Gemini: A Computation-Centric Distributed Graph Processing System

Xiaowei Zhu *Wenguang Chen* *Weimin Zheng*

Tsinghua University

Xiaosong Ma

Qatar Computing Research Institute

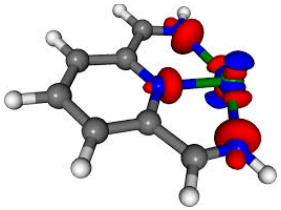
清華大學
Tsinghua University

معهد قطر لبحوث الحوسبة
Qatar Computing Research Institute

جامعة حمد بن خليفة
HAMAD BIN KHALIFA UNIVERSITY

Graphs, Platforms, and Systems



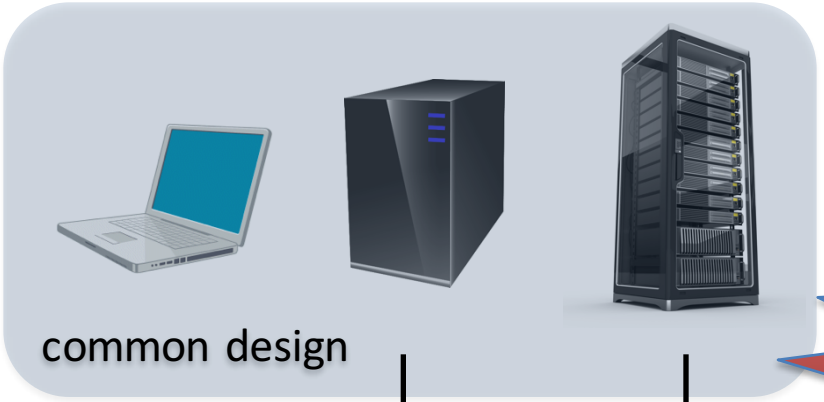





Shared-memory
Ligra[PPoPP '13], Galois[SOSP '13]
Efficiency☺ Scalability☹

Distributed
PowerGraph[OSDI '12], GraphX[OSDI '14]
Efficiency☹ Scalability☺

Connecting Shared-Memory and Distributed



Bottleneck in computation!

Profiling existing systems

Nodes System	1 Galois	8 PowerLyra[EuroSys '15]
Runtime (s)	19.3	26.9
Instructions	Distributed overhead	
Memory references		
Communication (GB)	-	38.1
Instructions per cycle	0.414	0.655
L3 cache miss rate	Sub-optimal computation	
CPU utilization		

Far from saturated for IB EDR (100Gbps) (38.1*8/2/26.9/8=**0.708**Gbps)

More instructions and memory traffic

Poorer access locality

Lower multi-core utilization

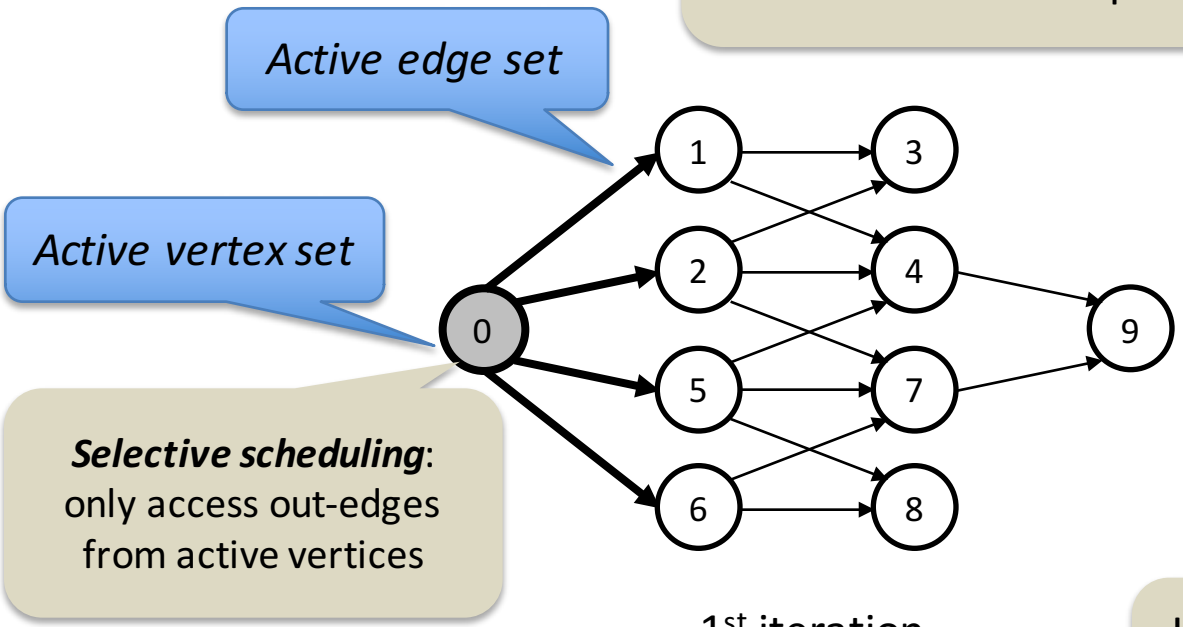
20 iterations of PageRank on twitter-2010 (41.7M vertices, 1.47B edges)

