GraBi: Communication-Efficient and Workload-Balanced Partitioning for Bipartite Graphs

[1]Feng Sheng, [1]Qiang Cao, [2]Hong Jiang, and [1]Jie Yao

[1]Huazhong University of Science and Technology

[2]University of Texas at Arlington

Outline

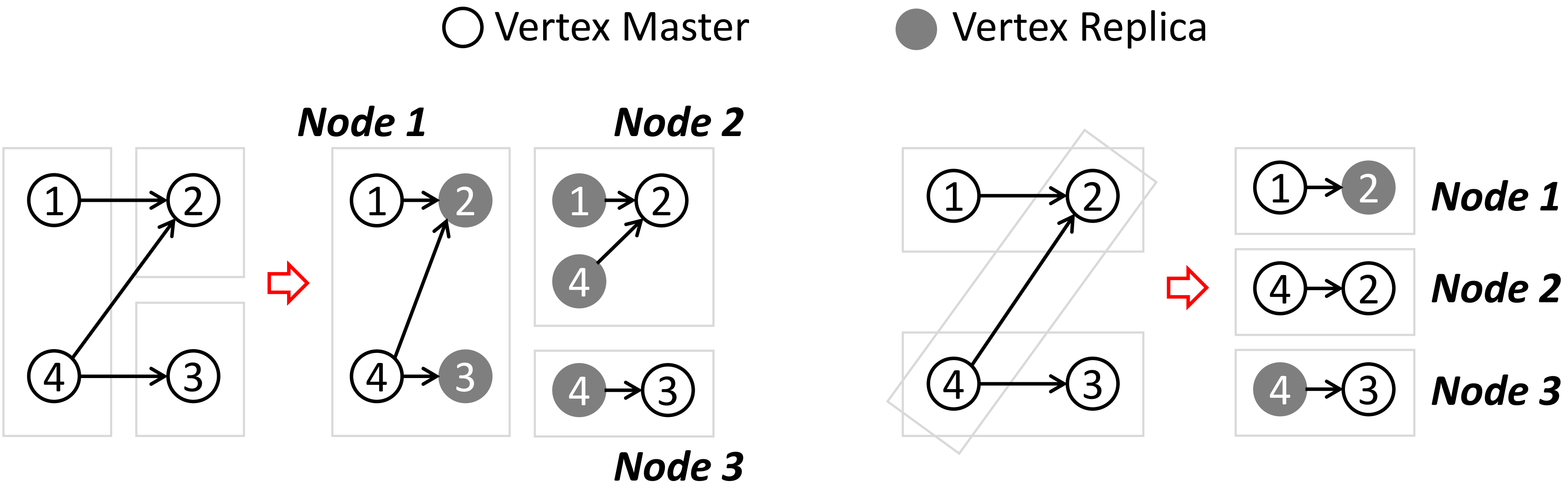
- ☑ Background
- ☐ Motivation
- ☐ Design of GraBi
  - ➢ Vertical Partitioning: Vertex-vector Chunking
  - ➢ Horizontal Partitioning: Vertex-chunk Assignment
- ☐ Evaluation
- ☐ Conclusion

Graph Partitioning

Graph partitioning distributes vertices and edges over computing nodes.

Graph Partitioning

Graph partitioning distributes vertices and edges over computing nodes.



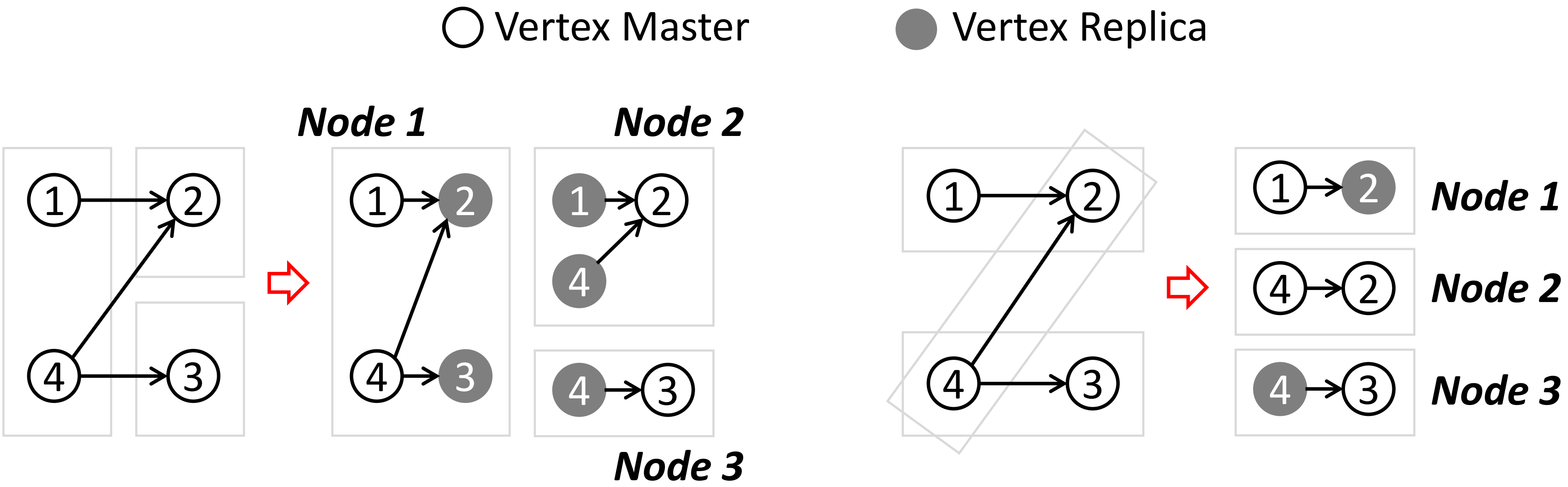
(a) Edge-cut

(b) Vertex-cut

- Edge-cut equally distributes vertices among nodes.
- Vertex-cut equally distributes edges among nodes.

Graph Partitioning

Graph partitioning distributes vertices and edges over computing nodes.



(a) Edge-cut

(b) Vertex-cut

- Edge-cut equally distributes vertices among nodes.
- Vertex-cut equally distributes edges among nodes.

- replication factor (λ): the average number of replicas per vertex.

Bipartite graphs & MLDM algorithms

- **Bipartite graphs**

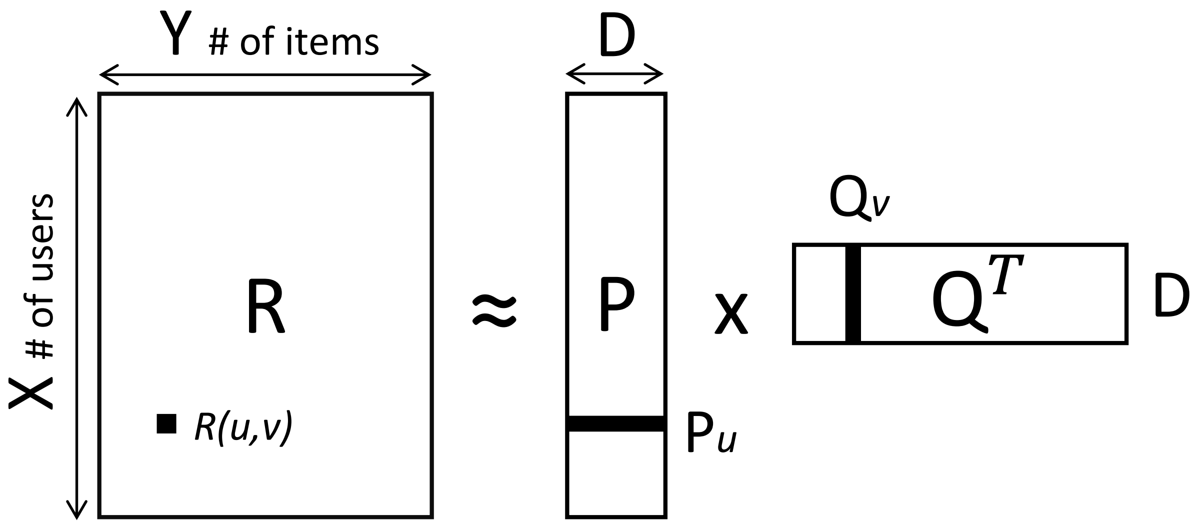
➢ Vertices are separated into two disjoint subsets.
➢ Every edge connects one vertex each from the two subsets.

Bipartite graphs & MLDM algorithms

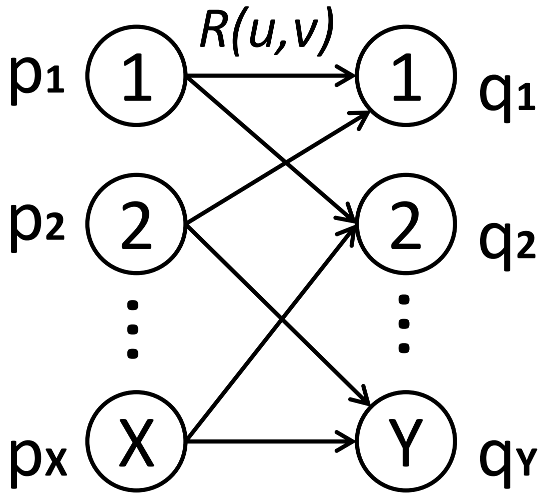
- **Bipartite graphs**
  - ➢ Vertices are separated into two disjoint subsets.
  - ➢ Every edge connects one vertex each from the two subsets.
- **Machine Learning and Data Mining (MLDM) algorithms**
  - ➢ Bipartite graphs have been widely used in MLDM applications .

Bipartite graphs & MLDM algorithms

- **Bipartite graphs**
  - ➢ Vertices are separated into two disjoint subsets.
  - ➢ Every edge connects one vertex each from the two subsets.
- **Machine Learning and Data Mining (MLDM) algorithms**
  - ➢ Bipartite graphs have been widely used in MLDM applications.



(a) View of Matrix



(b) View of Graph

Observations

- **Observation 1**: The vertex value in MLDM algorithms is a multi-element vector.

