SHENZHEN UNIVERSITY 深圳大学

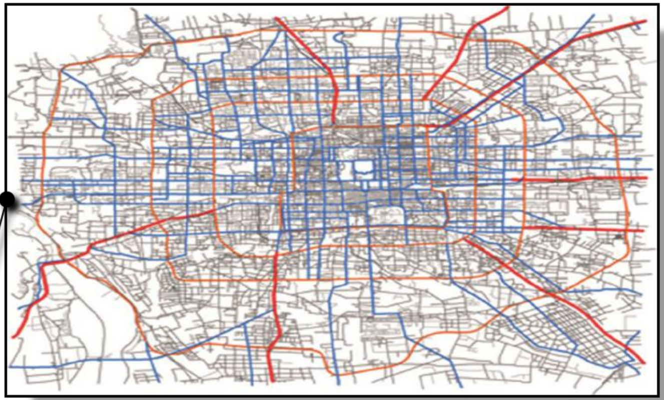
Accelerating PageRank in Shared-Memory for Efficient Social Network Graph Analytics

Baofu Huang,Zhidan Liu* ,Kaishun Wu

Shenzhen University,China

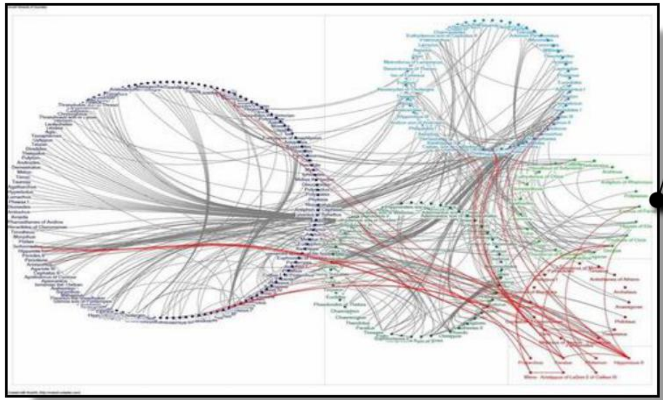
Graph Analytics

Social Network



Road Network

Graph is Ubiquitous



Web Network

Biological Network

Graph Computing

- Graph applications execute in two conceptual phases: message exchange(ME) and local update(LU)

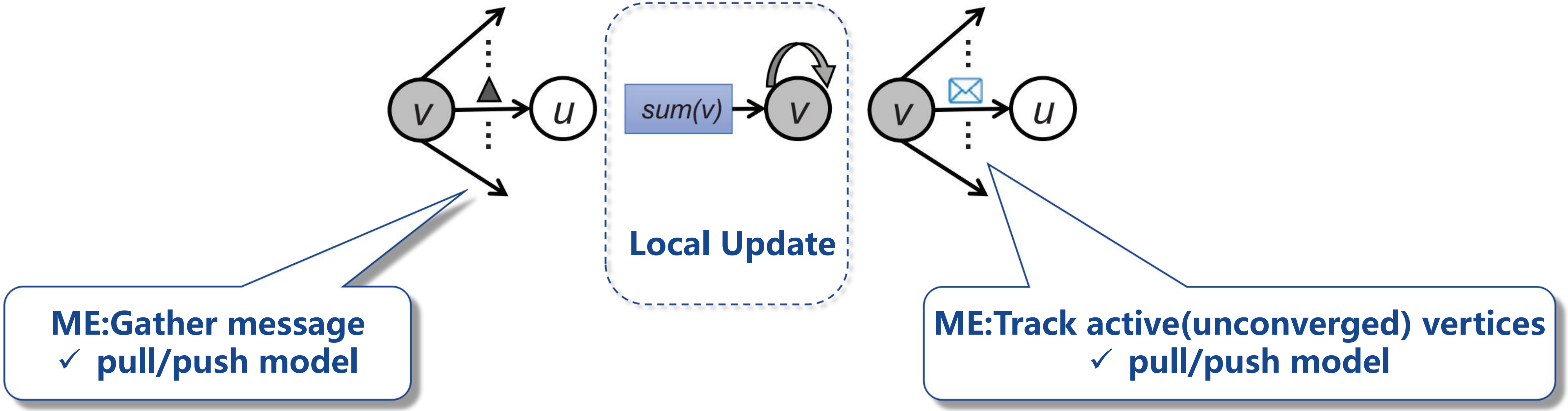
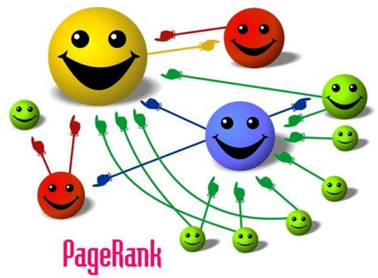


Figure. Graph Computing in GAS model[1]

1. J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. PowerGraph: distributed graph parallel computation on natural graphs. InUSENIX OSDI, 2012.

