International Journal of Parallel Programming (2020) 48:98–118
https://doi.org/10.1007/s10766-019-00651-0

CSMqGraph: Coarse-Grained and Multi-external-storage Multi-queue I/O Management for Graph Computing

Shuo Chen[1] · Zhan Shi[1] · Dan Feng[1] · Shang Liu[1] · Fang Wang[1] · Lei Yang[1] · Ruili Yu[1]

Received: 11 August 2019 / Accepted: 11 November 2019 / Published online: 15 November 2019

Abstract
As graphs continue growing, external storage graph processing systems serve as a promising alternative to distributed in-memory solutions for low cost and high scalability. To obtain high I/O throughput, these systems usually use multiple external storage devices. They adopt the operating system I/O management method based on striped volume, resulting in unsatisfactory performance, such as low sequential bandwidth utilization of each external storage device, limited I/O parallelism and expensive management overhead. In this paper, we analyzed the problems of the operating system I/O management method based on striped volume. Then we designed CSMqGraph, a graph processing system adopts coarse-grained striping method matching sequential large I/O to fully utilize the maximum sequential bandwidth of each external storage device and an I/O management strategy based on multi-external-storage multi-queue making I/O threads dedicated to each external storage device to further improve I/O throughput and fully exploit the parallelism of multiple external storage devices. For different graph algorithms and datasets, our evaluation shows that CSMqGraph consistently outperforms state-of-the-art engines GridGraph by up to 40%, and has better I/O scalability.

Keywords Graph processing system · External storage processing · Multiple external storage devices · Coarse-grained · I/O management

1 Introduction

Graphs are powerful data structures that have been used broadly to represent the relationships among various entities, and are being widely used in data modeling in various fields, such as social networking [2,4], web search [7,10,17], road networks [16], protein networks [3,9], healthcare [8], information and cyber-physical systems.

Extended author information available on the last page of the article

Therefore, large-scale graph analysis has emerged as a fundamental computing pattern in both academia and industry.

Generally speaking, many graph algorithms, such as breadth-first search, PageRank need to access the adjacency lists of the vertices. While graph algorithms that perform edge traversals on graphs induce many small, random I/Os [29] because edges are encoded non-local structure among vertices. It will make even the simplest graph traversal algorithm become extremely slow when the graph is massive and has to be stored in slow external storage devices. So I/O becomes a major performance bottleneck in external storage graph processing systems.

To tackle the I/O challenges in large-scale graph processing, Many distributed graph processing systems like HybridGraph [26], Chaos [20] have been proposed in the past few years. They are able to handle large-scale graphs by exploiting the powerful computation resources of clusters. However, load imbalance [11,19], synchronization overhead [27] and fault tolerance overhead [25] are still challenges for graph processing in distributed environment.

Recent studies have shown that single machine external storage graph processing systems can process large-scale graphs with billions of vertices and hundreds of billions of edges, and achieve the performance comparable with a distributed system. They focus on accelerating data access on external storage devices. However, a single external storage device cannot provide enough bandwidth, which performance is limited by slow external storage access. Therefore, multiple external storage devices serve as a better alternative when external storage graph processing systems require high I/O throughput for processing large-scale graphs. Existing multiple external storage graph processing systems such as GridGraph [30] adopt the operating system I/O management method based on striped volume. The ideal result of using multiple external storage devices is that the graph processing systems can use each device in full parallel and balance, and make full use of the maximum sequential bandwidth of each external storage device. However, the operating system I/O management method based on striped volume suffers not only from complex I/O management but also the lower I/O throughput. It cannot make full use of the maximum sequential bandwidth of each device and give full play to the parallelism of multiple external storage devices.

In order to solve the above problems, First, we analyzed the problems of the operating system I/O management method based on striped volume. Second, we analyzed the graph data access characteristics of state-of-the-art out-of-core graph engines. We presented a coarse-grained striping method matching sequential large I/O to fully utilize the maximum sequential bandwidth of each external storage device. Third, for I/O requests still across two or more external storage devices, we proposed an I/O management strategy based on multi-external-storage multi-queue, which performs I/O management in the application layer, such as address mapping, decomposition, prefetch merging and dispatching of I/O requests, and makes I/O threads dedicate to one external storage device. Then we have designed and developed CSMqGraph, a graph processing system based on coarse-grained and multi-external-storage multi-queue I/O management strategy.

For different graph algorithms and datasets, the results show that comparing with GridGraph, CSMqGraph's performance is improved in all cases.It outperforms external storage graph processing system GridGraph by up to 40% and has better I/O

scalability. Then we perform the comparison test of device I/Os and I/O throughput, which proves that the I/O management strategy can effectively reduce device I/Os and improve I/O throughput.

The remainder of this paper is organized as follows. Section 2 discusses the motivation of this work. Section 3 outlines our approach, followed by experimental evaluation in Sect. 4. Section 5 gives a survey of related work. Finally, we conclude this paper in Sect. 6.

2 Problem Presentation and Motivation

The operating system I/O management method based on striped volume is a common method for the multiple external storage graph processing systems, such as GridGraph, G-Store [12], NXgraph [1], etc. They take fine-grained stripes, which doesn't consider the characteristics of the processed graph data. The distribution of data completely relies on transparent volume I/O management. The parallelism of multiple external storage devices and the maximum sequential bandwidth of each external storage device cannot be fully utilized to achieve ideal I/O performance. Figure 1 shows the average throughput and bandwidth utilization of each external storage device while GridGraph runs different graph algorithms on Twitter [13]. The average throughput of each external storage device is in the range of 138–156 MB/s. It does not reach its maximum sequential bandwidth. The bandwidth utilization is 66–75 %, there is still a lot of room for improvement. Figure 2 shows the average throughput and bandwidth utilization of each external storage device when GridGraph executes the WCC algorithm on Twitter with different numbers of external storage devices. The maximum bandwidth utilization of each device is 73% and decreases as the number of external storage devices increases. And the minimum of that is 66%.

