Fargraph: Optimizing Graph Workload on RDMA-based Far Memory Architecture

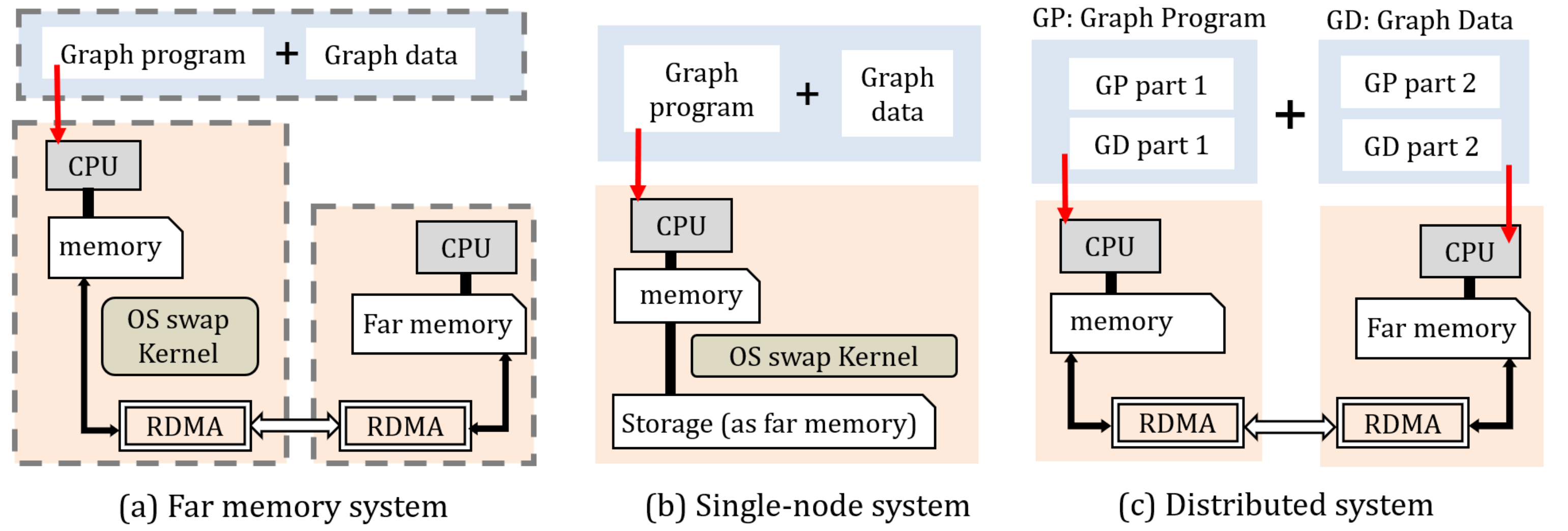
Jing Wang, Chao Li*, Taolei Wang, Lu Zhang, Pengyu Wang, Junyi Mei, Minyi Guo

Shanghai Jiao Tong University

Introduction & Background

Disaggregated architecture[1] brings new opportunities to memory-consuming applications like graph analytics by borrowing memory from a remote node.

- Far memory system provides a new option for scaling out graph processing on both single-node and distributed-computing system.
- OS swap mechanism can be leveraged to design transparent far memory access on RDMA, such as Infiniswap[4] and Fastswap[2].



(a) Far memory system (b) Single-node system (c) Distributed system

Fig 1: Different architectures that can be leveraged for graph processing.

We take the first step to design a far memory coordination strategy for enhancing graph processing applications on top of RDMA-based far memory system.

