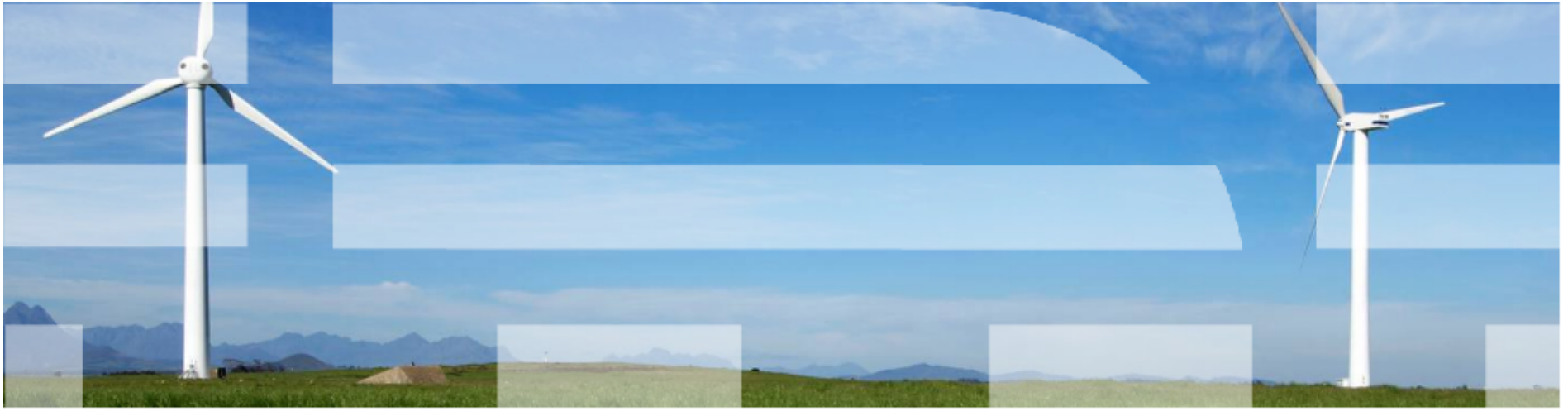
Graph Community Detection Algorithm for Distributed Memory Parallel Computing Systems

(Credit: X. Que, F. Checconi, F. Petrini, J. A. Gunnels — IBM Research)