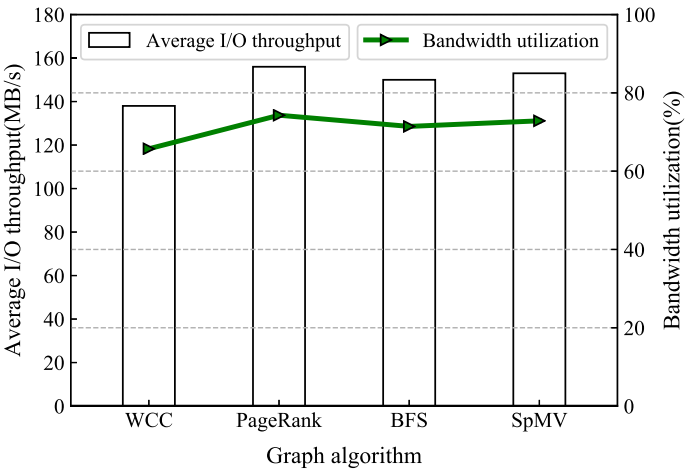

Fig. 1 Average I/O throughput and bandwidth utilization on Twitter graph with different graph algorithms

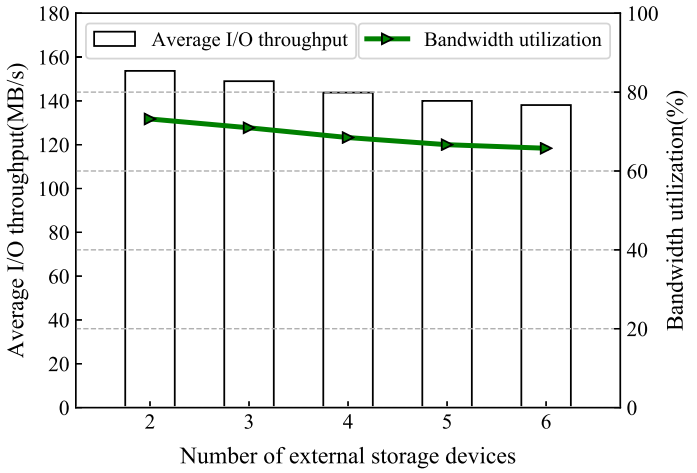

Fig. 2 Average I/O throughput and bandwidth utilization of WCC on Twitter graph with different numbers of external storage devices

External storage graph processing systems adopts the operating system I/O management method based on striped volume. The distribution of data completely relies on transparent volume I/O management. The boundaries of partitions and each I/O request will not be guaranteed to be aligned with the boundary of the stripe unit. So it is inevitable that one I/O request is distributed to multiple external storage devices. And each I/O thread cannot be dedicated to a single external storage device. It cannot effectively utilize optimization strategies such as I/O sorting and merging in the operating system. The parallelism of multiple external storage devices and the maximum sequential bandwidth of each external storage device cannot be fully utilized to achieve higher I/O performance. It also causes unnecessary I/O service overhead and locks contention overhead. However, in the multiple external storage devices graph processing systems, the sequentiality of each external storage device and the parallelism of multiple external storage devices are not contradictory. By analyzing the access characteristics of the graph data, we can adjust the size of the stripe depth to better utilize the sequential bandwidth of each external storage device and the parallelism of multiple external storage devices. These observations motivate us to develop a solution for efficient use of multiple external storage devices and the data access channel to achieve higher throughput.

3 Coarse-Grained I/O Management Strategy Based on Multi-External-Storage Multi-Queue

When using multiple external storage devices to improve the performance of I/O and alleviate I/O pressure, there are some problems, such as insufficient utilization of sequential bandwidth, limited parallel I/O capability, various locks contention overhead and synchronization overhead. Therefore, the I/O performance is not ideal

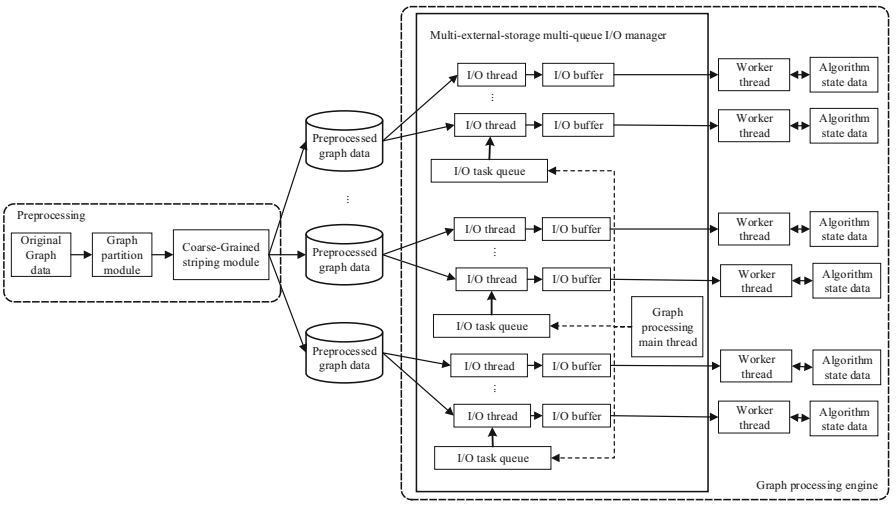

Fig. 3 CSMqGraph architecture overview

enough. We proposed a coarse-grained striping method matching sequential large I/O to maximize the sequential bandwidth of each external storage device. Aiming at the problems of limited parallel I/O capability and various locks contention overhead for multiple external storage devices, we proposed an I/O management strategy based on multi-external-storage multi-queue. Then we designed and implemented CSMqGraph, a graph processing system based on coarse-grained striping method matching sequential large I/O and multi-external-storage multi-queue I/O management strategy. The architecture is shown in Fig. 3.