Preliminary & Observations

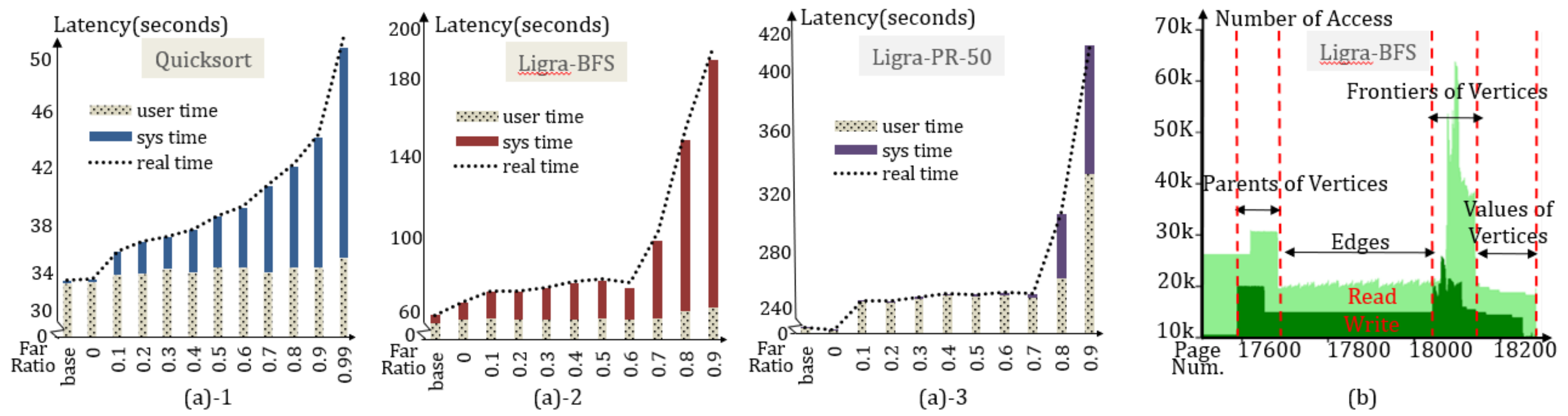


Fig 2: a)-1/2/3 are the runtime trends of three workloads on growing far memory proportions. b) is r/w access number over each page and shows distinguished data segments of graph programs.

Task Runtime Analysis by changing far memory ratio in Fig2-(a)-1/2/3

- **Observation1:** The overhead of far memory runtime mainly comes from OS-level swap, which is more serious with far memory ratio increasing.
- **Observation2:** Graph tasks show turning points of latency trends, different from the continuous trends of computation-intensive program Quicksort.

Graph Working Set Analysis by counting page access number in Fig2-(b)

- **Observation3:** Edges are often accessed less compared to vertex-related data while memory occupation of edges is much larger than vertices.
- **Observation4:** Graph shows obvious page allocation areas of each data segment such as parents of vertices, frontiers of vertices, edge lists, etc.

Fargraph Design

Fargraph Overview

- The front-end proposes a **working set partition strategy** for graph programs.
- The back-end designs **far memory interaction optimization** for far memory access.