PageRank

- Important benchmark for evaluating graph analytic frameworks
- Fundamental node ranking algorithm
  - ➢ Iteratively compute weighted sum of neighbor's PR[v_i]



$$PR_{i+1}(\mathrm{u}) = \frac{1-d}{|\mathrm{V}|} + d \sum_{v \in N_i(u)} \frac{PR(v)}{|N_o(v)|}$$

  - ➢ where d is the damp factor, $N_i(u)$ and $N_o(v)$ represent u's in-neighbors and vertex v's out-neighbors, respectively[1]

1. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: bringing order to the web. Technical report, Stanford InfoLab,1999.

Efficient PageRank Computing

- As the magnitude of graph data grows rapidly，how to compute PageRank efficiently ?
  - Serial computing or parallel computing
  - Single-machine computing or distributed computing

1. Frank McSherry, Michael Isard, and Derek G. Murray. Scalability! but at what cost? InProceedings of the 15th USENIX Conference on Hot Topics in Operating Systems, HOTOS'15,pages 14–14. USENIX Association, 2015.

Efficient PageRank Computing

- As the magnitude of graph data grows rapidly， how to compute PageRank efficiently ?
  - ☐ Serial computing or parallel computing
  - ☐ Single-machine computing or distributed computing

1. Frank McSherry, Michael Isard, and Derek G. Murray. Scalability! but at what cost? InProceedings of the 15th USENIX Conference on Hot Topics in Operating Systems, HOTOS’ 15,pages 14–14. USENIX Association, 2015.

Efficient PageRank Computing

- As the magnitude of graph data grows rapidly, how to compute PageRank efficiently ?
  - Serial computing or parallel computing
  - Single-machine computing or distributed computing

Many distributed systems can not defeat graph computing in single thread because of their expensive communication cost [1]

1. Frank McSherry, Michael Isard, and Derek G. Murray. Scalability! but at what cost? InProceedings of the 15th USENIX Conference on Hot Topics in Operating Systems, HOTOS' 15,pages 14–14. USENIX Association, 2015.

Efficient PageRank Computing

- As the magnitude of graph data grows rapidly，how to compute PageRank efficiently ?
  - Serial computing or parallel computing
  - Single-machine computing or distributed computing

Many distributed systems can not defeat graph computing in single thread because of their expensive communication cost [1]

1. Frank McSherry, Michael Isard, and Derek G. Murray. Scalability! but at what cost? InProceedings of the 15th USENIX Conference on Hot Topics in Operating Systems, HOTOS’ 15,pages 14–14. USENIX Association, 2015.

Limitation 1: Push Direction PageRank

Algorithm. *Push*-based Parallelled PageRank Computing

```
parallel_for (int vSrc = 0; vSrc < numVertices; ++vSrc) {
    if (!frontier.contains(vSrc)) continue;
    for (int d = 0; d < vertex[vSrc].outdegree; ++d) {
        const int vDst = vertex[vSrc].outneighbor[d];
        if (converged.contains(vDst)) continue;
        atomicCAS(vertex[vDst].value,
            compute(vertex[vSrc].value, vertex[vDst].value)); } }
```

