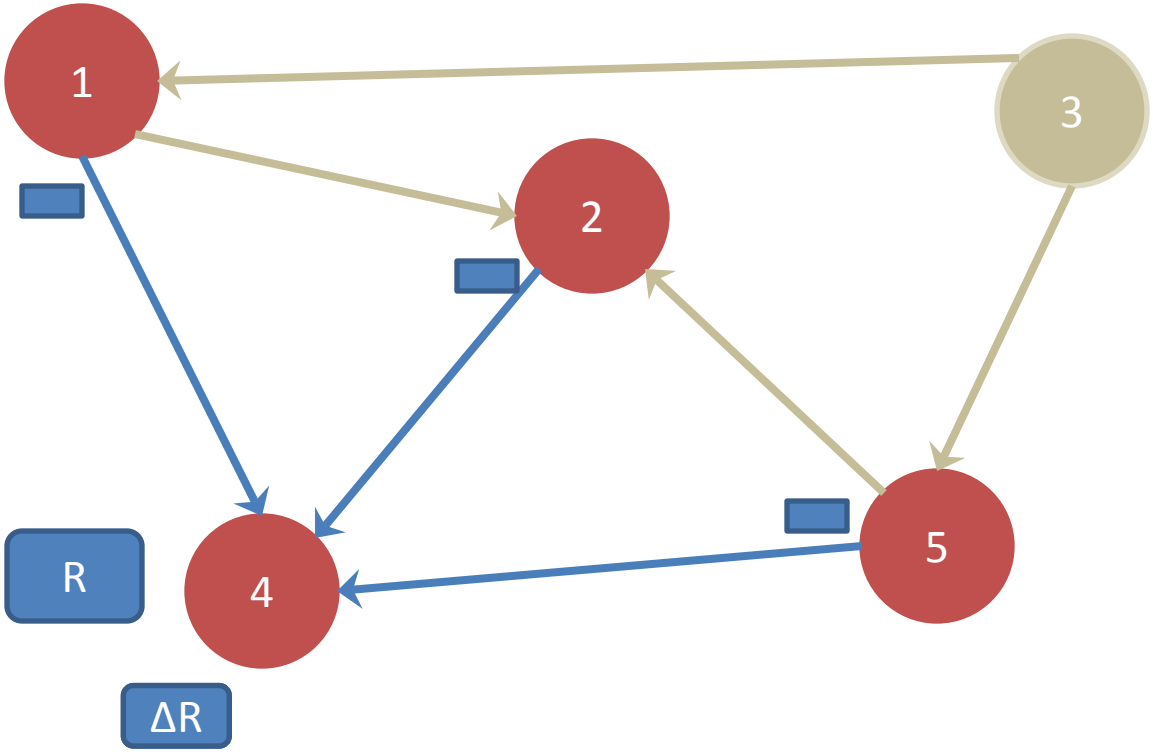
$$R_j^k = d \cdot \sum_{\{i|(i \to j) \in E\}} \frac{R_i^{k-1}}{|N(i)|} + (1-d)$$



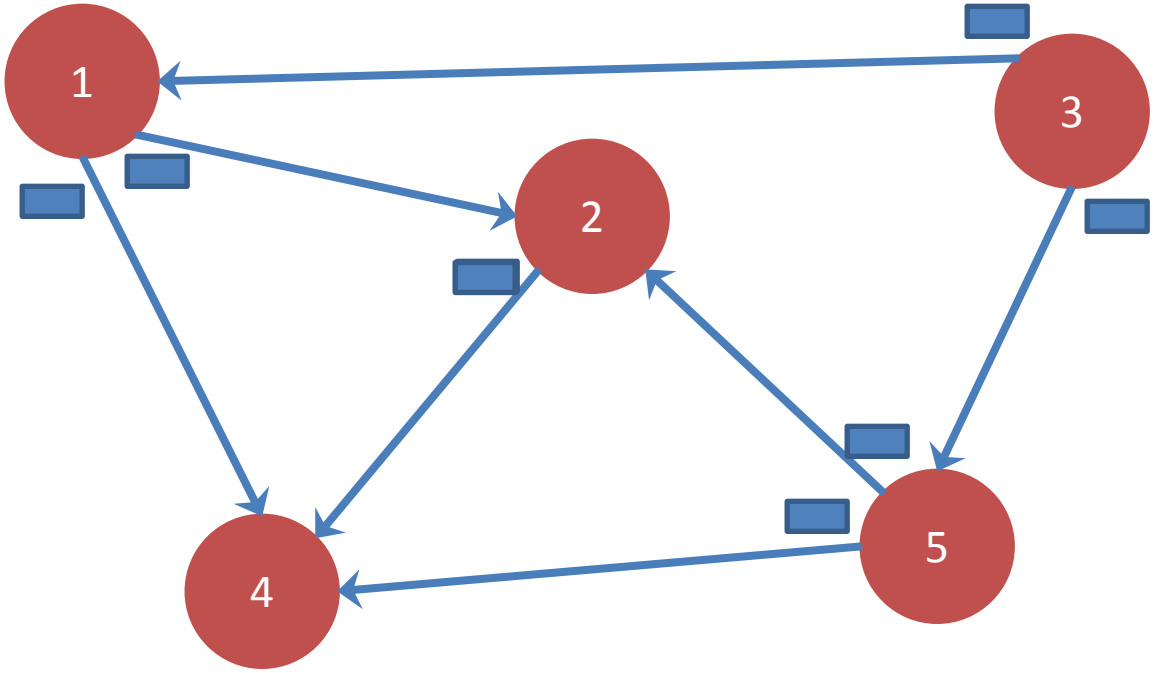
Traditional Approach: synchronous communication, synchronous computation, concurrent update