Observations

- **Observation 1**: The vertex value in MLDM algorithms is a multi-element vector.

- The authors of $CUBE^{[1]}$ associate each vertex with a vector of up to 128 elements.
- The users of $PowerGraph^{[2]}$ can configure each vertex value as a vector of thousands of elements

[1] M. Zhang, Y. Wu, K. Chen, et al. Exploring the Hidden Dimension in Graph Processing. In OSDI 2016.
[2] J. E. Gonzalez, Y. Low, H. Gu, et al. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In OSDI 2012.

Observations

- **Observation 1**: The vertex value in MLDM algorithms is a multi-element vector.

➢ The authors of $CUBE^{[1]}$ associate each vertex with a vector of up to 128 elements.
➢ The users of $PowerGraph^{[2]}$ can configure each vertex value as a vector of thousands of elements

- **Observation 2**: The sizes of two vertex-subsets in a bipartite graph can be highly lopsided.

[1] M. Zhang, Y. Wu, K. Chen, et al. Exploring the Hidden Dimension in Graph Processing. In OSDI 2016.
[2] J. E. Gonzalez, Y. Low, H. Gu, et al. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In OSDI 2012.

Observations

- **Observation 1**: The vertex value in MLDM algorithms is a multi-element vector.

- The authors of $CUBE$[1] associate each vertex with a vector of up to 128 elements.
- The users of $PowerGraph$[2] can configure each vertex value as a vector of thousands of elements

- **Observation 2**: The sizes of two vertex-subsets in a bipartite graph can be highly lopsided.

- In $Netflix$[3], the number of users is about 27x that of movies.
- In $English\ Wikipedia$[4], the number of articles is about 98x that of words.

[1] M. Zhang, Y. Wu, K. Chen, et al. Exploring the Hidden Dimension in Graph Processing. In OSDI 2016.
[2] J. E. Gonzalez, Y. Low, H. Gu, et al. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In OSDI 2012.
[3] http://www.netflixprize.com/community/viewtopic.php?pid=9857
[4] https://dumps.wikimedia.org/

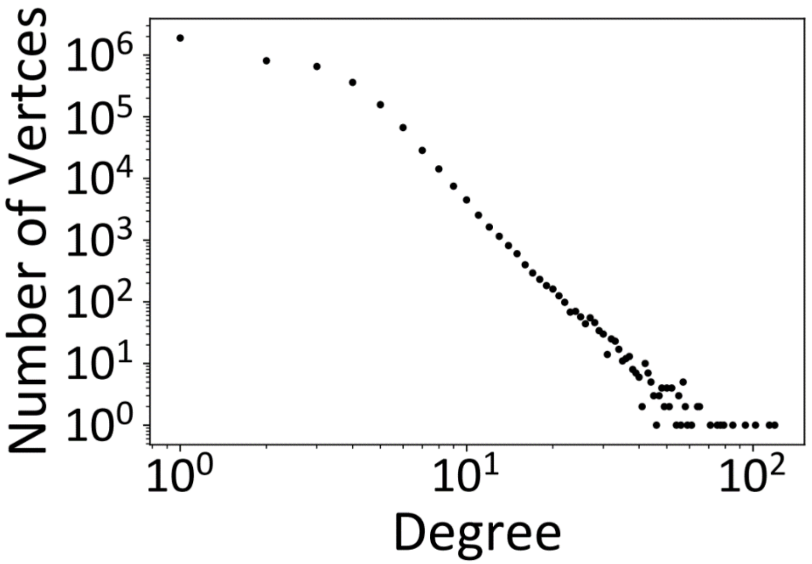
Observations

- **Observation 3:** Within a vertex-subset, the vertices usually exhibit power-law degree distribution

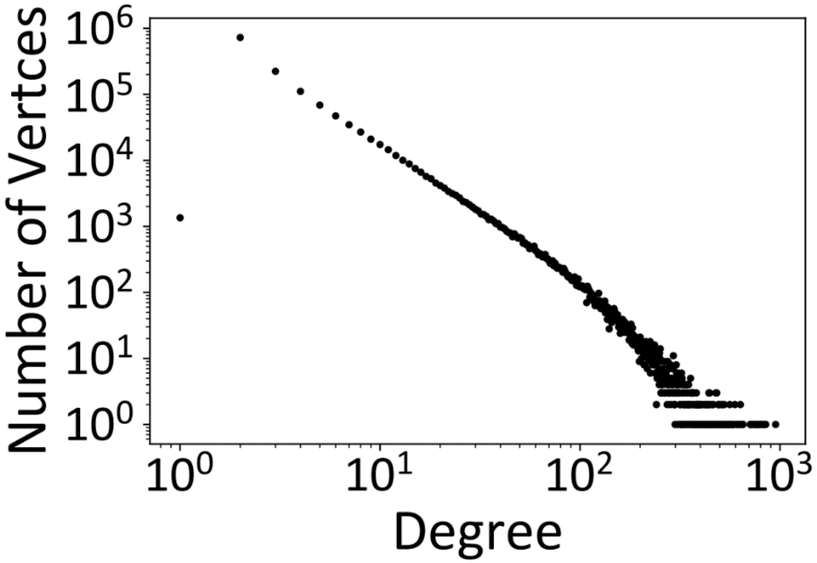
Observations

- **Observation 3:** Within a vertex-subset, the vertices usually exhibit power-law degree distribution

➢ Both the two vertex-subsets in $DBLP^{[1]}$ exhibit power-law degree distribution.



(a) Author Degree Distribution



(b) Publication Degree Distribution

[1] https://dumps.wikimedia.org/

Opportunities

- **Observation 1**: The vertex value in MLDM algorithms is a multi-element vector.

⇨ *Each vertex vector can be divided into multiple sub-vectors.*

Opportunities

- **Observation 1**: The vertex value in MLDM algorithms is a multi-element vector.

⇨ *Each vertex vector can be divided into multiple sub-vectors.*

- **Observation 2**: The sizes of two vertex-subsets in a bipartite graph can be highly lopsided

⇨ *The two vertex-subsets can be processed with different priorities.*

Opportunities

- **Observation 1**: The vertex value in MLDM algorithms is a multi-element vector.

⇨ *Each vertex vector can be divided into multiple sub-vectors.*

- **Observation 2**: The sizes of two vertex-subsets in a bipartite graph can be highly lopsided

⇨ *The two vertex-subsets can be processed with different priorities.*

- **Observation 3**: Within a vertex-subset, the vertices usually exhibit power-law degree distribution

⇨ *The vertices of different degrees should be* distinguished.

Overview of GraBi

- GraBi is a communication-efficient and workload-balanced partitioning framework for bipartite graphs.

Overview of GraBi

- GraBi is a communication-efficient and workload-balanced partitioning framework for bipartite graphs.

- GraBi comprehensively exploits the above three observations of bipartite graphs and MLDM algorithms.

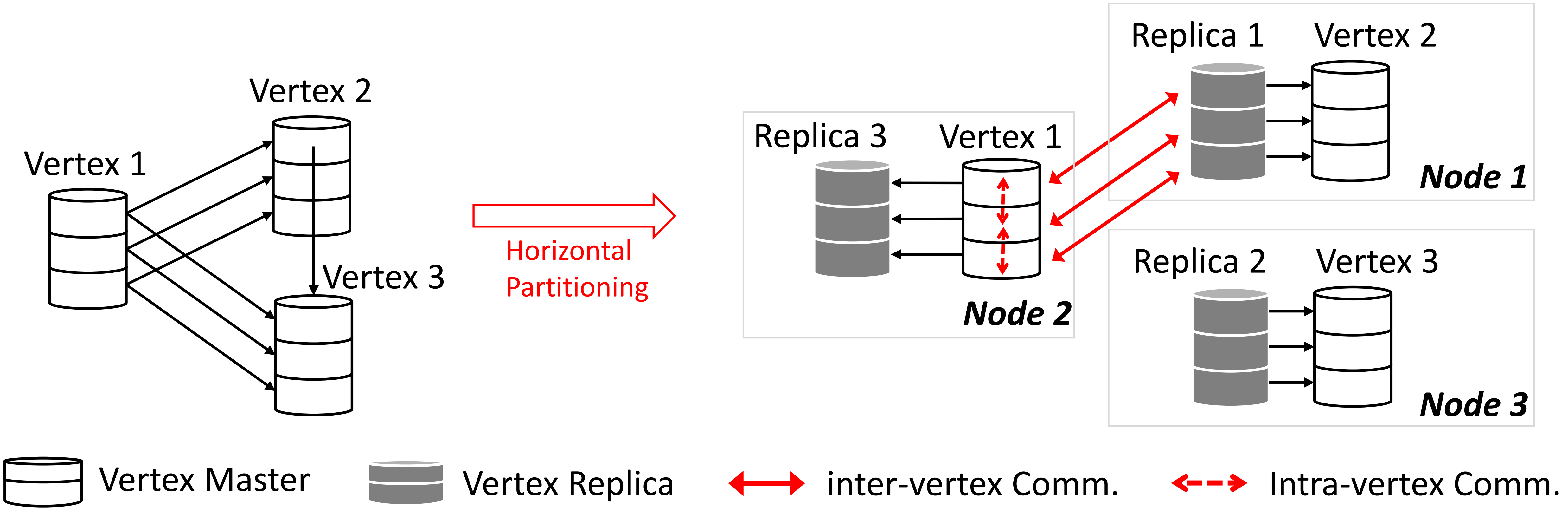
Overview of GraBi

- GraBi is a communication-efficient and workload-balanced partitioning framework for bipartite graphs.