Alexander Pozdneev

Technical Consultant — High Performance Computing, IBM

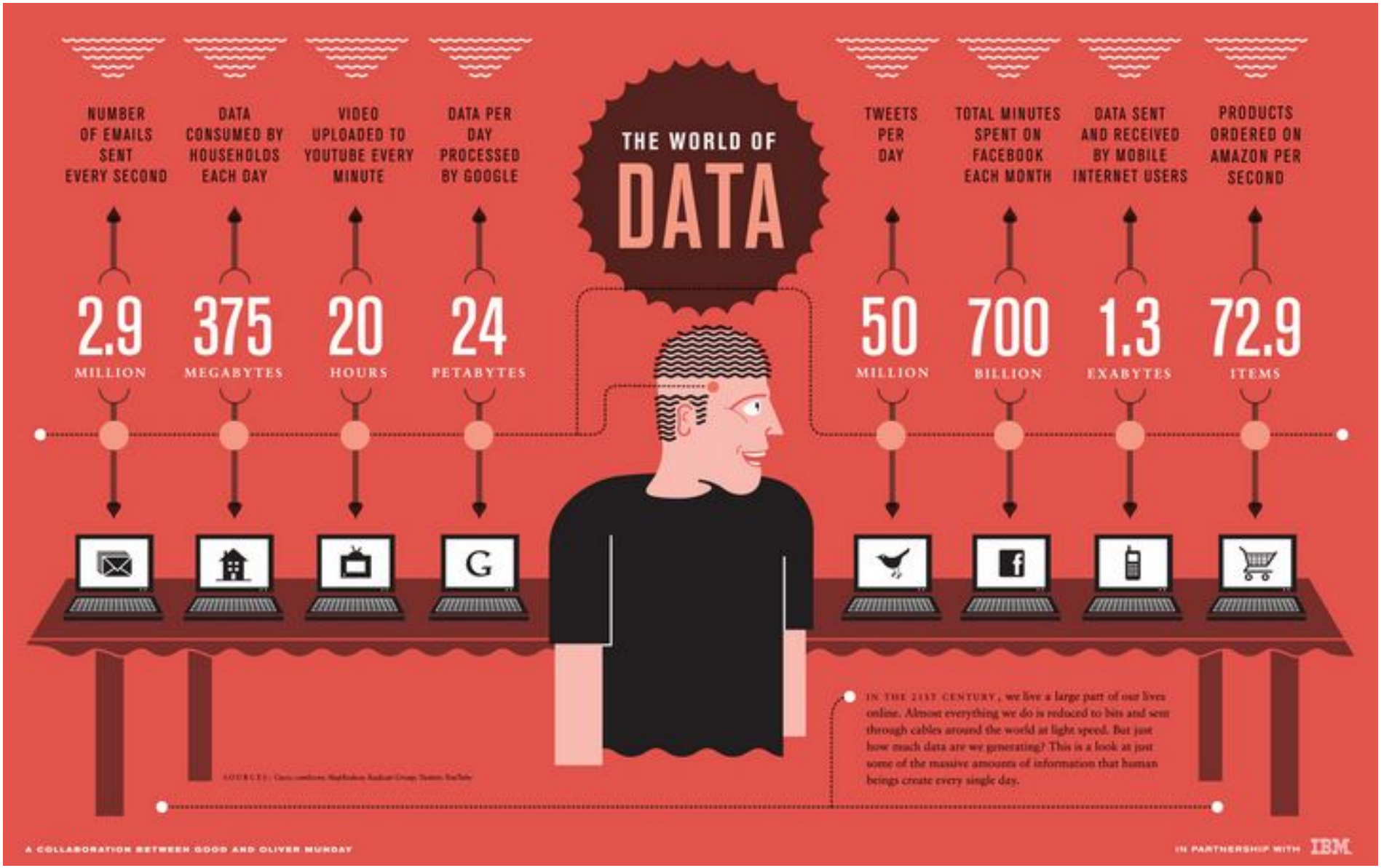
March 5, 2015 — GraphHPC-2015



Outline

Outline

Big Data challenges



IBM leadership in graph processing: Graph500

Date	#	System	Model	Nodes	Cores	Scale	GTEPS
Nov'14	1	Sequoia	Q	96k	1.5M	41	23751
Jun'14	2	Sequoia	Q	64k	1M	40	16599
Nov'13	1	Sequoia	Q	64k	1M	40	15363
Jun'13	1	Sequoia	Q	64k	1M	40	15363
Nov'12	1	Sequoia	Q	64k	1M	40	15363
Jun'12	1	Sequoia/Mira	Q	32k	512k	38	3541
Nov'11	1	BG/Q prototype	Q	4k	64k	32	253
Jun'11	1	Interpid/Jugene	P	32k	128k	38	18
Nov'10	1	Interpid	P	8k	32k	36	7

Outline

Reference

X. Que, F. Checconi, F. Petrini, J. Gunnels.
"Scalable Community Detection with the Louvain Algorithm".
29th IEEE International Parallel & Distributed Processing Symposium,
Hyderabad International Convention Centre, Hyderabad, INDIA,
May 25-29, 2015.
`http://dx.doi.org/10.1109/IPDPS.2015.59`

The graph community detection problem

- Important problem that spans many research areas:
  - health care
  - social networks
  - systems biology
  - power grid optimization, etc.
- Graph community detection algorithms attempt to identify
  - modules
  - their hierarchical organization
- Challenges that limit the overall scalability and performance
  - fine-grained communication
  - irregular access pattern to memory and interconnect
- Open research problem
  - strong scalability
  - high quality of community detection

Graph community detection: problem definition

- A weighted directed graph $G = (V, E)$:
  - V — set of vertices
  - E — set of edges
  - when u, $v \in V$, an edge $e(u, v) \in E$ has weight $w_{u,v}$
- The goal of *community detection* is to partition graph G into a set C of disjoint communities c_i:

$$\cup\, c_i = V, \quad \forall c_i \in C$$
$$c_i \cap c_j = \varnothing, \quad \forall c_i, c_j \in C$$

- Vertices in the *same* community are *densely* connected
- Vertices in *different* communities are *sparsely* connected
- The *modularity* [Newman, 2004] quantifies a community structure
- Empirically, the higher a modularity value, the better a partition quality

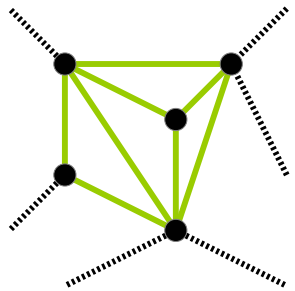
Graph community detection: the modularity metric

Modularity Q,

$$Q = \sum_{c \in C} \left[\frac{\Sigma^c_{\text{in}}}{2m} - \left(\frac{\Sigma^c_{\text{tot}}}{2m} \right)^2 \right]$$

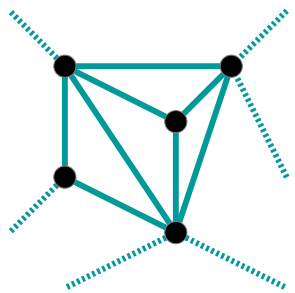
Σ^c_{in} — the sum of the weights from all internal edges of community c,

$$\Sigma^c_{\text{in}} = \sum_{\substack{u,v \in c \\ e(u,v) \in E}} w_{u,v}$$



Σ^c_{tot} — the sum of the weights from edges incident to any vertex in c,

$$\Sigma^c_{\text{tot}} = \sum_{\substack{u \in c \text{ or } v \in c \\ e(u,v) \in E}} w_{u,v}$$



m — normalization factor, the sum of the weights across the graph,

$$m = \sum_{e(u,v) \in E} w_{u,v}$$

Sequential Louvain algorithm

Louvain algorithm [Blondel, 2008] is a popular greedy algorithm for community detection.

1. Put all vertices into distinct communities (one per vertex)
2. Refine communities
   - For each vertex i
     - Compute $\Delta Q_{i \to c(j)}$ for each neighbor j
     - Join the community $c(j)$ that yields the largest gain in ΔQ
   - Repeat until no movement yields a gain
3. Reconstruct the graph
   - The partitions become supervertices
   - The weights of edges between communities are summed
4. Repeat steps 2 and 3 until convergence

Modularity gain

Modularity gain when moving vertex u into community c:

$$\Delta Q_{u \to c} = \frac{w_{u \to c}}{2m} - \frac{w(u) \Sigma^c_{\text{tot}}}{2m^2}$$