CSMqGraph does preprocessing in the following way: First, it partitions the original unordered edge list using the 2D grid partitioning method. This 2D grid partitioning provides not only flexibility but also efficiency since the higher level partitioning is virtual and CSMqGraph is able to utilize the outcome of lower level partitioning thus no more actual overhead is added. Second, it strips the partitioned files into multiple external storage devices according to the predefined coarse-grained stripe depth.

CSMqGraph does computation in the following way: First, the main thread pushes tasks to the I/O task queue, containing the file, offset, and length to issue each I/O request. Second, I/O threads fetch tasks from the queue until empty, read data to the I/O buffer and write data from a specified location. Third, worker threads fetch data from the I/O buffer and process each edge.

3.1 Coarse-Grained Striping Method Matching Sequential Large I/O Characteristics

In order to saturate the external storage bandwidth as much as possible, the graph processing systems adopt multiple I/O threads for concurrent access and maintain a large I/O buffer for each I/O thread (24 MB in the GridGraph, 16 MB in X-Stream) to access edge blocks. When the vertex partition size P satisfies the locality requirement,

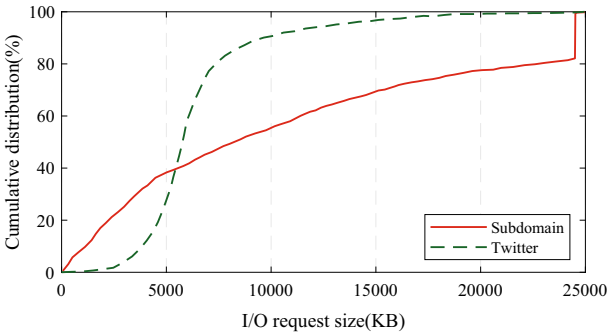

Fig. 4 I/O request size distribution

most of the edge blocks are large. The I/O requests also have the characteristics of large and don't exceed the I/O buffer size.

Through experiments, we further verify that the external storage graph processing systems, which execute graph algorithm based on the 2D partitioning and sequential external storage access optimization principle have the characteristics of large I/O. Figure 4 shows the distribution of I/O requests size when GridGraph performs a round of iteration of the graph algorithm. The Twitter graph uses 36*36 partitions and the Subdomain graph uses 70*70 partitions. It can be seen that in the Twitter graph, 92% of I/O requests are larger than 1 MB, and the average size is 11MB. 99% of I/O requests in the Subdomain graph are larger than 1MB, and the average size is 7 MB.

Studies have shown that stripe depth affects the system I/O performance. The choice of stripe depth is closely related to the application I/O characteristics [15]. The stripe depth of the operating system based on the striped volume is usually 2–512 KB. This stripe depth is very fine-grained relative to the I/O request of the external storage graph processing system. It can't fully exploit the bandwidth of the external storage device. However, the current graph processing system relies on the operating system I/O management method based on the striped volume to use multiple external storage devices. Instead of selecting the appropriate stripe depth based on the graph data access characteristics and the number of I/O threads, it directly uses the Linux operating system default 512 KB stripe depth. By analyzing the data access characteristics of the graph processing system, we propose a coarse-grained striping method that matches the large I/O features to perform the sequential distribution of graph data across multiple external storage devices.

Choosing the size of the strip depth is the key. If the stripe depth is smaller than the I/O request size, it will result in an I/O request spanning two or more external storage devices. It is hard for external storage devices to achieve full sequential bandwidth since more time will be spent on seeking to potentially different positions. And multiple block positioning requires multiple logical address to physical address mapping. The overhead generated by multiple address mappings is not negligible. If the stripe depth is larger than or equal to the I/O request size, the I/O request will span no more than two external storage devices. When the stripe depth is N times the size of the I/O request, if each I/O request start offset is aligned with the stripe unit x/N (x = 0, 1, ...

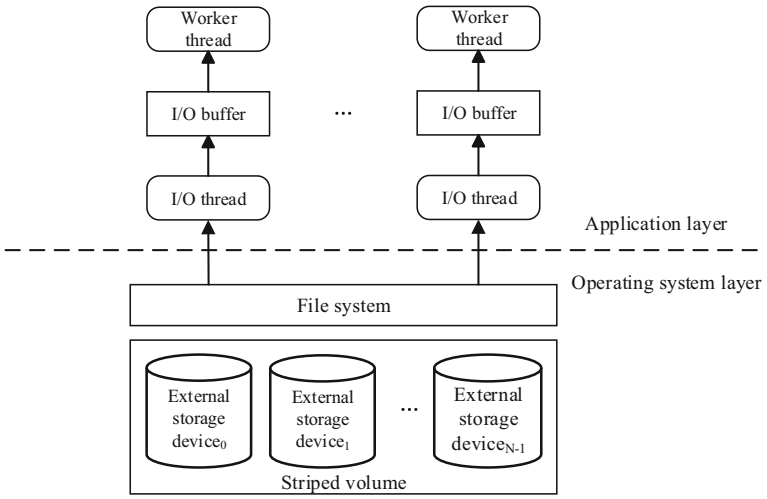

Fig. 5 I/O model of operating system based on striped volume

N-1), which ensures that each I/O request served by only one external storage device. But when the I/O concurrency is high, if I/O requests data is only distributed on one external storage device, concurrent I/O requests result in centralized access to the hot device. And if the stripe depth is too large, it may cause the imbalance distribution of graph data among multiple external storage devices and increase the computing time that cannot overlap with I/O.

