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

ABSTRACT

Disaggregated architecture brings new opportunities to memory-consuming applications like graph analytics. It allows one to outspread memory access pressure from local to far memory, providing an attractive alternative to disk-based processing. In this paper, we take the first step to analyze the impact of graph processing workloads on disaggregated architecture. We design Fargraph, a far memory coordination strategy for enhancing graph processing workloads on top of RDMA-based far memory system.

KEYWORDS

disaggregation, far memory, RDMA, graph processing, working set

1 INTRODUCTION AND BACKGROUND

Nowadays various data-intensive applications such as graph processing, machine learning, and data mining demand continued research on better memory performance. An important trend is to build disaggregated memory resource pools and enable *far memory* (i.e., remote main memory) accesses [3]. Far memory system allows one to opportunistically borrow memory resources from a remote node (Figure 1 (a)) which provides a new option for scaling out graph processing on both single-node (Figure 1 (b)) and distributed computing (Figure 1 (c)). Far memory helps to outspread memory flexibly as a good complementary of existing systems.

However, transparent far memory management may not provide the best performance. Far memory platforms often replace the original swap space with far memory space [1, 2], passively leaves all the pressure of deciding thrown-out parts to the OS kernel. This system overhead becomes the main challenge for swap-based far memory platform. On the other hand, one could miss great performance optimization opportunities when accessing far memory in a fine-grained, irregular manner [1]. Oftentimes, a graph consists of small amounts of write-intensive vertices and large amounts of read-only edges. We need to determine the appropriate data segments that should be moved out to far memory.

In this paper, 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. We investigate far-memory based graph programs from two aspects: working set partition and far memory interaction. We identify data segments that are most suitable to be placed on far memory. We also reconfigure the RDMA back-end system to fit graph workloads better.

2 PRELIMINARY AND OBSERVATIONS

To show the opportunity of graph processing on far memory environment, we analyze the impact of far memory usage on task latency in different scenarios. Specifically, we define a far memory ratio α, the ratio of far memory usage to all the memory occupation of applications. The base in x axis refers to no cgroup baseline.

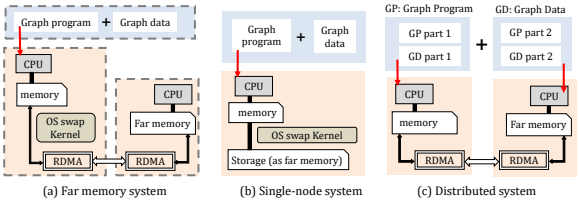

Figure 1: Different architectures for far memory platforms.

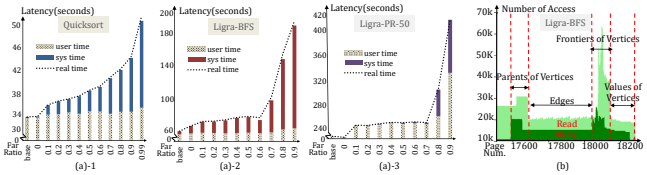

Figure 2: (a)-1/2/3: The runtime trends with growing far memory ratios. b): Page access frequency over each page.

Figure 2-(a) shows the the system time and user time of different applications on swap-based far memory platform Fastswap [1] with different far memory usage ratio α. We can see that the overhead of Fastswap mainly comes from OS-level swap mechanism operations, which is more obvious when far memory ratio increases. Additionally, we observe that graph programs have an obvious performance turning point other than a continuous runtime increase in Quicksort, a computation-intensive program. The latency increases rapidly if far memory usage is larger than the turning point.

Graph has distinguishable working sets. Figure 2 b) counts the read/write access number of each page in BFS. It shows edges are often accessed less compared to vertex-related data while memory occupation of edges is much larger than vertices. In addition, there are some obvious page allocation areas of each data segment such as parents of vertices, frontiers of vertices, edge lists, etc.

3 FARGRAPH DESIGN

Fargraph consists of two parts: the front-end working set partition strategy and the back-end far memory interaction optimization, as Figure 3 shows. The front-end mainly analyzes the memory access patterns of graph programs and partition data between local memory and remote memory. The back-end works cooperatively to build up efficient memory interactions for the entire program.

3.1 Front-end: working set partition

The front-end has two primary steps. (1) Classification: We analyze data segments of graph programs offline and classify them into *hot, warm and cold* working sets. We treat the warm and cold working sets to be transferable working sets (TWS) by default. (2)

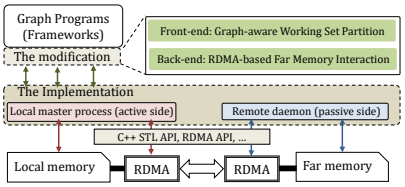

Figure 3: Fargraph platform overview.

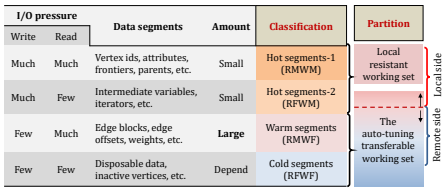

Figure 4: Working set partition.