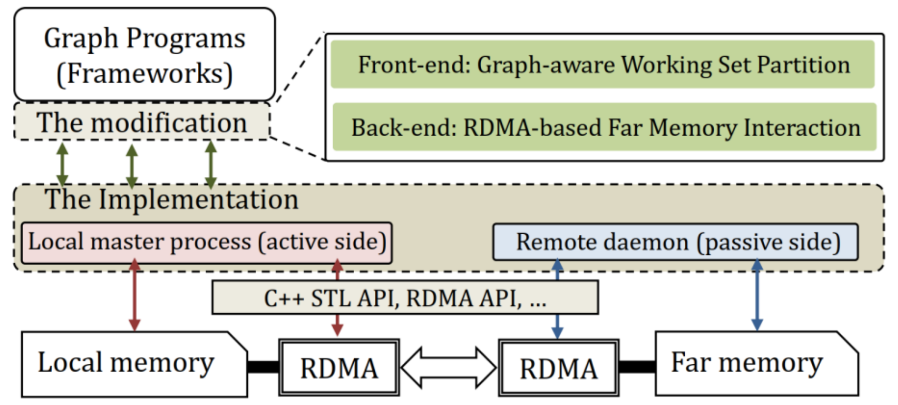
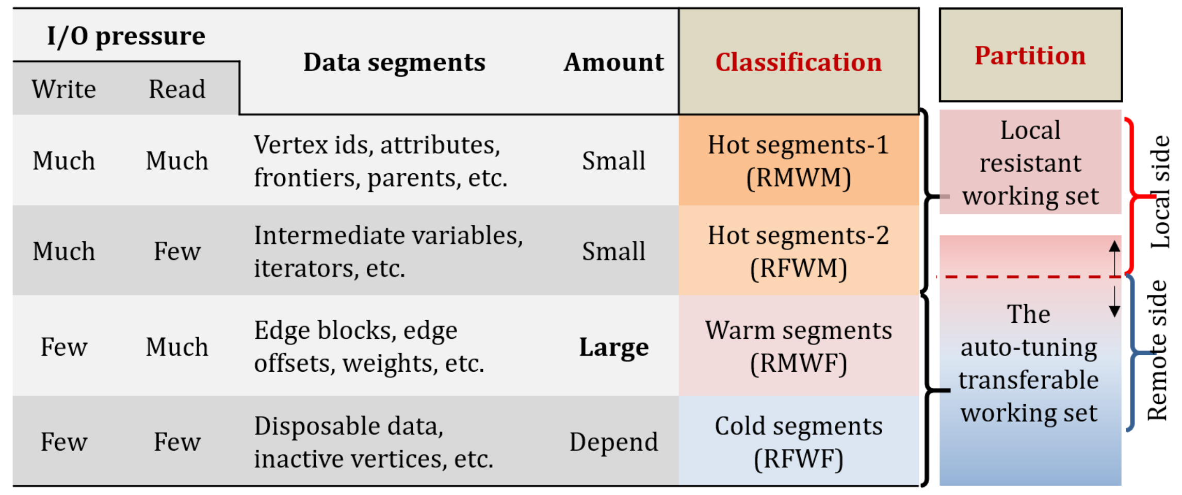


Fig 3: The overview of Fargraph platform organization.

Fargraph Front-end: working set partition

(1) Classification: We analyze data segments of graph programs offline and classify them into *hot*, *warm and cold* working sets, as shown in Fig 4.

(2) Partition: We determine data segments that are preferable to be transferred to remote side in advance for each particular working set.



I/O pressure		Data segments	Amount	Classification	Partition
Write	Read				
Much	Much	Vertex ids, attributes, frontiers, parents, etc.	Small	Hot segments-1 (RMWM)	Local resistant working set
Much	Few	Intermediate variables, iterators, etc.	Small	Hot segments-2 (RFWM)	
Few	Much	Edge blocks, edge offsets, weights, etc.	**Large**	Warm segments (RMWF)	The auto-tuning transferable working set
Few	Few	Disposable data, inactive vertices, etc.	Depend	Cold segments (RFWF)	

Fig 4: The data segments classification and working set partition strategy.

Auto-tuning Transferable Working Set (TWS):

- We give high priority to the read-only data in the transferable working set.
- The local and remote proportion of TWS is auto-tuning accordingly.

Fargraph Back-end: far memory interaction

***Key 1:* RDMA One-side Read and Write.**

We use user-level RDMA read and write operations to avoid the kernel overhead.

***Key 2:* Index configuration for data segments.**

We set indexes for each data segment to reduce the cost of calling data from far memory.

***Key 3:* Proper size of data chunk transfer.**

We transfer data based on a finer-grained unit data chunk and choose the optimal size of the chunk.

Fargraph Workflow: optimizing graph processing on far memory

Phase 1: Register memory regions and bind IP address to start connection.

Phase 2: Pre-Transfer the data segments in TWS to far memory and start local access.

Phase 3: One-sided read/write to send and fetch data in each iteration.

Phase 4: Finish and disconnection.