Therefore, we make a compromise approach to choose the average I/O request size $\sim$ maximum I/O request size as the stripe depth. This approach tries to avoid I/O requests spanning three or more external storage devices to reduce data block positioning overhead. It also reduces the I/Os of each external storage device. It relieves the potential problems of the imbalanced distribution of data and hot external storage devices centralized access caused by the excessive stripe depth. For I/O requests still across two or more external storage devices, we perform I/O request decomposition and prefetch merging in the application layer to realize the dynamic adjustment of the I/O request size and align stripe unit boundaries. It ensures that each I/O thread has launched an I/O request only on one external storage device.

3.2 I/O Management Strategy Based on Multi-External-Storage Multi-Queue

As shown in Fig. 5, the external storage graph processing systems use the operating system I/O management method based on striped volume to expand the I/O performance. In this I/O model, I/O threads transparently initiate I/O requests to the striped volume composed of multiple external storage devices. It is unavoidable that the data of one I/O request initiated by one I/O thread is across multiple external storage devices and the data of I/O requests initiated by multiple I/O threads may be on the same external storage device, resulting in contention overhead for external storage devices and file locks. In the context of an I/O thread, the I/O service tasks are performed in

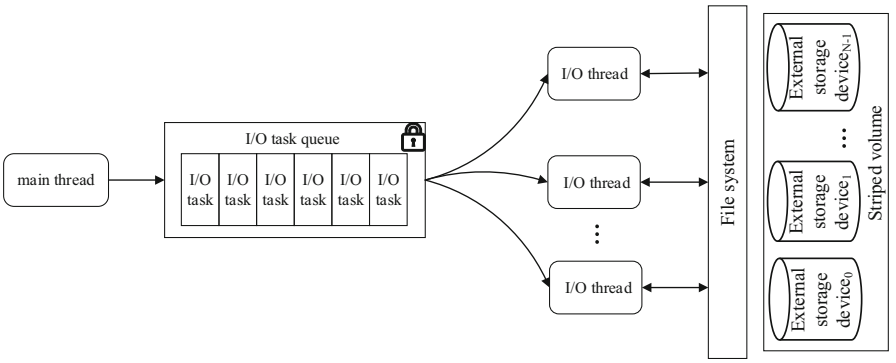

Fig. 6 Multi-external-storage single-queue thread pool model

a sequential polling manner, which limits the parallel capability of multiple external storage devices. After the I/O request decomposition by the volume manager of the operating system, the pluglist corresponding to an I/O thread in the block layer contains small I/O requests to different external storage devices. It is not efficient for the operating system to perform sorting and merging in such a pluglist. When the stripe depth is fine-grained, concurrent I/O threads cause sequential small I/O requests on each external device to be unpredictable out of order arrival, causing merge invalid. In some graph algorithms, only part of the data is active data. The merge is more inefficient, which increases the number of device I/Os.

Based on the operating system I/O management method of the striped volume, the external storage graph processing systems use the thread pool model of the single I/O task queue to initiate I/O requests concurrently in order to maximize the saturation bandwidth of the multiple external storage devices. As shown in Fig. 6, the main thread doesn't apply any I/O management strategies to multiple external storage devices. The main thread simply pushes the original I/O requests to the shared single I/O task queue, and then the multiple I/O threads fetch the tasks from the shared queue and initiate the corresponding I/O to the striped volume. The I/O requests are completely dependent on the operating system for management, such as I/O request decomposition, merging, and dispatching. It is not efficient for I/O sorting and merging. This thread pool model causes various lock conflicts in the operating system. For example, multiple I/O threads in the application layer compete for locks on a single shared I/O task queue.

To solve the above problems, we propose an I/O management strategy based on multi-external-storage multi-queue. The multi-external-storage multi-queue thread pool model is implemented, as shown in Fig. 7. It maintains a dedicated I/O task queue for each external storage device and performs I/O request decomposition, prefetch merging, and dispatching at the application layer. The decomposed new I/O request that doesn't span multiple devices is added to the corresponding I/O task queue. And bind the I/O threads to the I/O task queue, so that each I/O thread is dedicated to a specific external storage device. Figure 8 shows the I/O model based on multi-external-storage multi-queue. This I/O model avoids the case where an I/O request initiated by an I/O thread is served by multiple external storage devices. When multiple I/O threads

