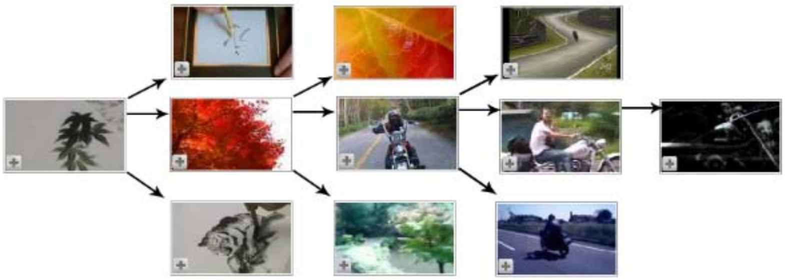
PrIter: A Distributed Framework for Prioritized Iterative Computations

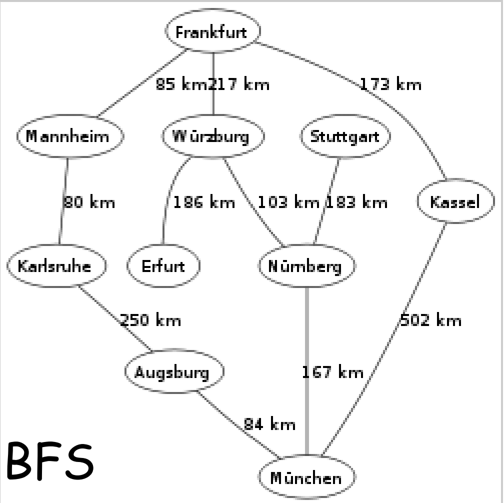
Yanfeng Zhang, Qixin Gao, Lixin Gao, Cuirong Wang