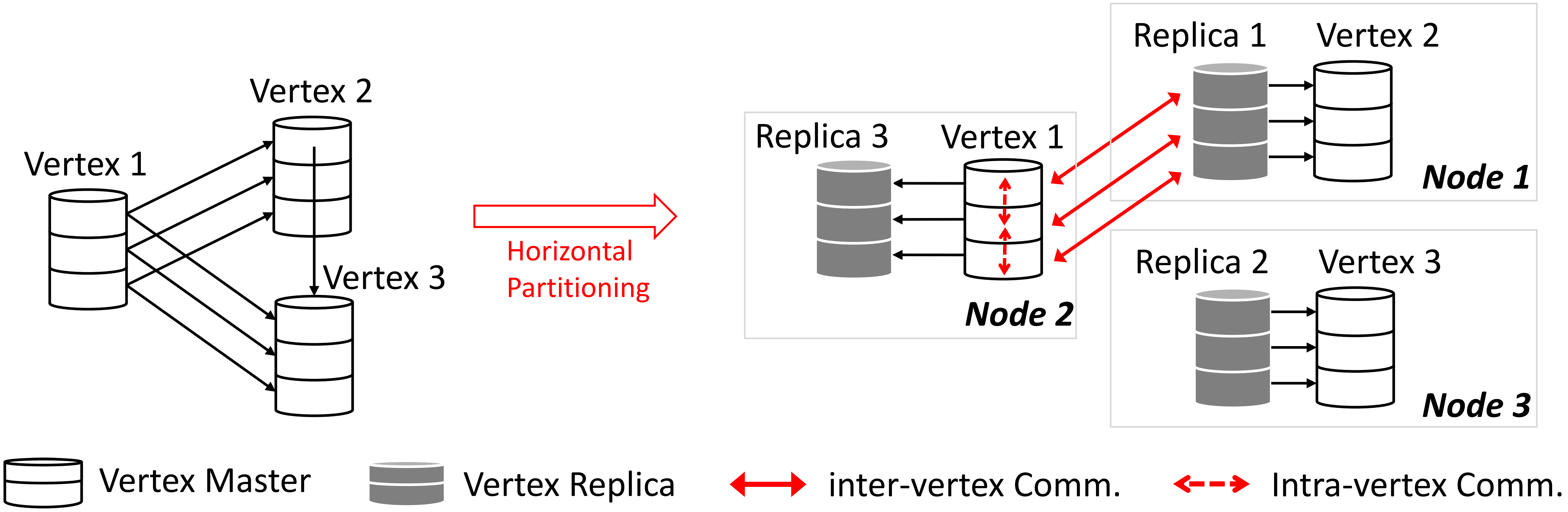
- GraBi comprehensively exploits the above three features of bipartite graphs and MLDM algorithms.

- GraBi partitions a bipartite graph first vertically, and then horizontally, to realize high-quality partitioning.

Vertical Partitioning: Vertex-vector Chunking

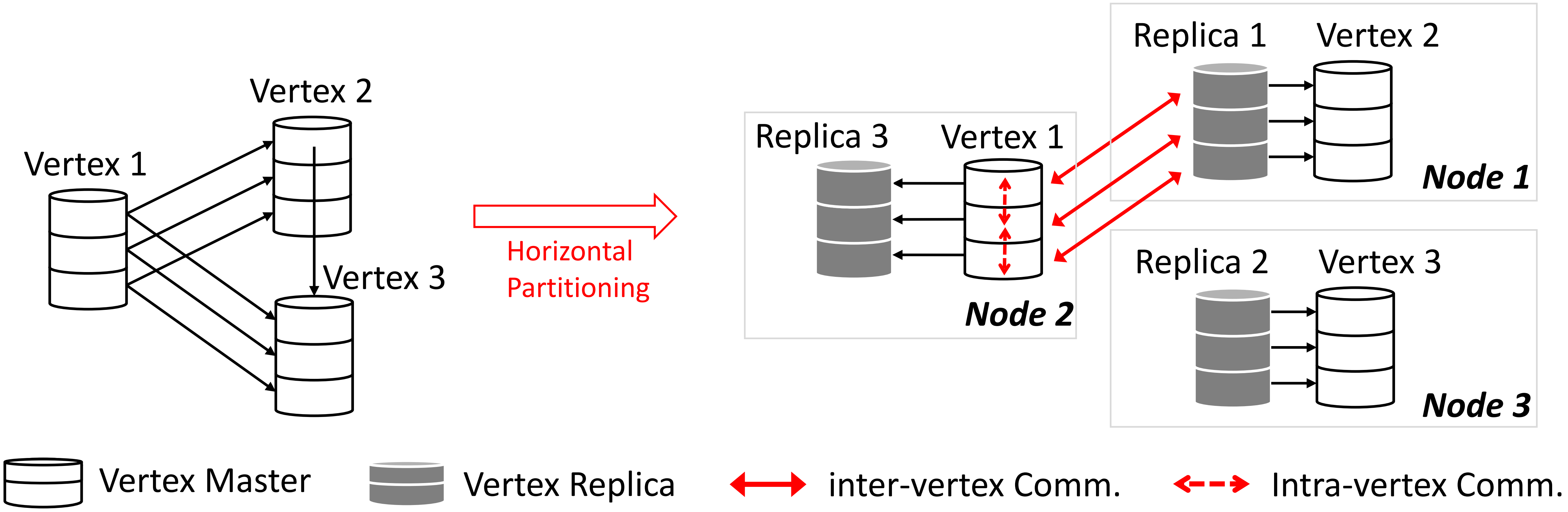


Vertical Partitioning: Vertex-vector Chunking

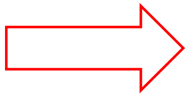


The whole vector of a vertex
is assigned to a computing node.

Vertical Partitioning: Vertex-vector Chunking

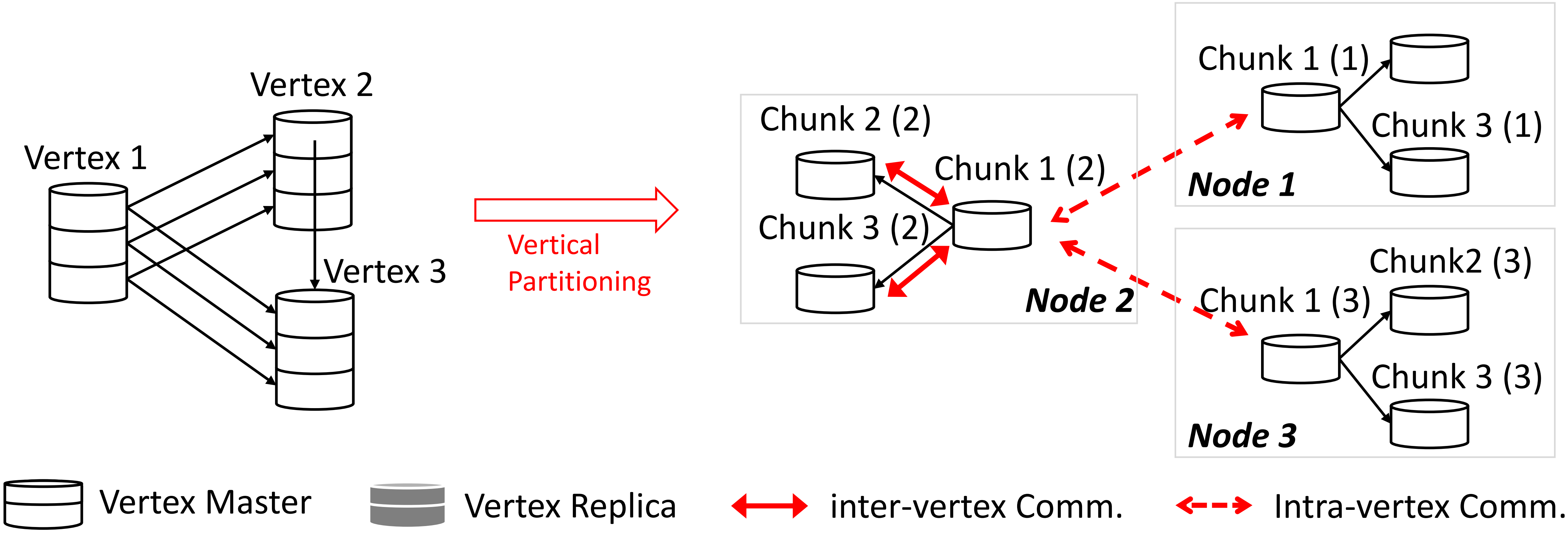


The whole vector of a vertex is assigned to a computing node.

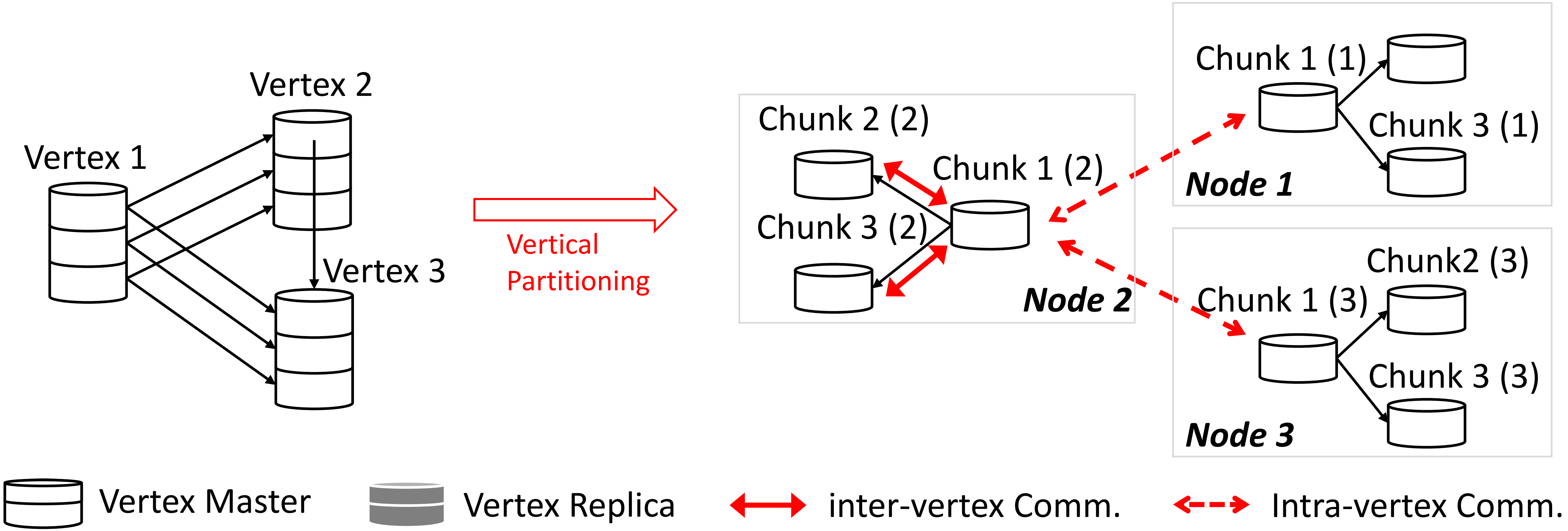


- Inter-vertex Communication happens between computing nodes
- Intra-vertex Communication happens within a computing node