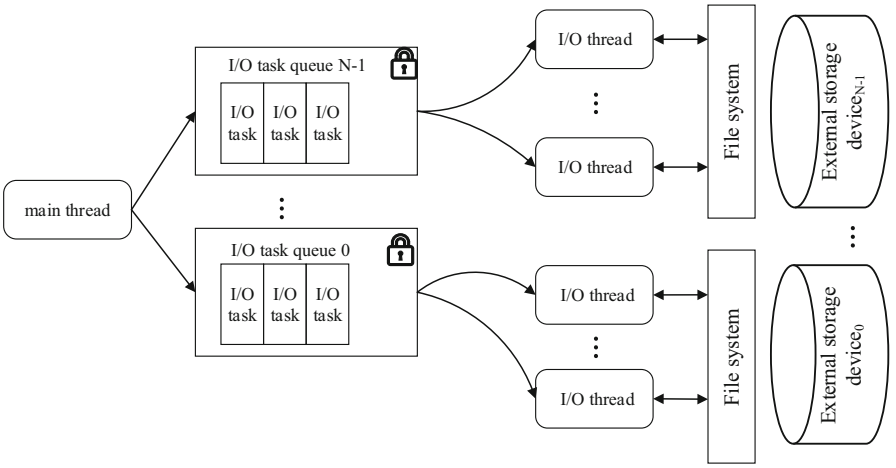

Fig. 7 Multi-external-storage multi-queue thread pool model

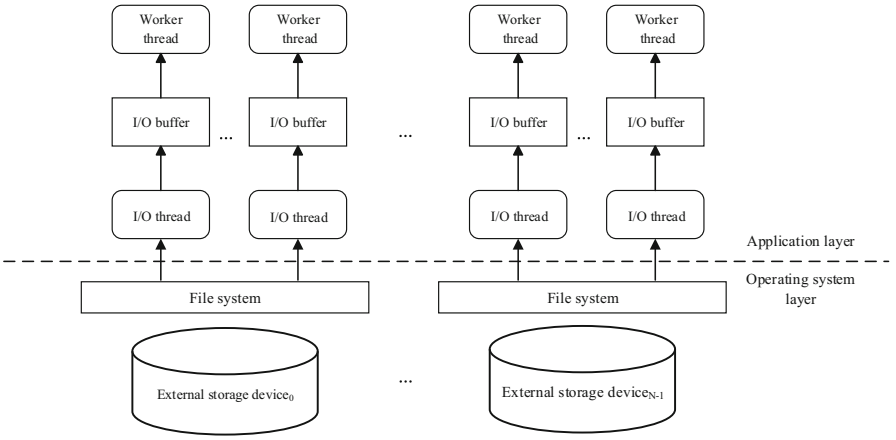

Fig. 8 I/O model based on multi-external-storage multi-queue

are concurrently access, multiple external storage devices can truly play their respective I/O performance in parallel. This I/O model not only fully exploits the parallelism of multiple external storage devices but also improves the I/O sorting and merging efficiency at the operating system level. It also reduces the contention overhead of I/O task queue locks in the application layer. By binding multiple I/O threads to each I/O task queue, we can fully exploit the maximum sequential bandwidth of each external storage device. Each I/O task queue is bound to the same number of I/O threads for I/O load balancing.

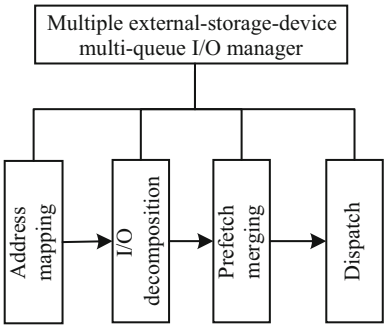

Fig. 9 I/O Manager module

3.3 Implementation

The I/O manager implements I/O management, such as address mapping, decomposition, prefetch merging and dispatch of I/O requests at the application layer, as shown in Fig. 9.

For the convenience of description, we assume that all edge block files are grouped into a large graph data file in the update order. The starting linear offset address of each edge block file in the merged graph data file is recorded. Striping maps a large linear address space to the N address spaces which is corresponding to the striped files. Define the following symbols:

S is the stripe depth, which equals to the size of stripe unit.

N is the number of striped files, which equals to the number of external storage devices.

D_l is the data length of an original I/O request.

O is the starting offset of the original I/O request in the original large linear address space.

SI_i is the starting striped file number of the ith new I/O request after the original I/O request address mapping and decomposition

SO_i is the starting offset in the striped file of the ith new I/O request after the original I/O request address mapping and decomposition.

Among $i = 0, 1\ldots$

Address mapping formula:

$$SI_i = \lfloor O/S \rfloor \% N \tag{1}$$

$$SO_i = \lfloor O/S/N \rfloor * N + O\%S \tag{2}$$

Address mapping stage The main thread is used to map the original I/O requests according to the address mapping formula. It obtains the starting striped file number of the new I/O request and the starting offset in the starting striped file.

I/O decomposition stage The main thread determines whether to decompose the original I/O requests according to the starting offset address, the I/O request data length, and the stripe unit boundary in the starting striped file.

$D_l \leq S - O\%S$. The original I/O request length doesn't exceed the remaining length of the stripe unit in the first stripe file. In this case, the original I/O request

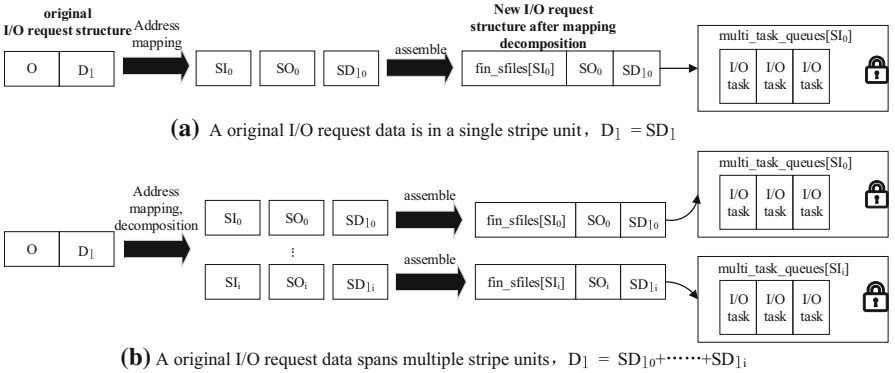

Fig. 10 An original I/O request address mapping and decomposition process

doesn't need to be decomposed. The striped file number of the corresponding new I/O request is SI_i. The starting offset address in the striped file is SO_i and the length is D_l.