Northeastern University, China
UMass Amherst

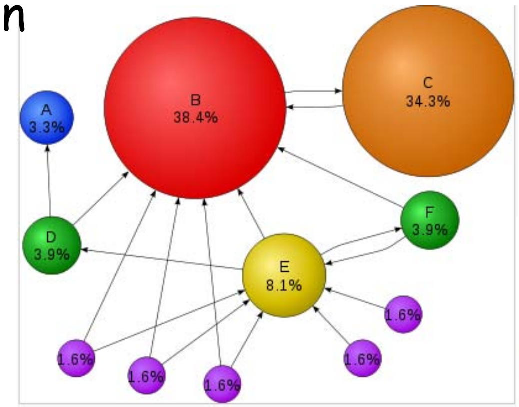
Iterative Computation is Everywhere



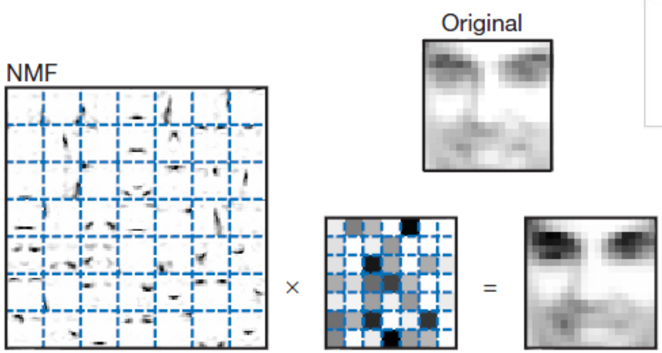
Youtube video suggestion



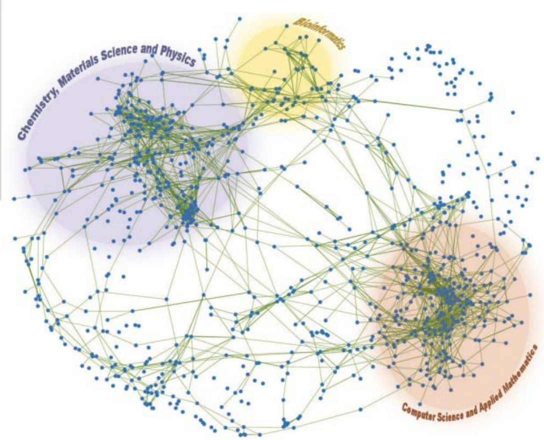
BFS



PageRank

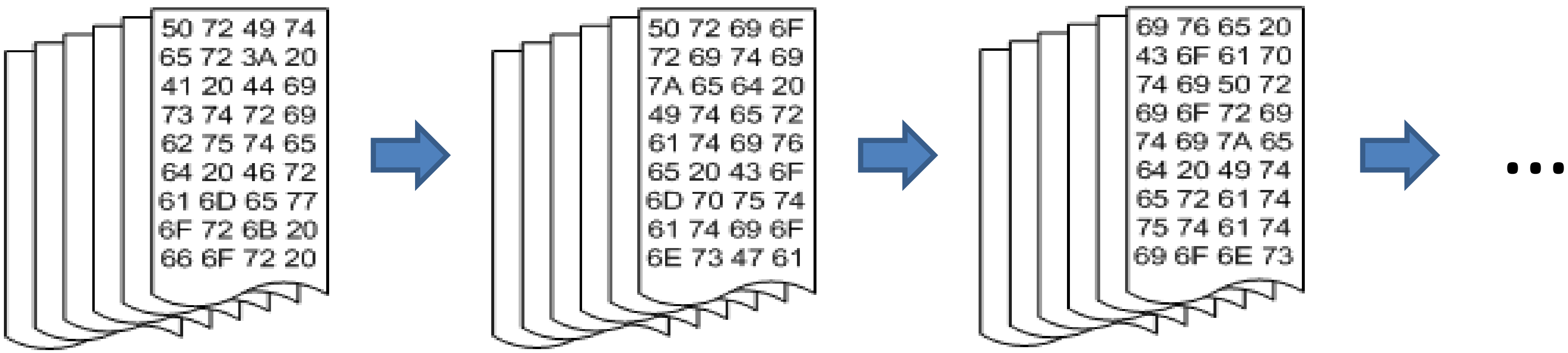


Pattern Recognition



Clustering

Large-Scale Iterative Computation



\+ Involve massive data sets

++ Process massive data iteratively

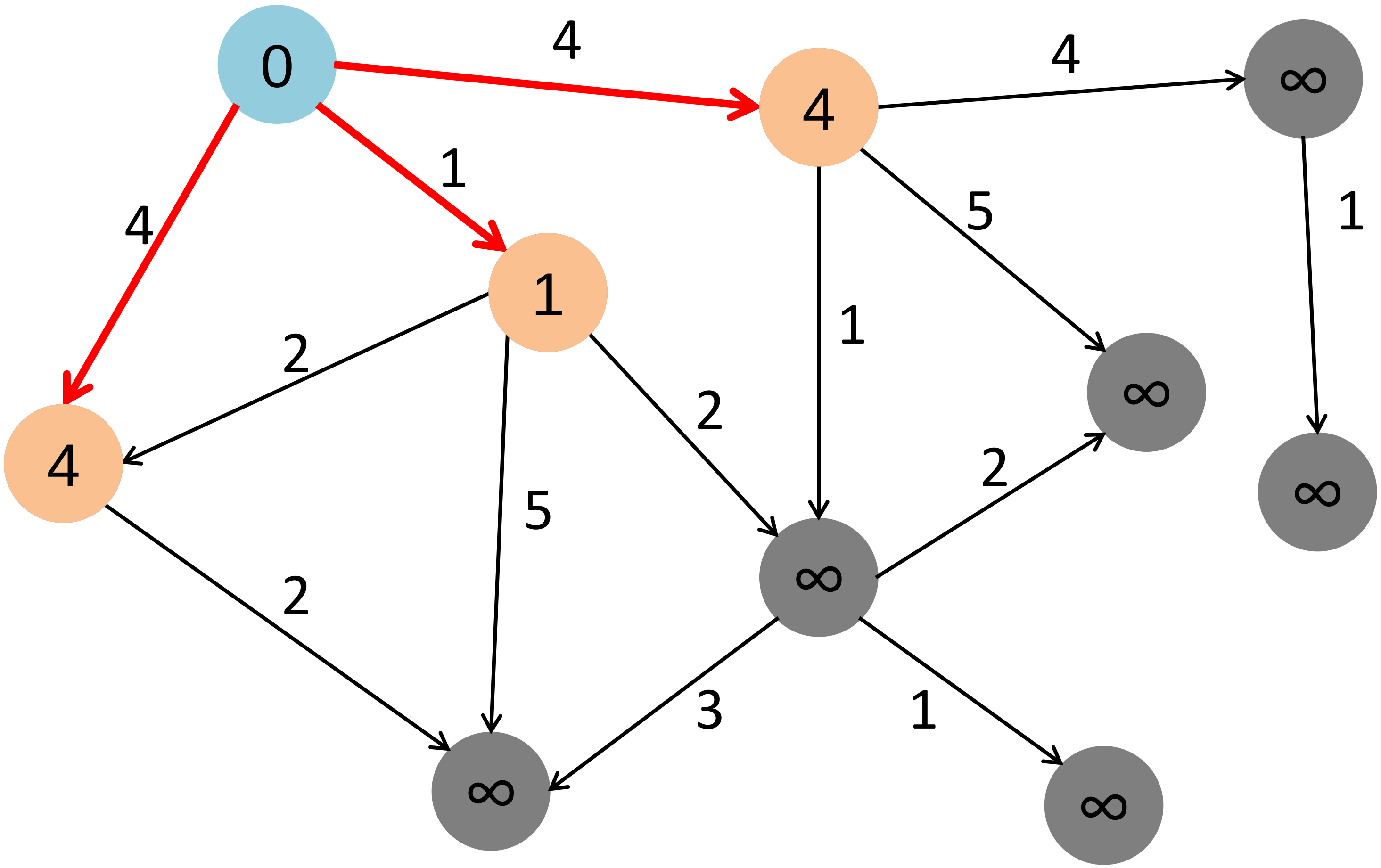
+++ Converge fast

Our Goal: a framework supporting iterative computation running in the cloud

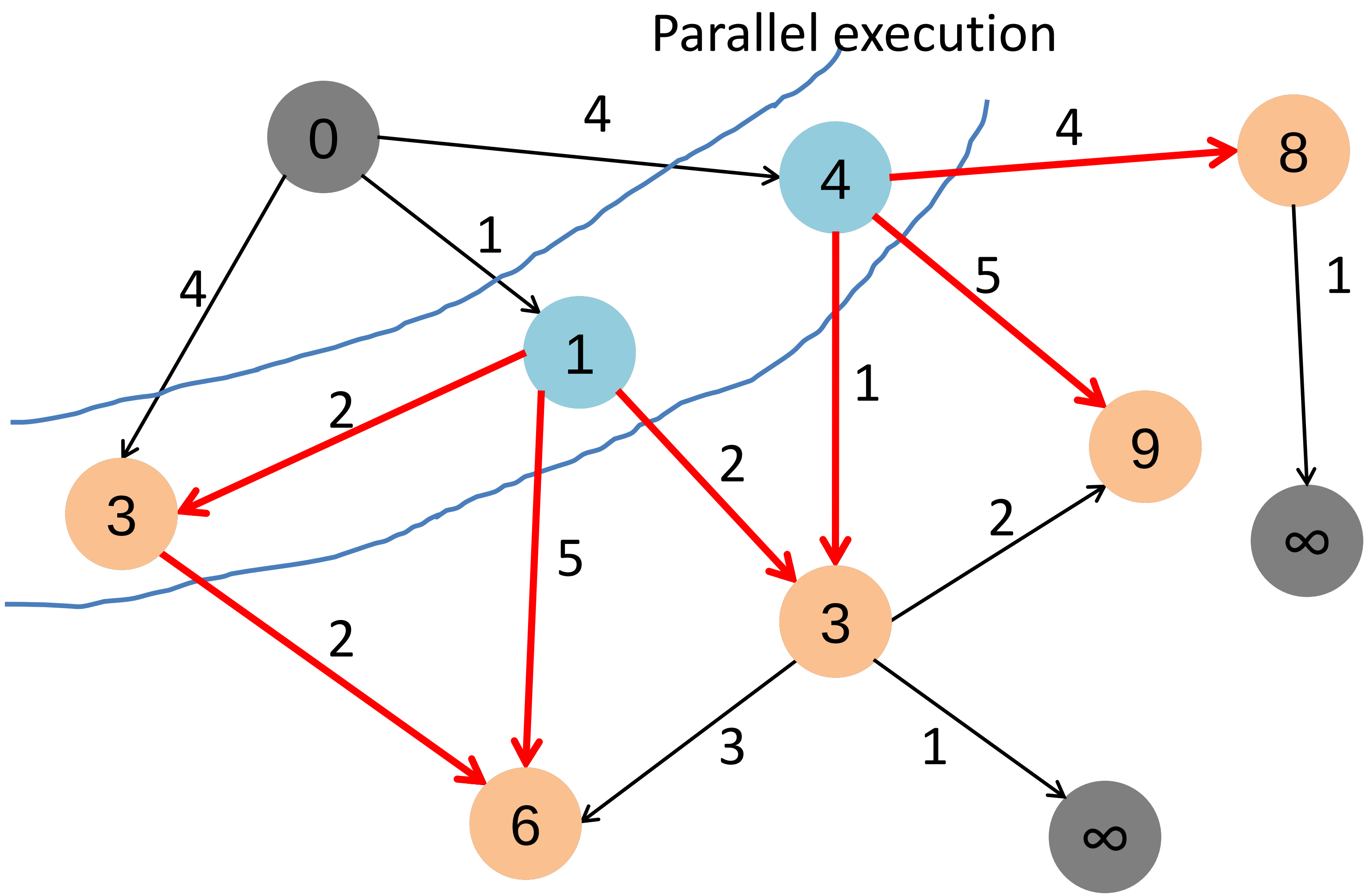
Outline

- Prioritized Iteration
- PrIter's Design & Implementation
- Evaluation
- Conclusions

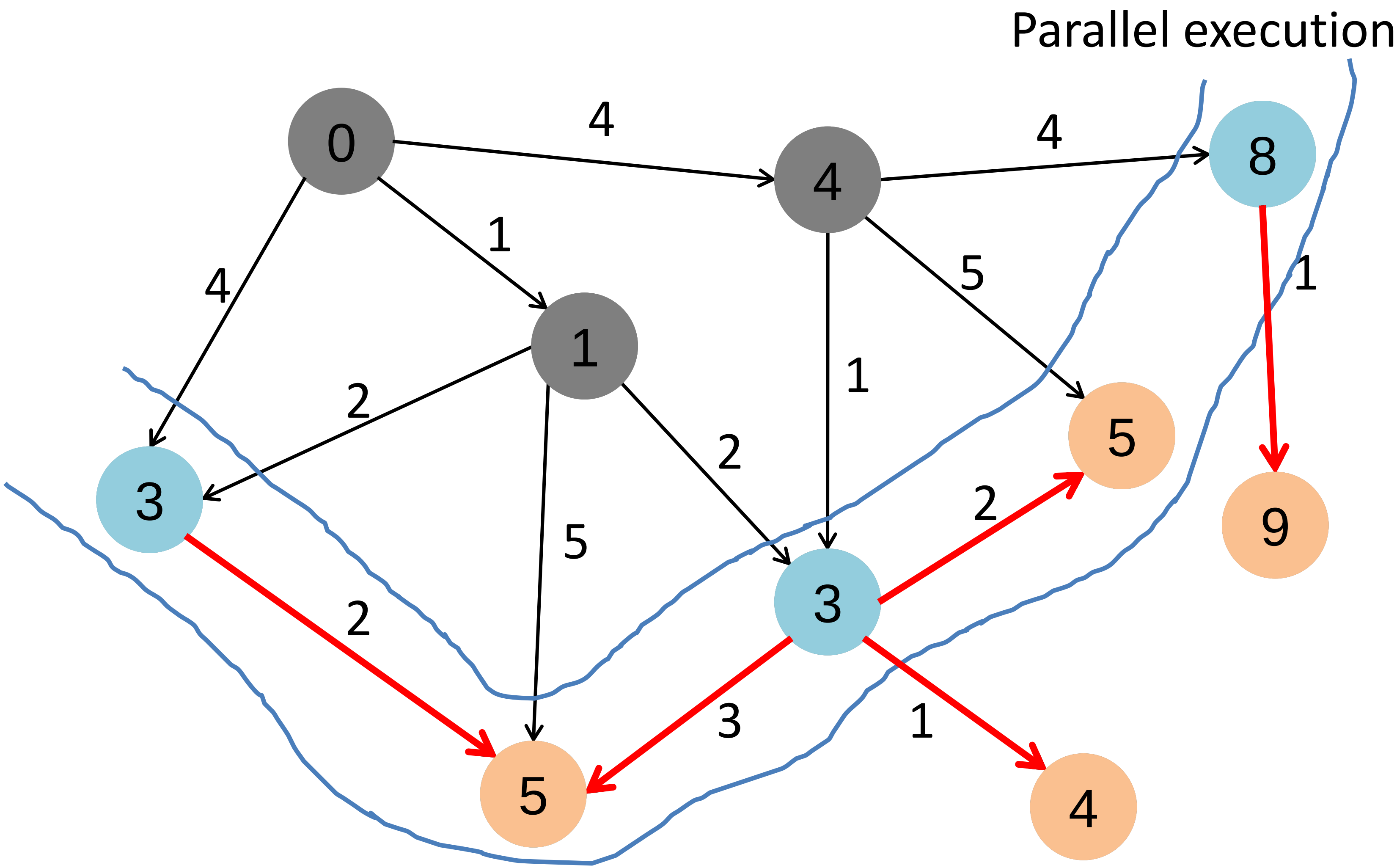
Parallel Parallel Single Source Shortest Path (SSSP): 1



