SHANGHAI JIAO TONG UNIVERSITY

Excavating the Potential of Graph Workload on RDMA-based Far Memory Architecture

Jing Wang, Chao Li, Taolei Wang, Lu Zhang, Pengyu Wang, Junyi Mei, Minyi Guo

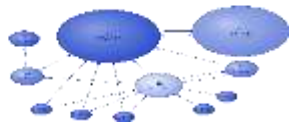
Department of Computer Science and Engineering, Shanghai Jiao Tong University ,

Shanghai, China

Outline

- Background
- Motivation
- System Design
- Evaluation
- Conclusion

Background: Graph Processing

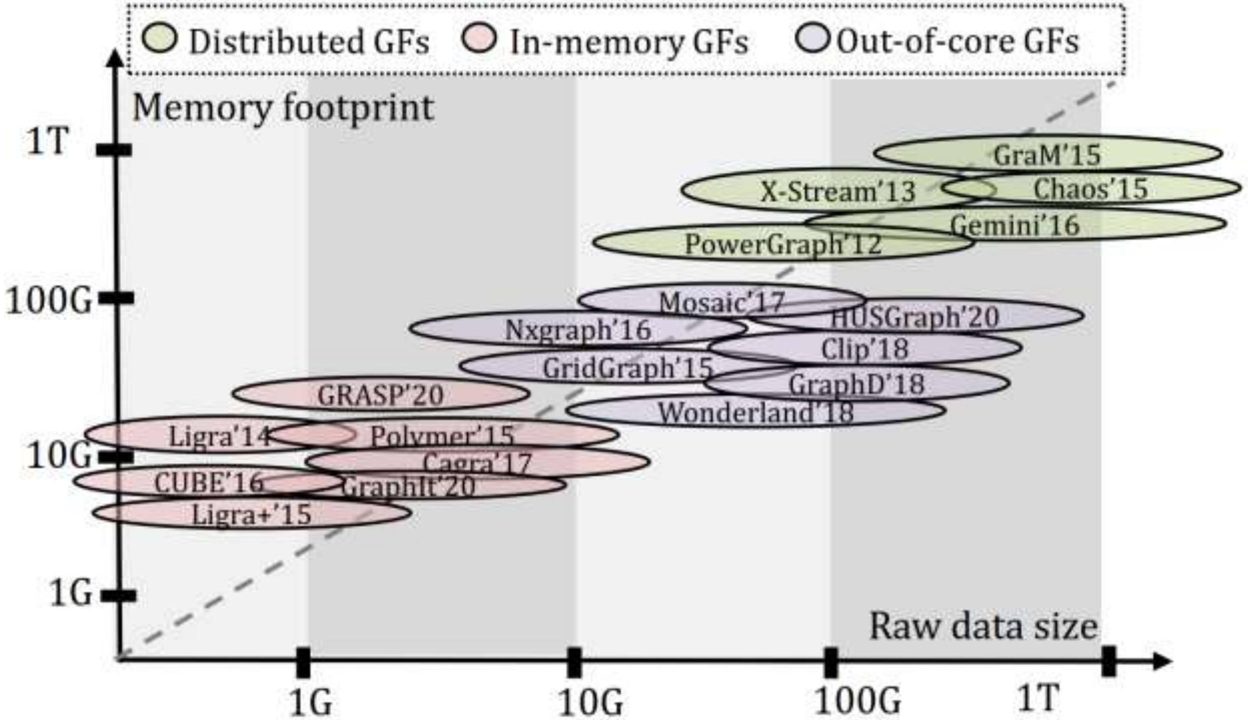


Pagerank

Kmeans

Label Propagation

Graph Applications



Existing Graph Processing Frameworks:

***In-memory* Frameworks**

→ Small size graphs

→ Memory footprint > raw data size

***Out-of-core* Frameworks**

→ Medium-size graphs

→ Performance issue due to I/O bottleneck

***Distributed* Frameworks**

→ Very large graphs

→ Communication overhead

Graph applications demand better memory performance on large memory sizes

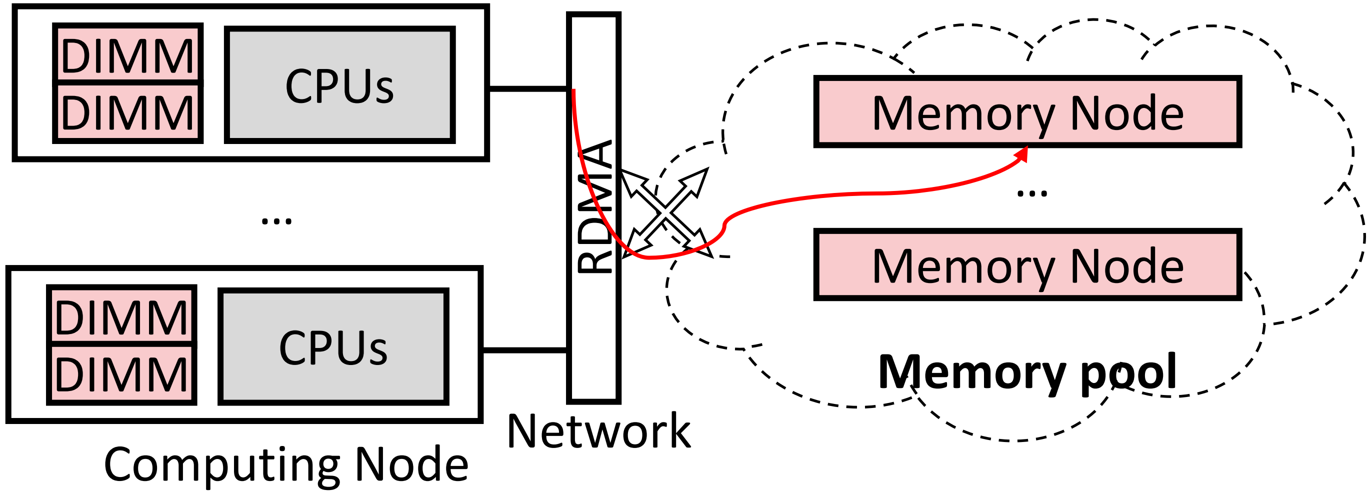
Background: Disaggregated Architecture

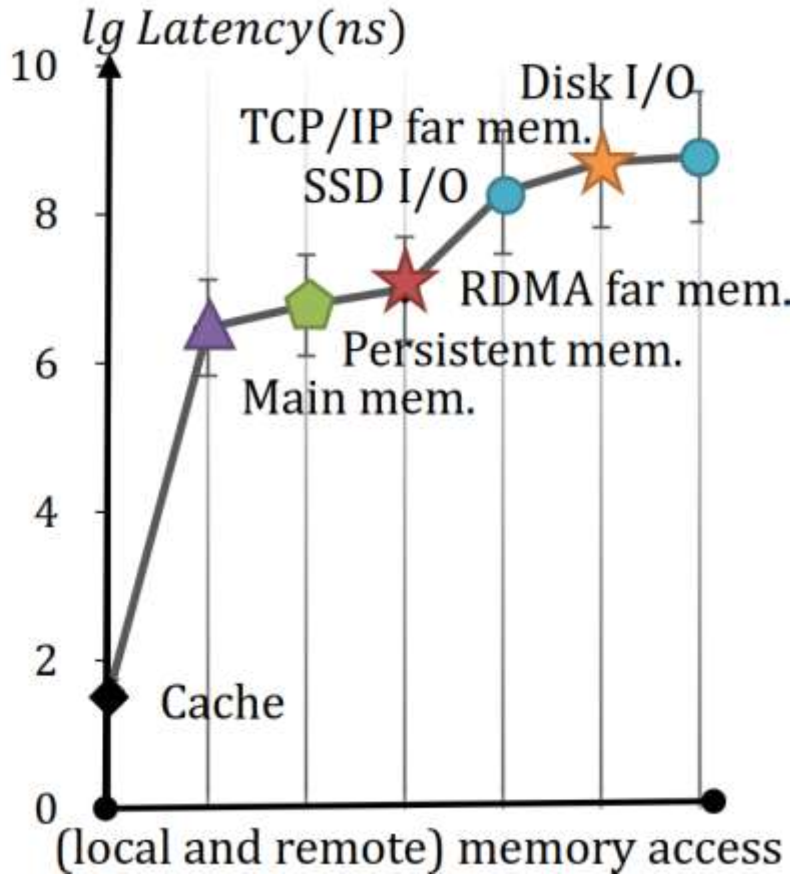
Disaggregated Architecture Opportunities:

- Large memory capacity
- Scalability and elasticity

RDMA-based Far Memory：

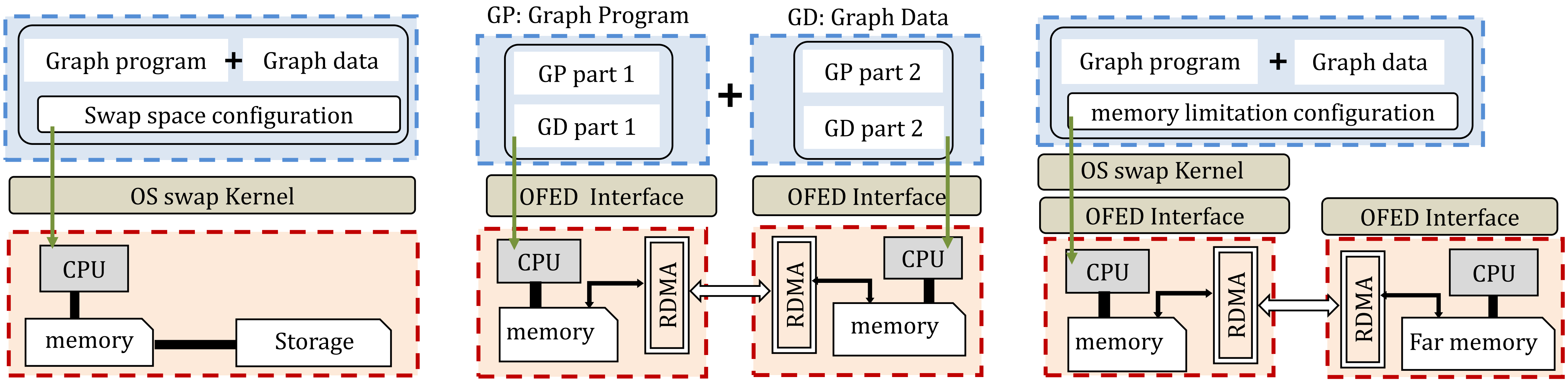
- No-CPU-involved execution model
- Near-DRAM application performance





Background: Comparison of Execution Models

- **There are three architectures of memory expansion ways for graph processing.**



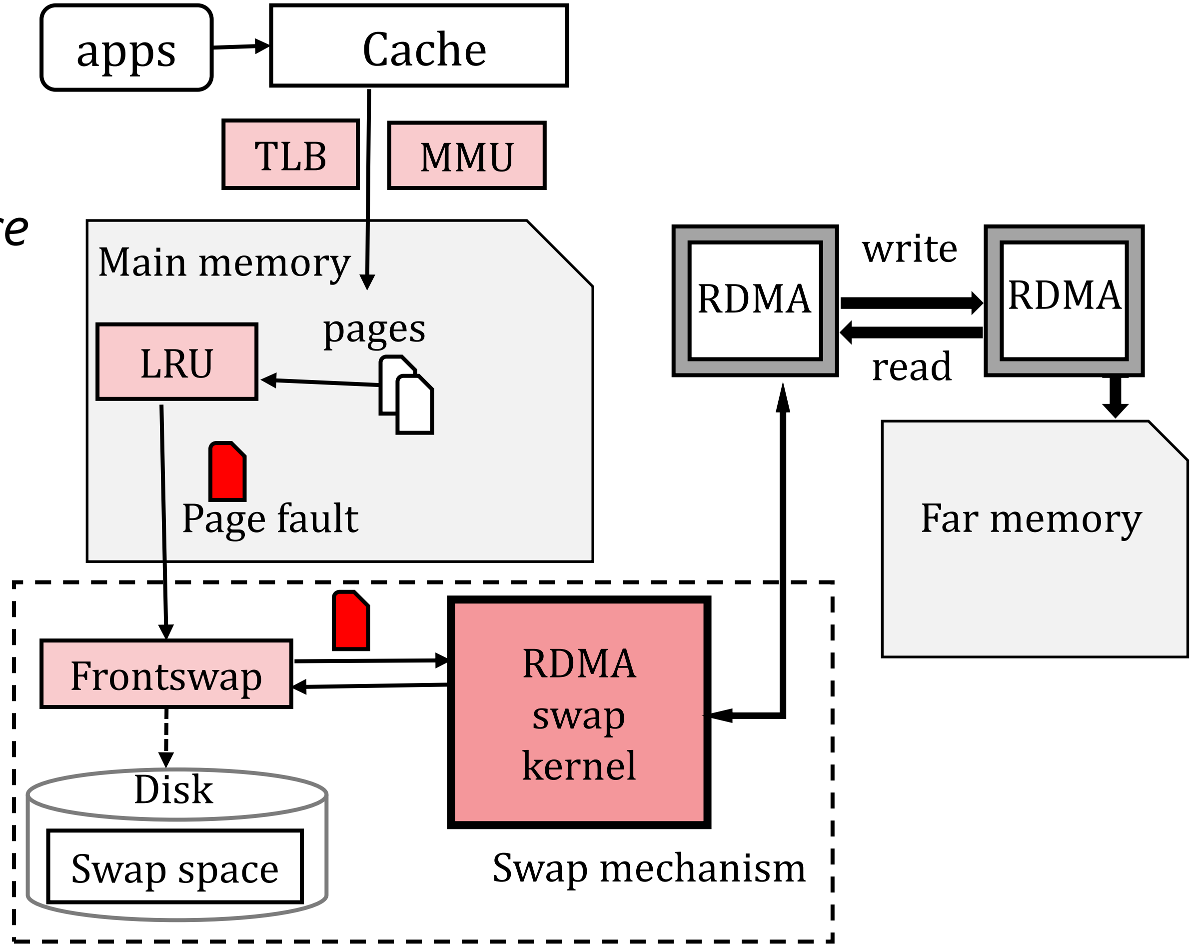
Single-node system **Distributed system** **Far memory system**

Far memory system provides a new option for scaling out graph processing on both single-node and distributed-computing systems.

Background: Far Memory Current Issues

Current Issues:

- ***Kernel overhead***
  → *Undermine kernel-bypass RDMA performance*
  → *Significant context switching overhead*
- **Swap-based data replacement strategy**
  → *Passively triggered*
  → *No decision for thrown-out parts*
- **Page-size based data transfer**
  → *Offloading data size is fixed*
  → *Not efficient enough*



Designing user-level and application-aware data offloading method is necessary.