$D_L > S - O\%S$. The original I/O request length exceeds the remaining length of the stripe unit in the first stripe file, spanning multiple external storage devices. In this case, the original I/O request needs to be decomposed multiple new I/O requests. An original I/O request address mapping and decomposition process is shown in Fig. 10.

Prefetch merging stage The main thread maps and decomposes the original I/O requests so that each new I/O request is located in a stripe unit of an external storage device. However, the boundary of each edge block after partition cannot be guaranteed to be aligned with the boundary of the stripe unit. Therefore, there must be a situation where a stripe unit contains two or more small new I/O requests data. Therefore, when processing an original I/O request that spans multiple stripe units or falls in the stripe unit but doesn't exceed the right boundary of the stripe unit, we consider mapping next I/O requests that are left-aligned with the original I/O request. We preprocess address mapping and decomposition of the I/O requests that are corresponding to the adjacent active edge blocks. The current I/O request can be merged with some new I/O requests that fall into the same stripe unit into a large, continuous new I/O request. The upper bound of the merging is MAX_IOSIZE. (MAX_IOSIZE is the size of the I/O buffer, which is the maximum size of the original I/O request.) Figure 11 shows the possible distribution of new I/O requests after address mapping and decomposition.
Dispatch stages The merged new I/O request is dispatched to the corresponding I/O task queue. Then I/O threads fetch I/O tasks from the I/O task queue.

4 Experimental Evaluation

We evaluate CSMqGraph on several real world social graphs and web graphs. And CSMqGraph shows significant performance improvement compared with current out-of-core graph engines. The hardware platform used in our experiments is a server containing 2-way 8-core 2.10 GHz Intel Xeon CPU E5-2670 and each CPU has 20 MB

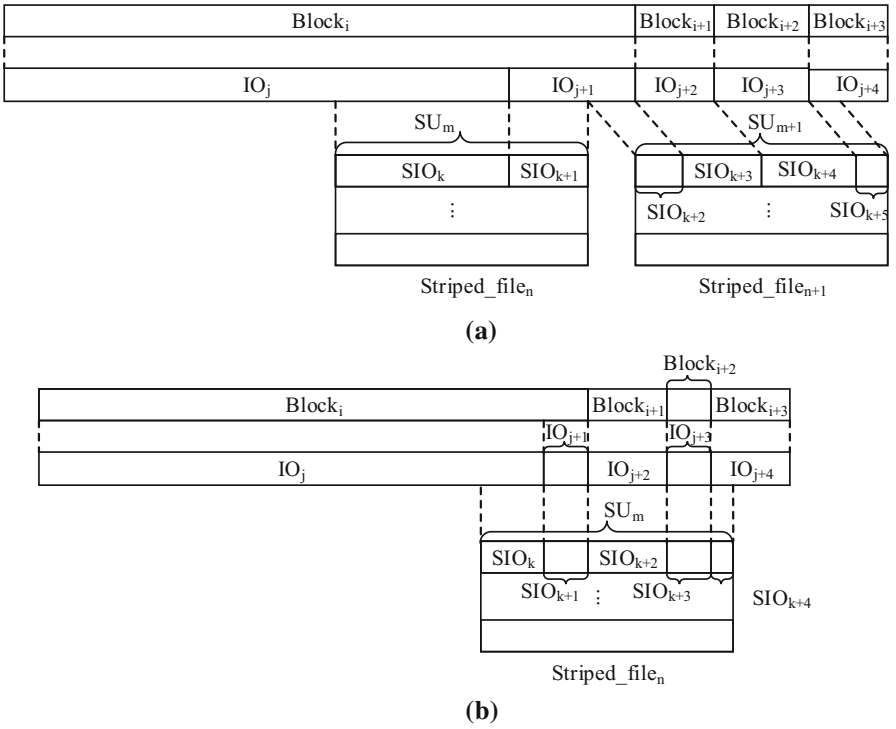

Fig. 11 Distribution of new I/O request after address mapping and decomposition

last-level cache, running a Linux operation system with kernel version 4.13.8-1. Its memory is 125 GB. We limit the available memory to 8 GB to illustrate the out-of-core performance. It has 6 899.6 GB hard disk drives and the sequential bandwidth of each HDD is approximate 210 MB/s.

It spawns a worker for each core to run benchmarks. In experiments, four popular graph algorithms from web applications and data mining are employed as benchmarks: (1) weakly connected component(WCC); (2) PageRank; (3) breadth first search(BFS); (4) sparse matrix Vector multiplication(SpMV). For the WCC and BFS algorithms, we run them until they converge. For PageRank, we specify run 10 iterations and SpMV runs only one iteration to compute the product. The dataset used for these graph algorithms are described in Table 1.The performance of CSMqGraph is compared with GridGraph. CSMqGraph is open-sourced for public access. The detailed configurations are currently available at https://github.com/shuochenok/CSMqGraph.