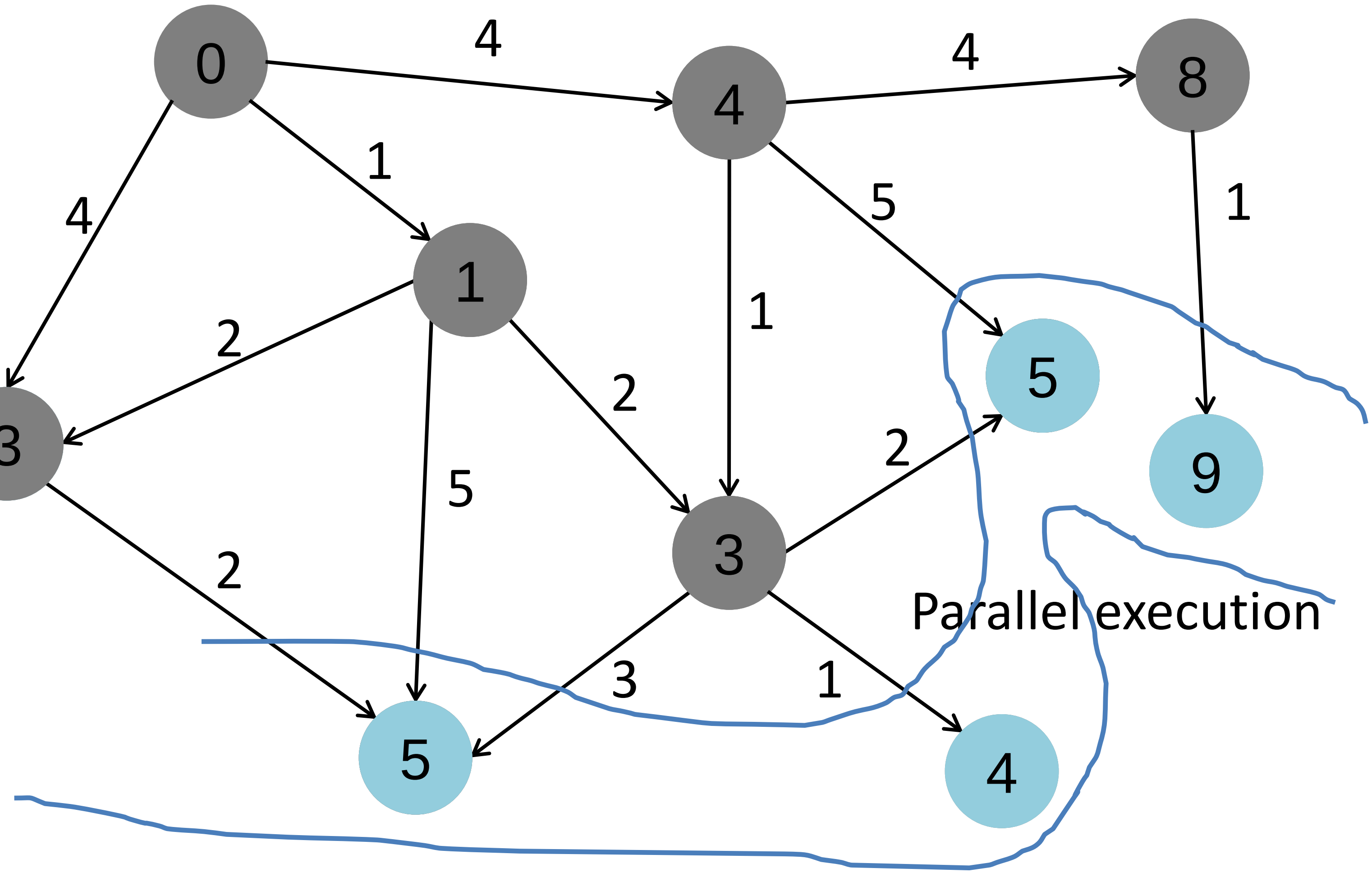
SSSP: 1+3



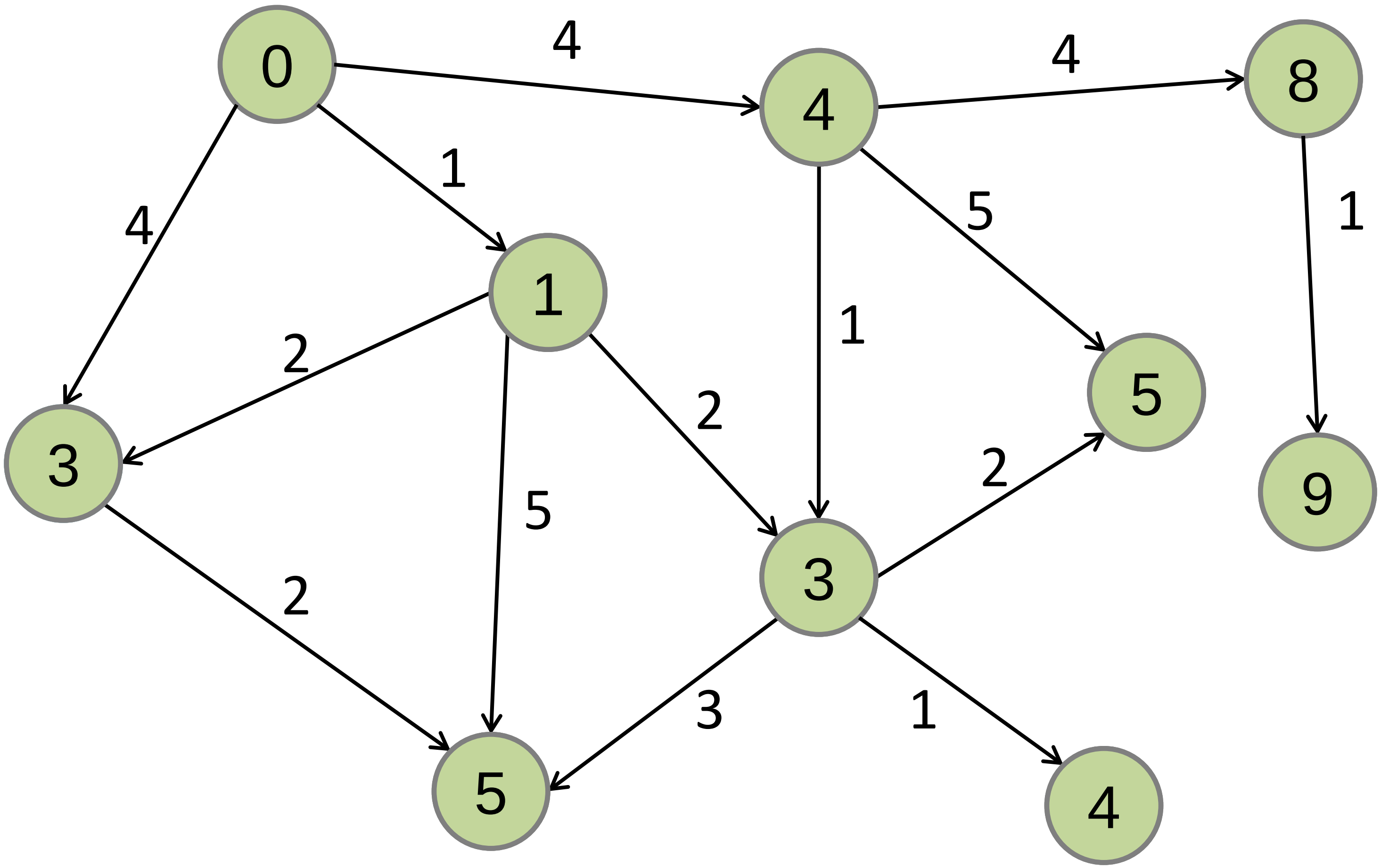
Parallel SSSP: 1+3+5



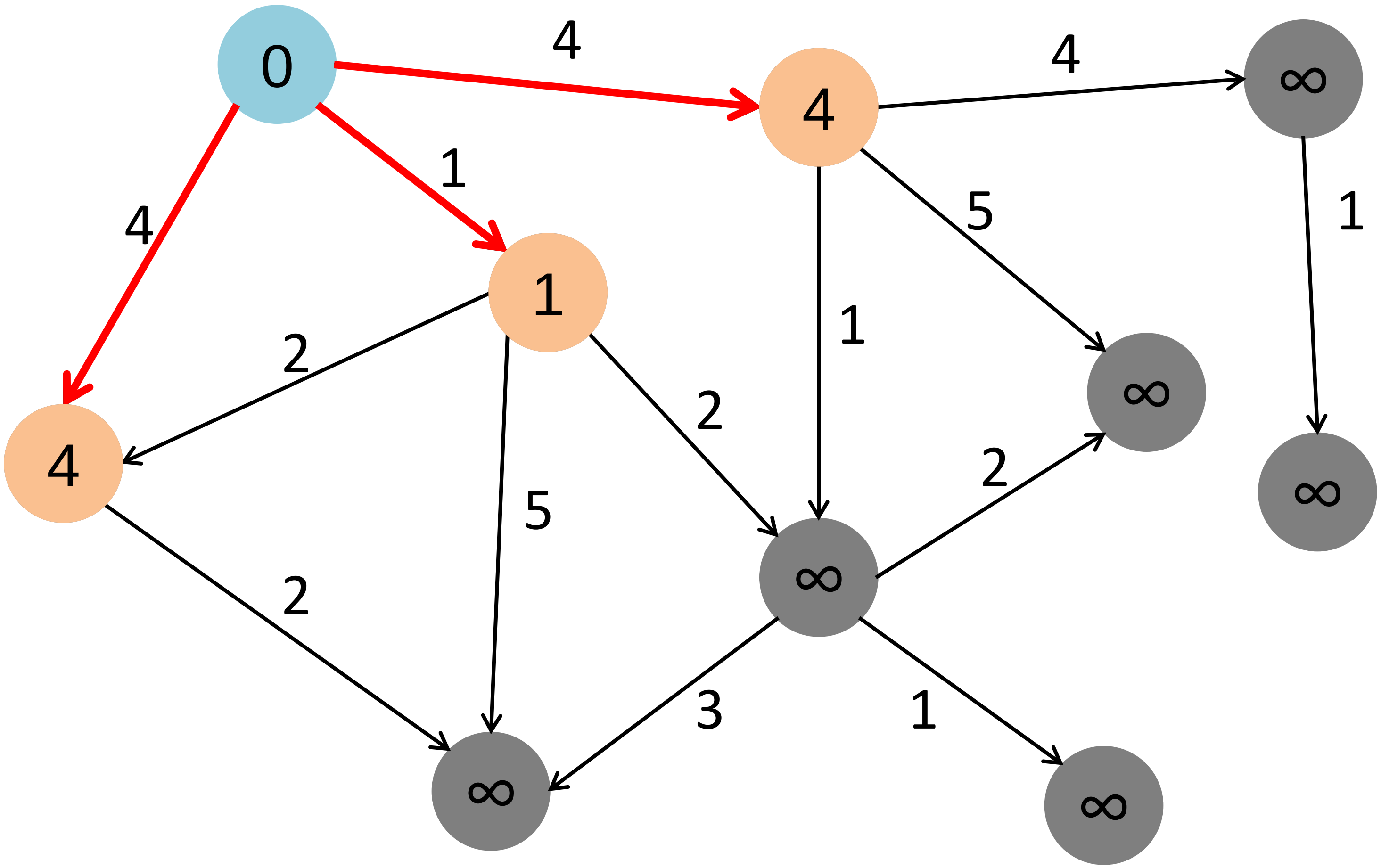
Parallel SSSP: 1+3+5+4



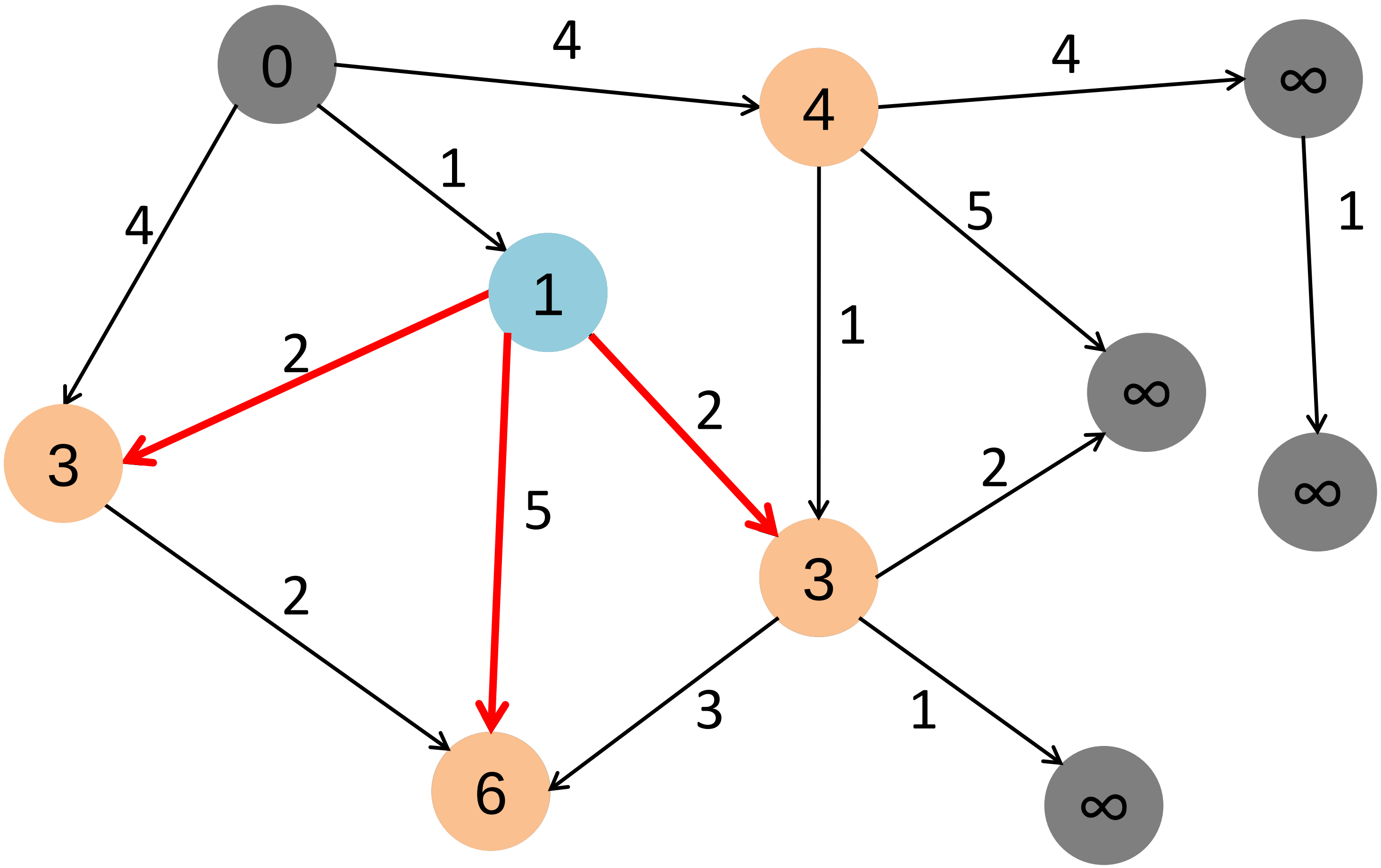
Parallel SSSP: 13 updates



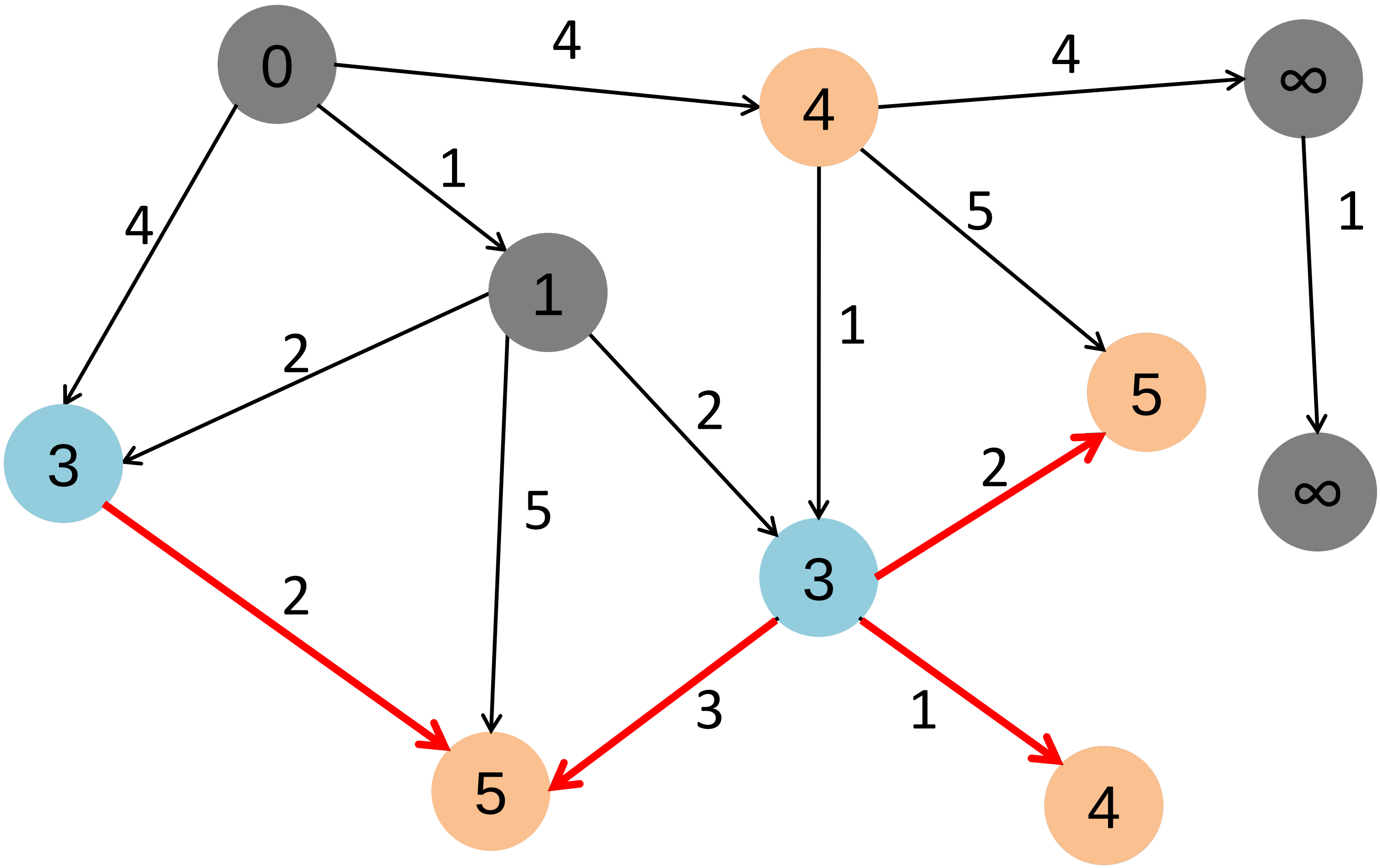
Prioritized SSSP: 1



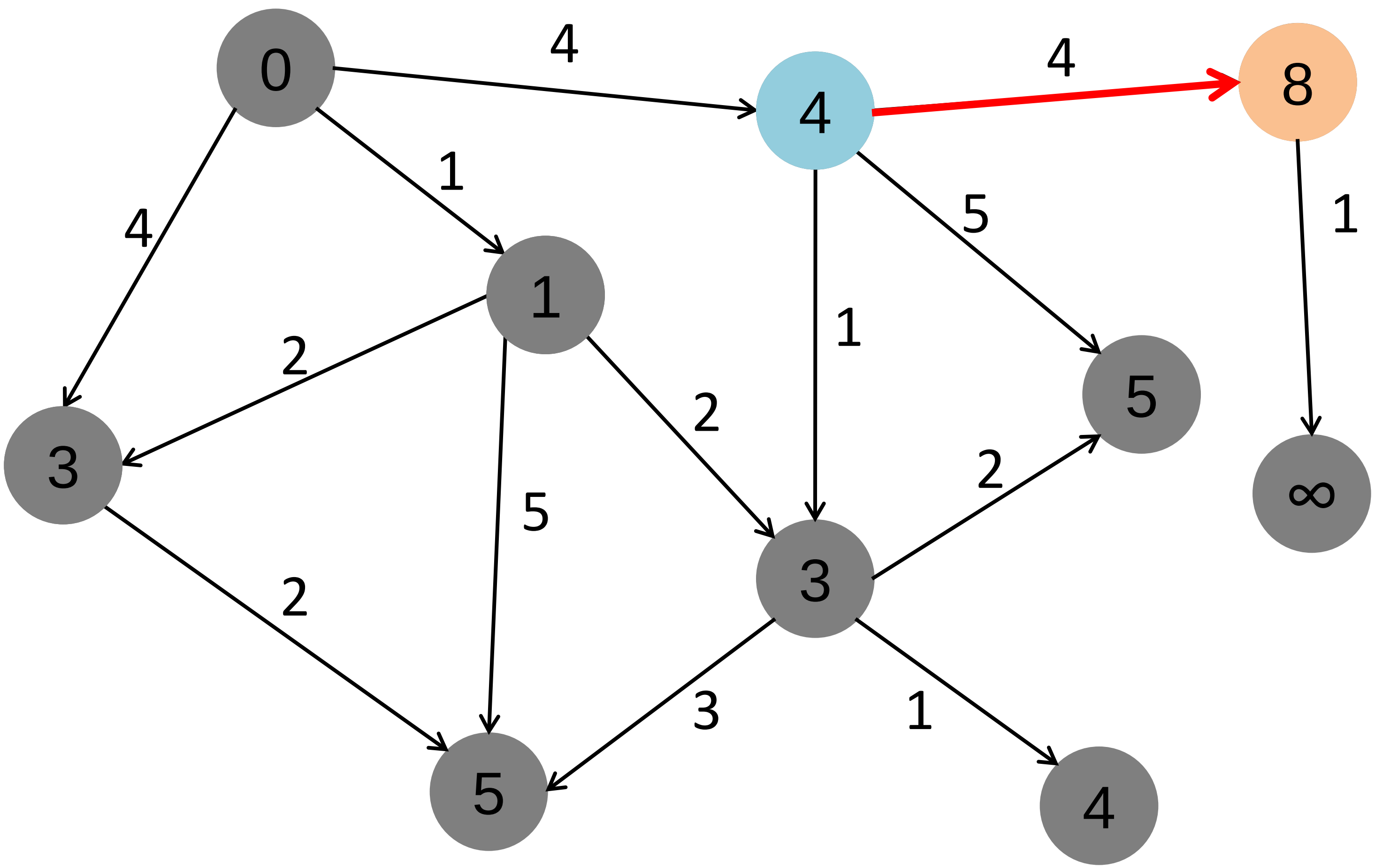
Prioritized SSSP: 1+1



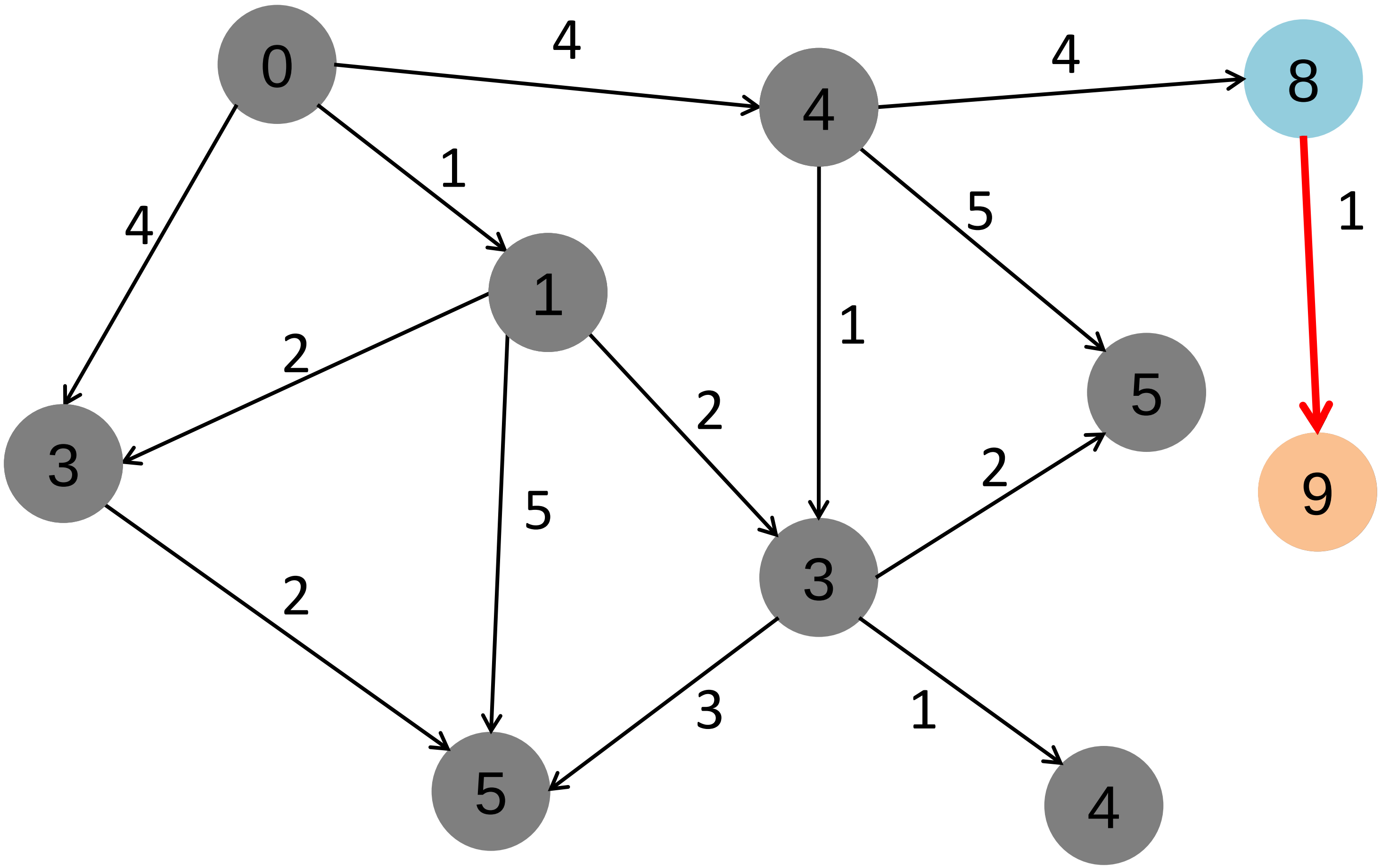
Prioritized SSSP: 1+1+2



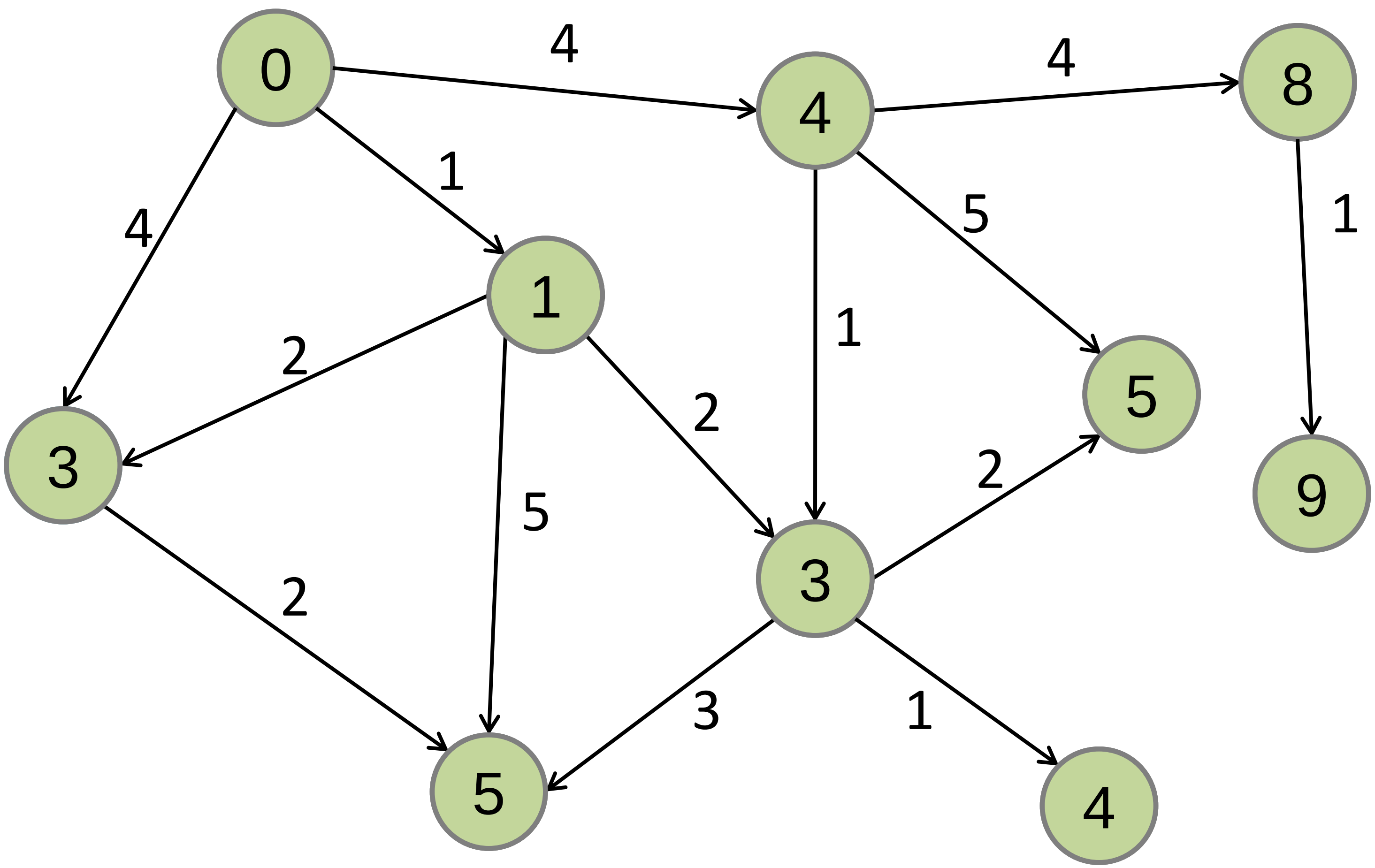
Prioritized SSSP: 1+1+2+1



Prioritized SSSP: 1+1+2+1+1



Prioritized SSSP: 7 updates



PageRank

$$R^{(k)} = dWR^{(k-1)} + (1-d)E$$

- *R* : size-n ranking vector (n)
- *W* : nxn matrix, W_{ij}=1/deg(j)