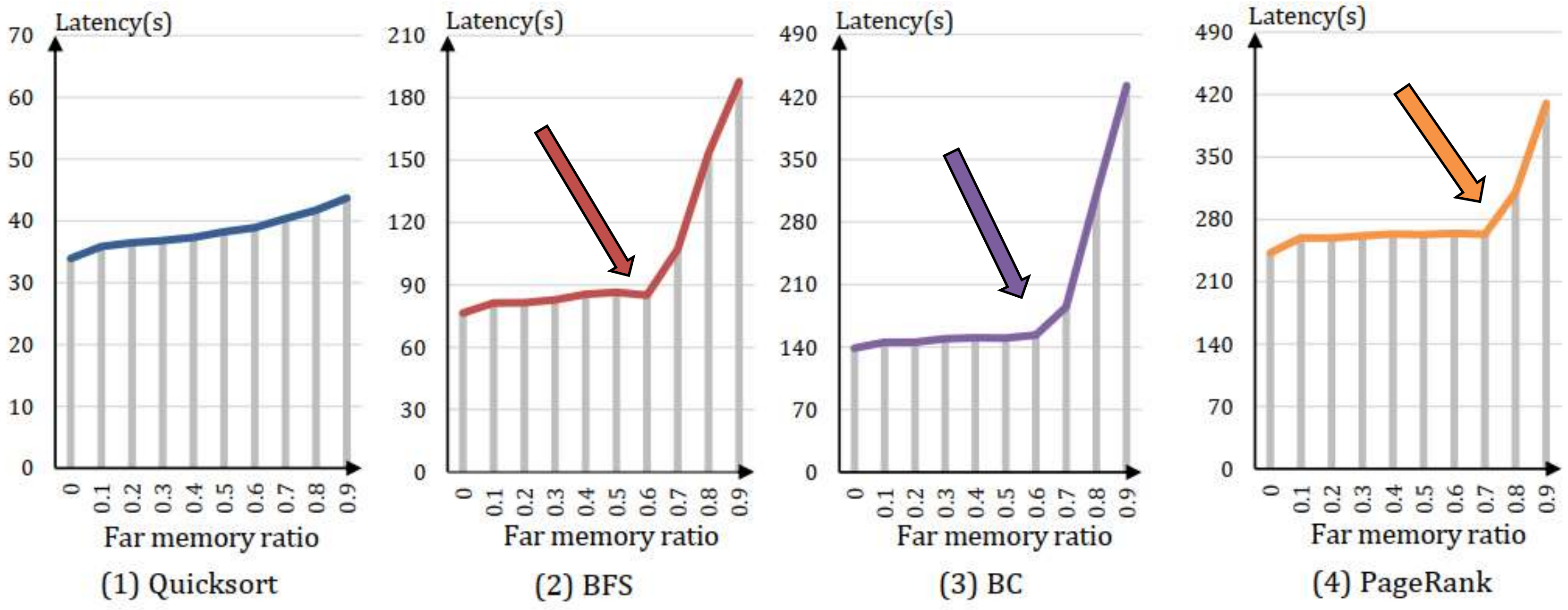
Outline

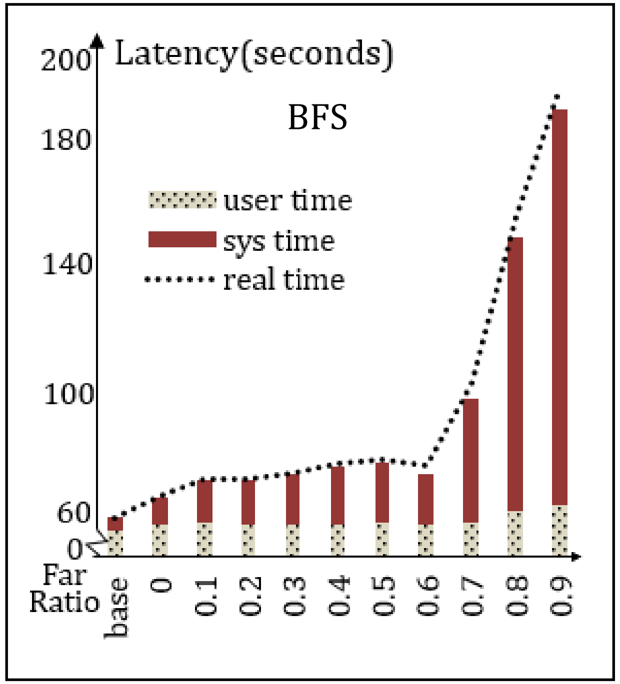
- Background
- **Motivation**
- System Design
- Evaluation
- Conclusion

Motivation: Duration Analysis

We run graph applications on far memory and analyze the behavior of task durations.

- **Observation 1:** Turning points of latency trends
  → Memory offloading of graph workloads should be careful.
- **Observation 2:** Duration increase is caused by page faults at the kernel level
  → Designing user-level far memory access operations is essential.

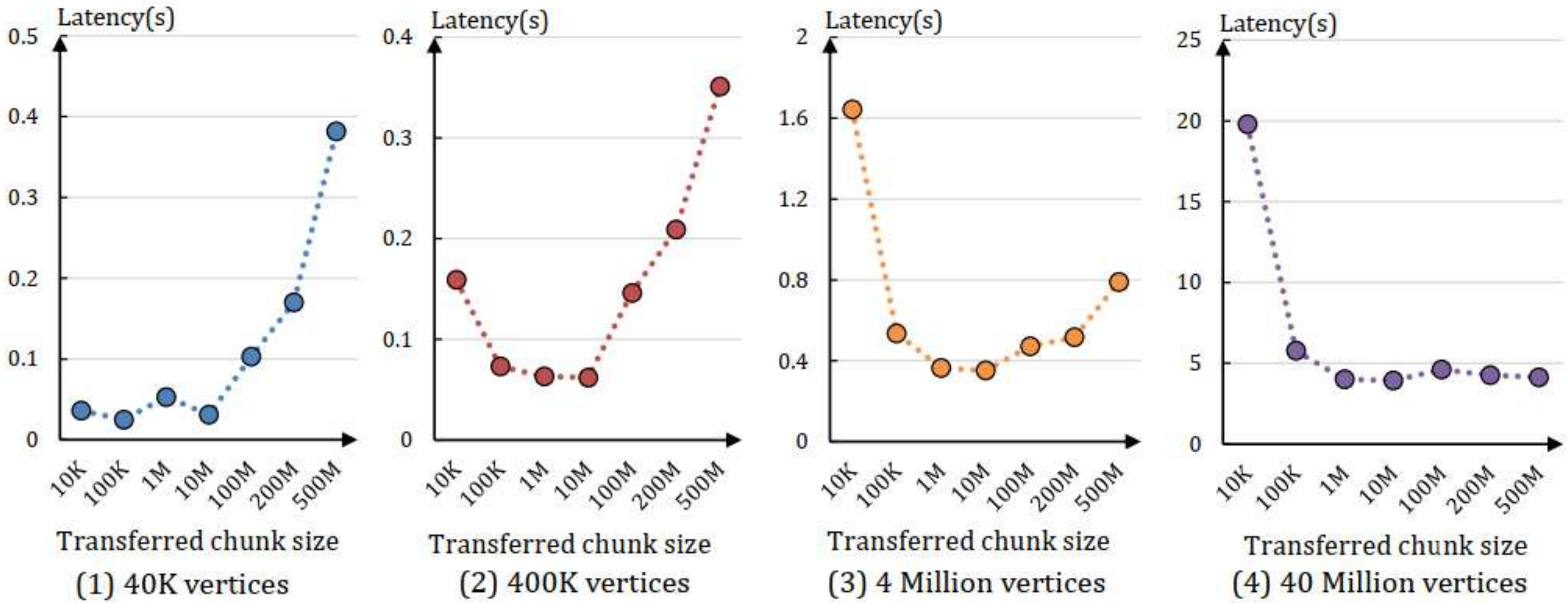




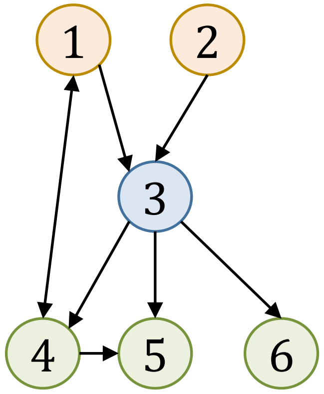
Motivation: Efficiency Issue

We transfer data through RDMA with different chunk sizes and compare the overall latency.