We Propose: ***Gemini***

- Build **scalability** on top of **efficiency**
  - Avoid unnecessary "distributed" side-effects
  - Optimize computation on partitioned sub-graphs
- Shift of design focus
  - Designed for distributed, but **computation-centric**
    - Modern clusters have fast interconnects
    - Computation-communication overlap in place
- Major optimizations
  - Efficiency
    - Adaptive push-/pull-style computation
    - Hierarchical chunk-based partitioning
  - Scalability
    - Locality-aware chunking
    - Chunk-based work-stealing

Dual Mode: BFS Example (1)

$|Active\ edge\ set| / |E| < threshold$
⇒ ***Sparse*** *mode*
⇒ ***Push*** operations



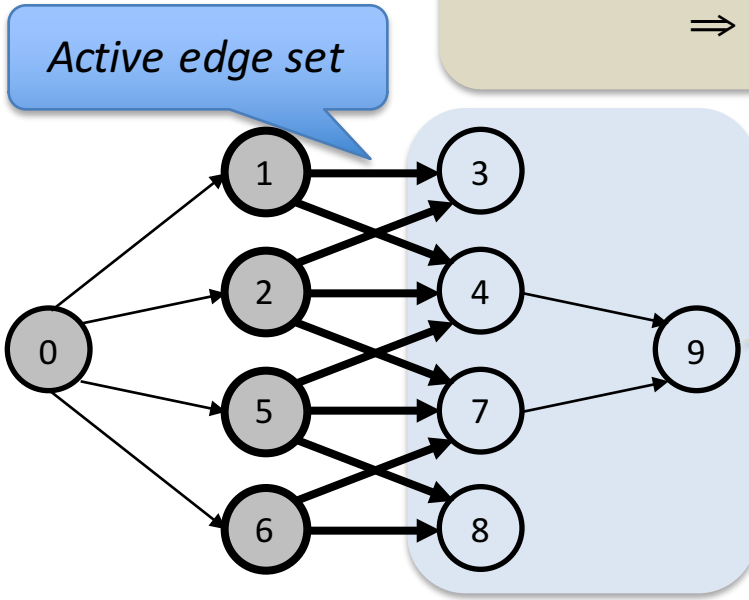
Selective scheduling:
only access out-edges
from active vertices

1st iteration

Locks/atomic operations
required for correctness
of concurrent updates

Dual Mode: BFS Example (2)

$|Active\ edge\ set| / |E| > threshold$
⇒ ***Dense*** *mode*
⇒ ***Pull*** operations