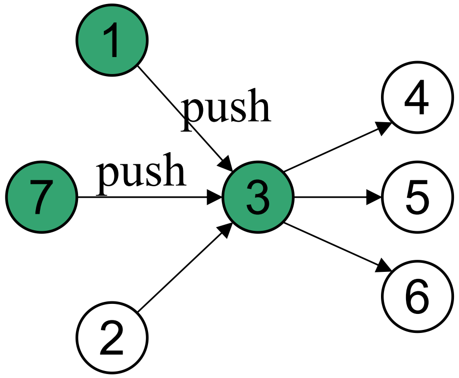


Fig 1. Push Model

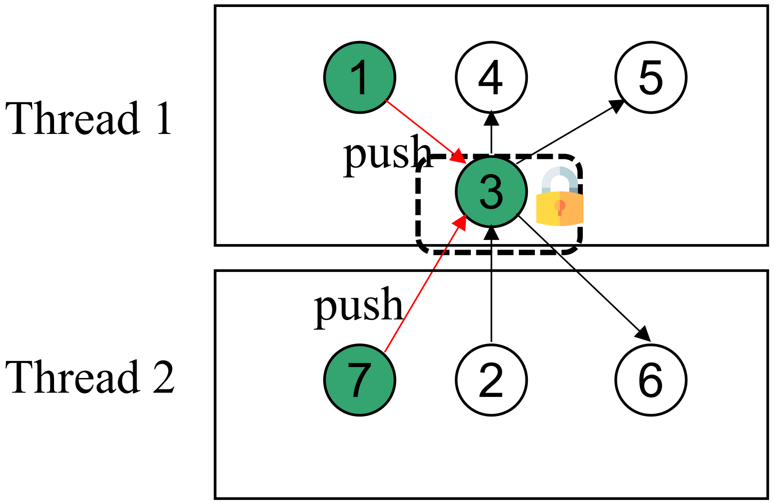


FIg 3. CAS for conflict

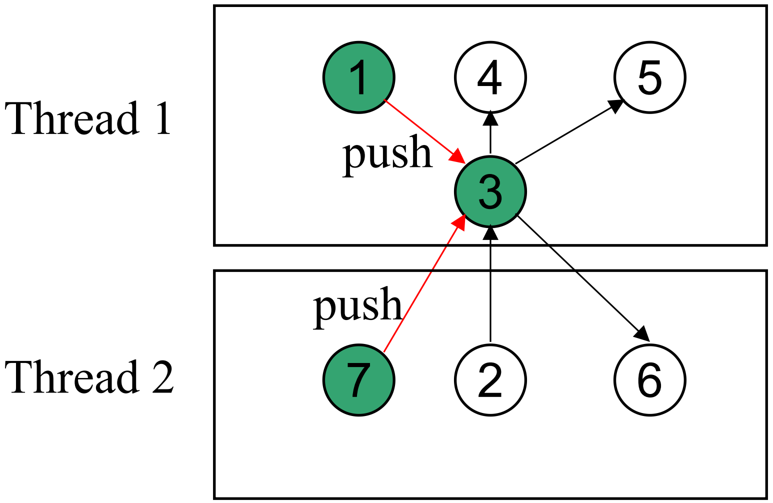


FIg 2. Multi-thread computing

Limitation 1: Push Direction PageRank

Algorithm. *Push*-based Parallelled PageRank Computing

```
parallel_for (int vSrc = 0; vSrc < numVertices; ++vSrc) {
    if (!frontier.contains(vSrc)) continue;
    for (int d = 0; d < vertex[vSrc].outdegree; ++d) {
        const int vDst = vertex[vSrc].outneighbor[d];
        if (converged.contains(vDst)) continue;
        atomicCAS(vertex[vDst].value,
            compute(vertex[vSrc].value, vertex[vDst].value)); } }
```