4.1 Overall Performance Comparison

To verify the effectiveness of the coarse-grained striping method that matches the characteristics of sequential large I/O and the I/O management strategy based on multi-external-storage multi-queue, we compare CSMqGraph against GridGraph when running various algorithms. Both CSMqGraph and GridGraph take the same graph

Table 1 Graph datasets used in evaluation

Dataset	Vertexes	Edges	Directed	Type
Twitter	61.6 million	1.47 billion	Directed	Social network
Friendster	124.8 million	1.8 billion	Undirected	Social network
Subdomain	101.7 million	2.0 billion	Directed	Web graph

partitioning method, so the preprocessing time of CSMqGraph does not increase compared to GridGraph. Both CSMqGraph and GridGraph systems use the same parameter configuration. The memory budget parameter is 8GB and the number of threads is 3*N. N is the number of external storage devices. CSMqGraph strips multiple graph data files to multiple external storage devices with the stripe depth of 12 MB. Figure 12 shows the speedup of CSMqGraph over CridGraph for four algorithms. CSMqGraph performs better than GridGraph under different numbers of external storage devices. For different datasets and different numbers of external storage devices, the speedup of WCC is by up to 1.4. For BFS, the speedup is by up to 1.28. For PageRank, the speedup is by up to 1.24. For SpMV, the speedup is by up to 1.25. We identify that the improvement of the execution time of CSMqGraph mainly due to maximizing the sequential bandwidth of each external storage device and fully exploiting the parallelism of multiple external storage devices.

4.2 Comparison of Device I/Os

We evaluated the average number of device I/Os per external storage device when the CSMqGraph and GridGraph execute the algorithm over multiple external storage devices. As shown in Fig. 13, the average number of I/Os to each external storage device in CSMqGraph is significantly lower than GridGraph when the WCC algorithm is executed on Twitter with 2 external storage devices. Overall, the total number of each external storage device I/Os in GridGraph during the entire WCC algorithm execution is 577 K, while the total number of each external storage device I/Os in CSMqGraph is 419 K. When both graph processing systems access the same amount of graph data, the total number of I/Os in CSMqGraph is significantly reduced and I/O performance is significantly improved.

4.3 I/O Throughput Comparison

We evaluated the I/O throughput when CSMqGraph and GridGraph run the graph algorithm with different numbers of external storage devices. Figure 14 depicts the average real-time I/O throughput of each external storage device when they run WCC algorithm on Twitter. We observe that during each iteration of the WCC algorithm execution, the average I/O throughput of each external storage device in CSMqGraph is significantly higher than that in GridGraph. For CSMqGraph, the average throughput of each external storage device is 193.5 MB/s. For GridGraph, that is only 143.8 MB/s. Figure 15 shows the average I/O throughput and promotion ratio of each external

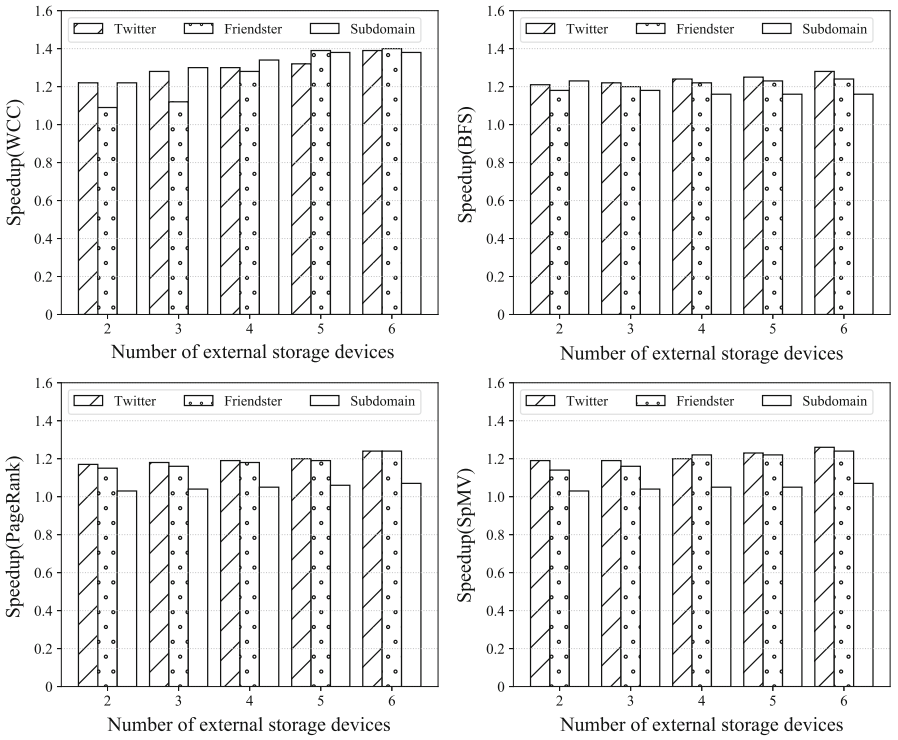

Fig. 12 The speedup of different algorithms on different graph datasets with different numbers of external storage devices

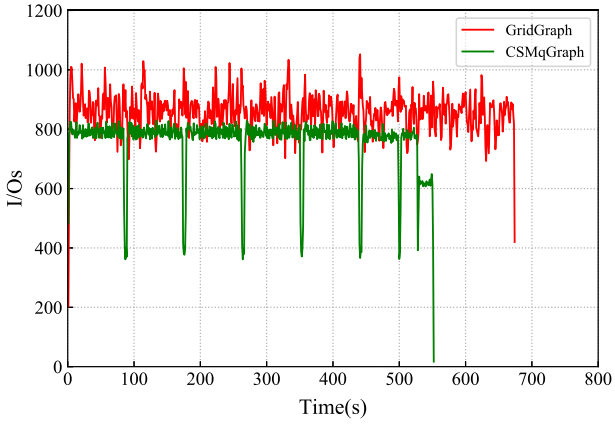

Fig. 13 Average I/Os of each external storage device

storage device when CSMqGraph and GridGraph run WCC graph algorithm on Twitter with different numbers of external storage devices. We observe that the average I/O throughput of each external storage device in CSMqGraph is better than GridGraph.