Vertical Partitioning: Vertex-vector Chunking

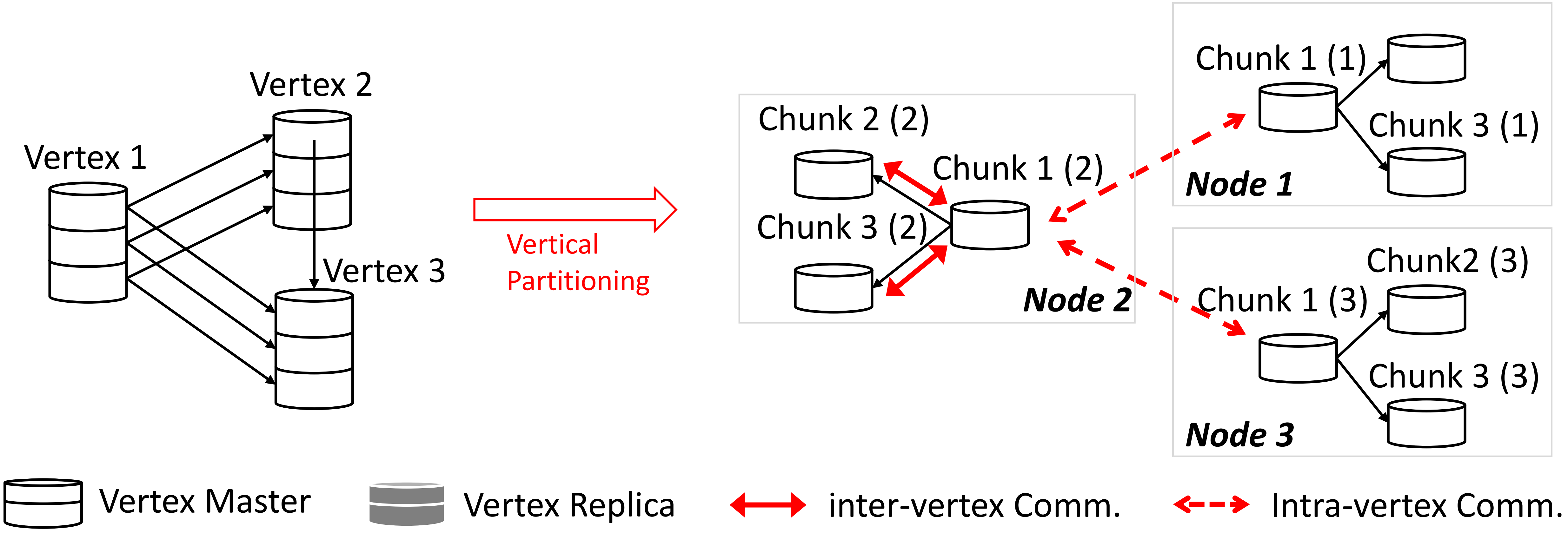


Vertical Partitioning: Vertex-vector Chunking



The whole vector of a vertex is divided into vertex-chunks.

Vertical Partitioning: Vertex-vector Chunking



The whole vector of a vertex is divided into vertex-chunks.

- Inter-vertex Communication happens within a computing node
- Intra-vertex Communication happens between computing nodes

Vertical Partitioning: Vertex-vector Chunking

➢ **Number of Layers *L***

- L=1, horizontal partitioning, inter-vertex communication dominates
- L=N, vertical partitioning, intra-vertex communication dominates

→ $\boldsymbol{L}$ **= 1, 2, … ,*N***

Vertical Partitioning: Vertex-vector Chunking

➢ **Number of Layers *L***

- *L*=1, horizontal partitioning, inter-vertex communication dominates
- *L*=*N*, vertical partitioning, intra-vertex communication dominates

→ L = 1, 2, … ,*N*

L is set as the *Greatest Common Divisor* (GCD) of D and N
D is the number of elements in each vector, *N* is the number of computing nodes.

→ Each vertex-chunk consists of D/L elements, Each layer is assigned to N/L nodes.

Vertical Partitioning: Vertex-vector Chunking

- **Number of Layers *L***

- *L*=1, horizontal partitioning, inter-vertex communication dominates
- *L*=*N*, vertical partitioning, intra-vertex communication dominates

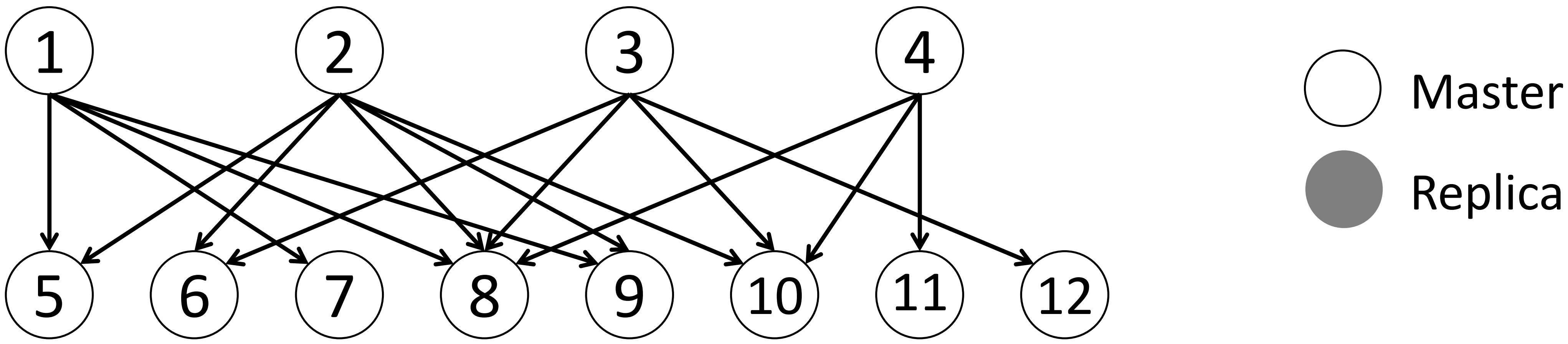
→ L = 1, 2, … ,*N*

L is set as the *Greatest Common Divisor* (GCD) of D and N
D is the number of elements in each vector, *N* is the number of computing nodes.

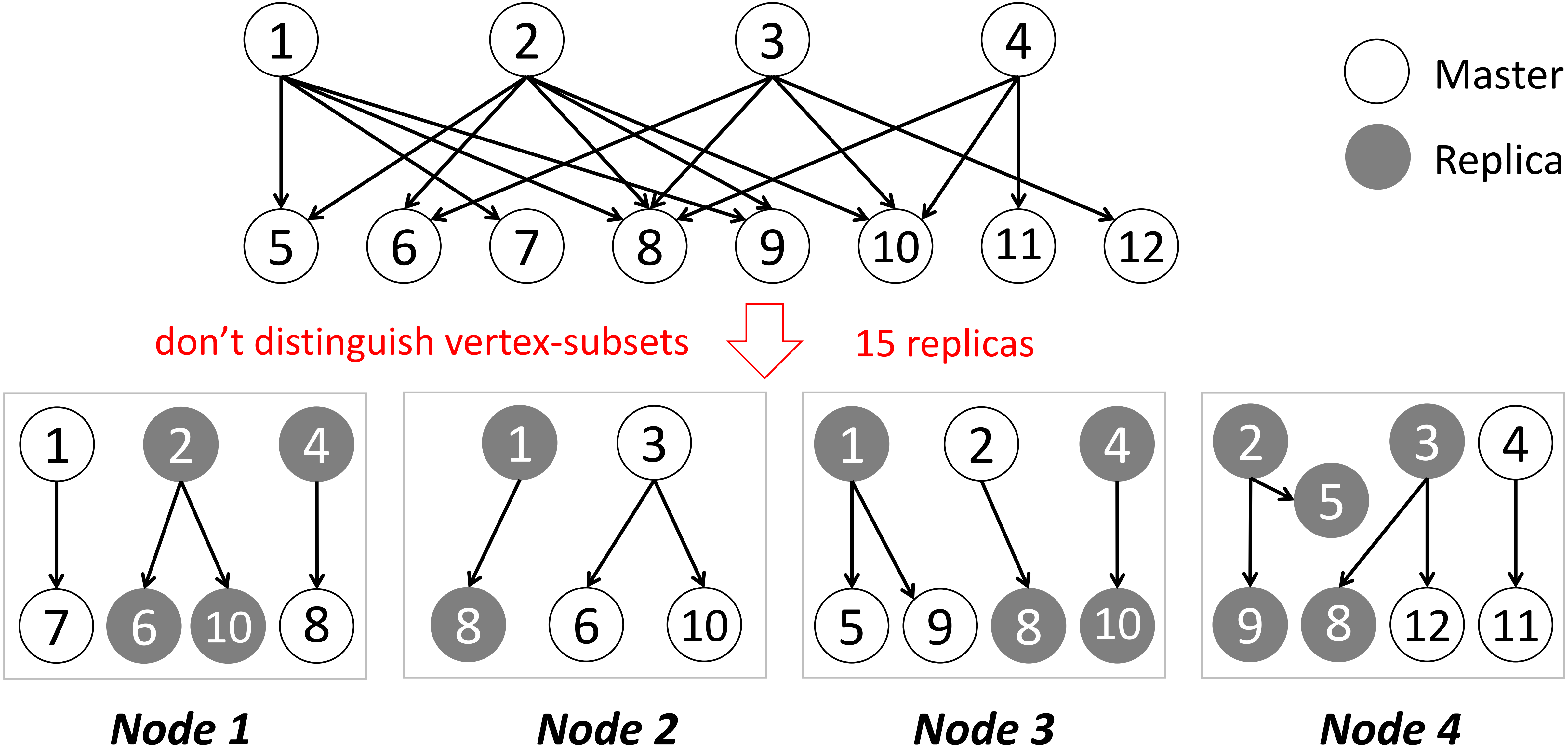
→ Each vertex-chunk consists of D/L elements, Each layer is assigned to N/L nodes.