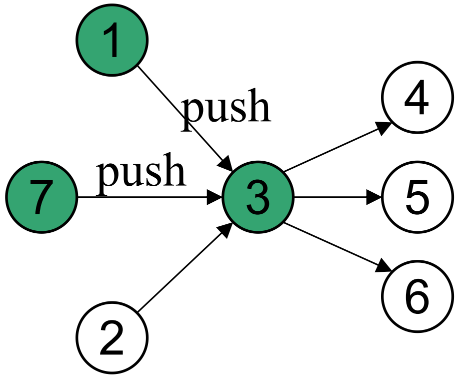


Fig 1. Push Model

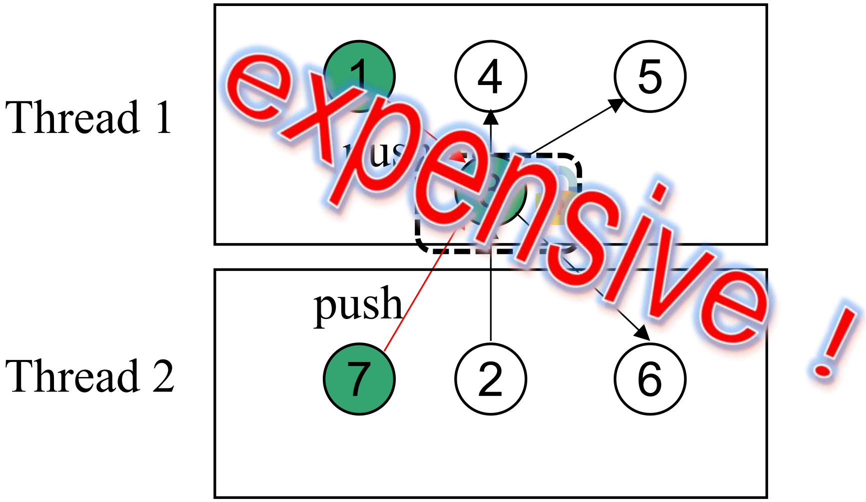


FIg 3. CAS for conflict

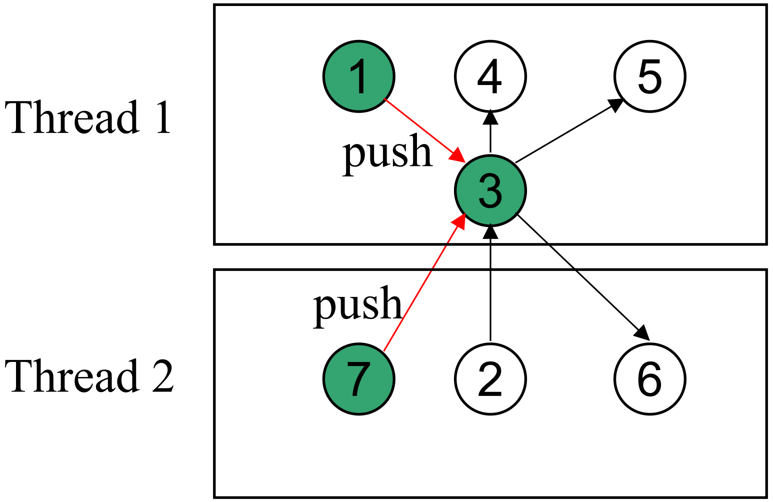


FIg 2. Multi-thread computing

Limitation 2: Computing Redundancy

Pseudocode for PageRank

1. initData(v): v.rank = 0.15; Δ = -0.85;
2. initMsg: **Activate**(u), $u \in$ V
3. **......**

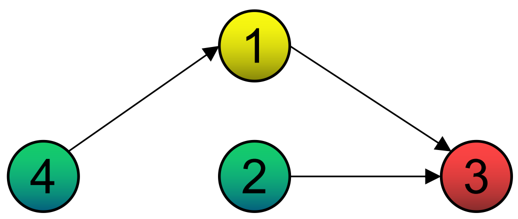


FIg. example graph

1st iteration:

$PR_1[3]$ = compute{$PR_0[1]$,$PR_0[2]$} -->unconverged
$PR_1[1]$ = compute{$PR_0[4]$} -->unconverged
$PR_1[2]$ = compute{}; -->converged
$PR_1[4]$ = compute{}; -->converged

2nd iteration:

$PR_2[3]$ = compute{$PR_1[1]$} -->unconverged
$PR_2[1]$ = compute{}; -->converged

3rd iteration:

$PR_3[3]$ = compute{} -->converged

Limitation 2: Computing Redundancy

Pseudocode for PageRank

1. initData(v): v.rank = 0.15; Δ = -0.85;
2. initMsg: **Activate**(u), $u \in$ V
3. **......**

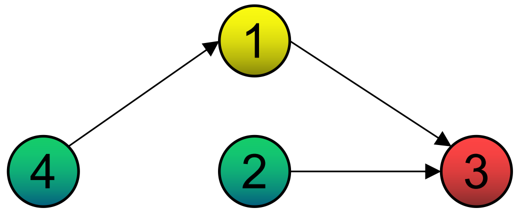


FIg. example graph

1st iteration:

$PR_1[3]$ = compute{$PR_0[1]$,$PR_0[2]$} -->unconverged
$PR_1[1]$ = compute{$PR_0[4]$} -->unconverged
$PR_1[2]$ = compute{}; -->converged
$PR_1[4]$ = compute{}; -->converged

2nd iteration:

$PR_2[3]$ = compute{$PR_1[1]$} -->unconverged
$PR_2[1]$ = compute{}; -->converged

3rd iteration:

$PR_3[3]$ = compute{} -->converged

- A vertex will not converge until all it's in-neighbors have become converged
- Not all vertics need to start computing from the beginning,e.g. vertex 3

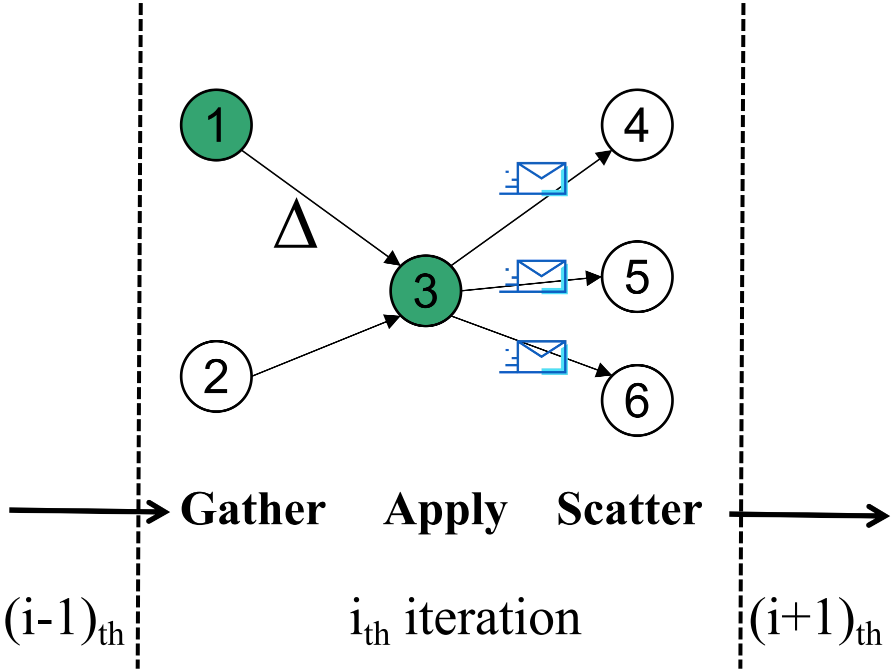
Limitation 3: Communicating Redundancy

```
Gather (v, n):
  return n.rank/#outNbrs(v)
Acc (a, b): return a + b

Apply(v, sum)
  v.rank = 0.15 + 0.85 * sum

Scatter (v, n):
  if ( !converged(v) )
    activate(n)
```