PageRank with Accumulative Updates

$$\begin{cases} \Delta R_j^{k+1} = d \cdot \sum_{\{i|(i\to j)\in E\}} \frac{\Delta R_i^k}{|N(i)|}, \\ R_j^{k+1} = R_j^k + \Delta R_j^{k+1} \end{cases}$$



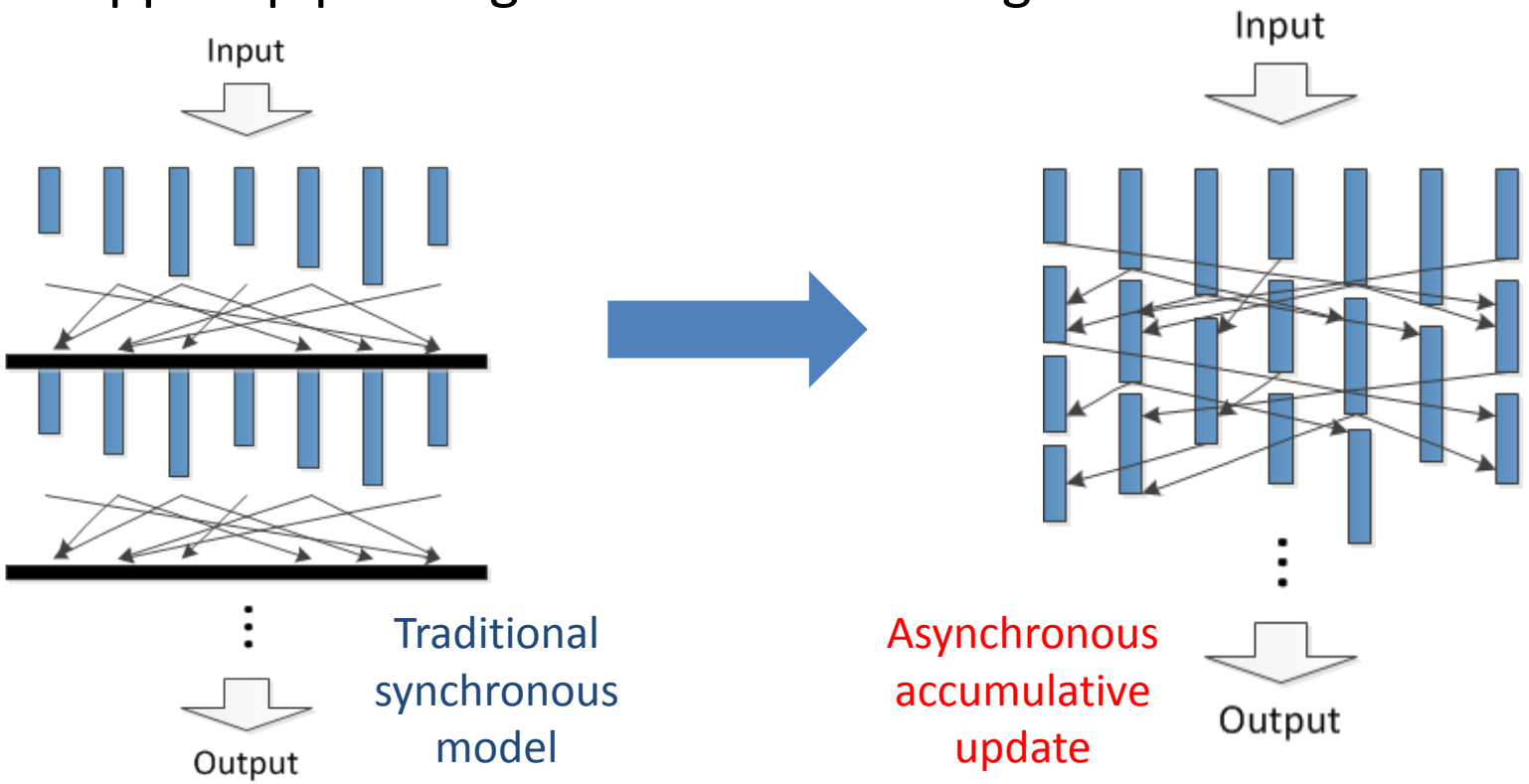
PageRank with Asynchronous Accumulative Updates