The vertical partitioning stage, *Vertex-vector Chunking*, is simple element-grouping for every vectored vertex.

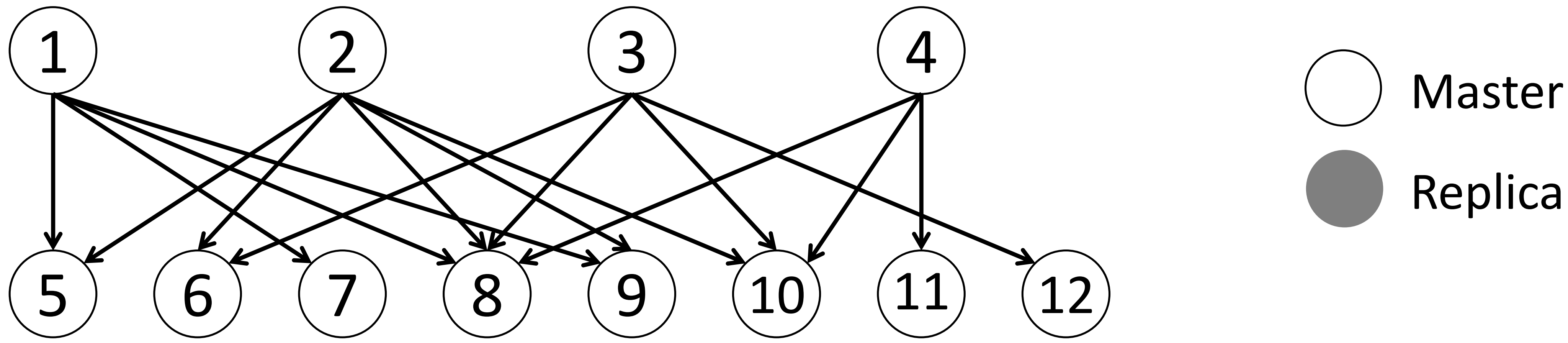
Horizontal Partitioning: Vertex-chunk Assignment



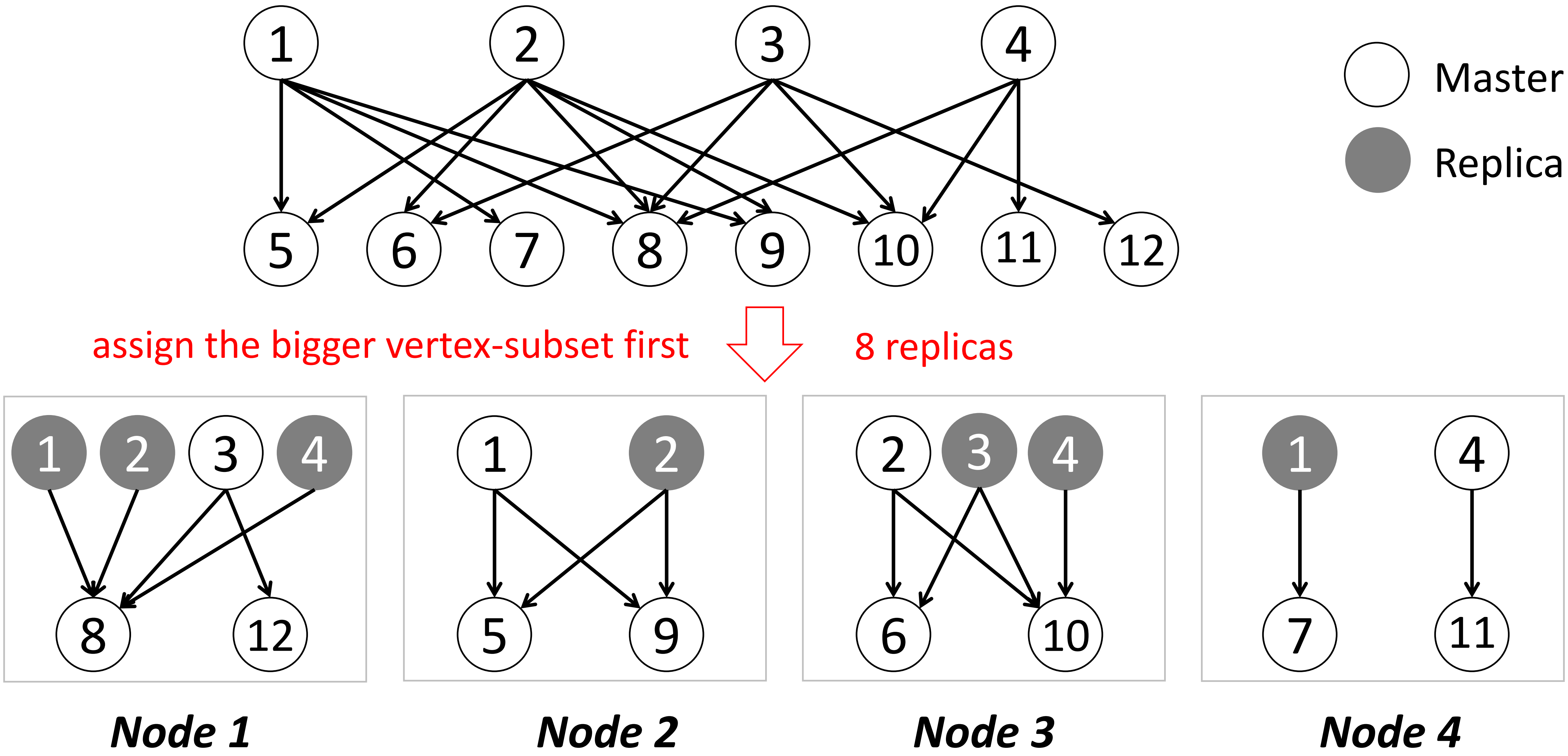
Horizontal Partitioning: Vertex-chunk Assignment



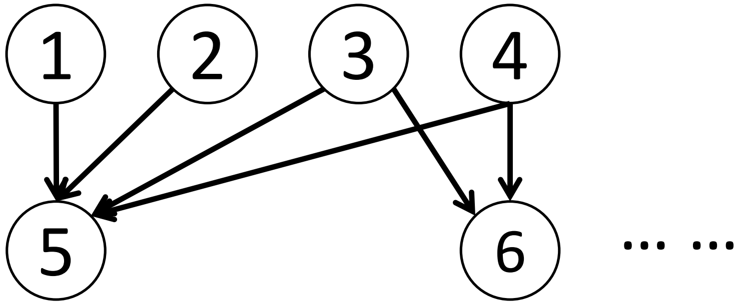
Horizontal Partitioning: Vertex-chunk Assignment



Horizontal Partitioning: Vertex-chunk Assignment



Horizontal Partitioning: Vertex-chunk Assignment



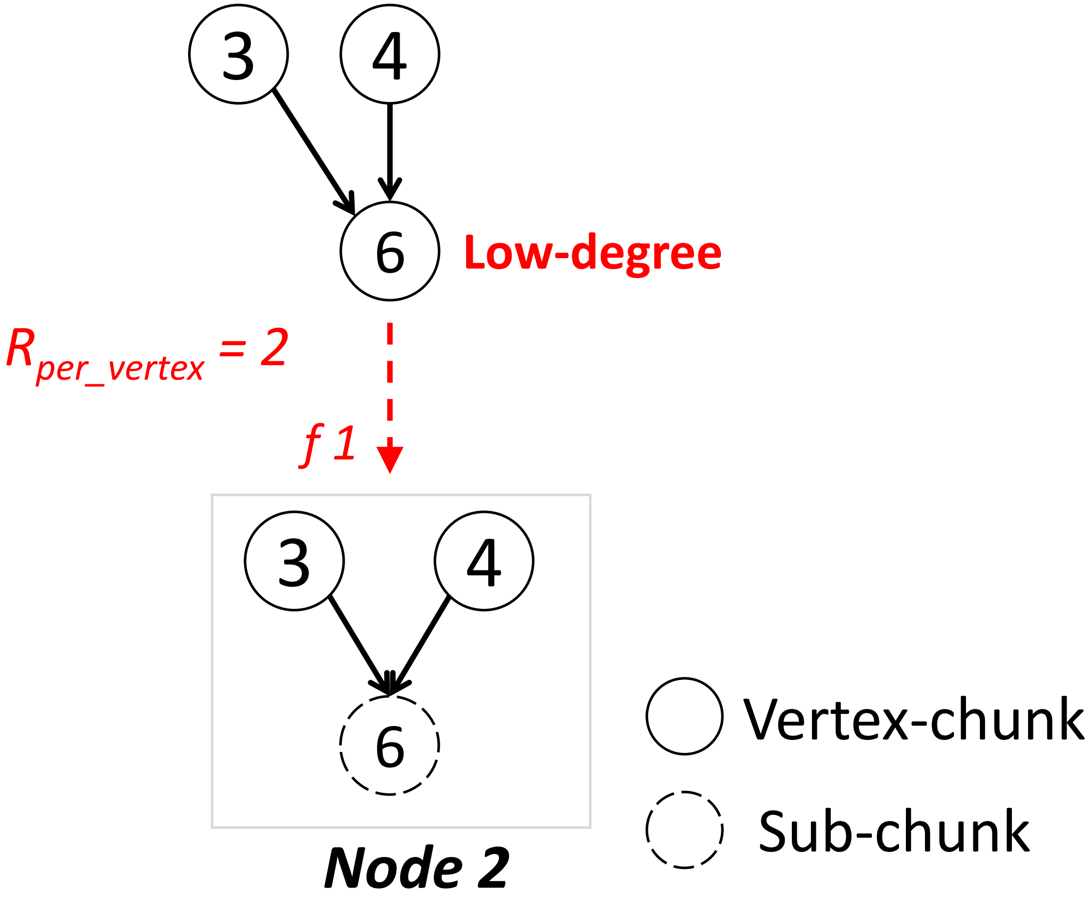
$$R_{per\_vertex} = \alpha \times \left(\frac{|E|}{|U|}\right)$$