Figure. The sample code of PageRank on various systems.[1]

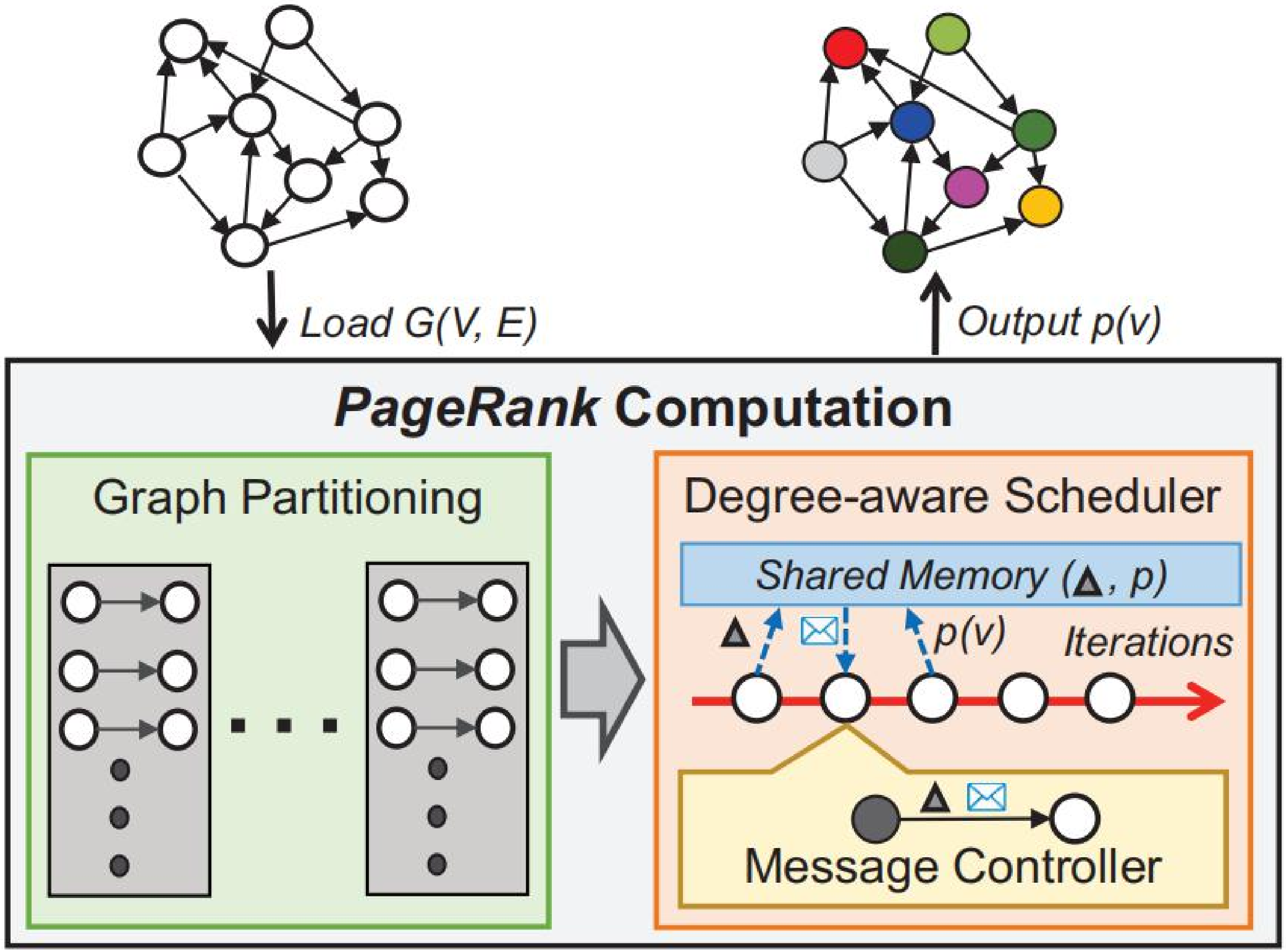


- Unconverged Vertics have to communicate with their neighbors twice per iteration

1. J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. PowerGraph: distributed graph parallel computation on natural graphs. InUSENIX OSDI, 2012.