- **Observation 1:** Latency differs on different chunk size
  → Choose a proper chunk size for better total performance
- **Observation 2:** Data may be overwritten if using RDMA operation unproperly
  → Set buffers when communication with far memory

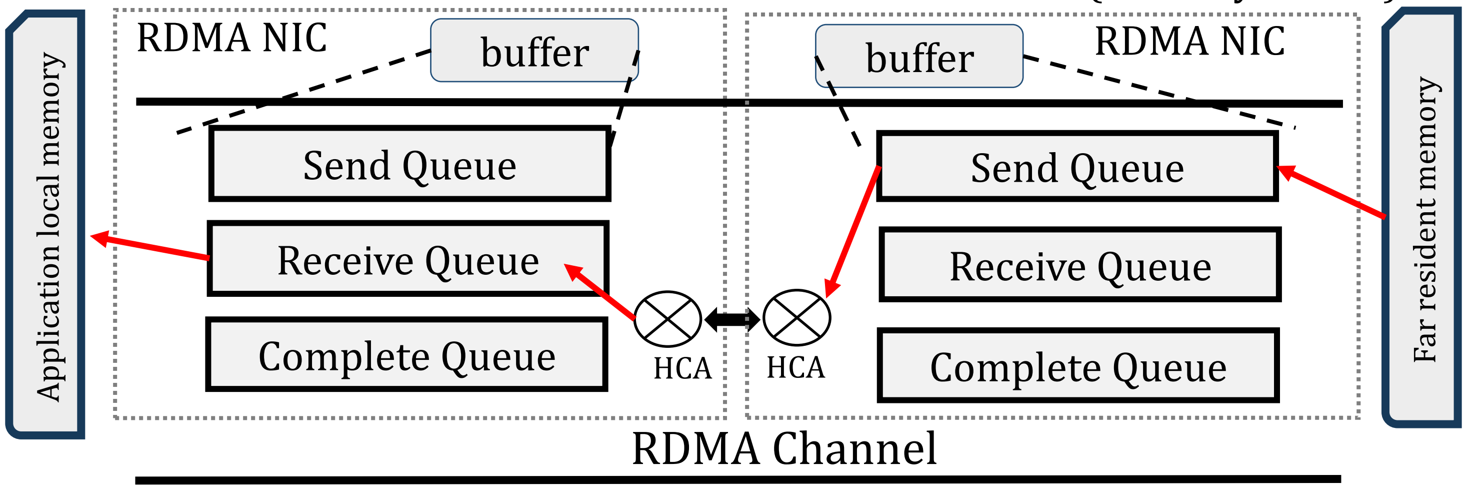


Motivation: Opportunities



Graph data blocks

	1,2,3	4,5,6
1, 2, 3	1,3 1,4 2,3	3,4 3,5 3,6
4, 5, 6	4,1	4,5



Key Oppty. of Optimizing Graph Workload:

- Distinctive data segments
- Large size of read-only edge data
- Iterative execution model

Key Oppty. of Utilizing RDMA Mechanism:

- High-throughput memory access
- High-performance one-sided operations
- Turning knob configurations

Our System:

A high-performance graph-aware data offloading and fetching strategy

Outline

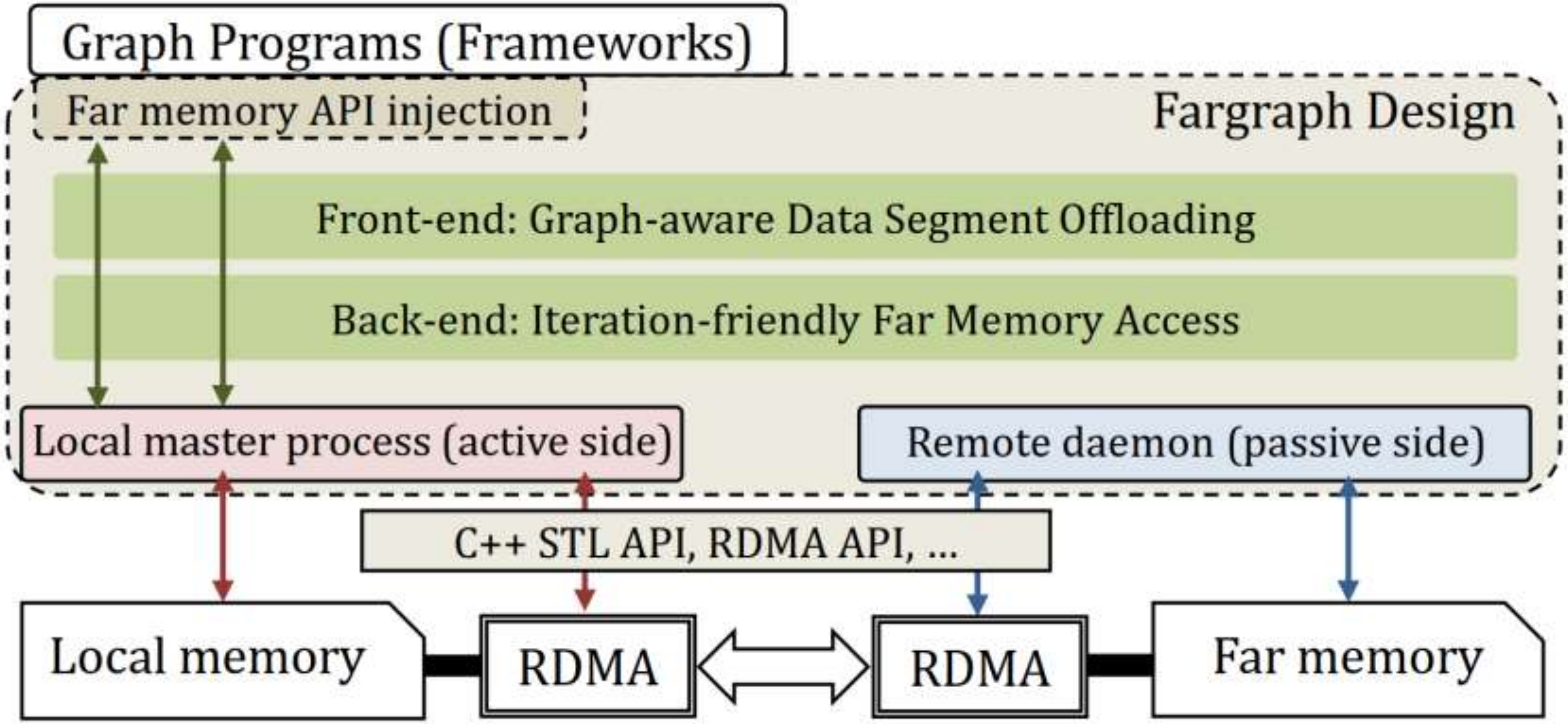
- Background
- Motivation
- **System Design**
- Evaluation
- Conclusion

Fargraph Design

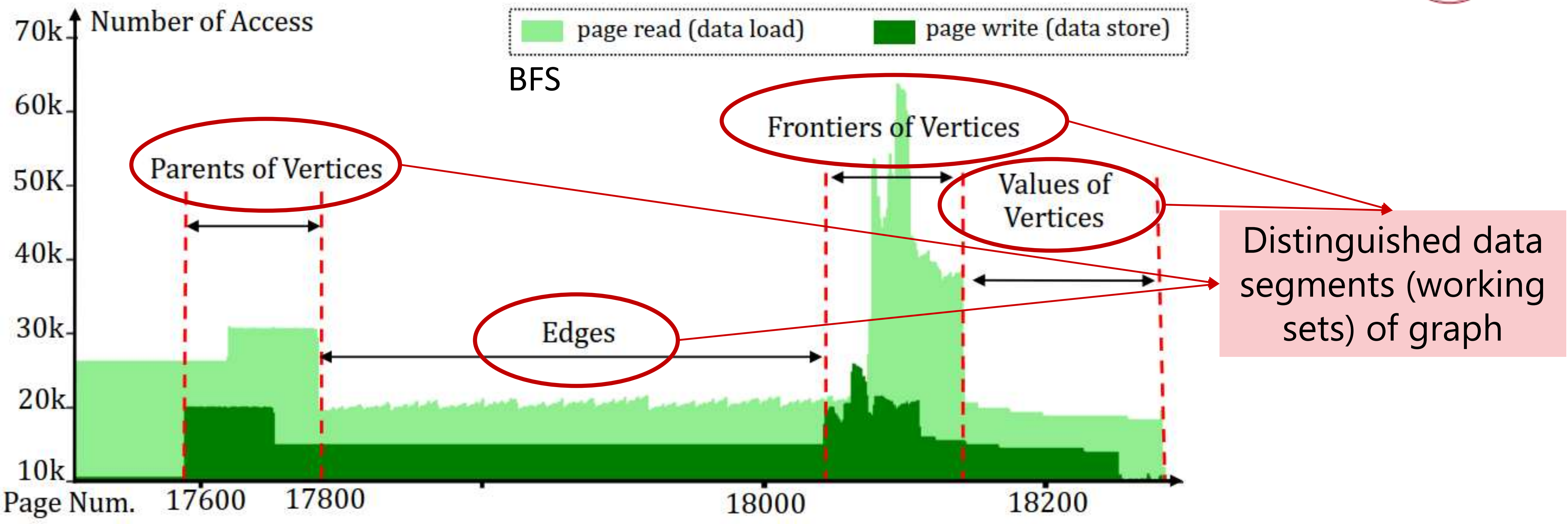
Fargraph mainly consists of two parts:

The **front-end:** ***graph-aware data segment offloading strategy***

The **back-end:** ***iteration-friendly far memory interaction optimization***



Fargraph Front-end: Data Segment Offloading



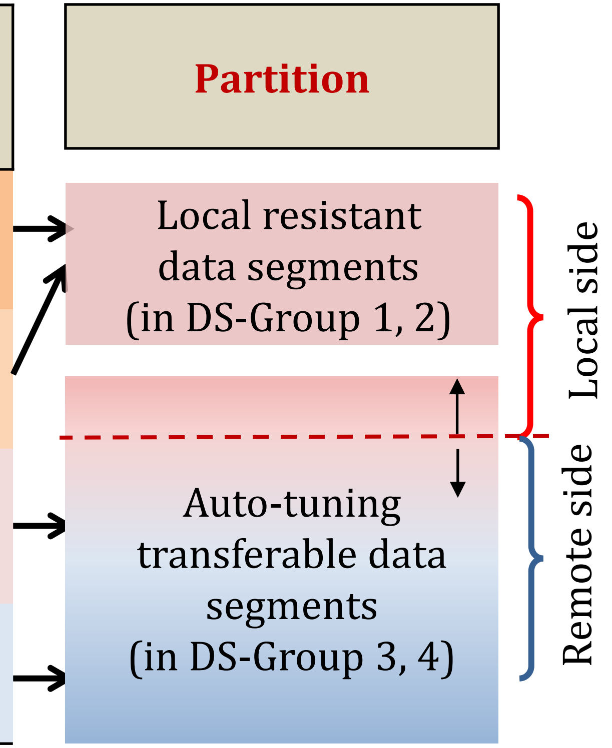
- Graph applications show obvious page allocation areas of each data segment such as parents of vertices, frontiers of vertices, edge lists, etc.
- Different data segments show obviously different memory read/write behaviors.

Fargraph Front-end: Data Segment Offloading

(1) Graph Data Segment Grouping: We analyze data segments of graph programs and classify them into 4 groups according to the memory offloading sensitiveness.

(2) Flexible Data Segment Offloading: We give preferable offloading orders of data segments to offload data in an efficient and auto-tuning way.

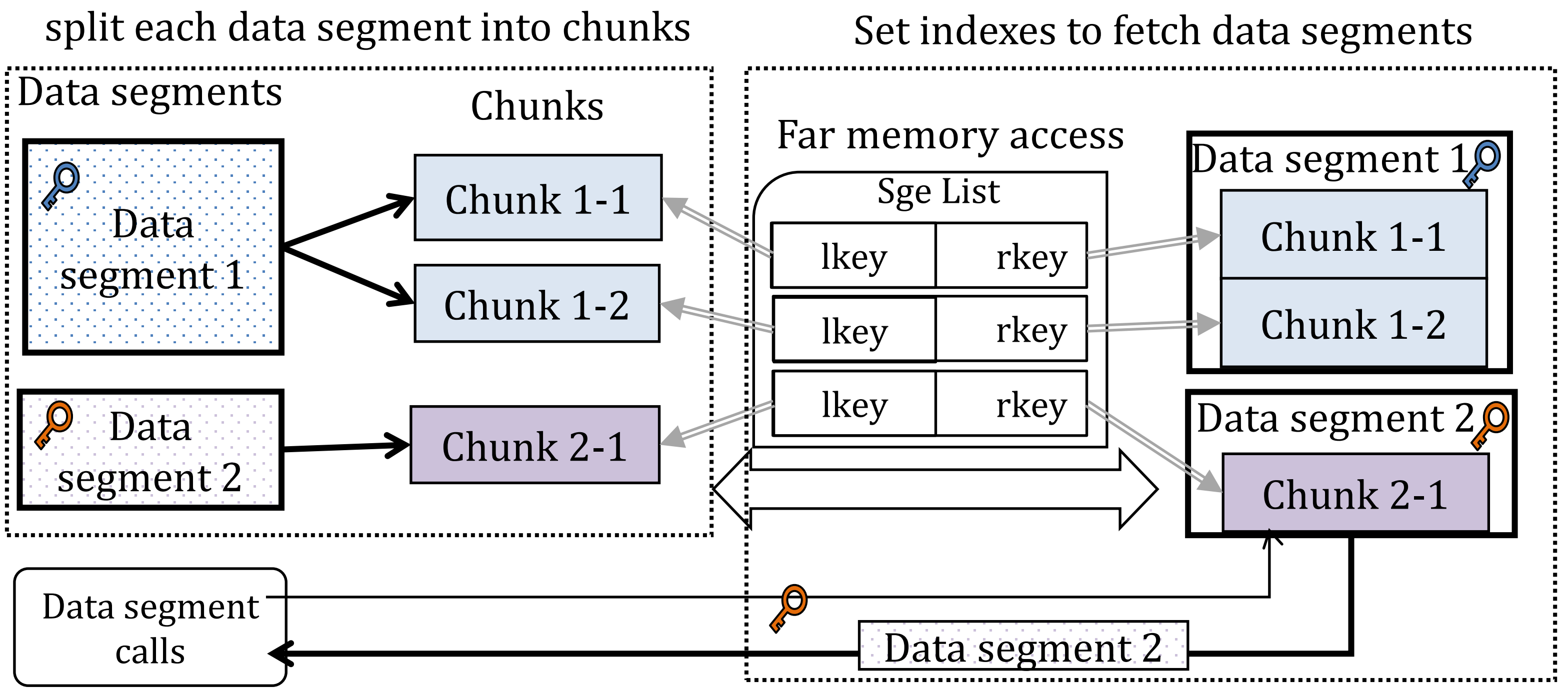
Data segments (DS)	Amount	I/O pressure		Classification
		Write	Read	
Vertex ids, attributes, frontiers, parents, etc.	Small	Much	Much	**DS-Group 1** (MO sensitive)
Intermediate variables, iterators, etc.	Small	Much	Few	**DS-Group 2** (MO less sensitive)
Edge blocks, edge offsets, weights, etc.	Large	Few	Much	**DS-Group 3** (MO less insensitive)
Disposable data, inactive vertices, etc.	Depend	Few	Few	**DS-Group 4** (MO insensitive)

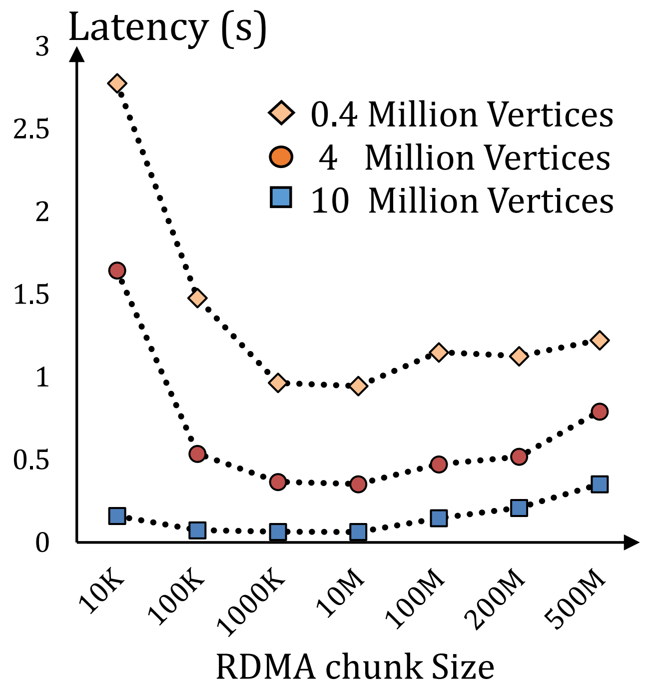


Fargraph Back-end: Far Memory Interaction Optimization

(1) Data Segment Splitting:

- Split data segments into chunks
- Use indexes to facilitate data segment fetching from far memory
- Choose proper chunk size to have better overall performance