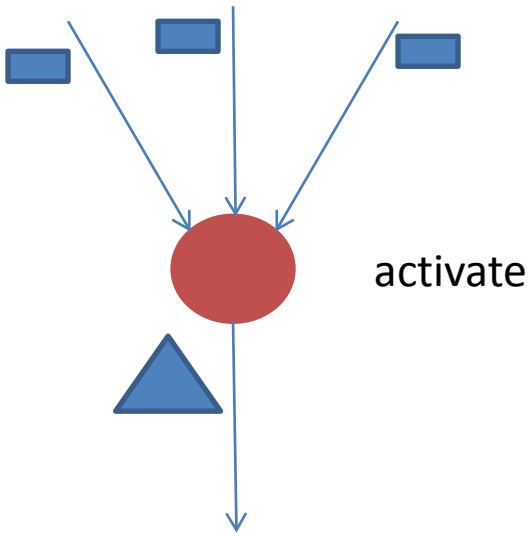
- Asynchronous communication
- Asynchronous computation

Asynchronous Accumulative Update

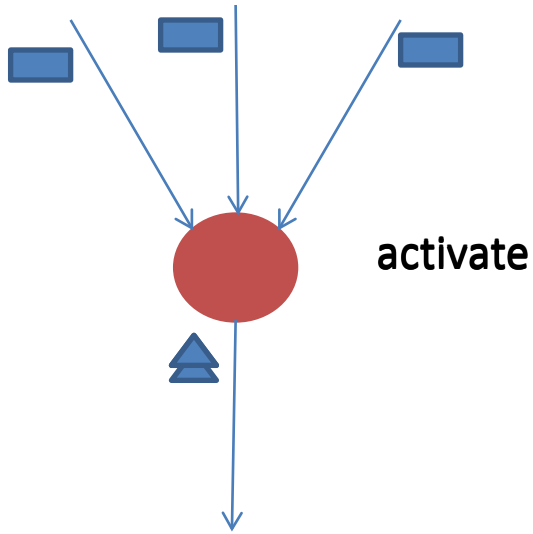
- Asynchronous computation model
  - Remove synchronization barriers
  - Support pipelining and data streaming



What kind of iterative computations can perform asynchronous accumulative update?



Δ



Δ 1

Δ 2

Δ 3

Condition: Δ = Δ 1 + Δ 2 + Δ 3

Sufficient condition for performing asynchronous accumulative update

1

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

$$v_j^k = g_{\{1,j\}}(v_1^{k-1}) \oplus g_{\{2,j\}}(v_2^{k-1}) \oplus \ldots \oplus g_{\{n,j\}}(v_n^{k-1}) \oplus c_j$$

2 $g_{\{i,j\}}(x)$ has distributive property over '$\oplus$'

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)

$$v_j^k = g_{\{1,j\}}(v_1^{k-1}) \oplus g_{\{2,j\}}(v_2^{k-1}) \oplus \ldots \oplus g_{\{n,j\}}(v_n^{k-1}) \oplus c_j$$

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

- Describe AAU by specifying a 3-tuple

algorithm	c_j	$g_{\{i,j\}}(x)$	$\oplus$
SSSP	$0\ (j=s)$ or $\infty\ (j \neq s)$	$x + w(i,j)$	min
Connected Components	j	x	max
PageRank	$1-d$	$d \cdot \frac{x}{\lvert N(j)\rvert}$	+
Adsorption	$p_j^{inj} \cdot I_j$	$p_i^{cont} \cdot W(i,j) \cdot x$	+
HITS (*authority*)	1	$d \cdot A^T A(i,j) \cdot x$	+
Katz metric	$1\ (j=s)$ or $0\ (j \neq s)$	$\beta \cdot x$	+
Jacobi method	$\frac{b_j}{a_{jj}}$	$-\frac{a_{ji}}{a_{jj}} \cdot x$	+
Rooted PageRank	$1\ (j=s)$ or $0\ (j \neq s)$	$P(j,i) \cdot x$	+