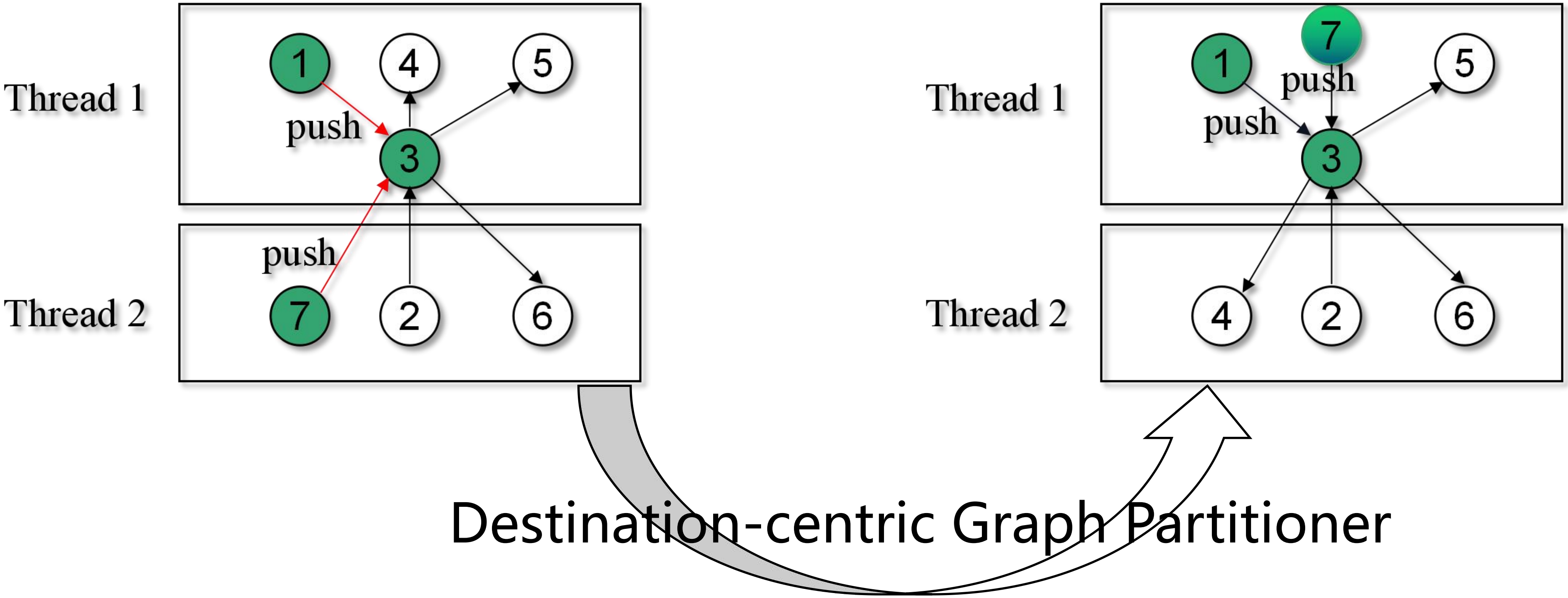
Overview

- **Components of APPR**
  - ✓ **Graph Partitioner**
  - ✓ **Degree-aware Scheduler**
  - ✓ **Message Controller**



Opt 1: Destination-Centric Graph Partitioning

- Partitioning is done by grouping edges based on destination
- It works well in most cases



Opt 2: Degree-Aware Computation Scheduler

- Low in-degree(L) vertices compute ahead of High in-degree(H) vertices
- This lazy strategy does not affect the correctness of computing results