α is an amplification factor

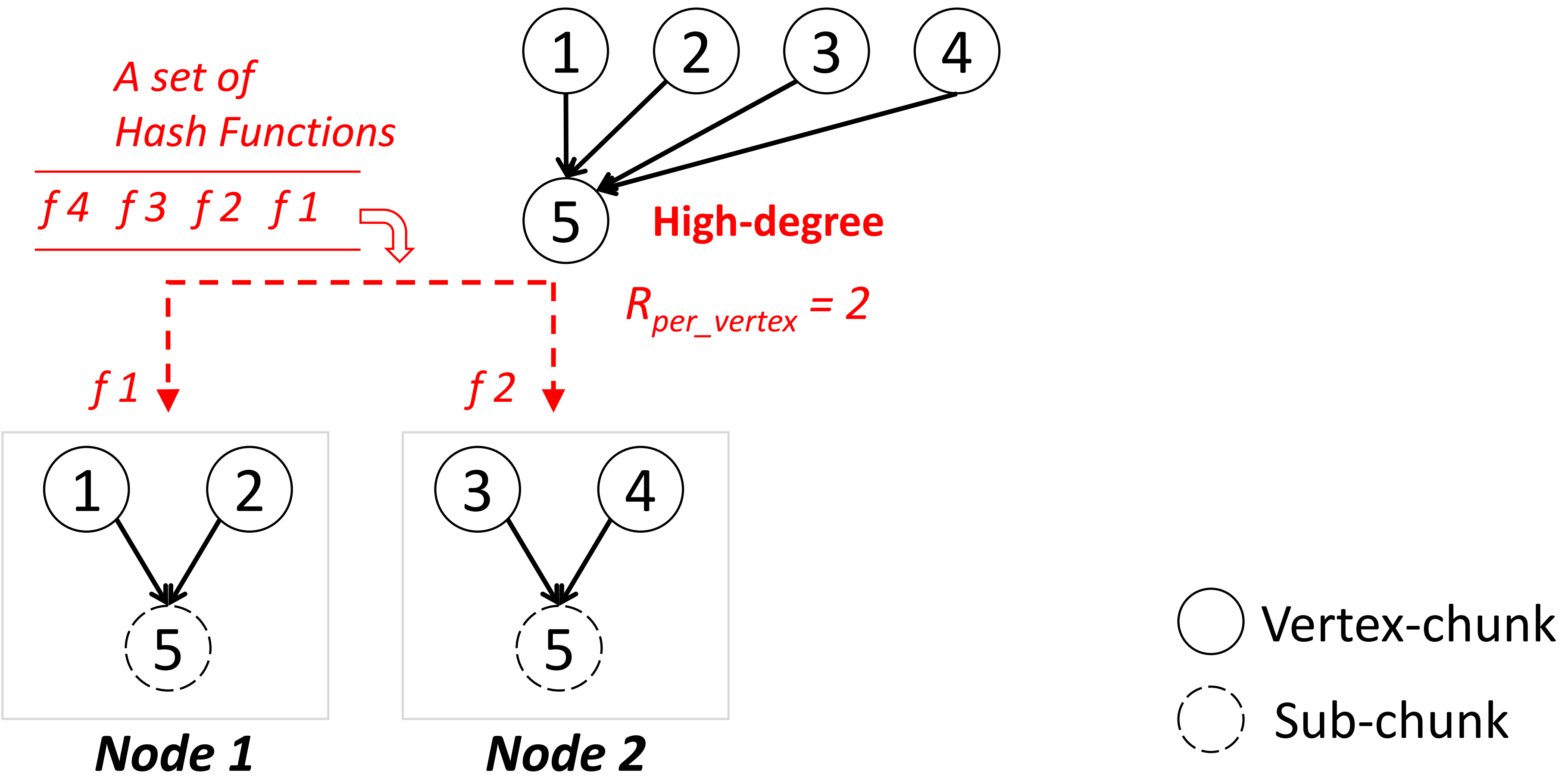
Horizontal Partitioning: Vertex-chunk Assignment

A set of
Hash Functions

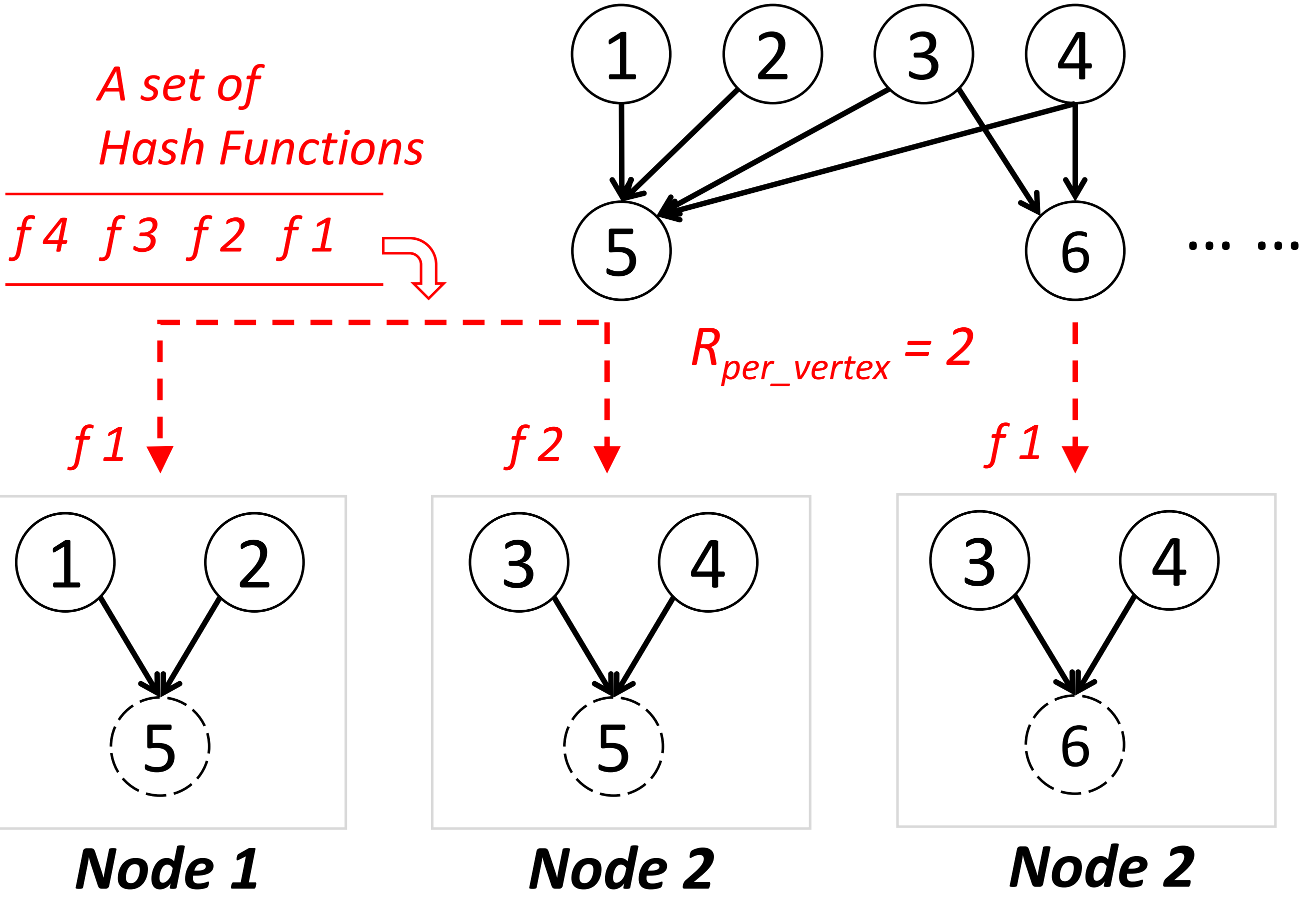
f 4 f 3 f 2 f 1



Horizontal Partitioning: Vertex-chunk Assignment

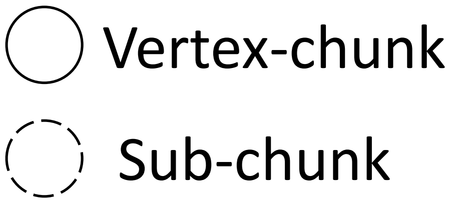


Horizontal Partitioning: Vertex-chunk Assignment



Global Table

vertex ID	functions
5	1, 2
6	1
…	…
k	1,2,4



Summary of GraBi

- **Vertical Partitioning:**

Divide a bipartite graph into several layers
→ trade off between inter-vertex communication and intra-vertex communication

Summary of GraBi

- **Vertical Partitioning:**

  Divide a bipartite graph into several layers
  → trade off between inter-vertex communication and intra-vertex communication

- **Horizontal Partitioning:**

  Assign the bigger vertex-subset first within each layer
  → decrease the number of replicas

  Cut each high-degree vertex-chunk into multiple sub-chunks
  → balance the computation time among vertices

Summary of GraBi

- **Vertical Partitioning:**

  Divide a bipartite graph into several layers
  → trade off between inter-vertex communication and intra-vertex communication

- **Horizontal Partitioning:**

  Assign the bigger vertex-subset first within each layer
  → decrease the number of replicas

  Cut each high-degree vertex-chunk into multiple sub-chunks
  → balance the computation time among vertices

GraBi

- **Fine-grained, high-quality**
- **Light-weight**
- **Generalizable to most MLDM algorithms**

Experimental Setup

- **Implementation**

- GraBi is implemented on a open-source distributed graph-processing system $PowerLyra$[1].
- The two important parameters in GraBi, L and α, are set as 4 and 2 respectively.