- *d* : damping factor (0<d<1, usually 0.8)
- *E* : size-n vector (1/n,1/n…,1/n)
- *k* : iteration number

Prioritized PageRank

- Priority is not that obvious as in SSSP

Incremental Update Function

$$R^{(k)} = dWR^{(k-1)} + (1-d)E$$

$$R^{(k)} = (1-d)E\sum_{l=0}^{k} d^l W^l + d^k W^k R^0$$

Incremental update function

$$R^{(i)} = R^{(i-1)} + \Delta R^{(i)} \quad , \quad \Delta R^{(i)} = dW\Delta R^{(i-1)}$$

$$R^{(0)} = 0 \qquad \Delta R^{(0)} = (1-d)E$$

Prioritized Incremental PageRank

$$R^{(i)} = R^{(i-1)} + \Delta R^{(i)}, \quad \Delta R^{(i)} = dW\Delta R^{(i-1)}$$

Measure convergence speed by $||R^{(i)}||_1$, monotonously increasing to $||R^{(\infty)}||_1 = 1$

$$||R^{(i)}||_1 = ||R^{(i-1)}||_1 + ||\Delta R^{(i)}||_1$$

Suppose in each iteration, update
of $\Delta R^{(i-1)}$, that is all the other e

$$||\Delta R^{(i-1)}||_1 = \Delta R^{(i-1)}(j)$$

$$||\Delta R^{(i)}||_1 = ||dW\Delta R^{(i-1)}||_1 = d\Delta R^{(i-1)}(j)$$

W is column normalized matrix $||w_j||_1 = 1$

Priority is given to la

Theoretical Results

- Incremental PageRank converges to the same vector as PageRank
- Prioritized Incremental PageRank
  - Converges to the same vector as PageRank
  - Converges faster than Incremental PageRank
- Same can be proved for many other iterative comptation
  - Personalized PageRank
  - Katz Measure Computation
  - Label Propagation such as Adsorption
  - Single Source Shortest Path