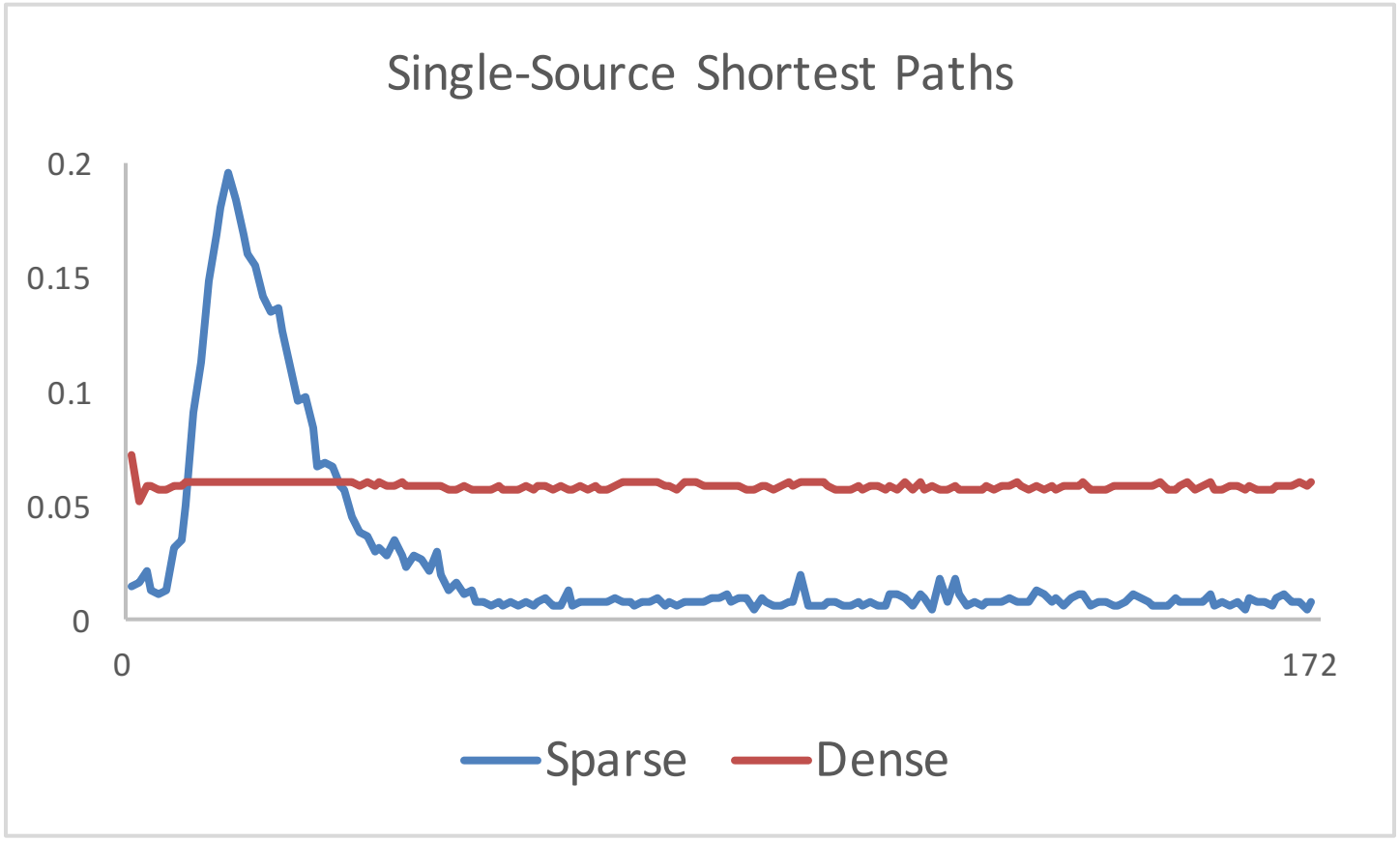


Useless computation

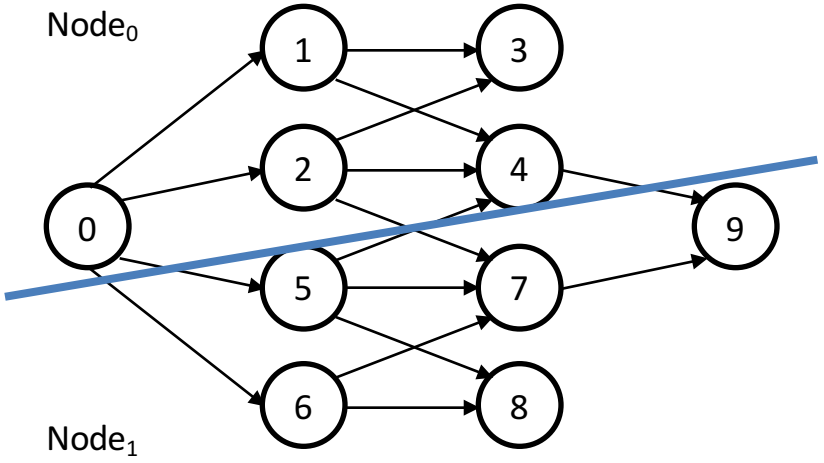
2nd iteration

Vertices pulling along in-edges
Contention-free updating