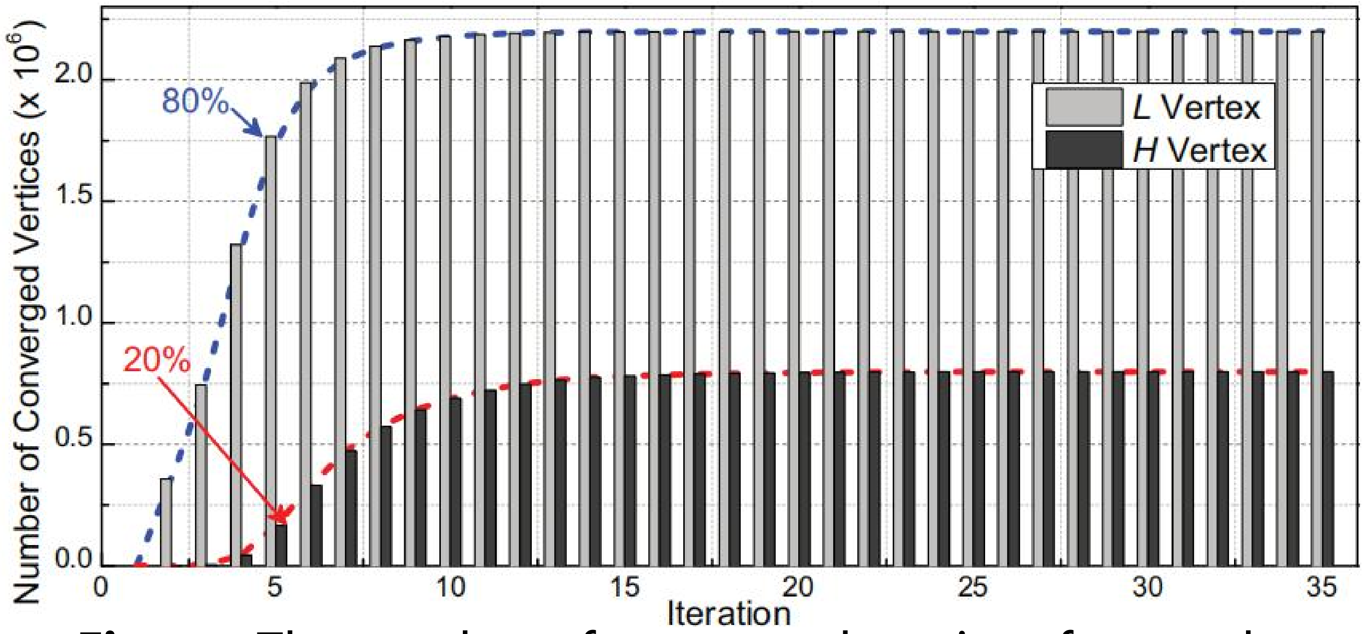


Figure. The number of converged vertices for graph orkut in each iteration.

Opt 3：Message Controller

- How does it work ?
    1. The active vertics all push their value to their neighbors
    2. A vertex will push both status and new update to it's neighbors **at the same time** if it's still not converged， or it will take no actions

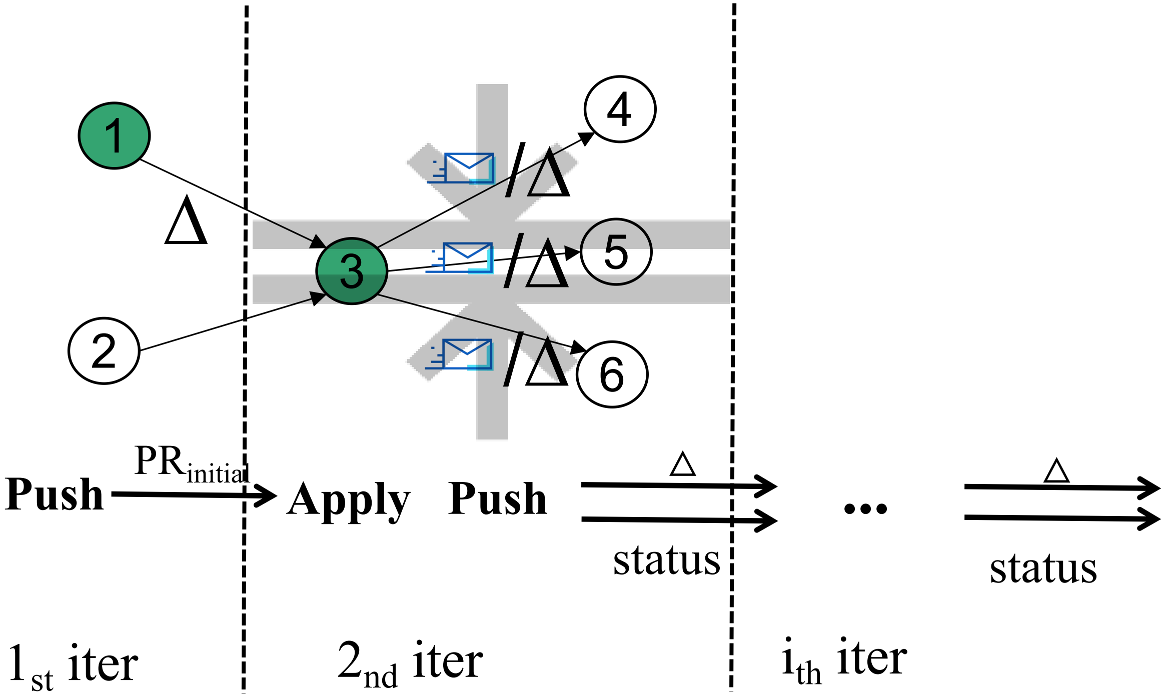


Figure. Message Controller of APPR

- In one iteration, a vertex needs to communicate with its neighbors at most once

Experimental Setup

- Baseline：
  - ✓ **pullPR**： *PullPR* implements PageRank in the pull direction[1]
  - ✓ **pushPR**： *PushPR* implements PageRank in the push direction
  - ✓ **PCPM**： *PCPM* is the state-of-the-art method that optimizes the parallel PageRank computation based on a partition centric processing methodology

✓ **All in parallel**

- Platform:
  - ✓ Intel(R) Xeon(R) E5-2630 v4 processors @2.20GHz
  - ✓ Dual-socket --- 10 cores per socket with 192 GB memory
- Dataset