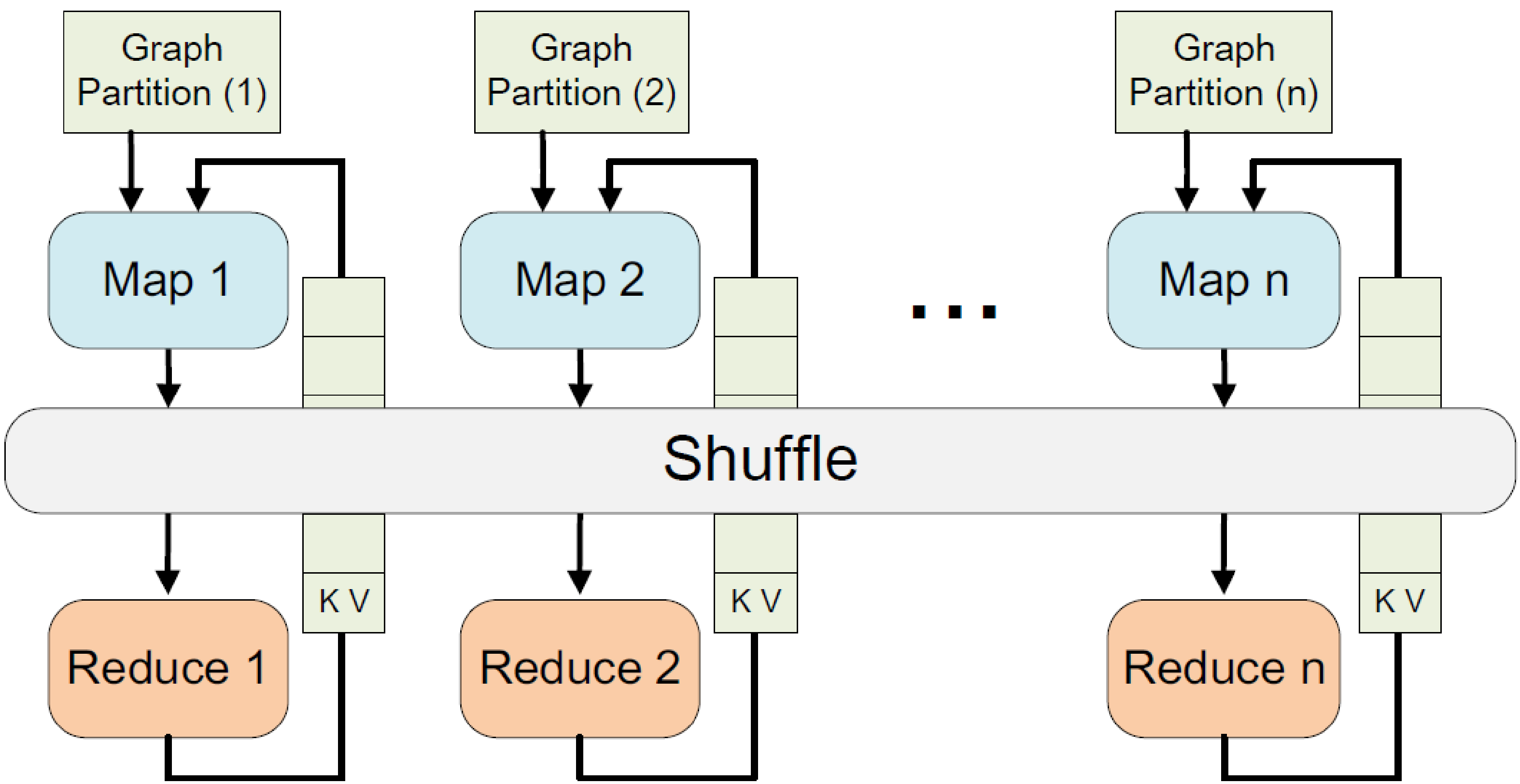
Existing Frameworks

- MapReduce, Dryad
- Support Iterative Computation in the cloud
  - HaLoop (VLDB ‘10)
  - Spark (HotCloud ‘10)
  - Piccolo (OSDI ‘10)
  - Pregel (SIGMOD ‘10)
  - Twistter (MapReduce ‘10)
  - CIEL (NSDI ‘11)
  - iMapReduce (DataCloud ‘11)
- GraphLab
  - Multicore enviroment

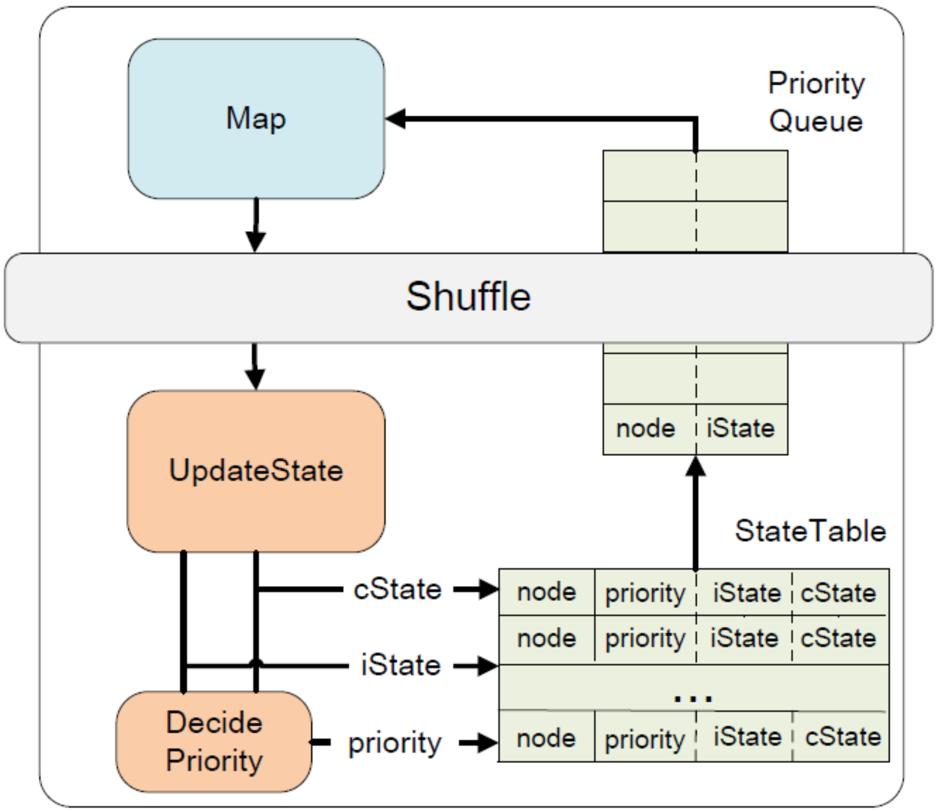
Outline

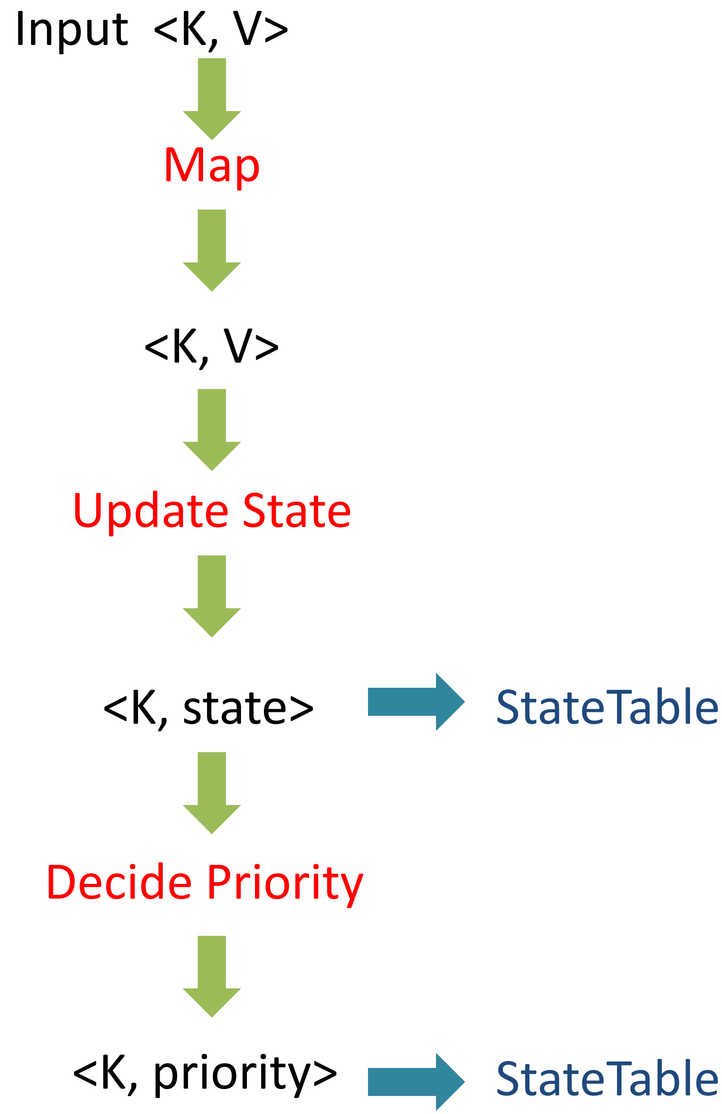
- Prioritized Iteration
- PrIter's Design & Implementation
- Evaluation
- Conclusions