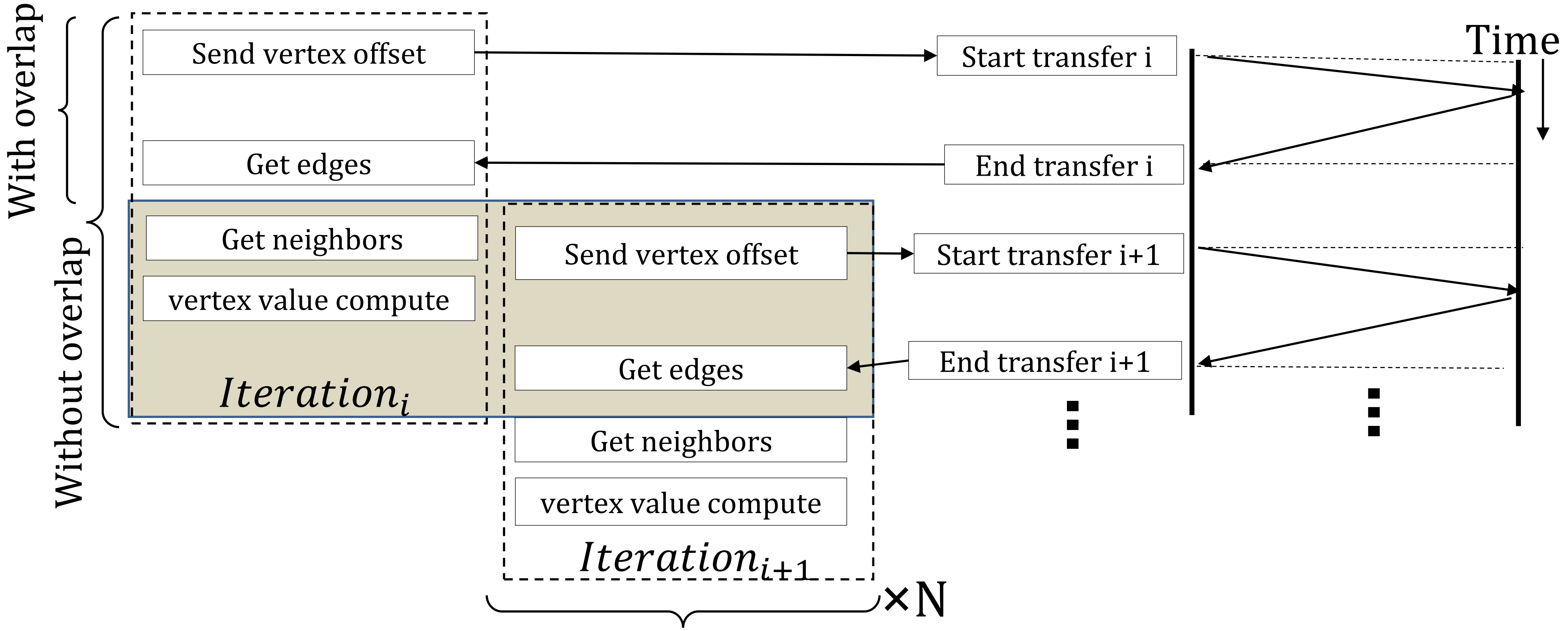




Fargraph Back-end: Far Memory Interaction Optimization

(2) Data Segment Buffering:

- Use RDMA read and write operations to avoid the kernel overhead
- Design buffers that support iteration pipeline overlap
- Hide data transfer time into execution time in each iteration



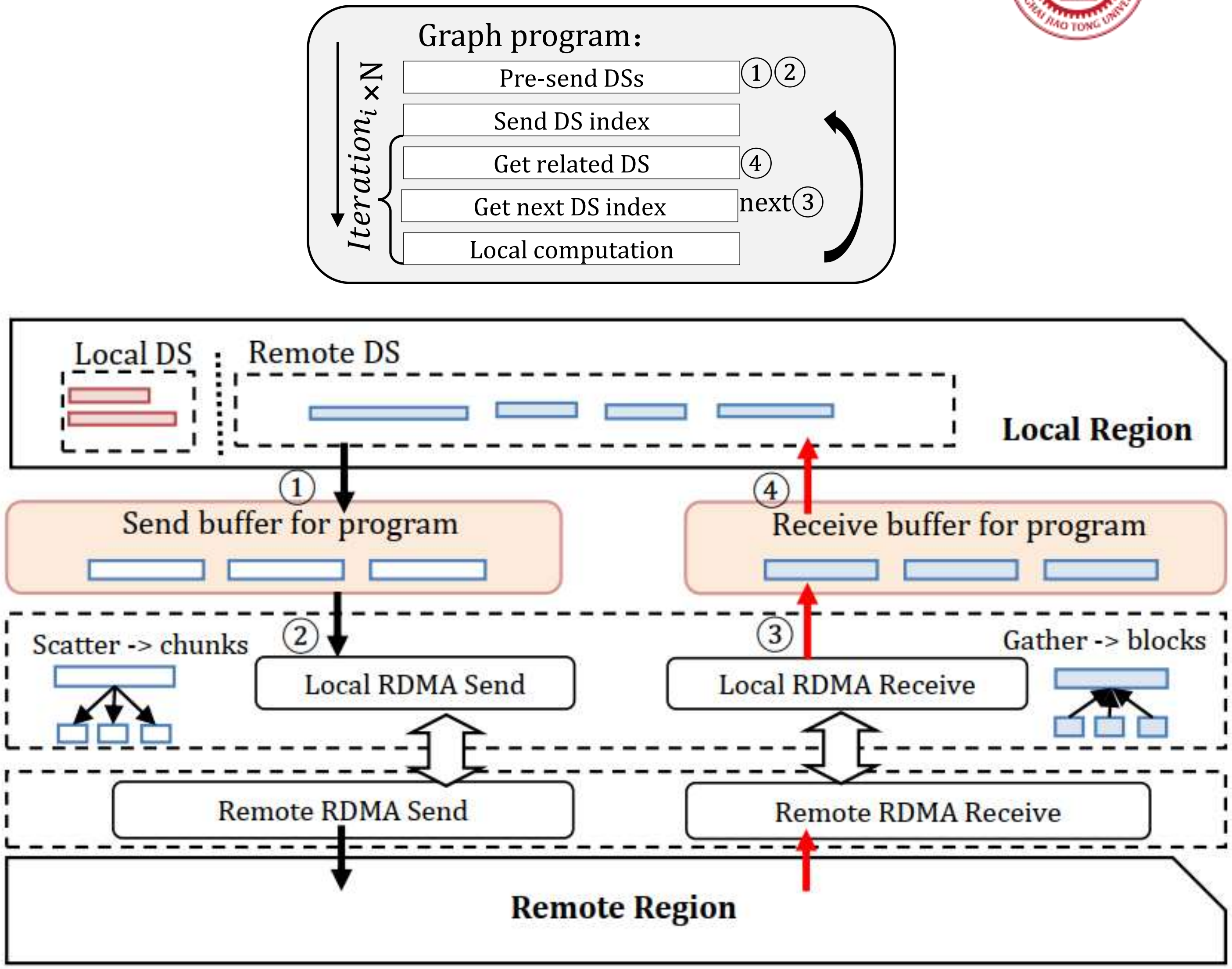
Fargraph Workflow

(1) Preprocessing ①②

- Build RDMA connection
- Prepare buffers
- Add transferable labels
- Set indexes
- Pre-transfer data to far memory

(2) Far Memory Coordination. ③④

- Send indexes
- Fetch them back in order
- Write the data into buffers
- Copy buffer to the local region



Directive-like Implementation

We provide such interfaces :

- Add_transferable_flag() → makes offloading decisions
- Build connection() → starts RDMA network connection
- Far_write_start() → triggers the memory registration and start writing data to far memory
- Far_write_complete() → returns once the sending data is accomplished
- Far_read_start() → starts fetching each data segments by one-sided read
- Far_read_complete() →returns the rkey and index of the fetched data when the data transmission finishes