Maiter

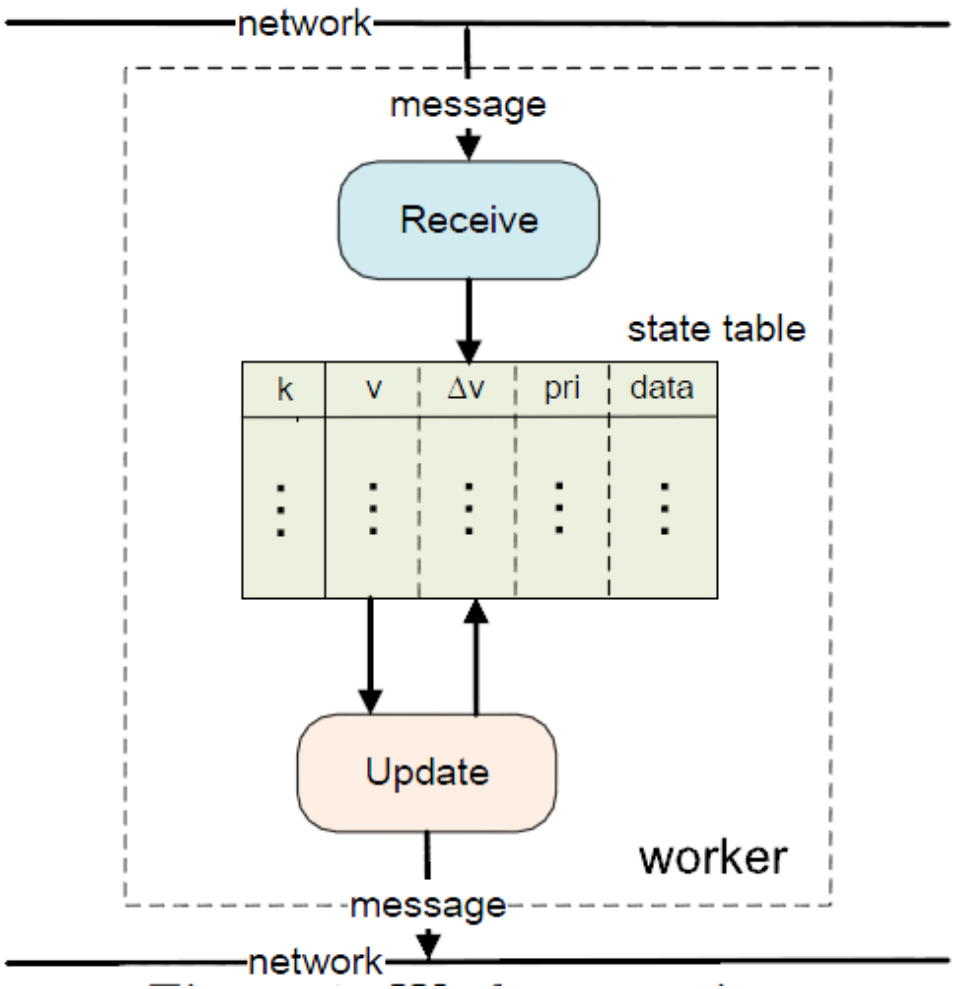
- Build Maiter by modifying Piccolo

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 (Cambrige [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])