Iterative Processing

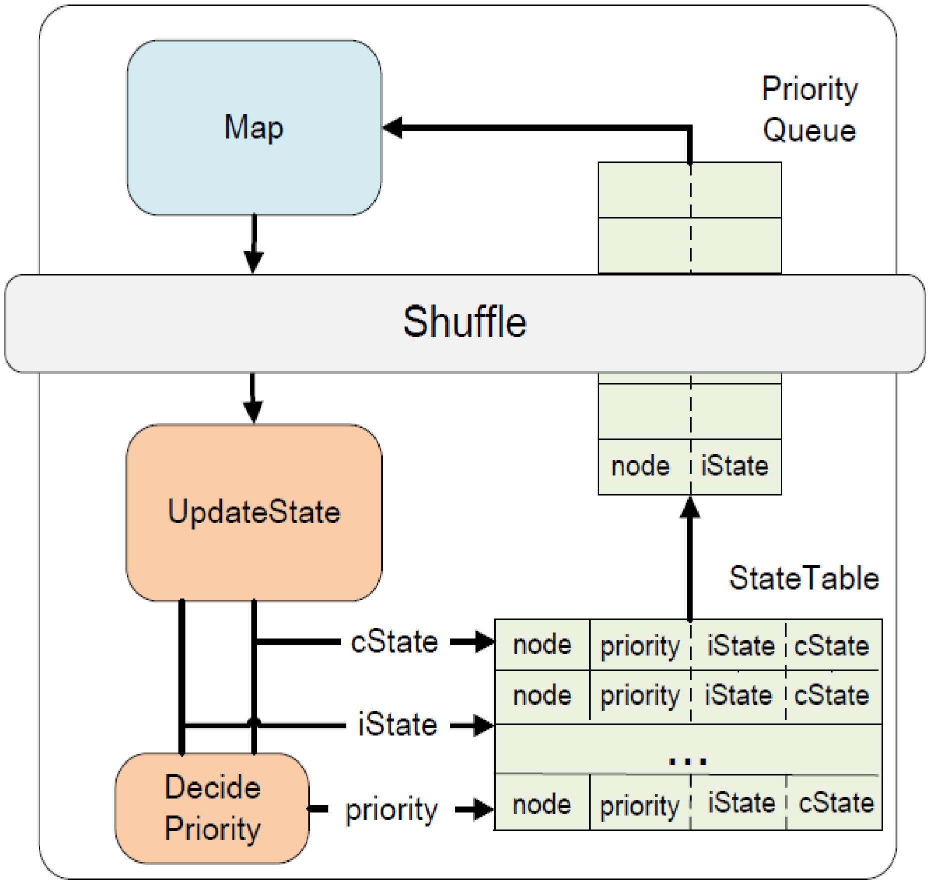


Update StateTable





Extract PriorityQueue



- Extract <K,V>s that have higher priority
  - Sampling method to retrieve a threshold
  - A single pass of StateTable
  - Priority larger than threshold, enqueue

Optimal Queue Size

$$q^* = \sqrt{\frac{\beta \cdot N \cdot T_{ovhd}}{\alpha \cdot T_{proc}}}$$

- Use the default value
- Set a value based on N

Outline

- Prioritized Iteration
- PrIter's Design & Implementation
- Evaluation
- Conclusions

Experiment Cluster

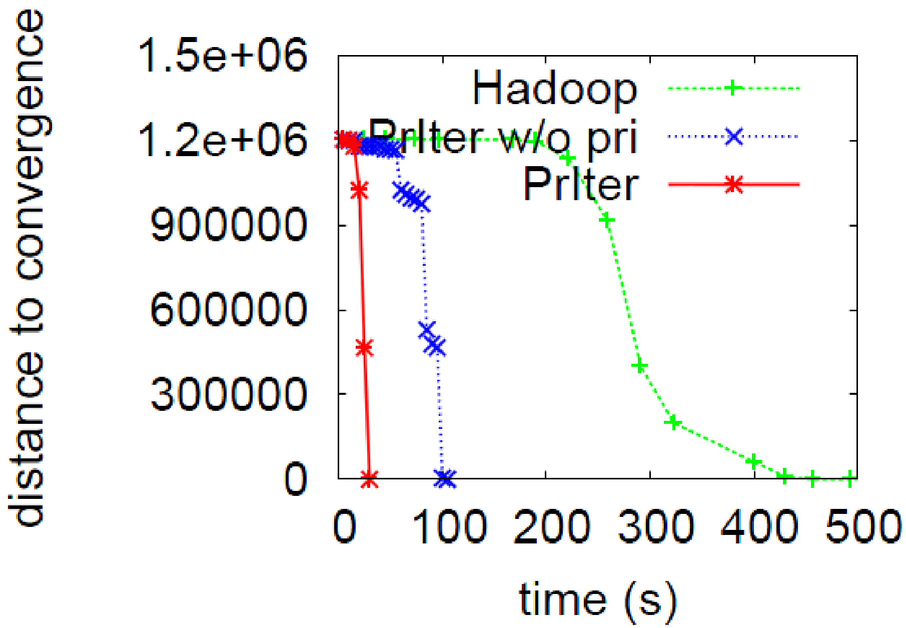
- 4-node local cluster
  - E8200 dual-core 2.66GHz CPU, 3GB of RAM, and 160GB storage
- Amazon EC2 cloud
  - 100 medium instances

Data Sets

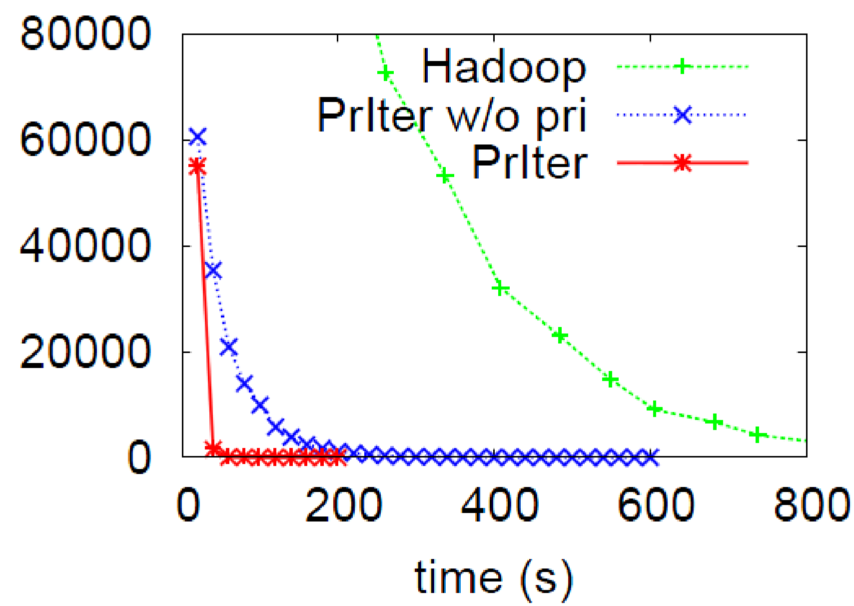
Table 1: Data Sets Summary

Algorithm	Graph	Nodes	Edges
SSSP	Facebook	1,204,004	20,492,148
	LiveJournal	4,847,571	68,993,773
	roadCA	1,965,206	5,533,214
PageRank	Berk-Stan	685,231	7,600,595
	Google	875,713	5,105,039
	Notredame	325,729	1,497,134
	Synth_WEB	100,000,000	1,216,907,427
Adsorption	Facebook	1,204,004	20,492,148
	Youtube	311,805	1,761,812
	Amazon	403,394	3,387,388
ConnComp	Amazon	403,394	3,387,388
	Wiki-talk	2,394,385	5,021,410
	LiveJournal	4,847,571	68,993,773

Convergence Speed (Local Cluster)

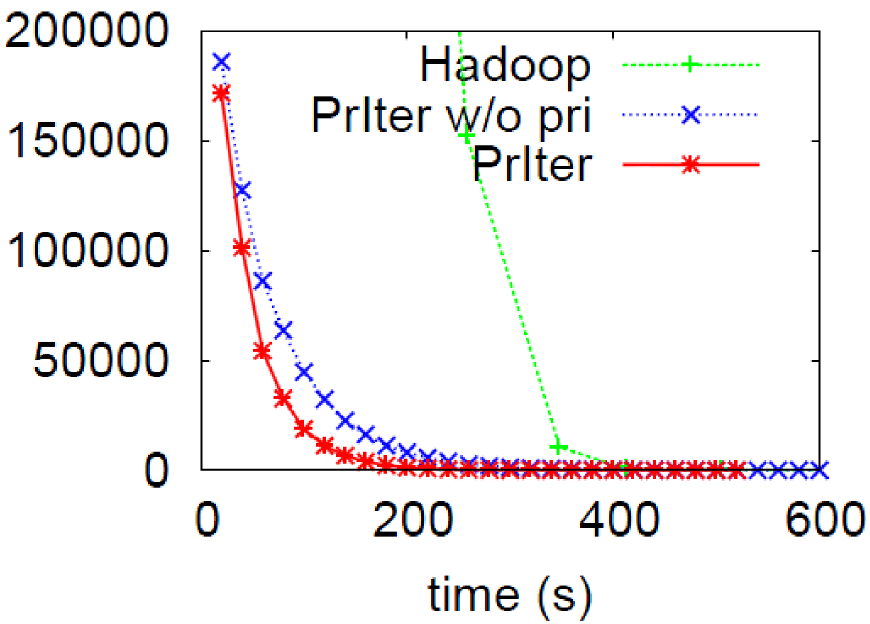


SSSP (Facebook)