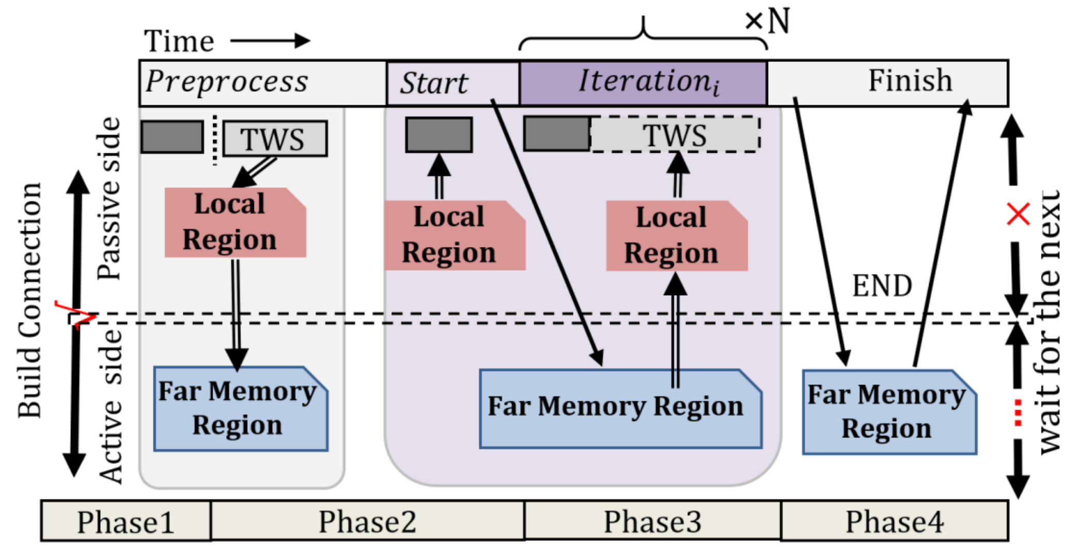


Fig 5: The four phases of Fargraph workflow.

Exprimental Evaluation

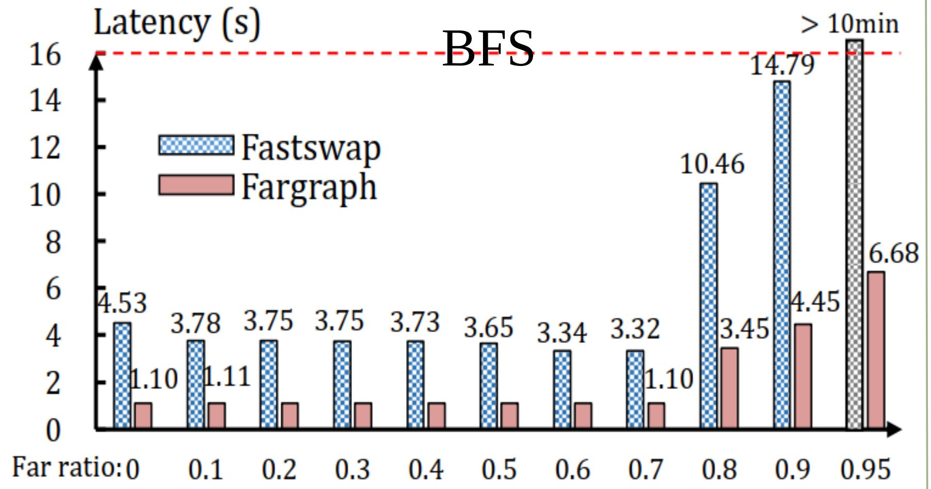
Table 1: Real graph datatsets

Graph Label	Real graph	$\|V\|$	$\|E\|$	Edge Size
LJ	Live Journal	4,847,571	68,993,773	1.08G
OR	Orkut	3,072,441	117,185,083	1.77G
TW	Twitter7	17,069,982	476,553,560	26.27G
FR	Friendster	65,608,366	1,806,067,135	32.36G