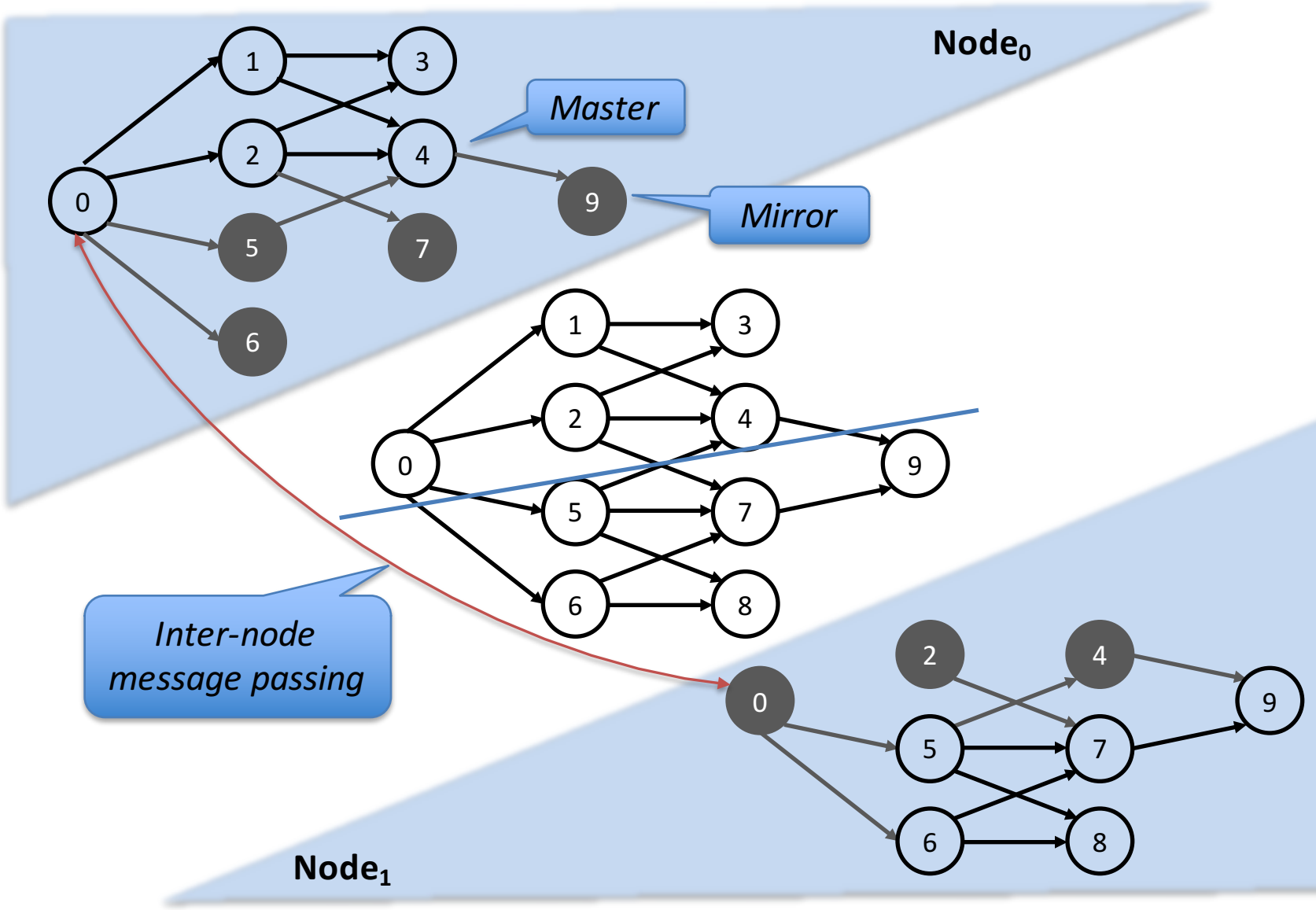
Push vs. Pull: Performance Impact



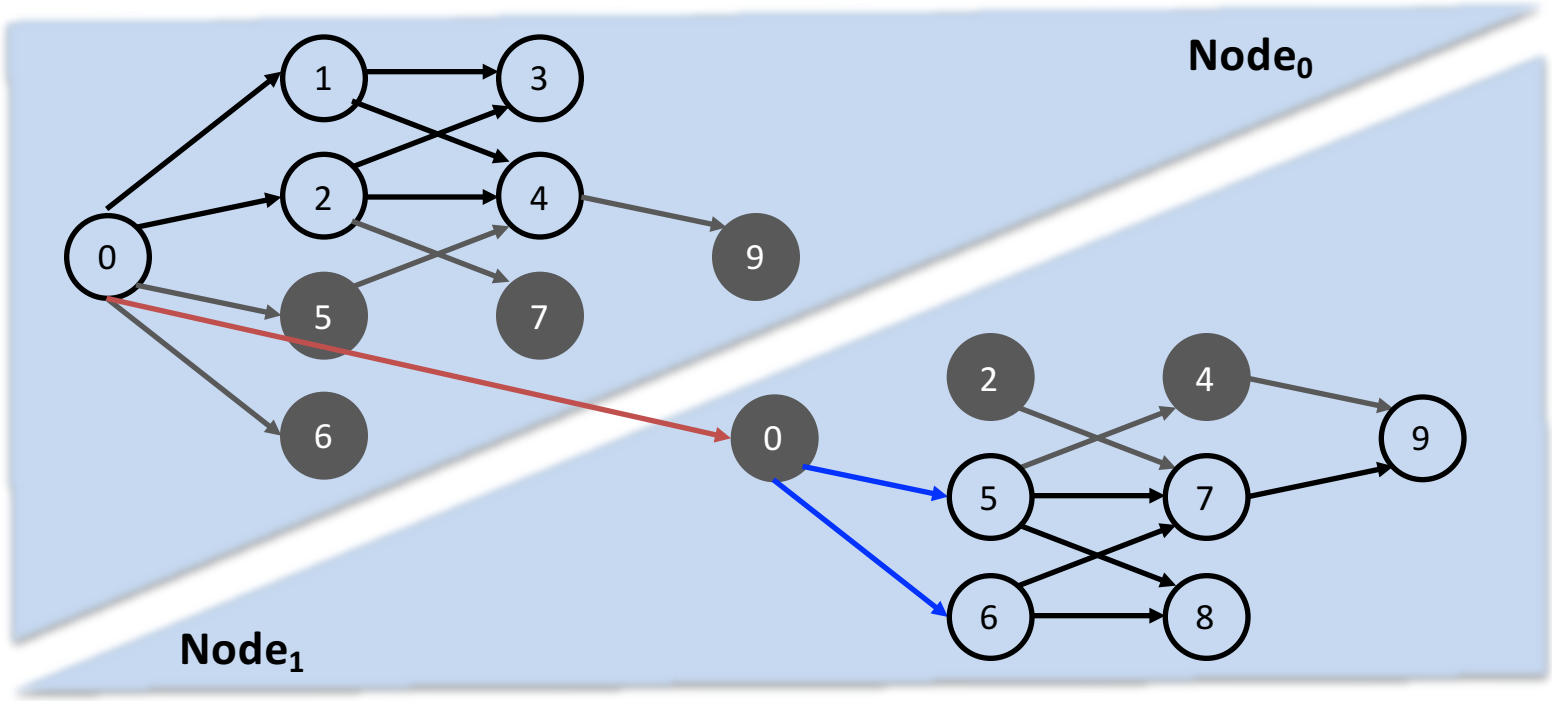
Distributed Dual-Mode Computation



When Distributed to 2 Nodes