[1] R. Chen, J. Shi, Y. Chen, et al. PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs. In EuroSys 2015.

Experimental Setup

➢ Implementation

- GraBi is implemented on a open-source distributed graph-processing system $PowerLyra$[1].
- The two important parameters in GraBi, L and α, are set as 4 and 2 respectively.

➢ Counterparts

- Hybrid-cut (Observation 3)
- Bi-cut (Observation 2)
- 3D-partitioner (Observation 1+ Observation 2)

[1] R. Chen, J. Shi, Y. Chen, et al. PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs. In EuroSys 2015.

Experimental Setup

- **Cluster Configuration**

  The experiments are conducted on an 8-node cluster.

  Each node has one Intel Xeon E5-2650 processor (8 cores) and 16GB DRAM.

Experimental Setup

- **Cluster Configuration**

  The experiments are conducted on an 8-node cluster.

  Each node has one Intel Xeon E5-2650 processor (8 cores) and 16GB DRAM.

- **Bipartite Graphs**

Graph	|U|	|V|	|E|	|U/V|
DBLP	4,000K	1,426K	8.6M	2.81
Netflix	480K	18K	100.5M	27.02
LiveJournal	7,489K	3,201K	112.3M	2.34
Yahoo	1,001K	625K	256.8M	1.60
Orkut	8,731K	2,783K	327.0M	3.14

Experimental Setup

- **Cluster Configuration**

The experiments are conducted on an 8-node cluster.

Each node has one Intel Xeon E5-2650 processor (8 cores) and 16GB DRAM.

- **Bipartite Graphs**

Graph	|U|	|V|	|E|	|U/V|
DBLP	4,000K	1,426K	8.6M	2.81
Netflix	480K	18K	100.5M	27.02
LiveJournal	7,489K	3,201K	112.3M	2.34
Yahoo	1,001K	625K	256.8M	1.60
Orkut	8,731K	2,783K	327.0M	3.14

- **MLDM Algorithms**

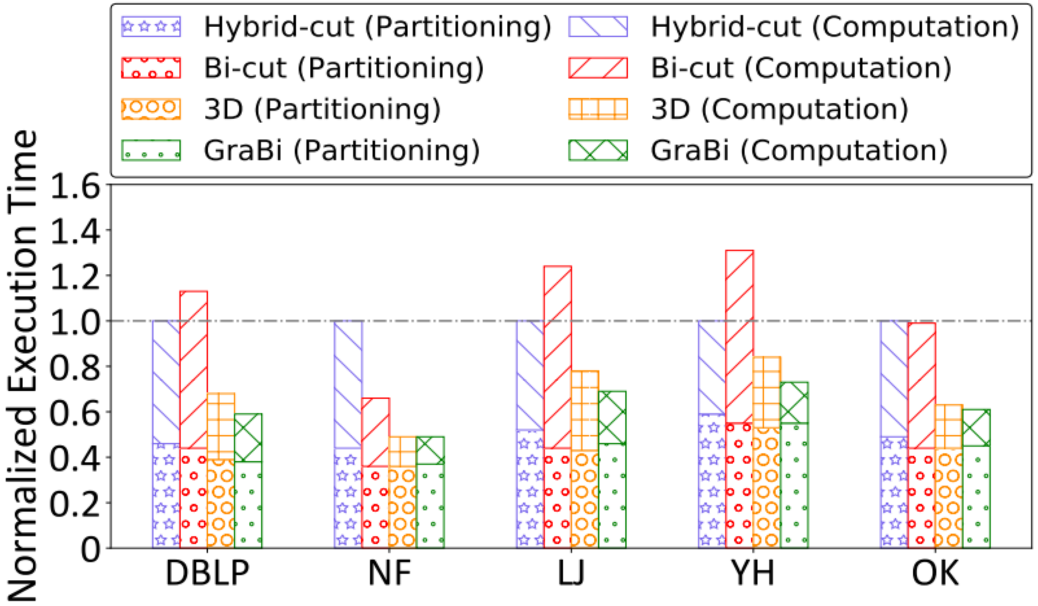
Alternating Least Squares (ALS)

Stochastic Gradient Descent (SGD)

Non-negative Matrix Factorization (NMF)

Overall Performance

➢ **Total Execution Time**



- GraBi improves the execution time by an average of 1.65x over Hybrid-cut, 1.70x over Bi-cut, and 1.09x over 3D-partitioner respectively.
- GraBi surpasses Hybrid-cut and Bi-cut in both the partitioning and computation phases.
- GraBi outperforms 3D-partitioner in the computation phase, but slightly underperforms in the partitioning phase.

Partitioning Phase

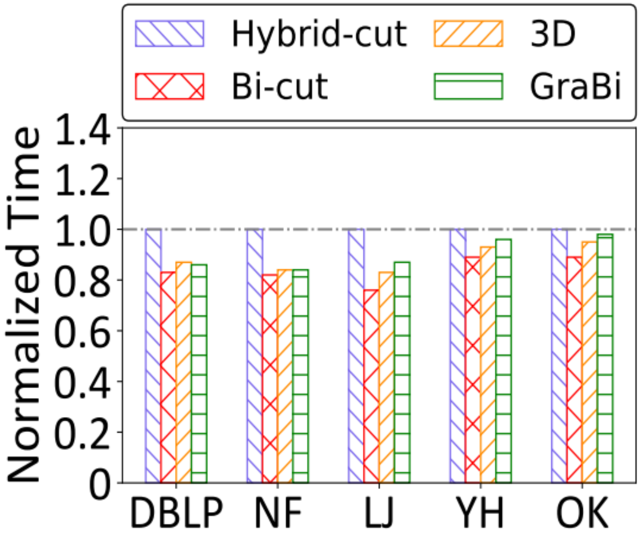
➢ **Replication Factor**

Graph	Hybrid-cut	Bi-cut	3D-partitioner	GraBi
DBLP	2.74	3.08	1.38	1.45
Netflix	3.37	2.14	1.16	1.20
LiveJournal	2.64	3.47	1.30	1.52
Yahoo	3.34	4.43	1.53	1.56
Orkut	3.34	4.43	1.53	1.56

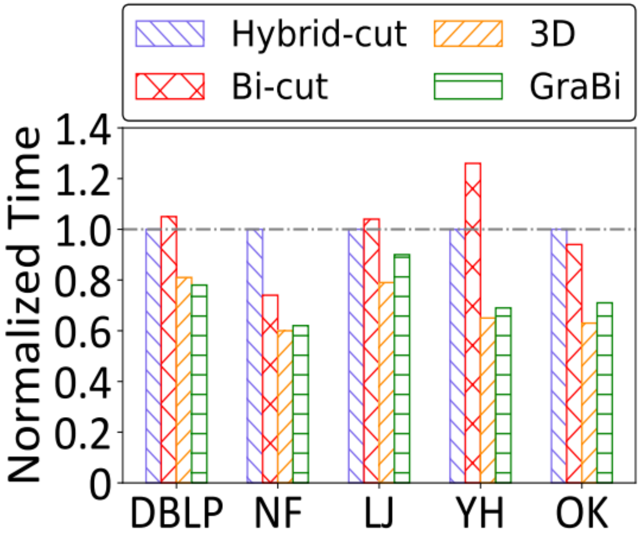
- A lower replication factor represents higher partitioning quality.
- The average of Hybrid-cut, Bi-cut, 3Dpartitioner, and GraBi are 3.06, 3.28, 1.36, and 1.45 respectively.

Partitioning Phase

➢ **Graph Partitioning Time**



(a) Loading & Distributing Time

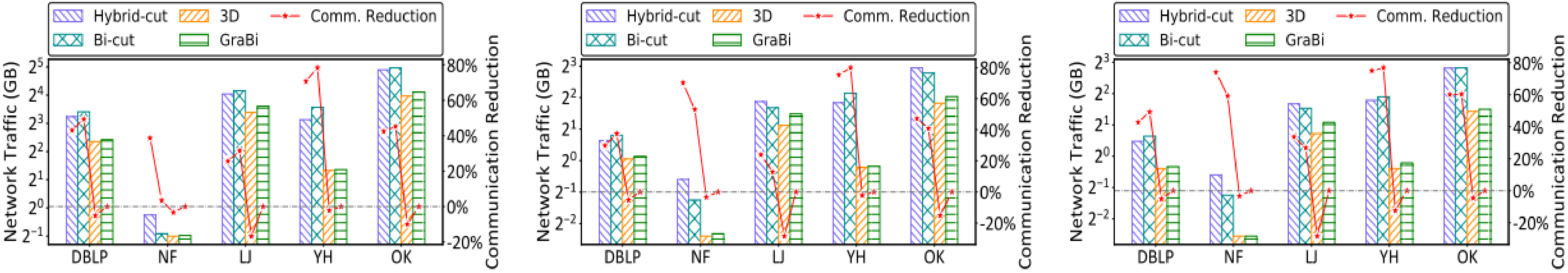


(b) Finalizing Time

- Bi-cut has the shortest loading & distributing time, and Hybrid-cut has the longest.
- The finalizing time is almost proportional to the corresponding replication factor.