Synchronous model

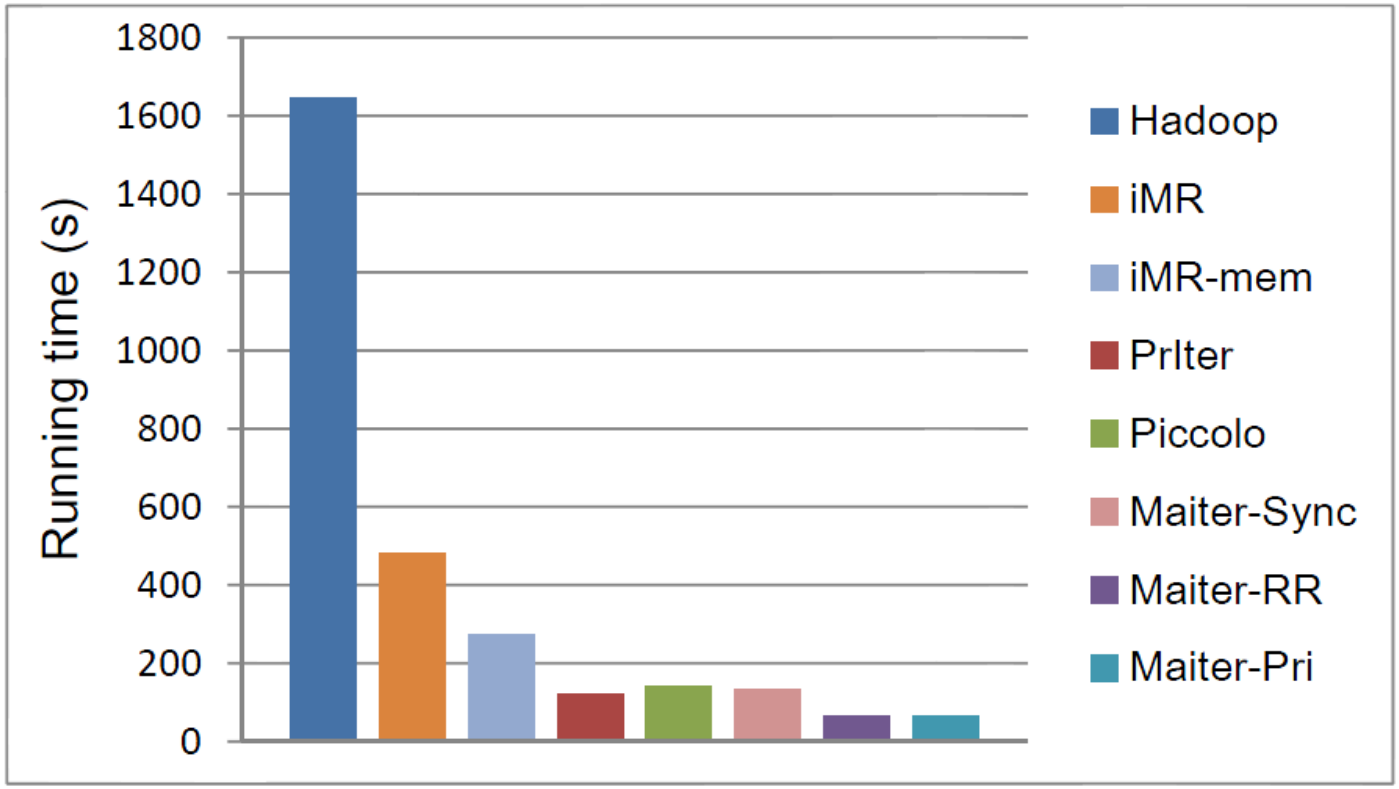
- GraphLab (CMU, [UAI '10, VLDB '12])

Evaluation

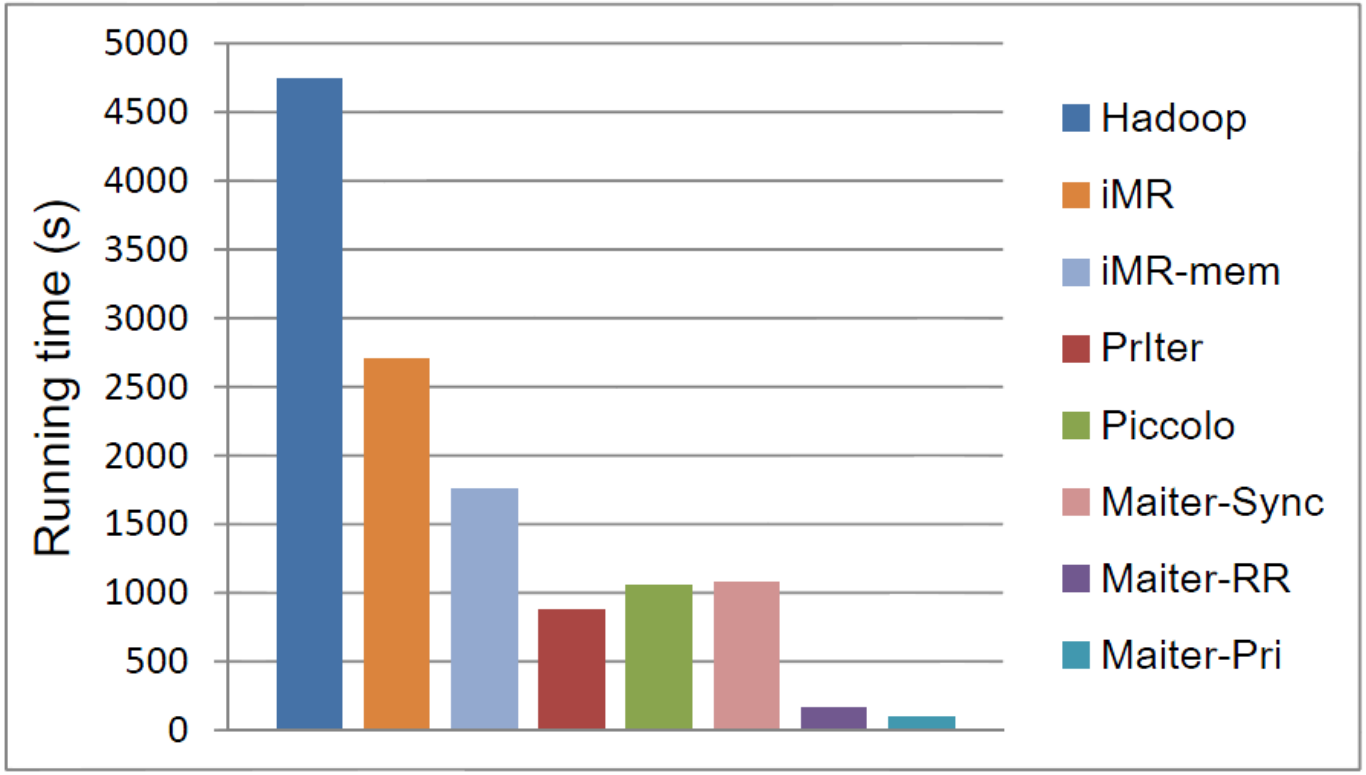
- 100 EC2 medium instances
- Algorithms: SSSP, PageRank, Adsorption, Katz metric
- Data set: 100-million-node graph
- Compared frameworks:

	Sep. Data	In Mem	Async. Comm.	Async. Comp.	Opt. Sched.
Hadoop	×	×	×	×	×
iMR	√	×	×	×	×
iMR-mem	√	√	×	×	×
PrIter	√	√	×	×	√
Piccolo	√	√	√	×	×
Maiter-Sync	√	√	×	×	×
Miater-RR	√	√	√	√	×
Maiter-Pri	√	√	√	√	√

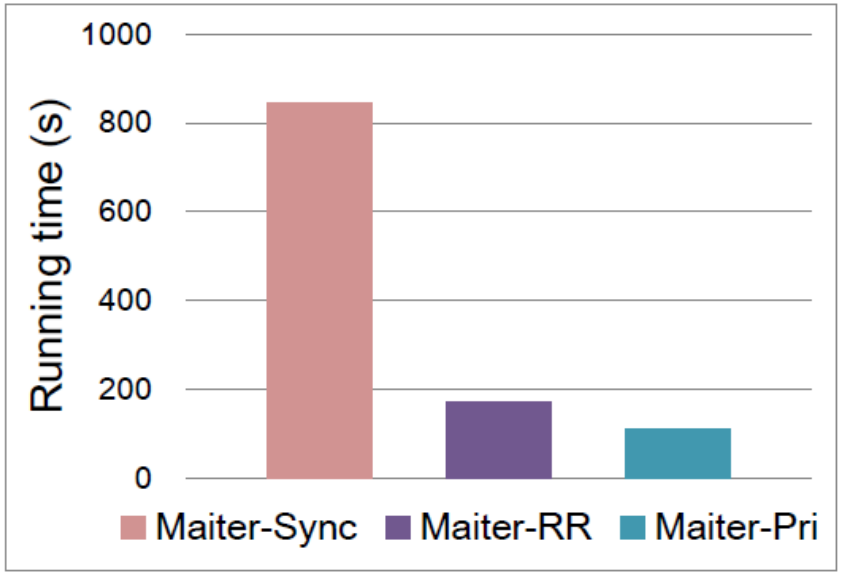
Convergence Time: SSSP



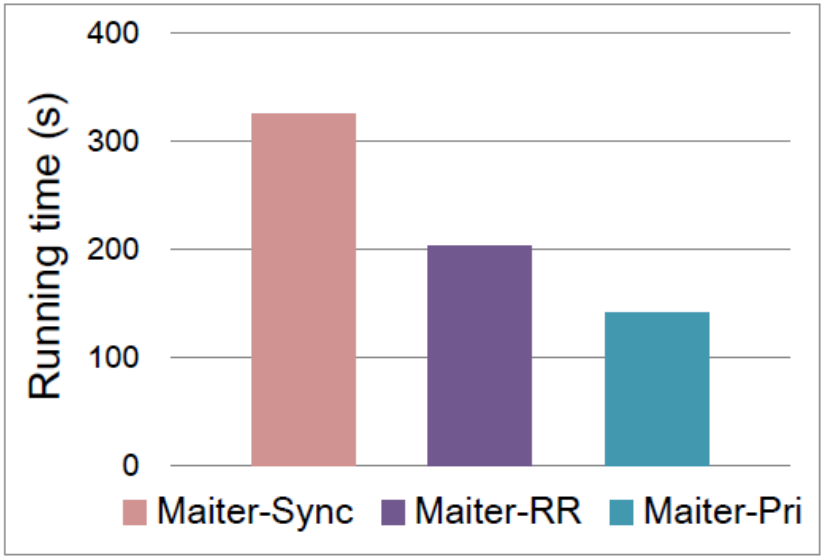
Convergence Time: PageRank



Convergence Time: Adsorption & Katz



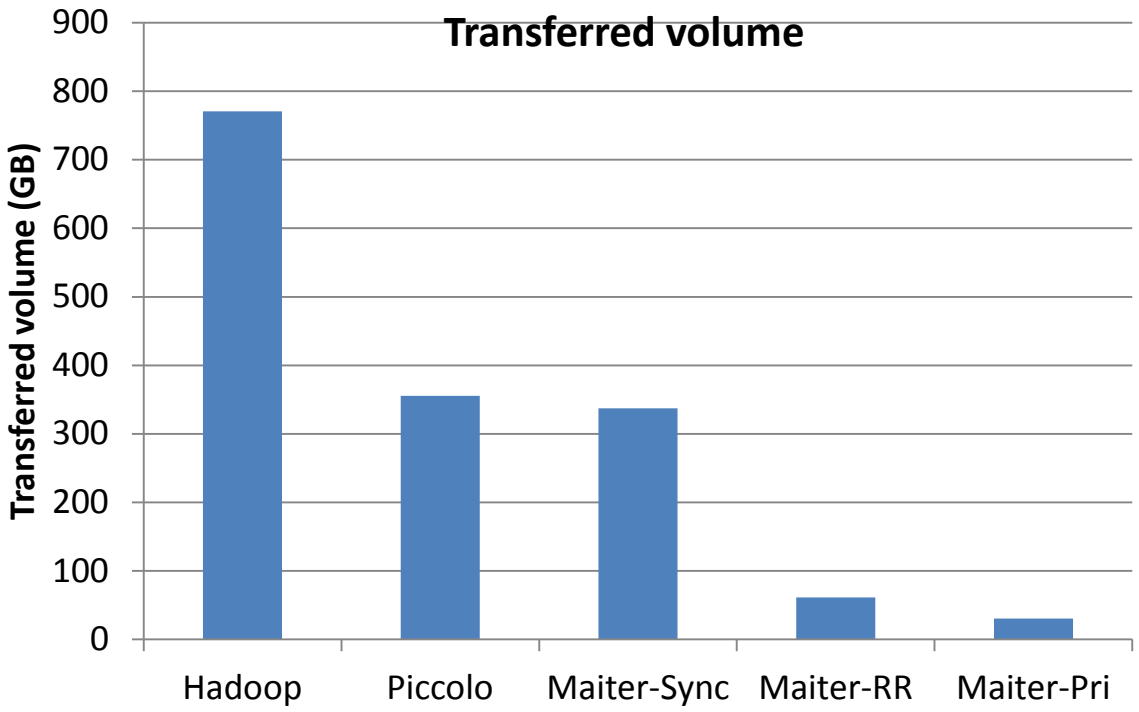
Adsorption



Katz metric

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

Q & A

- Thanks!

Backup Slides: Maiter API

algorithm	c_j	$g_{\{i,j\}}(x)$	$\oplus$	$\mathbf{0}$
SSSP	$0\ (j=s)$ or $\infty\ (j\neq s)$	$x+w(i,j)$	min	$+\infty$
Connected Components	j	x	max	$-\infty$
PageRank	$1-d$	$d\cdot\frac{x}{\lvert N(j)\rvert}$	$+$	0
Adsorption	$p_j^{inj}\cdot I_j$	$p_i^{cont}\cdot W(i,j)\cdot x$	$+$	0
HITS (*authority*)	1	$d\cdot A^T A(i,j)\cdot x$	$+$	0
Katz metric	$1\ (j=s)$ or $0\ (j\neq s)$	$\beta\cdot x$	$+$	0
Jacobi method	$\frac{b_j}{a_{jj}}$	$-\frac{a_{ji}}{a_{jj}}\cdot x$	$+$	0
Rooted PageRank	$1\ (j=s)$ or $0\ (j\neq s)$	$P(j,i)\cdot x$	$+$	0