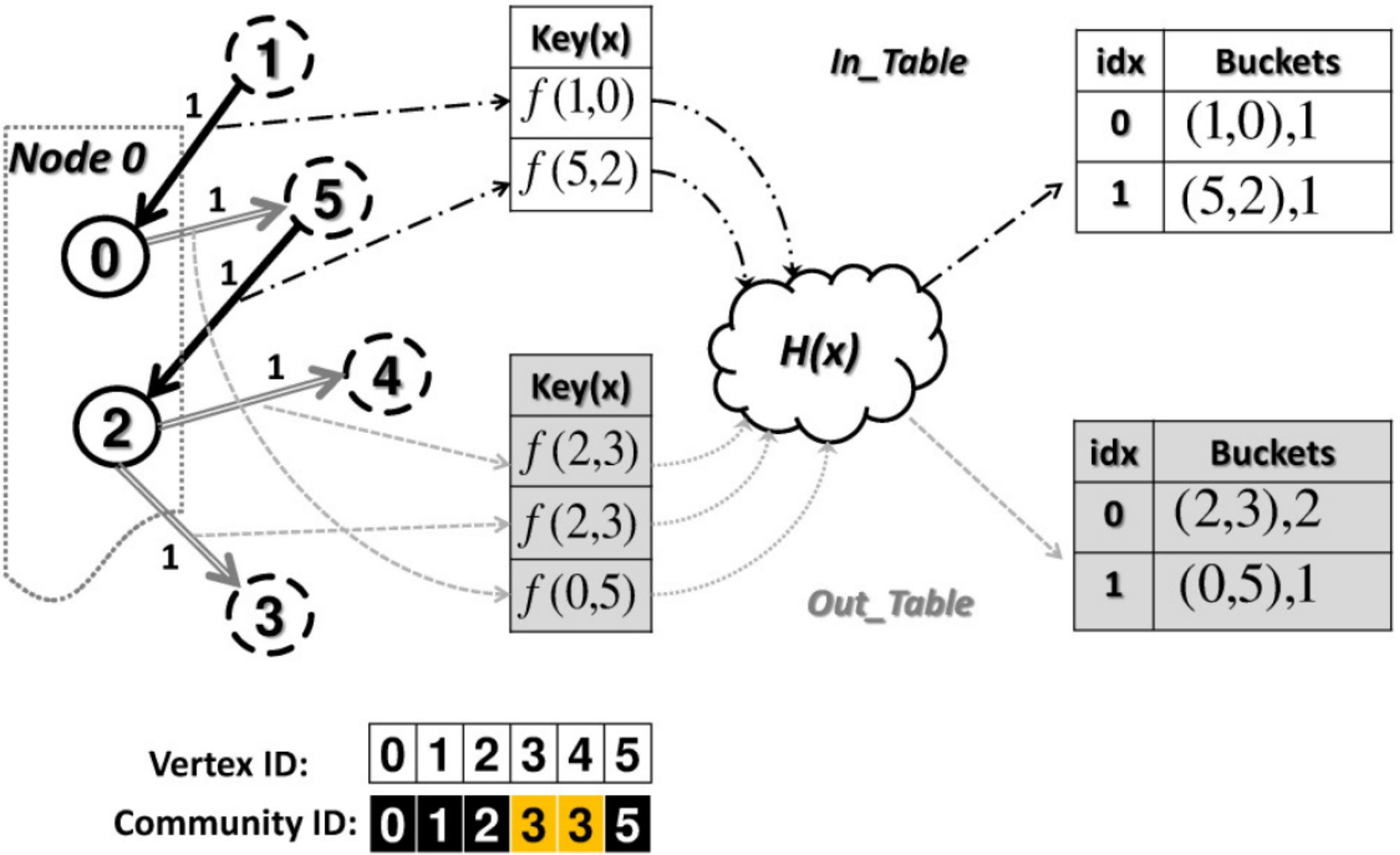
Σ^c_{tot} — the sum of the weights from edges incident to any vertex in c,

$$\Sigma^c_{\text{tot}} = \sum_{\substack{u \in c \text{ or } v \in c \\ e(u,v) \in E}} w_{u,v}$$

$w(u)$ — the sum of the weights of the edges incident to vertex u
$w_{u \to c}$ — the sum of the weights of the edges from vertex u to vertices in community c