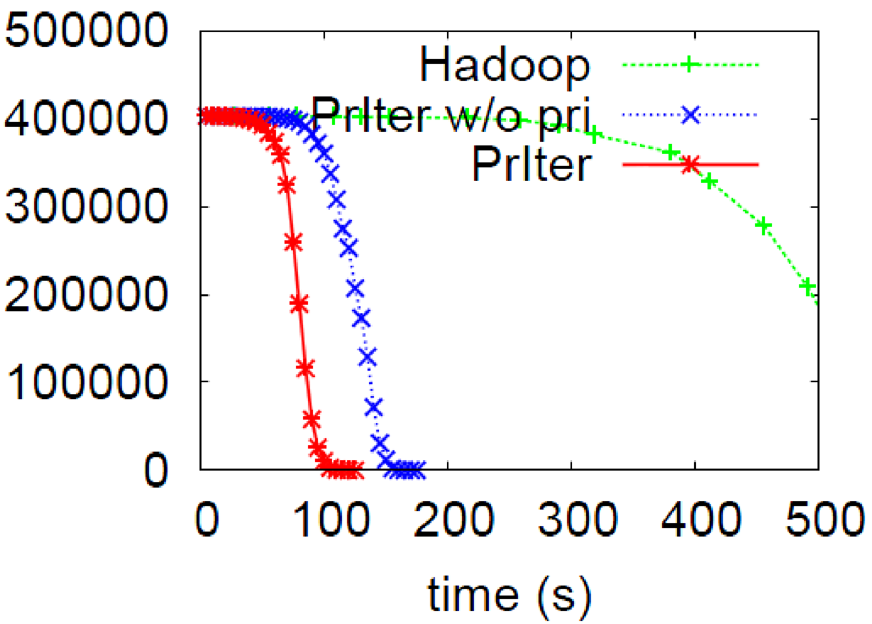


PageRank (BerkStan)

Convergence Speed (Local Cluster)



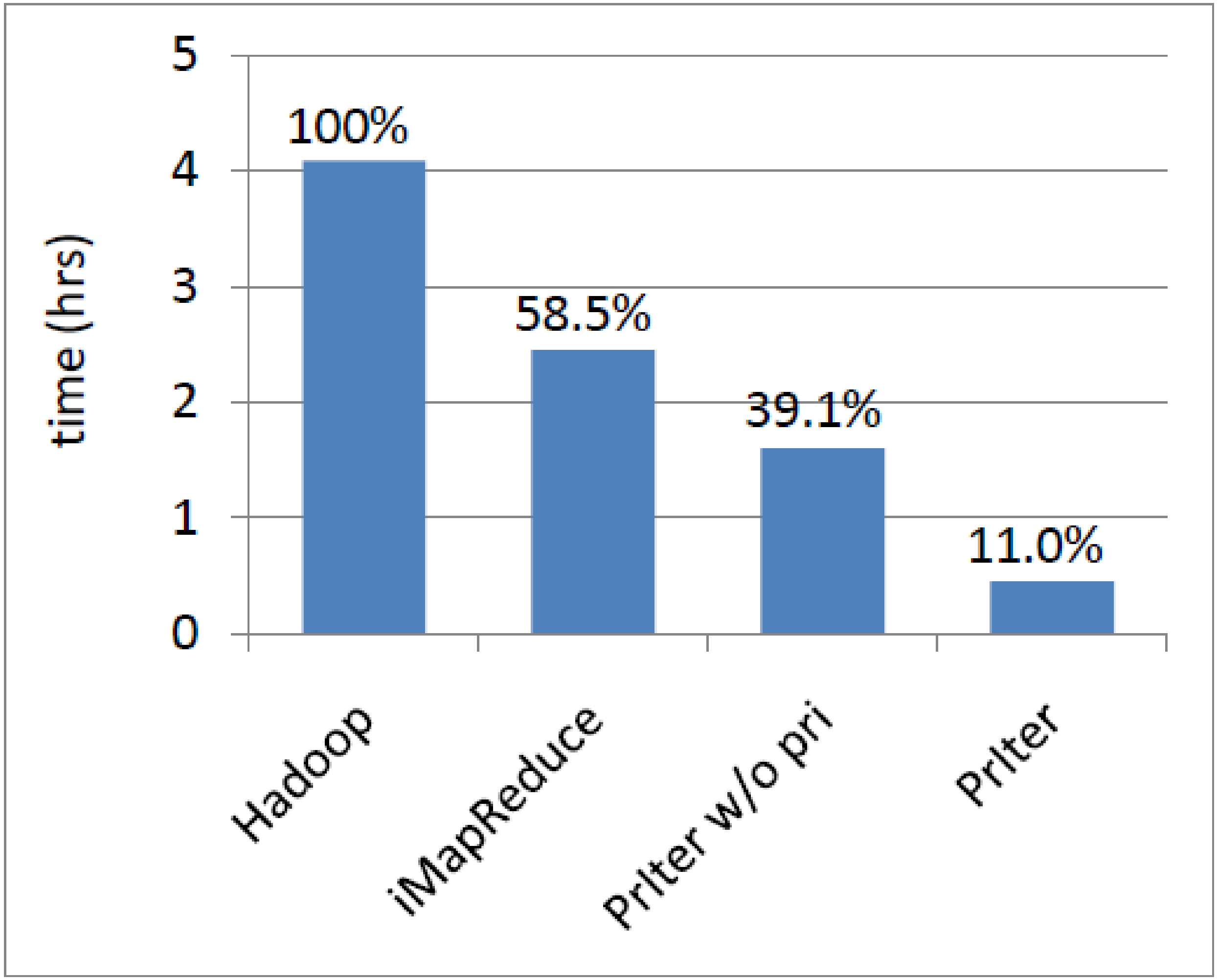
Adsorption (Facebook)



Connected Components (Amazon)

Convergence Time (EC2)

Pagerank

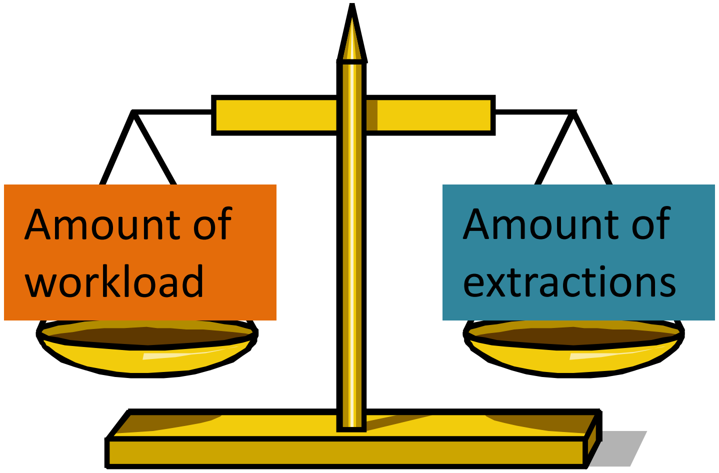


Conclusions

- Proven effectiveness of prioritized iterative computation
  - Converge faster
  - Derive the same converged result
- PrIter
  - Distributed framework that supports prioritized iteration
  - Reduces expensive data access by performing more effective update first
  - Achieve up to 50x speedup comparing to Hadoop
  - MapReduce like API
- Our source code is available at

  http://code.google.com/p/priter/

Optimal Queue Size



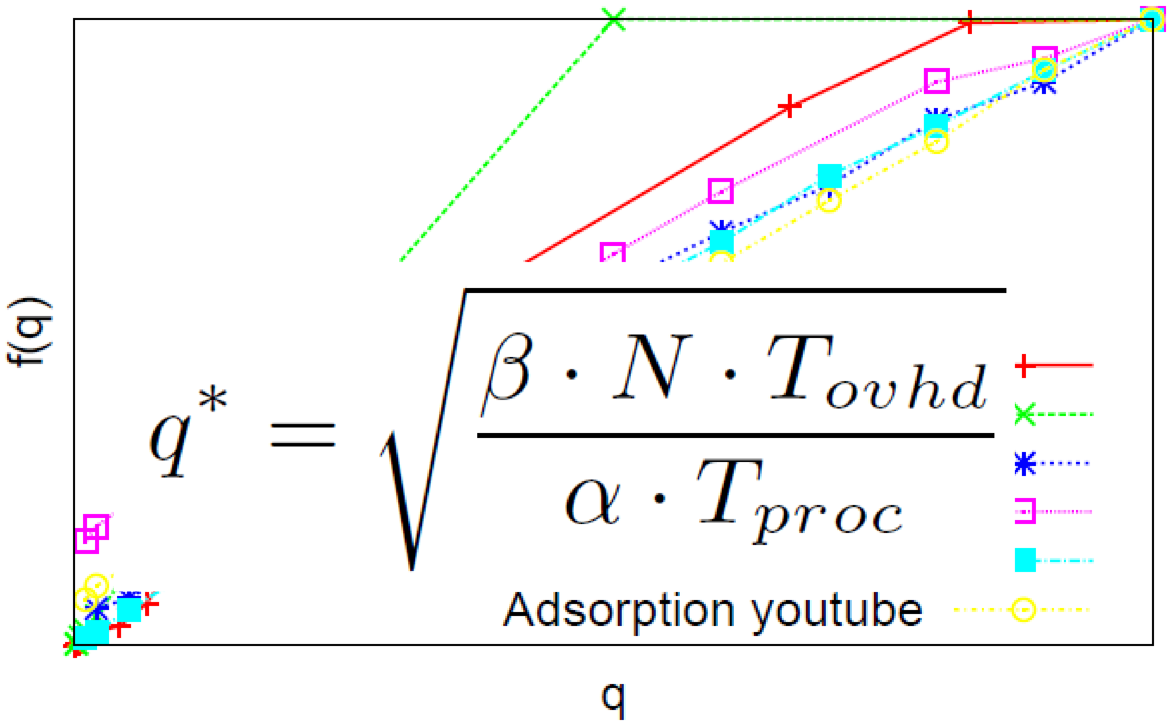
- The optimal case: update the most prioritized node
  - Need a pass of StateTable (overhead)
  - NOT practical
- The other extreme case: update all nodes in StateTable
  - NOT prioritized iteration
- Less updates (less workload) need more frequent queue extractions
- An optimal queue size that balances these two

Optimal Queue Size

$$\min_{q}\left\{ f(q) \cdot T_{proc} + \frac{f(q)}{q} \cdot N \cdot T_{ovhd} \right\}$$

- q: queue size
- f(q): total updates needed
- f(q)/q: number of queue extractions
- N: StateTable size
- T_{proc}: processing time for a node
- T_{ovhd}: scanning time for table entry

Optimal Queue Size



$$q^* = \sqrt{\frac{\beta \cdot N \cdot T_{ovhd}}{\alpha \cdot T_{proc}}}$$

- Online estimation
- Set a default value based on N