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

template <class K, class V>
struct TermChecker {
    virtual double estimate_prog(LocalTableIterator<K, V>*
                                 table_itr) = 0;
    virtual bool terminate(list<double> local_reports) = 0;
};
```

Backup Slides: Accumulative Computation Model

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

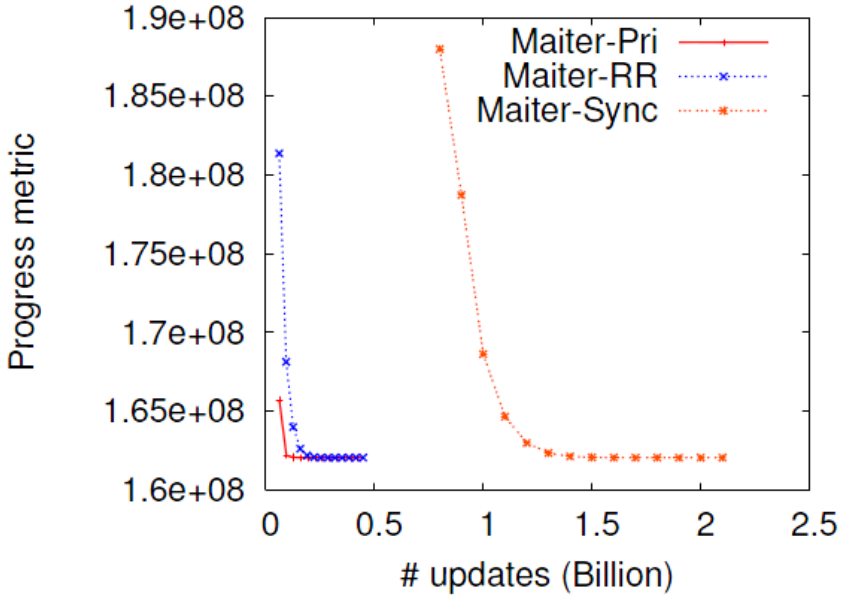
$$\texttt{receive:}\begin{cases}\texttt{Whenever a value } m_j \texttt{ is received,}\\ \quad \Delta\check{v}_j \leftarrow \Delta\check{v}_j \oplus m_j.\end{cases}$$

$$\texttt{update:}\begin{cases}\check{v}_j \leftarrow \check{v}_j \oplus \Delta\check{v}_j;\\ \texttt{For any } h\texttt{, if } g_{\{j,h\}}(\Delta\check{v}_j) \neq \mathbf{0},\\ \quad \texttt{send value } g_{\{j,h\}}(\Delta\check{v}_j) \texttt{ to processor } h;\\ \Delta\check{v}_j \leftarrow \mathbf{0},\end{cases}$$

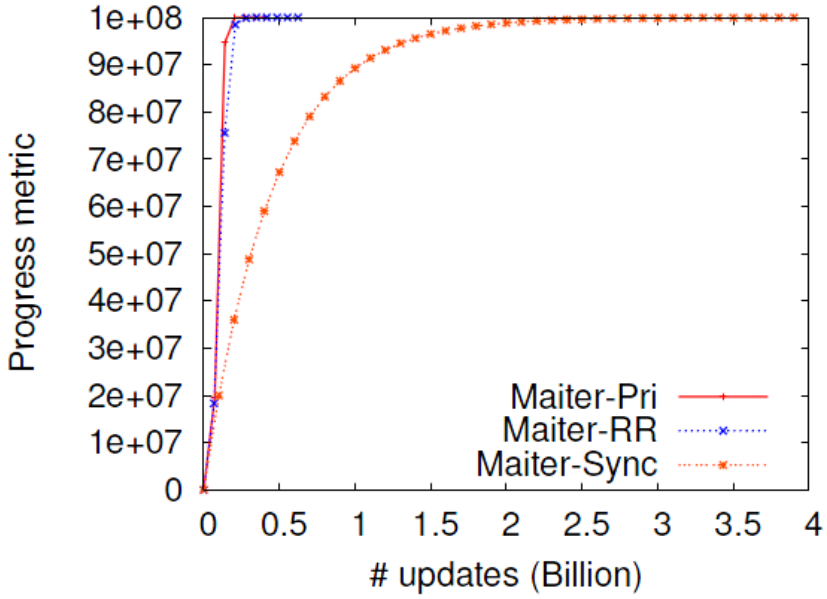
Prioritized Scheduling

Backup Slides: Efficiency of Accumulative Updates

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



SSSP



PageRank