$$w_{u \to c} = \sum_{v \in c} w_{u,v}$$

Hash-based data organization

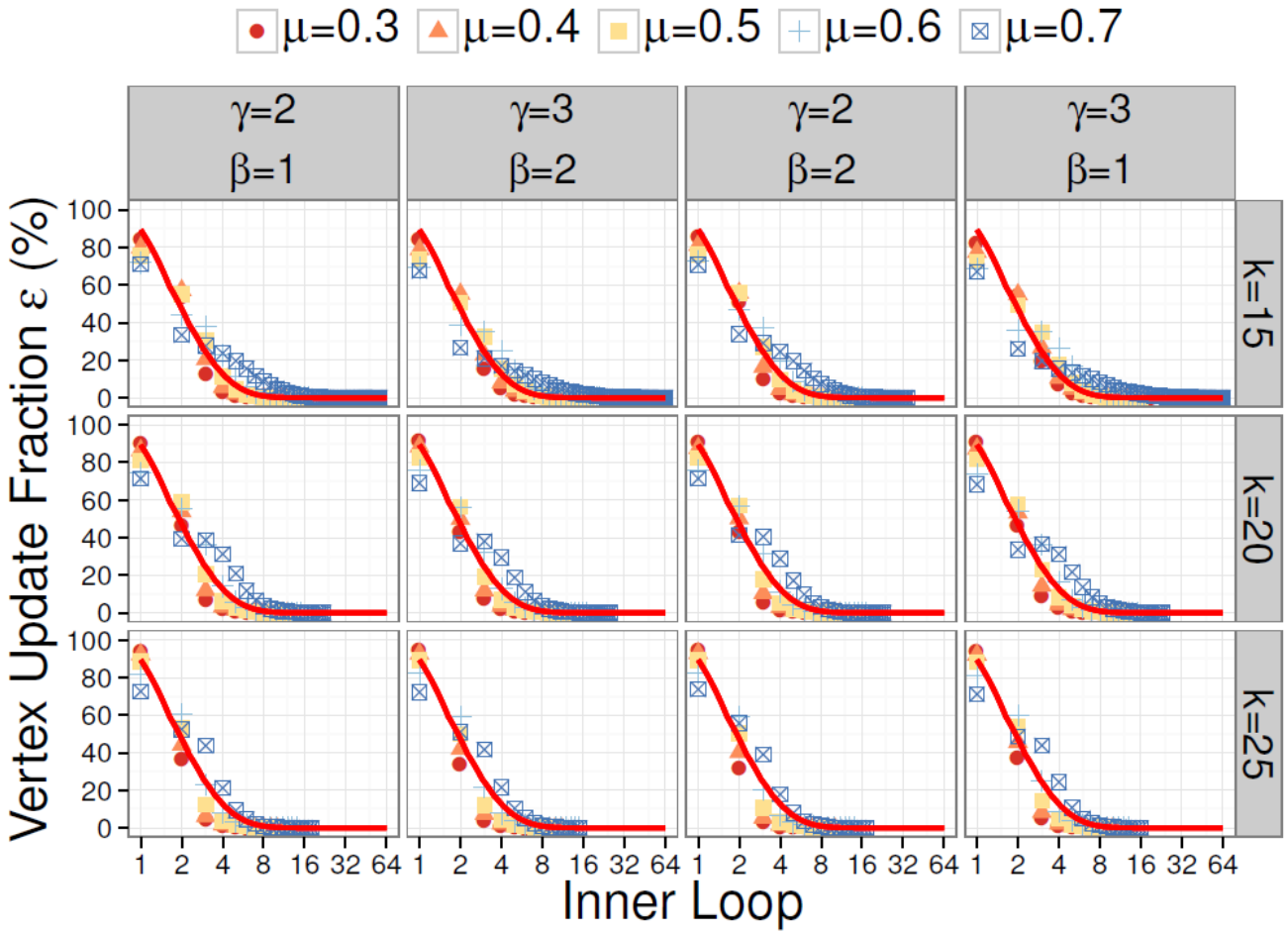


Novel convergence heuristic

The fraction of vertices updated during each iteration of the inner loop:

$$\varepsilon = p_1 \cdot e^{-p_2 \cdot \text{iter}} \tag{7}$$

Regression analysis and the dynamical threshold for the LFR benchmark:

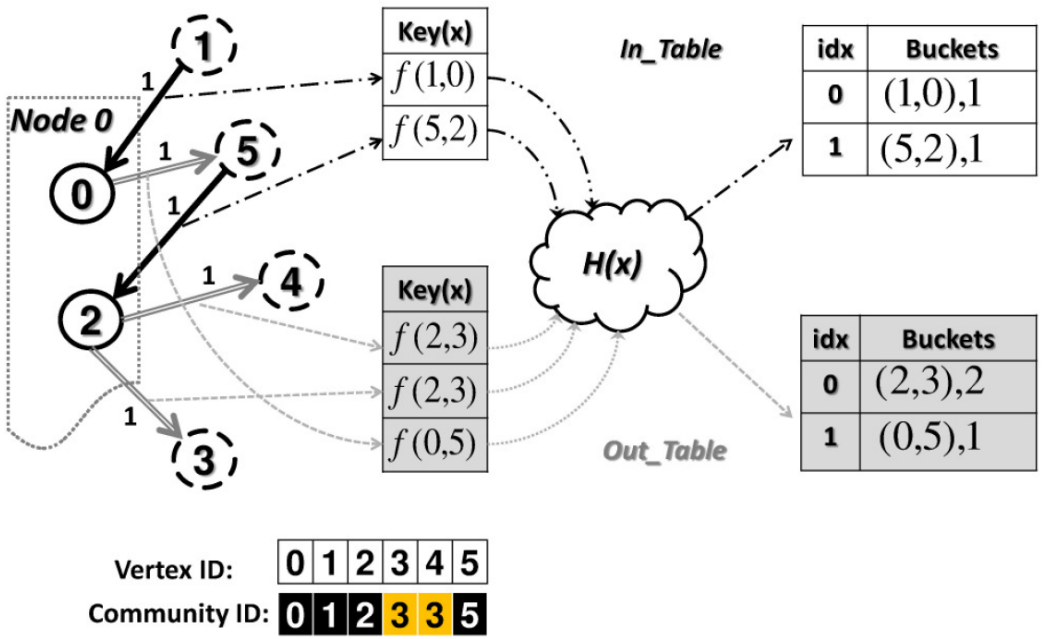


Parallel Louvain algorithm

Algorithm 2: Parallel Louvain Algorithm.

Input: p: process;
$G=(V,E)$: graph representation;
V_p: vertices owned by process p.
Output: C: community sets at each level;
Q: modularity at each level.
Var: C_p: community set owned by process p;
In_Table_p: In_Table owned by process p;
Out_Table_p: Out_Table owned by process p.