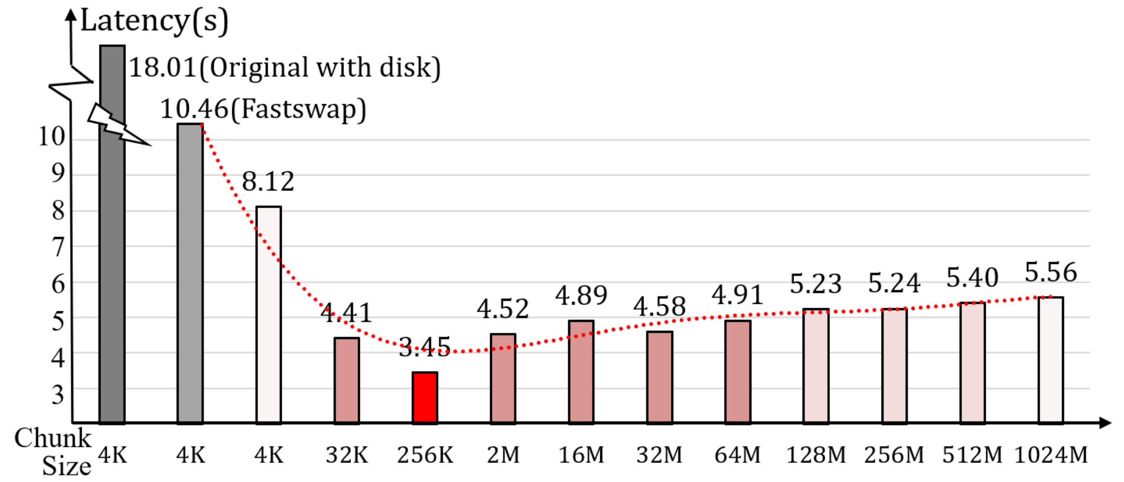
The far memory platform environment:

- Original(Gridgraph[3])
- Gridgraph+Fastswap[2]
- Gridgraph+Fargraph

There are 4 datasets in our experiment. The benchmark algorithms are BFS and PageRank.



Result 1: Fargraph successfully reduces the latency when using far memory, improving performance by up to 4x compared to Fastswap.



Result 2: We plot a smile-like runtime curve of BFS on LiveJournal with various chunk size. Our experiments show that we can obtain the best performance at 256KB for each workload.

Table 2: The total latency comparison of graph workload execution on with 80% far memory and 256K Chunk size.

Far ratio=0.8 Chunk size=256K	BFS				PageRank			
	LiveJournal	Orkut	Twitter	Friendster	LiveJournal	Orkut	Twitter	Friendster
Original (Gridgraph)	9.84	6.08	235.91	637.17	39.20	74.80	848.00	1153.60
Gridgraph + Fastswap	10.46	7.03	262.24	639.00	25.53	40.80	966.03	1662.00
Gridgraph + Fargraph	**3.45**	**4.53**	**75.61**	**107.40**	**7.44**	**26.80**	**153.19**	**233.63**
Speedup (Fastswap)	**3.0x**	**1.6x**	**3.5x**	**5.9x**	**3.4x**	**1.5x**	**6.3x**	**7.1x**

Our Fargraph is more efficient than disk-swap Original baseline and Fastswap baseline. Comparing the results of BSF and Pagerank, the speedup of Pagerank is larger than BFS.

Conclusion

In this paper we explore graph processing on RDMA based far memory architecture.

- Graph processing can benefit from good working set partition when deploying on far memory.
- Tuning critical metrics on RDMA properly brings performance chances for far memory access.
- We can speedup the state-of-the-art far memory platform Fastswap by up to 7.1x.

Reference

[1] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. Legoos: Adisseminated, distributed OS for hardware resource disaggregation. InOSDI’18

[2] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ouster-hout, Marcos K Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker.2020. Can far memory improve job throughput?. InEuroSys’20. 1–16

[3] Xiaowei Zhu, Wentao Han, and Wenguang Chen. GridGraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning.

[4] . Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang GShin. 2017. Efficient memory disaggregation with Infiniswap. In NSDI’17. 649–667