Graph	Description	#vertics(M)	#edges(M)	*d*	Disk size(G)
livej	Social network	7.5	112.3	15	1.6
twitter	Social network	21.3	265.0	12	5.2
orkut	Social network	3.0	106.3	35	1.6
pld	Web Pages	42.9	623.1	15	10.9
sd	Web Pages	94.9	1937.5	20	34.4

1. S. Beamer, K. Asanovi ́c, and D. Patterson. The GAP benchmark suite. arXiv preprint arXiv:1508.03619, 2015.

Experimental Results

- Overall performance

TABLE
COMPARISONS ON EXECUTION TIME (*Unit: seconds*)

Graph	***PullPR***	***PushPR***	***PCPM***	***APPR***	*Ratio*
livej	1.4	2.5	4.0	1.0	$1.4 \sim 4.0$
twitter	5.6	14.6	7.5	3.7	$1.5 \sim 3.9$
orkut	1.9	3.0	1.7	0.5	$3.4 \sim 6.0$
pld	29.5	59.5	13.6	11.6	$1.2 \sim 5.1$
sd	94.9	99.8	35.1	29.5	$1.2 \sim 3.4$

- ✓ Up to 4.0x speedup over PCPM
- ✓ Up to 6.0x speedup over PushPR
- ✓ Up to 3.8x speedup over PullPR

Experimental Results

- Pre-processing time

TABLE
COMPARISONS ON PRE-PROCESSING TIME (*Unit: seconds*)

Method	*livej*	*twitter*	*orkut*	*pld*	*sd*
PCPM	0.04	0.34	0.08	0.20	0.54
APPR	0.11	0.50	0.18	0.55	1.39

- APPR spends slightly more time to pre-process a graph than PCPM
- The pre-processing time of both methods is **proportional** to the graph size, i.e., a larger graph needs more pre-processing time

Experimental Results

- Evaluation of APPR Design
  - **APPR-S** <-> APPR without degree-aware scheduler
  - **APPR-M** <-> APPR without message controller

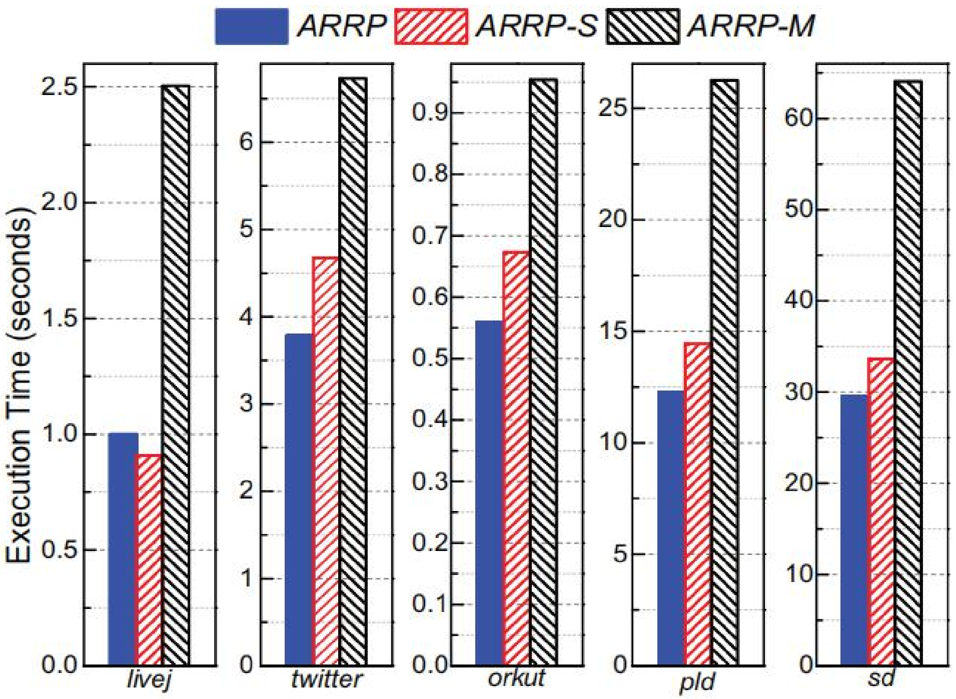


Figure. Evaluation of APPR modules.

- ✓ Degree-aware scheduler brings 14% ~ 24% improvements on execution time
- ✓ On average, the message controller module accelerates PageRank computation by 104%

Experimental Results

See paper for more results …

Conclusion

- We present APPR to accelerate parallel PageRank computation in the shared-memory platforms for large scale graphs
  - Destination-centric graph partitioner to avoid synchronization issues
  - Degree-aware computation scheduler to reduce unnecessary operations
  - Message controller to improve the efficiency of memory accesses

- APPR outperforms state-of-the-art methods with on average 2.4x speedup in execution time and 16.4x reduction in communication messages for social network graphs

深圳大学
SHENZHEN UNIVERSITY

2020

Thank You