1 $In\_Table_p \leftarrow ((v,u),w_{v,u}),\ \forall u \in V_p,\ \forall e(v,u) \in E$;
2 $Out\_Table_p \leftarrow \emptyset$;
3 $C_p \leftarrow \{\{u\}\},\ \forall u \in V_p$;
4 **Loop** ***outer***
5 STATE PROPAGATION() ;
6 *// Refine vertices' community set.*
7 REFINE() ;
8 print C_p and Q ;
9 *// Reconstruct the Graph.*
10 GRAPH RECONSTRUCTION() ;
11 **if** *No improvement on the modularity* **then**
12 **exit** ***outer*** **Loop**;

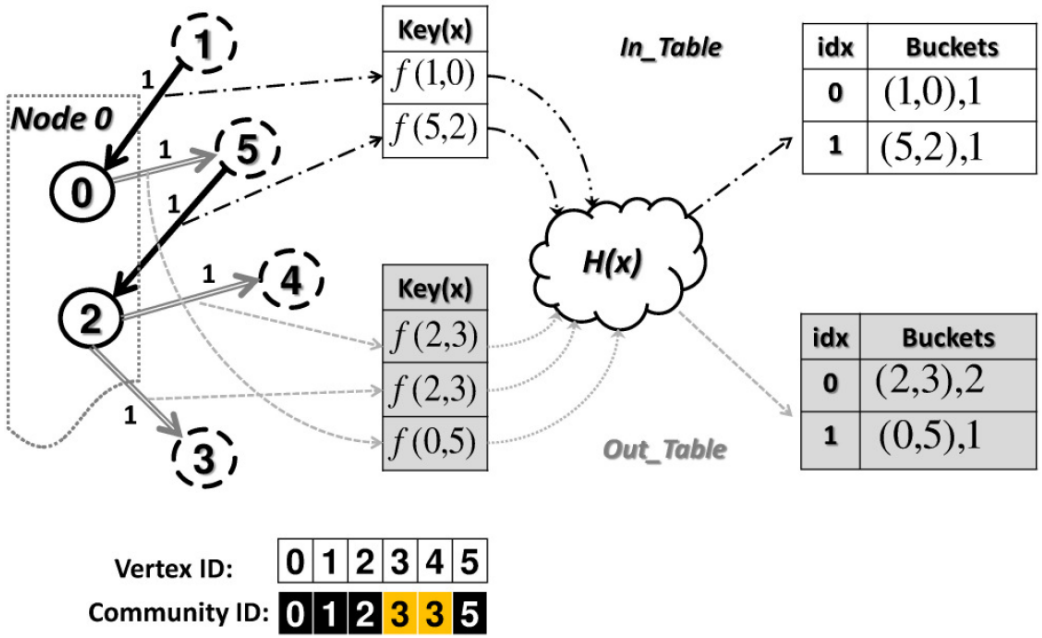


Community state propagation

Update of Out_Table

Algorithm 3: Community State Propagation.

```
1  function STATE PROPAGATION
2  begin
3      // Scan In_Table and send messages.
4      for ((v,u),w) ∈ In_Table_p do
5          send ((v,c),w) to process p'(v ∈ V_p', u ∈ c) ;
6      // Update Out_Table.
7      for ((u,c),w) received do
8          if ∃((u,c),w') ∈ Out_Table_p then
9              w' ← w' + w ;
10         else
11             place the triple with linear probing ;
```

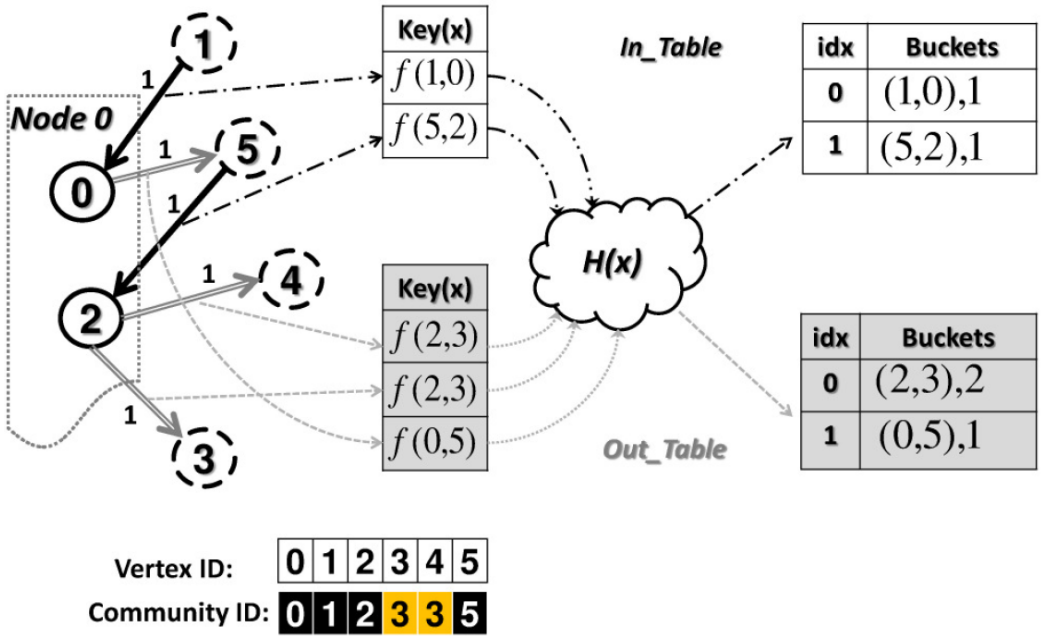


Community refinement

Algorithm 4: Refine.