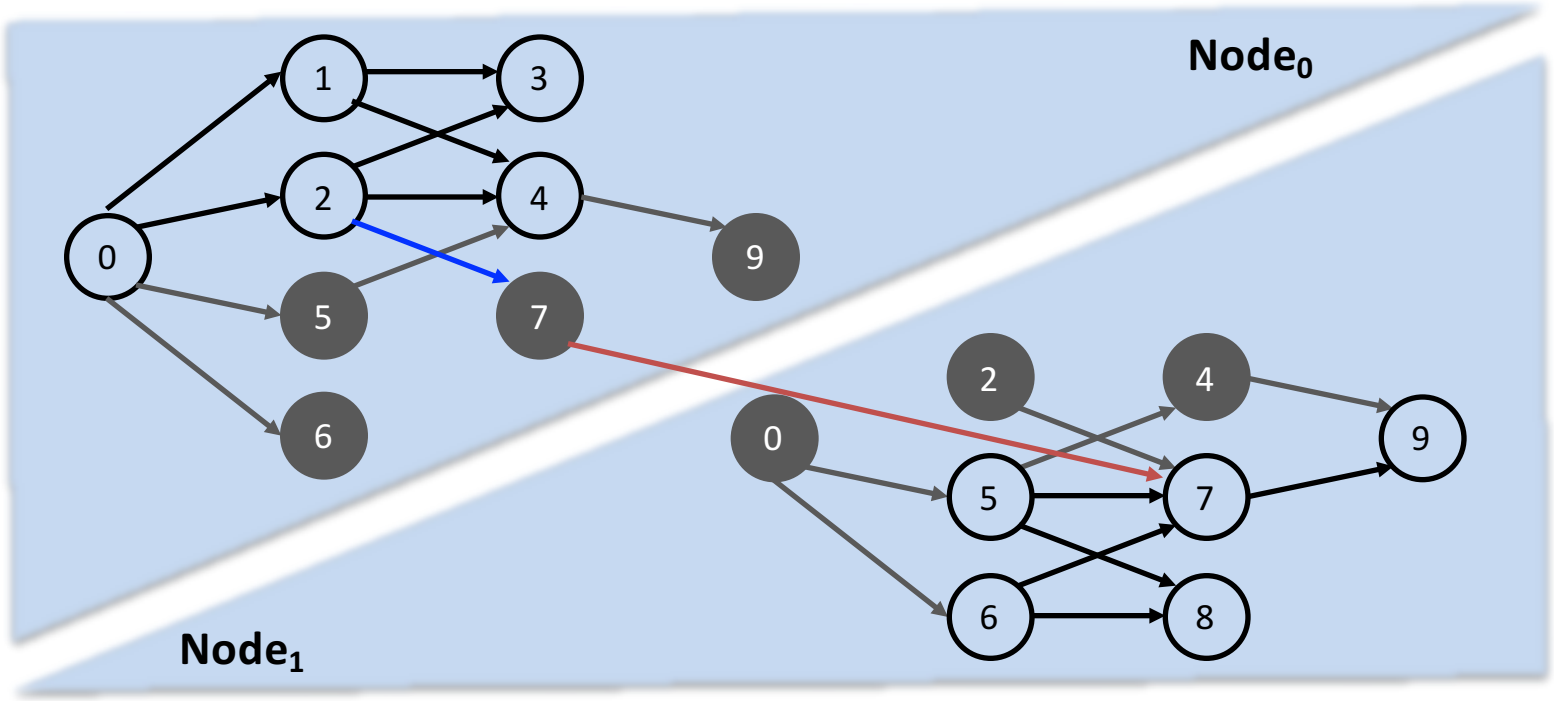


Gemini's Distributed Push



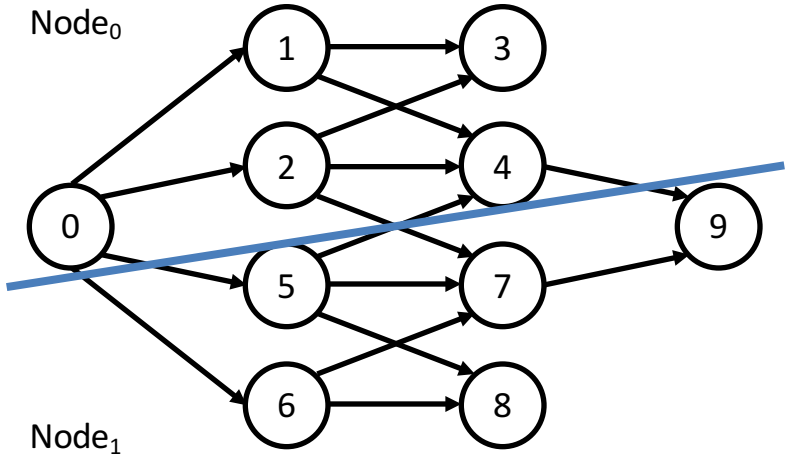
Masters message mirrors, who update their local neighbors