Algorithm 1 Program Adjustment with Fargraph Interfaces

```
Add_transferable_flag(DS_list, far_ratio, ...);
Build_connection(IP,port,memory_region_size, ...);
//send all TDSS to far memory when preparation
for each DS_i in transferable_DS_list do
    Far_write_start(transfer_flag, DS_i, index, lkey, ...);
end for
Far_write_complete(DS_indexes, rkey, ...);
...continue... //waiting for data segments calls
//start read far DS_Current in another process;
Far_read_start(DS_Current, index, rkey, ...);
while (in each processing loop) do
    ...continue... //original data process
    while calling DS_Current do
        if DS_Current is prepared then
            Far_read_complete(DS_Current,index, rkey, ...);
        end if
    end while
    // start receive the next DS;
    Far_read_start(DS_Next, index, rkey, ...)
    ...continue...//original data process
    if DS_Current finishes occupying then
        Free DS_Current in local RAM;
    end if
end while
```

Outline

- Background
- Motivation
- System Design
- Evaluation
- Conclusion

Evaluation: Experimental Environment

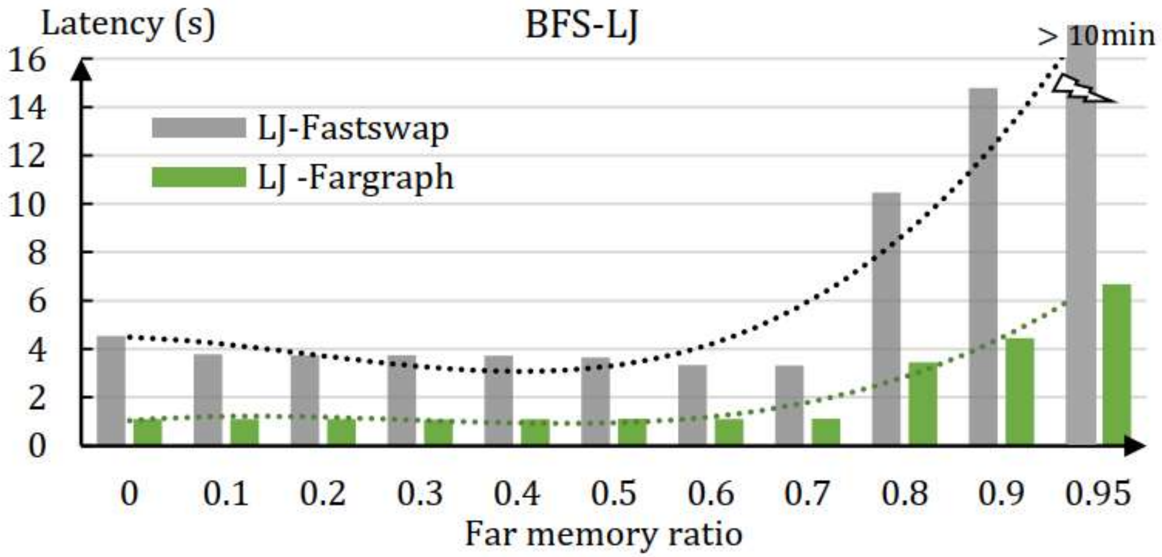
- **Hardware:**
  - Two server nodes
  - 128 GB of memory
  - Dual port Mellanox ConnectX-5 RDMA NIC.
- **Software:**
  - Cgroup2 to limit the memory usage
  - OFED v4.3.0 with RoCE protocol to use RDMA.

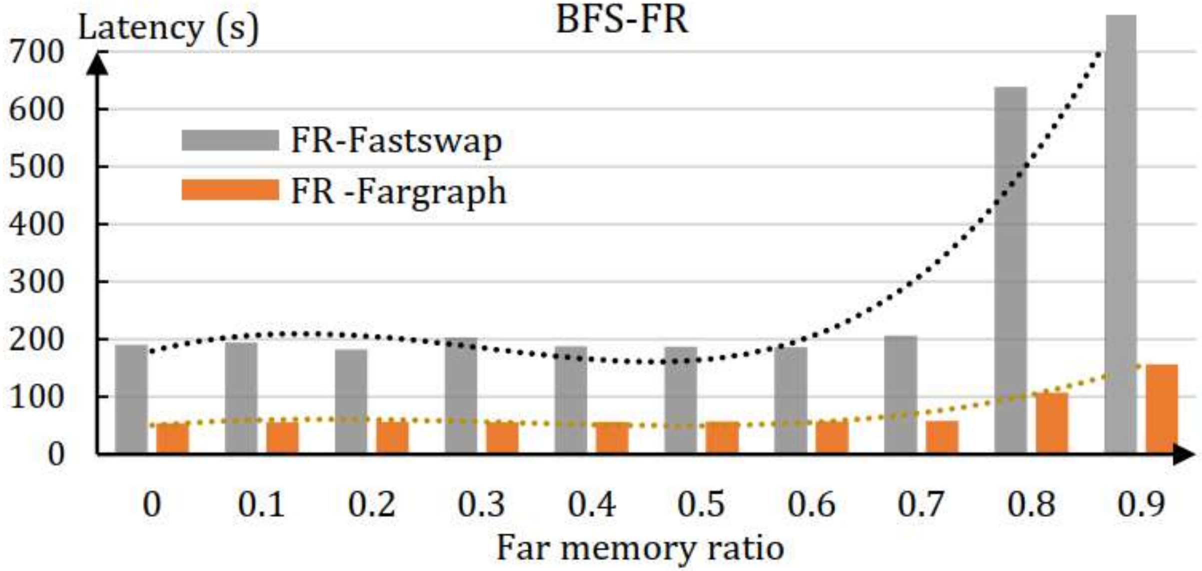
Schemes	Application Framework	Exe. Environment
Original	GridGraph with mem. limitations	Disk
Fastswap	GridGraph without mem. limitations	Fastswap (far memory)
Fargraph	GridGraph without mem. limitations	Fargraph (far memory)
Oracle	GridGraph without mem. limitations	Local main memory

Algorithms	Description	Memory Access Feature
BFS	breadth-first search	random I/O
WCC	weak connected components	random I/O
PageRank	web page ranking	random I/O and sequential I/O
Radii	graph radii estimation	random I/O and sequential I/O

Dataset	$\|V\|$	$\|E\|$	Edge Size	Mem. Footprint
Live Journal (LJ)	4,848 K	69 M	1.1 GB	2.4 GB
Orkut (OR)	3,072 K	117 M	1.8 GB	3.9 GB
Twitter7 (TW)	17 M	477 M	26.3 GB	47.7 GB
Friendster (FR)	65 M	1806 M	32.7 GB	60.4 GB

Evaluation: Efficiency of the Front-end Design



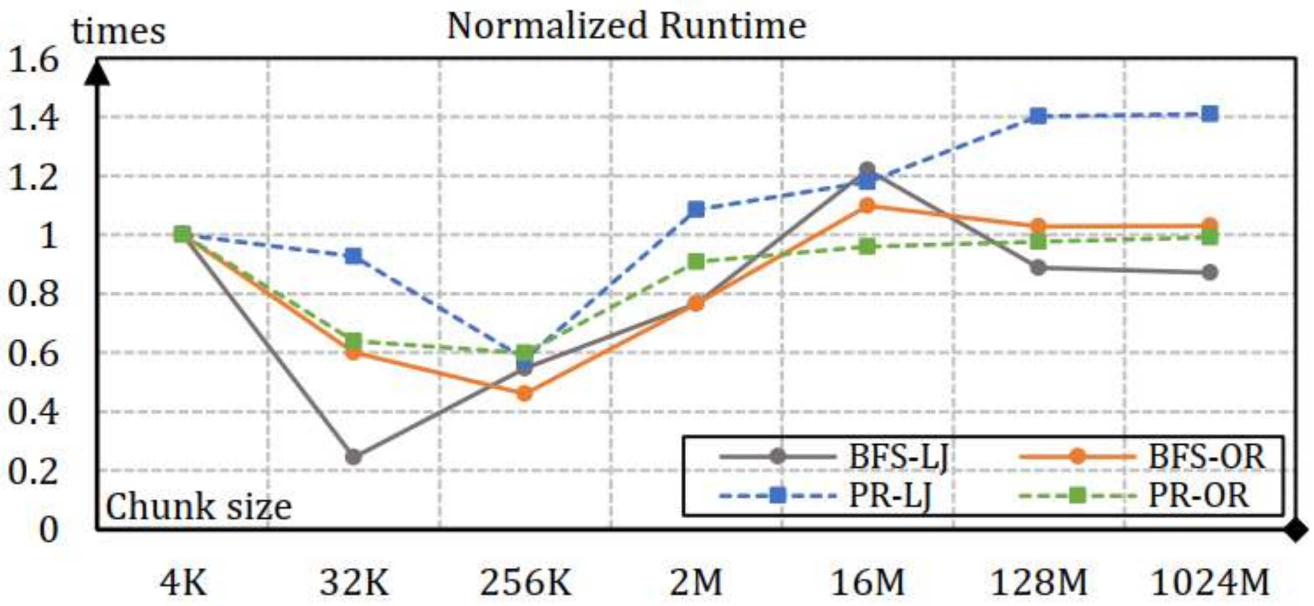


The duration of BFS on dataset LJ and FR on far memory platform Fargraph and Fastswap under rising far memory ratios

- We show lower task durations, especially when far memory ratio is large.
- We have a larger available far memory resource with acceptable performance.