Var: m_u: vertex u's maximum modularity gain;
$\hat{c_u}$: vertex u's best candidate community set.

```
1  function REFINE
2  begin
3      Loop inner
4          m_u ← 0, ĉ_u ← c, ∀u ∈ V_p, u ∈ c ;
5          // FIND BEST COMMUNITY.
6          for ((u,c'),w) ∈ Out_Table_p, u ∈ c do
7              if ΔQ_{u→c'} ≥ m_u then
8                  ĉ_u ← c' ;
9                  m_u ← ΔQ_{u→c'} ;
10         // Obtain updated threshold.
11         Compute threshold ΔQ̂ from Equation 7 ;
12         // UPDATE COMMUNITY INFORMATION.
13         ∀u ∈ V_p  if m_u ≥ ΔQ̂ then
14             Σ_tot^{ĉ_u} ← Σ_tot^{ĉ_u} + w(u) ;  Σ_tot^c ← Σ_tot^c − w(u) ;
15             ĉ_u ← ĉ_u ∪ {u} ;  c ← c − {u} ;
16         STATE PROPAGATION() ;
17         // Calculate Σ_in.
18         for ((u,c),w) ∈ Out_Table_p do
19             if (u ∈ c) then
20                 Σ_in^c ← Σ_in^c + w ;
21         // Calculate community set and modularity.
22         Q_p ← 0 ;
23         for c ∈ C_p do
24             Q_p ← Q_p + Σ_in^c/2m − (Σ_tot^c/2m)^2 ;
25         Q ← Allreduce(Q_p) ;
26         if No improvement on the modularity then
27             exit inner Loop;
28         Reset Out_Table_p and Σ_in^c ;
29         V_p ← C_p ;
```



Algorithm 3: Community State Propagation.

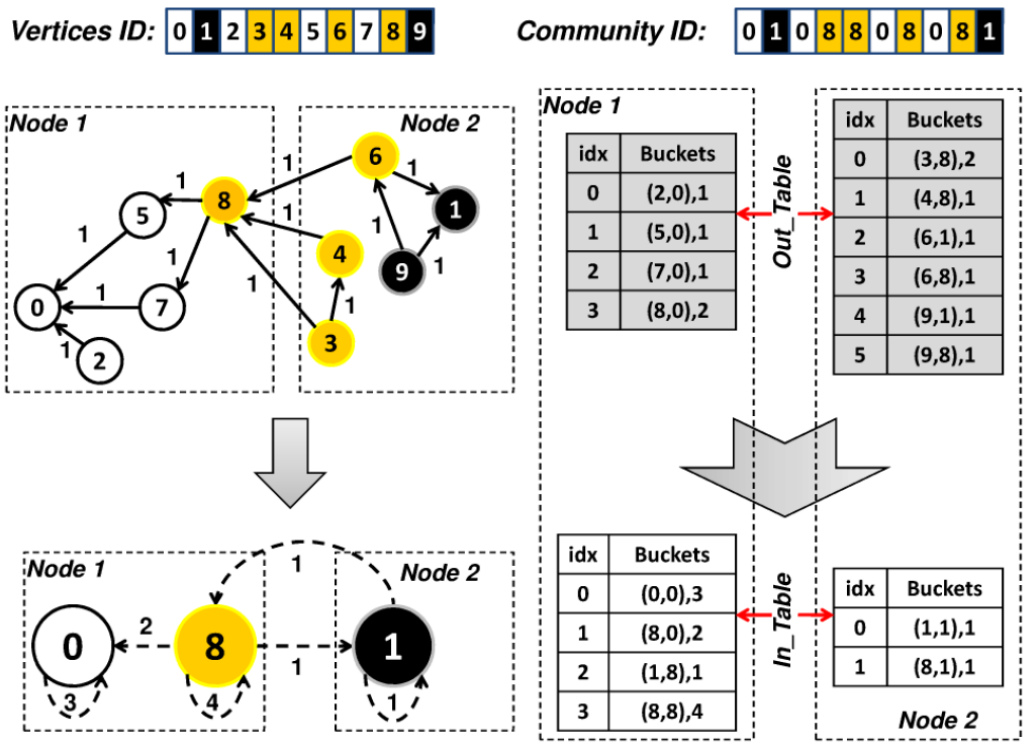
```
1  function STATE PROPAGATION
2  begin
3      // Scan In_Table and send messages.
4      for ((v,u),w) ∈ In_Table_p do
5          send ((v,c),w) to process p'(v ∈ V_p', u ∈ c) ;
6      // Update Out_Table.
7      for ((u,c),w) received do
8          if ∃((u,c),w') ∈ Out_Table_p then
9              w' ← w' + w ;
10         else
11             place the triple with linear probing ;
```