Gemini's Distributed Pull

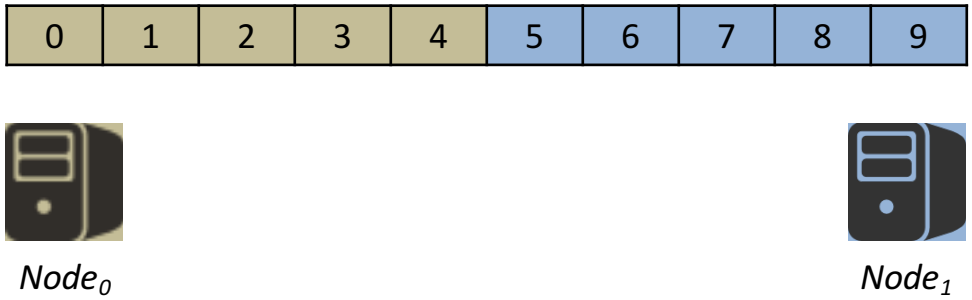


Mirrors pull updates from neighbors, then message masters

Gemini's Choice of Graph Partitioning



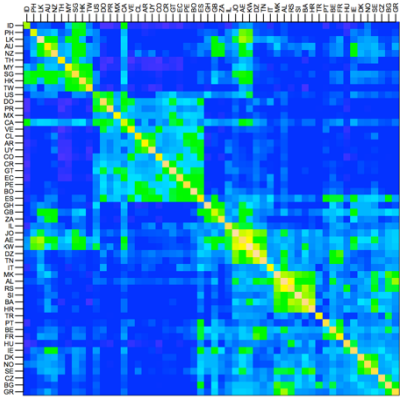
- Chunking
  - Divide vertex set ***V*** into ***p*** **contiguous** chunks



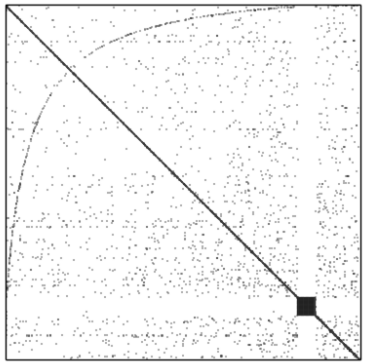
  - Dual-mode edge data distributed accordingly

Why Chunk-Based Partitioning?

- It preserves locality!
  - Fact: **locality** exists in many real-world graphs
    - Vertices "semantically" ordered



Facebook Country Adjacency Matrix[1]



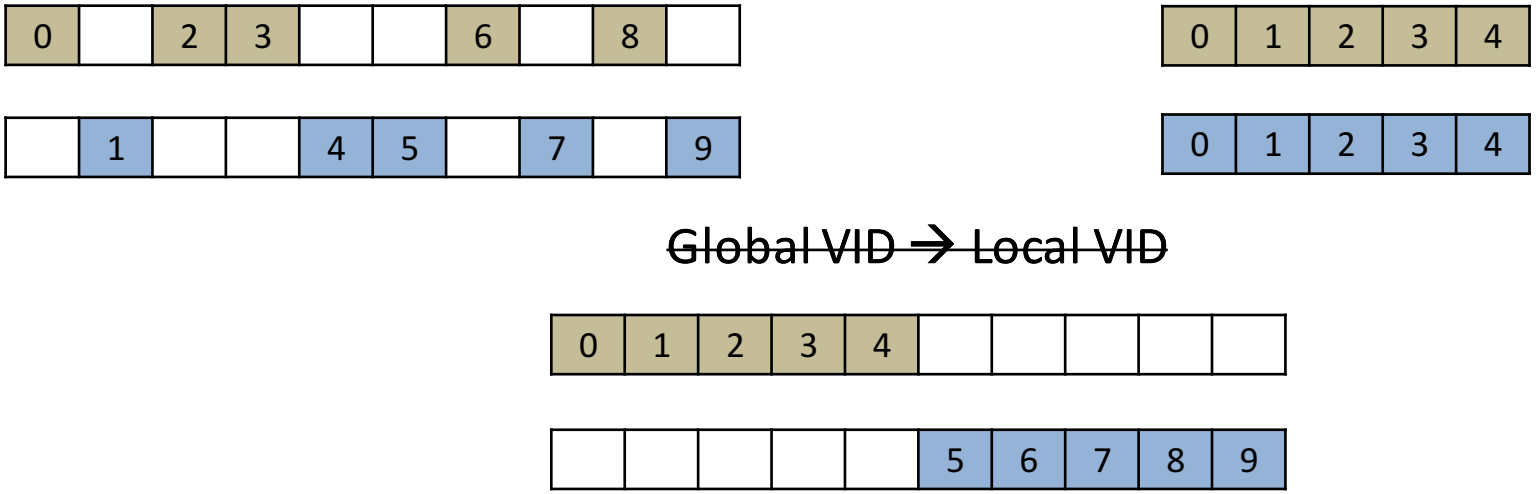
UK Web (2005) Adjacency Matrix

  - Preprocessing affordable when vertices unordered
    - E.g., BFS[Algorithms 09], LLP[WWW '11]