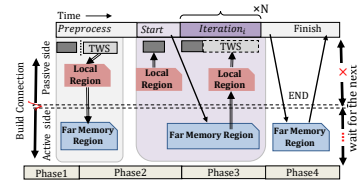

Figure 5: A four-phase workflow.

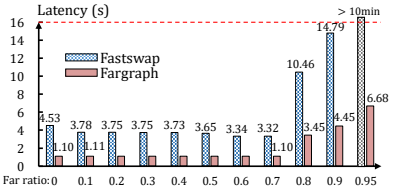

Figure 6: The runtime comparison of Fargraph and Fastswap under different far memory proportion.

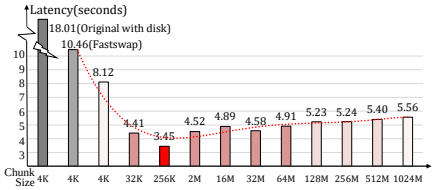

Figure 7: The performance of BFS on Fargraph with different chunk sizes (compared with 4K of Fastswap).

Far ratio=0.8 Chunk size=256K	LiveJournal	Orkut	Twitter	Friendster
	Size=1.08G	Size=1.77G	Size=26.27G	Size=32.36G
Original BFS(Gridgraph)	9.84s	6.08s	235.91s	637.17s
BFS+Fastswap	10.46s	7.03s	262.24s	639.00s
BFS+Fargraph	3.45s	4.53s	75.61s	107.40s
Speedup(Fargraph/Fastswap)	**3.0x**	**1.6x**	**3.5x**	**5.9x**
Original Pagerank(Gridgraph)	39.20s	74.80s	848.00s	1153.60s
Pagerank+Fastswap	25.53s	40.80s	966.03s	1662.00s
Pagerank+Fargraph	7.44s	26.80s	153.19s	233.63s
Speedup(Fargraph/Fastswap)	**3.4x**	**1.5x**	**6.3x**	**7.1x**

Figure 8: The total runtime comparison of 8 workloads on no-far-memory and Fastswap baselines.

Partition: We determine data segments that are preferable to be transferred to remote side in advance for each particular working set. A straightforward idea is to keep hot segments locally and move all the TWS to remote memory. However,to achieve a better trade-off between local memory saving and far memory performance, we define auto-tuning TWS, which means the size of TWS in local memory is changeable. We allow one to keep part of the TWS in local memory (as indicated in Figure 4). Most data segments of the TWS in graph workloads are read-only (e.g., edges and attributes), and we give high priority to the read-only data in our TWS.

3.2 Back-end: far memory interaction

In Fargraph back-end design, we optimize far memory access performance based on appropriate configurations of RDMA. We explore the benefits of RDMA from three key aspects below.

RDMA one-side read and write: Our far memory interaction can bypass the kernel and communicate on event queues with user-level RDMA read and write operations. RDMA one-sided read mechanism allows one to directly fetch data from remote memory without waiting for system handshaking.

Index configuration for data segments: We use indexes for fetching remote data segments. It reduces the traversal cost of finding the corresponding data segments in far memory. For example, we use the vertex IDs of each grid block as the indexes of edges that are stored in the far memory.

Proper size of data chunk transfer: We split each data segment into multiple data chunks(finer-grained units) when writing to the far memory, and merge these chunks back to the data segments when fetching back. Since the size of each data segment is somehow different, we transfer data based on an optimal size of the chunk.

Fargraph Workflow: In Figure 5, we show the workflow with all optimization involved. There are two procedures on the active (initiator client) side and the passive (responder server) side. The key strategies are all implemented on the active side with the aid of the passive side.

4 EVALUATION

We build our far memory platform based on two nodes with Dual-Port Mellanox ConnectX-5 RDMA NICs separately. We use Cgroup2 to limit the memory usage. The benchmark algorithms are BFS and Pagerank in GridGraph [4] graph framework. We compare Fargraph with the Original system (no far memory) and Fastswap platform.

As Figure 6 shows, Fargraph successfully reduces the runtime of Fastswap[1] on BFS[4] . When the far memory ratio reaches 0.95, we find that Fastswap is too slow to complete the execution even after tens of minutes of execution. In Figure 7, 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 chunk size for each workload. The reason for the smile-like curve is that the far memory chunk size is determined by the smaller one between RDMA bandwidth and PCIe bandwidth. Comparing the results of BSF and Pagerank in Figure 8, The speedup of Pagerank is larger than BFS due to the ratio of computation and memory access. The final result in Figure 8 shows that Fargraph can speedup disk-swap Original baseline and Fastswap baseline by up to 7.1x.

5 CONCLUSION

In conclusion, graph processing can benefit from good working set partition and proper RDMA tuning. We show that Fargraph has great performance potential on far memory architecture. We will continue to improve our design in the future work.

REFERENCES

[1] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can far memory improve job throughput?. In *EuroSys*. 1–16.

[2] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. 2017. Efficient memory disaggregation with infiniswap. In *NSDI*. 649–667.

[3] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. Legoos: A disseminated, distributed OS for hardware resource disaggregation. In *OSDI'18*.

[4] Xiaowei Zhu, Wentao Han, and Wenguang Chen. 2015. GridGraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning. In *ATC*. 375–386.