Graph reconstruction

Reconstruction of In_Table

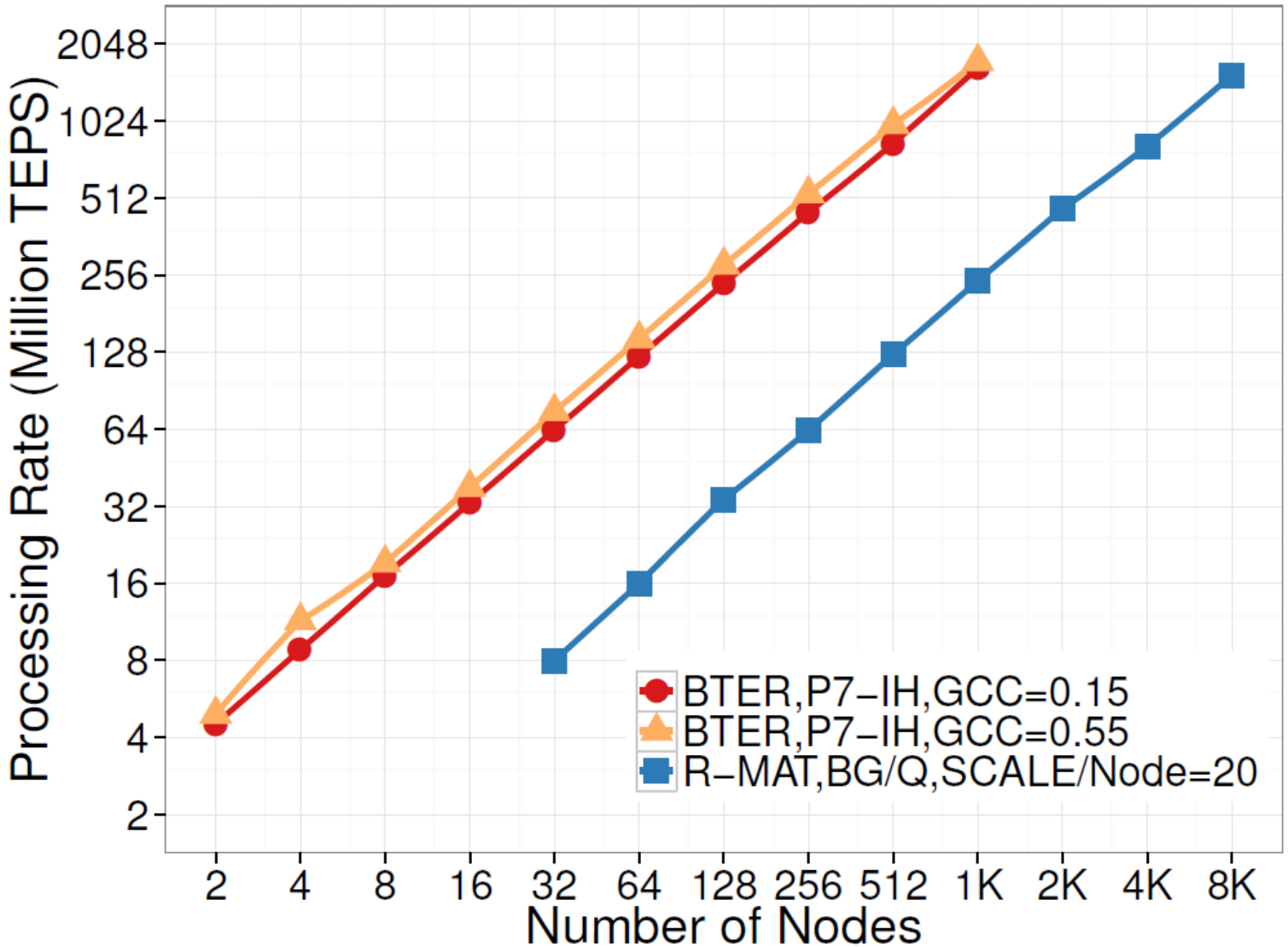
Algorithm 5: Graph Reconstruction.

```
1  function GRAPH RECONSTRUCTION
2  begin
3      Reset In_Table_p ;
4      for ((u,c),w) ∈ Out_Table_p do
5          send ((c',c),w) to process p'(c ∈ C_p', u ∈ c')
6      // Reconstruct In_Table.
7      for ((v,u),w) received do
8          if ∃((v,u),w') ∈ In_Table_p then
9              w' ← w' + w ;
10         else
11             place the triple with linear probing ;
12     Reset Out_Table_p ;
```