Evaluation: Efficiency of the Back-end Design

(1) Performance Impact of Data Segment Splitting



The normalized performance of four workloads (BFS and Pagerank on dataset LJ and OR) with different chunk sizes.

- The duration under 32K and 256K chunk size is higher than 4K(Page size).
- The best far memory chunk size is determined by the smaller one between RDMA bandwidth and PCIe bandwidth.

Evaluation: Efficiency of the Back-end Design

(2) Performance Impact of Data Segment Buffering

THE DURATION COMPARISON OF FARGRAPH DATA BUFFERING

Duration (s)		BFS			
		LJ	OR	TW	FR
Schemes	Fargraph w/o buffering	3.45	4.53	75.61	107.40
	Fargraph buffering	2.97	3.93	63.23	94.02
Duration reduction	Absolute value	↓ 0.48	↓ 0.6	↓ 12.38	↓ 23.38
	Relative value	14%	13%	12%	13%
		PageRank			
Schemes	Fargraph w/o buffering	7.44	26.80	153.19	233.63
	Fargraph buffering	6.92	23.52	123.70	200.85
Duration reduction	Absolute value	↓ 0.52	↓ 3.28	↓ 29.49	↓ 32.78
	Relative value	7%	12%	19%	14%

- Data segment buffering brings task duration down by up to 19%.
- The duration reduction of BFS is stable while that of PageRank may increase significantly on larger graph datasets.

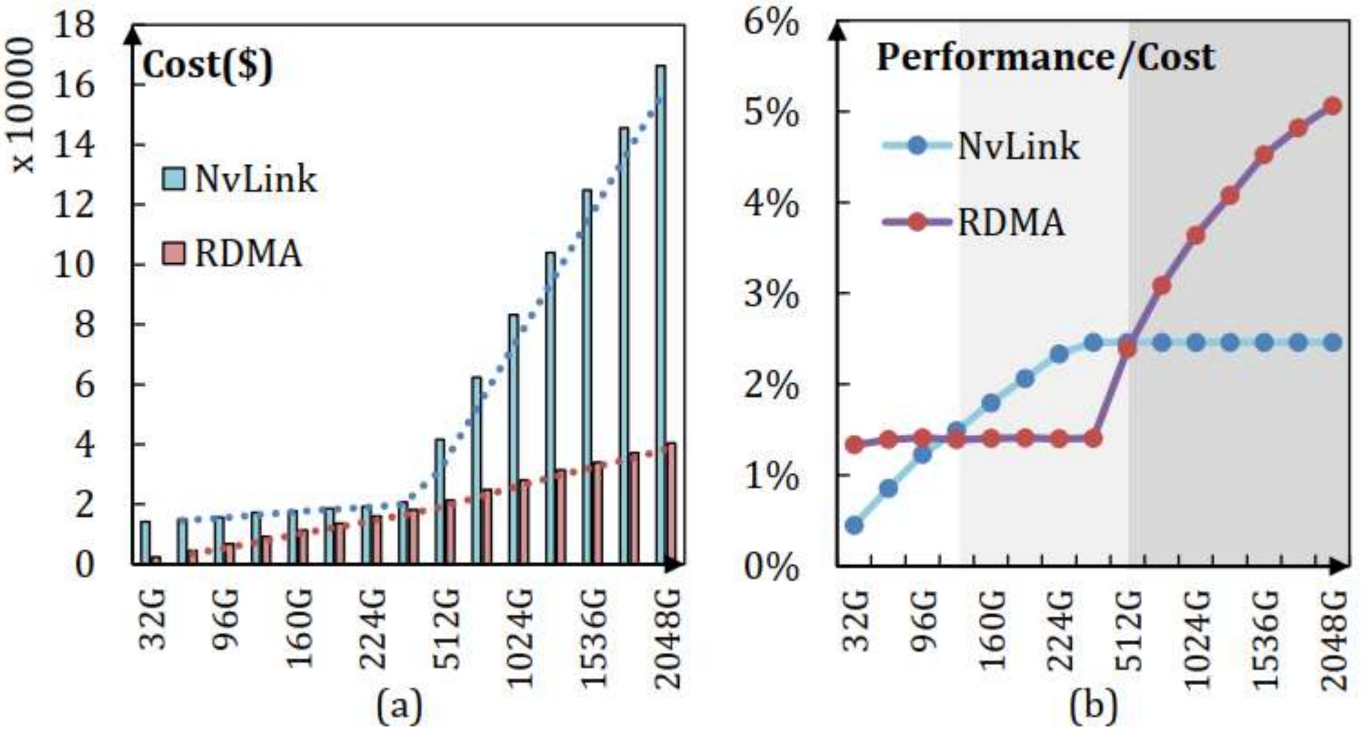
Evaluation: Overall Performance

The total performance comparison of 16 graph workloads with 80% far memory

Results (seconds)	BFS				WCC				PageRank				Radii			
	LJ	OR	TW	FR	LJ	OR	TW	FR	LJ	OR	TW	FR	LJ	OR	TW	FR
Oracle	2.63	2.61	27.88	54.33	0.16	1.59	38.5	69.6	5.33	7.73	115.24	137.76	2.74	9.44	150.36	320.45
Original	9.84	6.08	235.91	637.17	2.36	4.55	144.17	318.8	39.20	74.80	848.00	1153.6	20.75	110.3	1139.0	2578.0
Fastswap	10.46	7.03	262.24	639.0	2.56	6.52	256.4	523.1	25.53	40.80	966.03	1662.0	20.45	135.24	1524.6	3054.8
Fargraph	**2.97**	**3.93**	**63.23**	**94.02**	**1.32**	**2.98**	**70.2**	**98.2**	**6.92**	**23.52**	**123.70**	**200.85**	**5.48**	**20.49**	**350.26**	**652.24**
Sp(Original)	**3.3x**	**1.5x**	**3.7x**	**6.7x**	**1.8x**	**1.5x**	**2.1x**	**3.2x**	**5.7x**	**3.2x**	**6.9x**	**5.7x**	**3.8x**	**5.4x**	**3.3x**	**4.0x**
Sp(Fastswap)	**3.5x**	**1.8x**	**4.1x**	**6.8x**	**1.9x**	**2.2x**	**3.7x**	**5.3x**	**3.7x**	**1.7x**	**7.8x**	**8.3x**	**3.7x**	**6.6x**	**4.4x**	**4.7x**

- Computation-centric algorithms such as Pagerank and Radii performs better compared to the traversal-centric algorithms, such as BFS and WCC.
- Fargraph shows demonstrate the attractive scalability
- We can achieve 6.9× better performance compared to *Original*, and up to 8.3× performance compared to *Fastswap*.

Evaluation: Cost-Effectiveness of Memory Capacity



- The cost of each NVLink-based machine is almost 10-100x more expensive than a RDMA-based machine.
- The cost-effectiveness of RDMA-based design can be better when the requested extra memory capacity is in the range of 128-512G.

Conclusion

In this work, we explore the potential of graph processing on far memory.

- Capturing graph properties, the data segment grouping method can achieve better far memory offloading effectiveness.
- Configuring the data transfer carefully, the data splitting and buffering can improve performance of iterative graph execution model.
- Our design opens a door for more efficient big data analysis in the next-generation cloud on disaggregated architecture.
- We will continue to improve our design for better scalability and higher memory efficiency in the future work.

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Thank You

Jing Wang

jing618@sjtu.edu.cn