[1] The Anatomy of the Facebook Social Graph

More Benefits of Chunking

- Low-overhead distributed designs

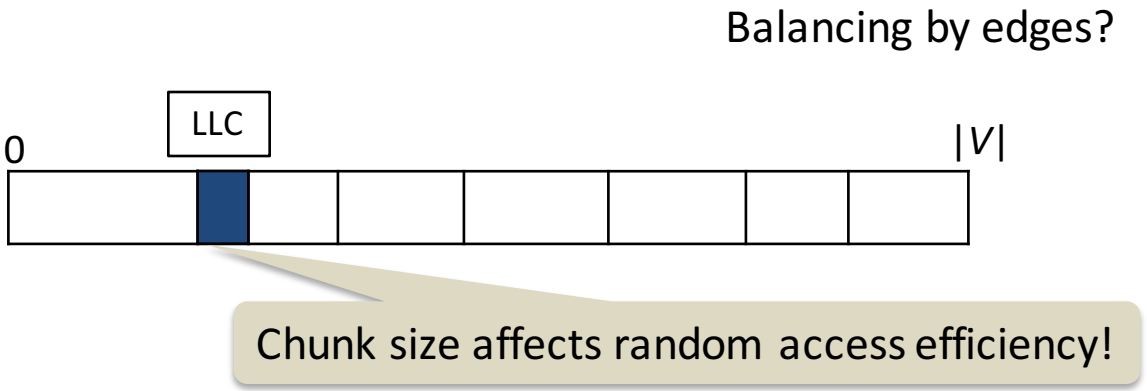


- Applied recursively at different levels
  - More on this later

Challenges with Distributed Chunking

- When scaling out
  - CSR/CSC vertex indices can become very **sparse**
  - *Solution:* **compress** vertex indices
- Modern servers built upon NUMA architecture
  - **Interleaved** layout sub-optimal
  - *Solution:* apply inter-socket **sub-partitioning**
- Chunking as vertex-centric (edge-cut) scheme
  - Vertex-centric solutions are not good at **load balancing** natural graphs
  - *Solution:* balancing workload in **locality**-aware manner

Locality-Aware Chunking



- Gemini considers both vertex and edge
  - Edge: the amount of work to be processed
  - Vertex: the processing speed of work (locality)
  - Hybrid metric: $\alpha \cdot |V_i| + |E_i|$

Chunking, Chunking All the Way

Per-node partition

cluster

Per-socket partition w. NUMA-aware placement

node

Data partitioning till socket-level

Per-core work chunk

socket

core

Mini-chunk for work-stealing

Work partitioning within socket

Locality-aware

Fine grain (64-vertex)

Evaluation

- Platform: 8-node cluster
  -  Intel Xeon E5-2670 v3 (12-core CPU), 30MB L3 cache
  -  2 sockets sharing 128 GB RAM (DDR4 2133MHz)
  -  Network: Mellanox Infiniband EDR 100Gbps

- Applications
  - PageRank (PR) (20 iterations)
  - Connected Components (CC)
  - Single-Source Shortest Paths (SSSP)
  - Breadth-First Search (BFS)
  - Betweenness Centrality (BC)

- Input graphs

Graph	|V|	|E|
enwiki-2013	4,206,785	101,355,853
twitter-2010	41,652,330	1,468,365,182
uk-2007-05	105,896,555	3,738,733,648
weibo-2013	72,393,453	6,431,150,494
clueweb-12	978,048,098	42,574,107,469

Single-Node Efficiency

Application	Ligra	Galois	Gemini
PR	21.2	19.3	**12.7**
CC	6.51	**3.59***	4.93
SSSP	**2.81**	3.33	3.29
BFS	**0.347**	0.528	0.468
BC	2.45	3.94*	**1.88**

Runtime in seconds (twitter-2010)

More iterations

More instructions

NUMA-aware memory accesses

System	Ligra	Gemini
Remote access ratio	50.1%	9.10%
L3 cache miss rate	52.6%	40.1%
Average access latency	183ns	125ns

Memory performance (BC)

"*" uses different algorithms.

Multi-Node Scalability: Larger Graphs

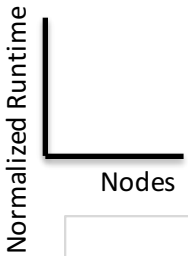
Graph	|V|	|E|
enwiki-2013	4,206,785	101,355,853
twitter-2010	41,652,330	1,468,365,182
uk-2007-05	105,896,555	3,738,733,648
weibo-2013	72,393,453	6,431,150,494
clueweb-12	**978,048,098**	**42,574,107,469**