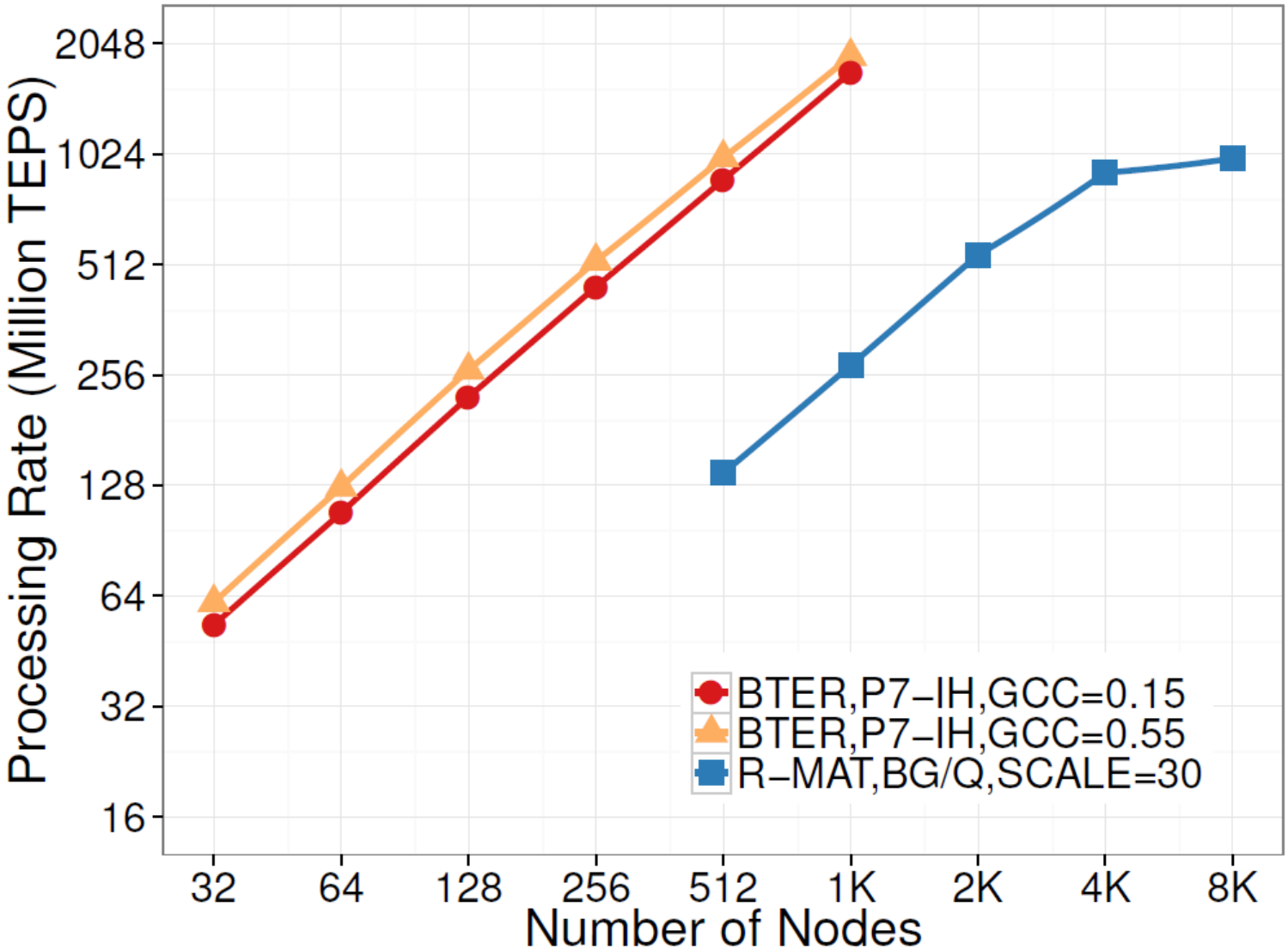


Scalability analysis: weak scaling

P7-IH, BTER: 2^{22} vertices per node, average degree of 32



Scalability analysis: strong scaling



Outline

Conclusion

- Highly scalable parallel Louvain algorithm for distributed memory systems
- Preserves/slightly improves
  - the convergence properties
  - the overall modularity
  - the quality of the detected communities
- A novel implementation strategy to store and process dynamic graphs
- Validation on a wide variety of real-world social graphs
- Scalability:
  - BTER, 4B vertices/138B edges — 1k P7-IH nodes (32k threads)
  - R-MAT, 8B vertices/138B edges — 8k BG/Q nodes (512k threads)

Outline

References

M. E. J. Newman, M. Girvan.
Finding and Evaluating Community Structure in Networks.
Phys. Rev. E, 69(2):026113, Feb. 2004.

V. Blondel et al.
Fast unfolding of communities in large networks.
J. Stat. Mech., P10008, 2008.

X. Que et al.
Scalable Community Detection with the Louvain Algorithm.
Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International, Hyderabad, India, 25-29 May 2015, pp. 28-37.

Outline

Modularity maximization techniques

- *Greedy optimization* — applies different approaches to merge vertices into communities for higher modularity
- *Simulated annealing* — adopts a probabilistic procedure for global optimization on modularity
- *Extremal optimization* — is a heuristic search procedure
- *Spectral optimization* — uses eigenvalues and eigenvectors of a special matrix for modularity optimization

Disclaimer

All the information, representations, statements, opinions and proposals in this document are correct and accurate to the best of our present knowledge but are not intended (and should not be taken) to be contractually binding unless and until they become the subject of separate, specific agreement between us.
Any IBM Machines provided are subject to the Statements of Limited Warranty accompanying the applicable Machine.
Any IBM Program Products provided are subject to their applicable license terms.
Nothing herein, in whole or in part, shall be deemed to constitute a warranty.
IBM products are subject to withdrawal from marketing and or service upon notice, and changes to product configurations, or follow-on products, may result in price changes.
Any references in this document to "partner" or "partnership" do not constitute or imply a partnership in the sense of the Partnership Act 1890.
IBM is not responsible for printing errors in this proposal that result in pricing or information inaccuracies.

Правовая информация

Упоминание в этой публикации продуктов или услуг корпорации IBM не означает, что IBM предполагает предоставлять их во всех странах, в которых осуществляет свою деятельность, информация о предоставлении продуктов или услуг может быть изменена без уведомления. За самой свежей информацией о продуктах и услугах компании IBM, предоставляемых в Вашем регионе, следует обращаться в ближайшее торговое представительство IBM или к авторизованным бизнес-партнерам.
Все заявления относительно намерений и перспективных планов IBM могут быть изменены без уведомления.
Информация о продуктах третьих фирм получена от производителей этих продуктов или из опубликованных анонсов указанных продуктов. IBM не тестировала эти продукты и не может подтвердить производительность, совместимость, или любые другие заявления относительно продуктов третьих фирм. Вопросы о возможностях продуктов третьих фирм следует адресовать поставщику этих продуктов.
Информация может содержать технические неточности или типографические ошибки. В представленную в публикации информацию могут вноситься изменения, эти изменения будут включаться в новые редакции данной публикации. IBM может вносить изменения в рассматриваемые в данной публикации продукты или услуги в любое время без уведомления.
Любые ссылки на узлы Web третьих фирм приведены только для удобства и никоим образом не служат поддержкой этим узлам Web. Материалы на указанных узлах Web не являются частью материалов для данного продукта IBM.