Computation Phase

➢ **Network Traffic**

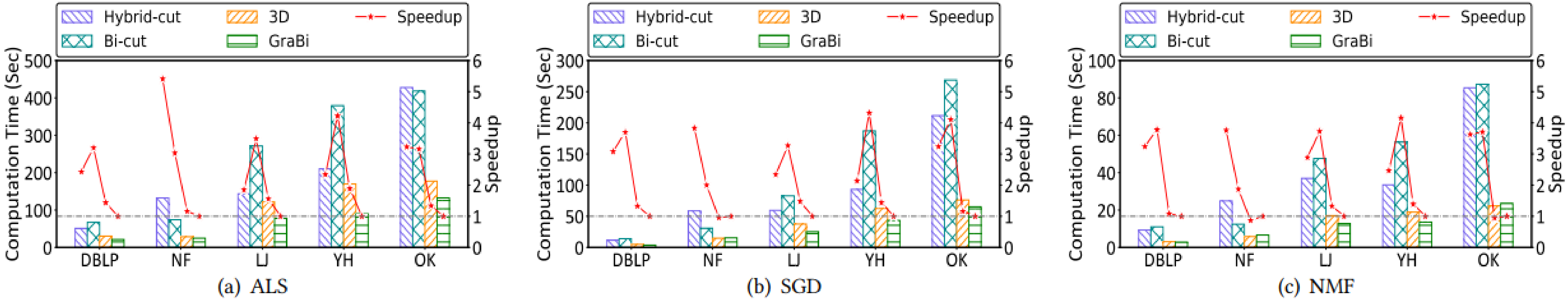


(a) ALS (b) SGD (c) NMF

- GraBi reduces the network traffic in Hybrid-cut and Bi-cut by an average of 45% and 49% respectively.
- GraBi incurs more network traffic than 3D-partitioner by an average of 11%.

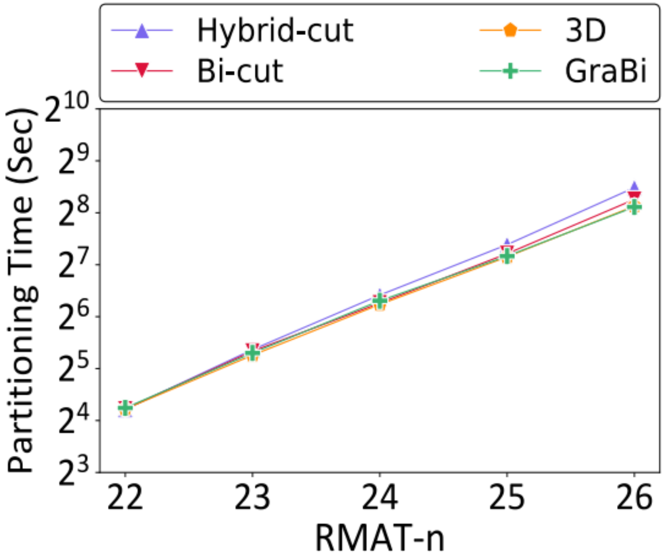
Computation Phase

➢ **Graph Computation Time**

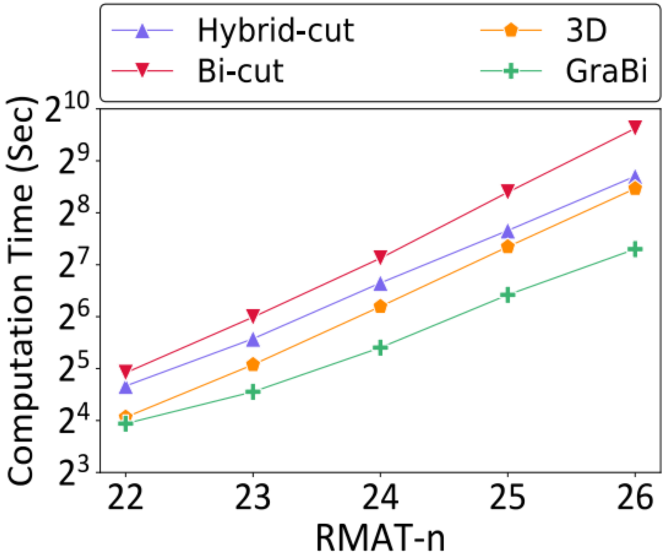


- GraBi outstrips Hybrid-cut, Bi-cut, and 3D-partitioner by an average of 3.12x, 3.41x, and 1.30x respectively.

Scalability



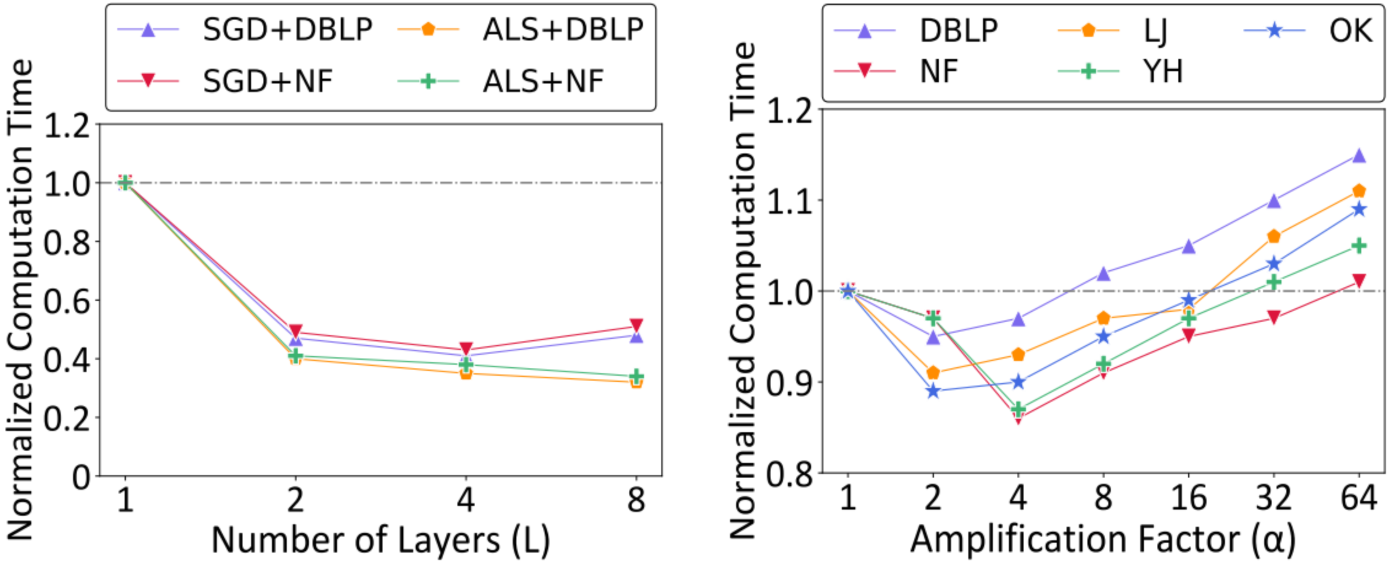
(a) Partitioning Time



(b) Computation Time

- As to the partitioning phase, the scalability of 3D-partitioner and GraBi is better than Hybrid-cut and Bi-cut.
- As to the computation phase, the scalability Hybrid-cut of and GraBi is better than Bi-cut and 3D-partitioner.

Impact of Parameters

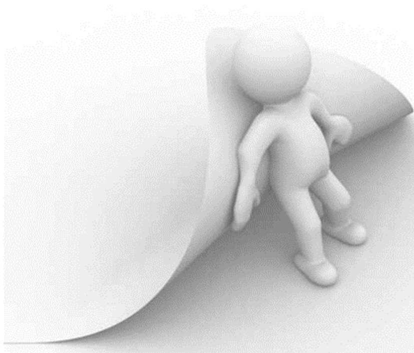


(a) Impact of L on ALS and SGD (b) Impact of α on ALS

- ALS and SGD algorithms behave best at different values of L.
- The impact of α is moderate and stable within a wide value range.

Conclusion

GraBi is a *communication-efficient* and *workload-balanced* partitioning framework for bipartite graphs.

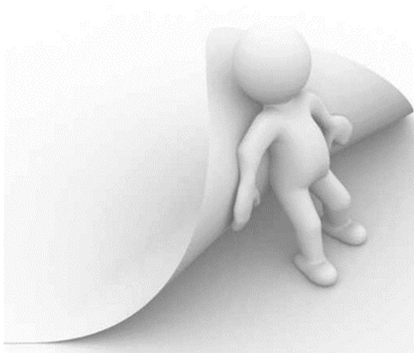


Conclusion

GraBi is a *communication-efficient* and *workload-balanced* partitioning framework for bipartite graphs.

- **Vertical Partitioning:**

Observation 1 ⇨ Divide a bipartite graph into several layers ⇨ Efficient Communication



Conclusion

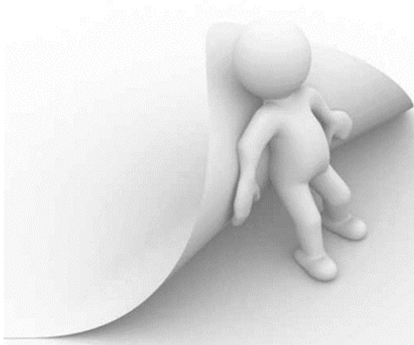
GraBi is a *communication-efficient* and *workload-balanced* partitioning framework for bipartite graphs.

- **Vertical Partitioning:**

Observation 1 ⇨ Divide a bipartite graph into several layers ⇨ Efficient Communication

- **Horizontal Partitioning:**

Observation 2 ⇨ Assign the bigger vertex-subset first within each layer ⇨ Efficient Communication



Conclusion

GraBi is a *communication-efficient* and *workload-balanced* partitioning framework for bipartite graphs.

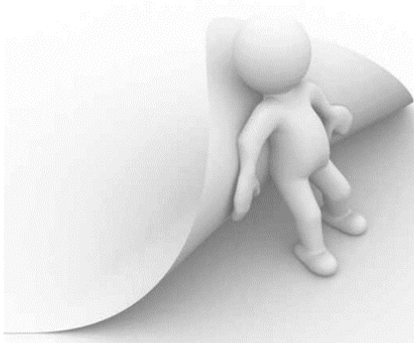
- **Vertical Partitioning:**

Observation 1 ⇨ Divide a bipartite graph into several layers ⇨ Efficient Communication

- **Horizontal Partitioning:**

Observation 2 ⇨ Assign the bigger vertex-subset first within each layer ⇨ Efficient Communication

Observation 3 ⇨ Decompose each high-degree vertex-chunk into sub-chunks ⇨ Workload Balance



Thank You !