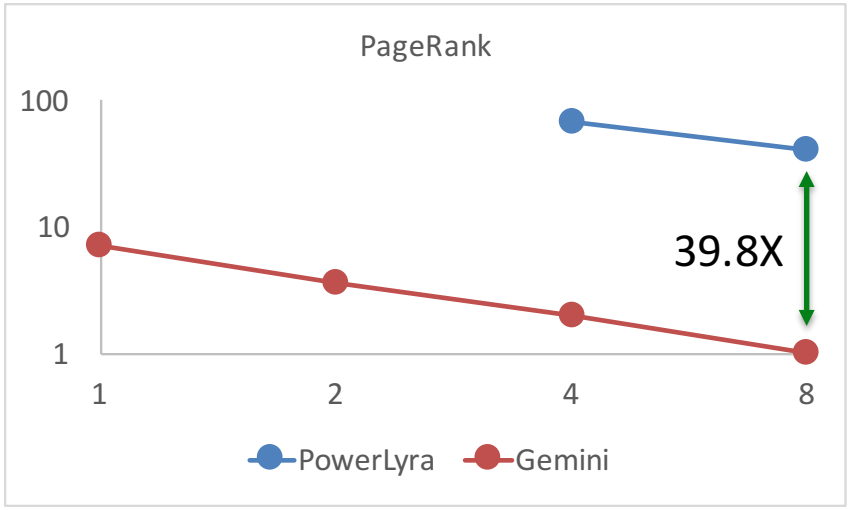
318GB input graph

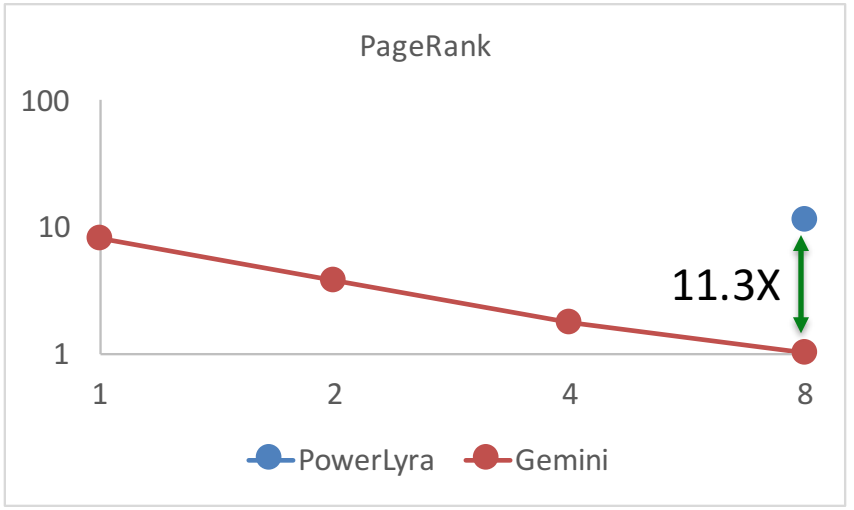
Application	PowerLyra	Gemini
PR	Out of memory	31.1
CC		25.7
SSSP		56.9
BFS		10.2
BC		45.3

Runtime in seconds (clueweb-12)

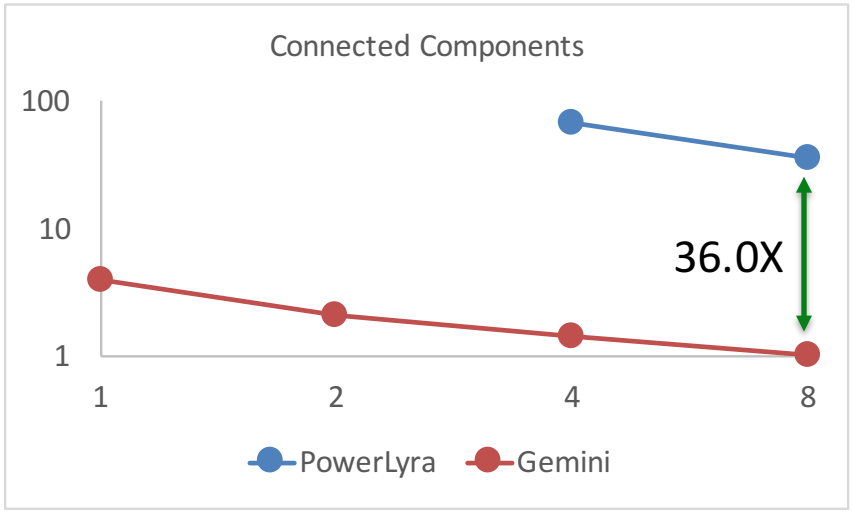
Multi-Node Scalability: Faster Speeds



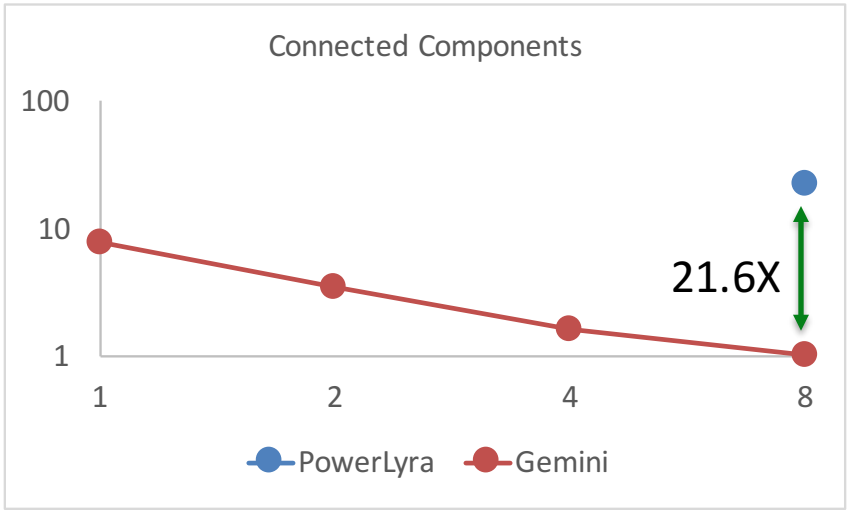




uk-2007-05



weibo-2013



Closing Remarks

Search **GeminiGraph** on Github

- What have we learned
  - Computation efficiency highlighted by fast network
    - Existing guidelines may not apply
  - **Chunking works!**
    - Multi-fold benefits
    - Enables series of optimizations

Thanks!

Q & A