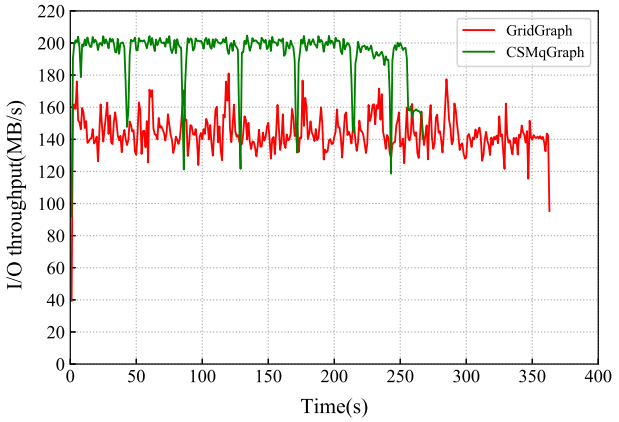

Fig. 14 Average I/O throughput of each external storage devices

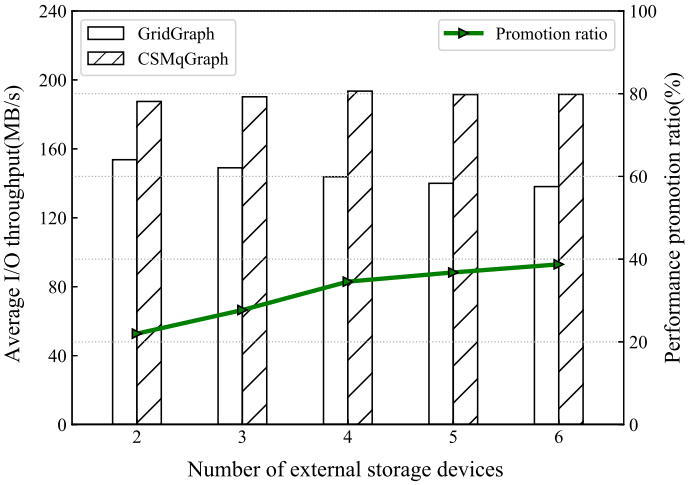

Fig. 15 Average I/O throughput and promotion ratio of WCC on Twitter graph with different numbers of external storage devices

The average I/O throughput of each external storage device in CSMqGraph is stable at around 190.9 MB/s and the I/O throughput increased by 22.0% to 38.7%. As the number of external storage devices increases, the proportion of promotion increases gradually.

4.4 Scalability of CSMqGraph

We evaluated the scalability of CSMqGraph by executing WCC algorithm and PageRank on Friendster with different numbers of external storage devices. The WCC algorithm is that some vertices are active vertices, and the PageRank algorithm is that all vertices are active vertices. Figure 16 gives the results relative to the execution

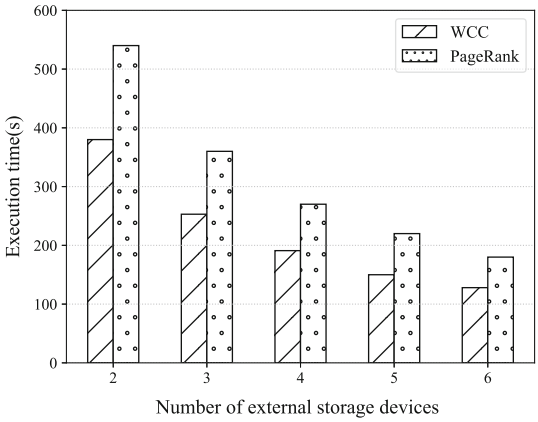

Fig. 16 CSMqGraph I / O scalability

time of WCC and PageRank algorithm in CSMqGraph and shows an almost linear decrease with the increase of external storage devices. The results indicate that CSMqGraph can make full use of the increasing bandwidth resources of multiple external storage devices and has better I/O scalability.

4.5 Influence of Stripe Depth

When the application layer strips 2D graph partition files to multiple external storage devices, the stripe depth will affect the system I/O performance. We evaluated the effect of stripe depth on the performance of CSMqGraph. Under different stripe depths, CSMqGraph executes PageRank algorithm on Twitter with 2 external storage devices and 6 I/O threads. As shown in Fig. 17, we can observe that when the strip depth is 12 MB, the best performance is achieved. The average size of the original I/O requests for Twitter is 11 MB, and the optimal stripe depth is 12 MB. The optimal stripe depth is between the average of the original I/O request size and the maximum of the original I/O request size. Because if the stripe depth is too small, it is hard for external storage devices to achieve full sequential bandwidth since more time will be spent on seeking to potentially different positions. And address mapping and decomposition are not efficient enough, which leads to more data access overhead and system call overhead. If the stripe depth is too large, it causes the problems, such as centralized access to the hot external storage device, unbalanced load, increased computation time that cannot overlap with I/O. In both cases, the I/O performance of multiple external storage devices cannot be fully utilized.

4.6 Impact of I/O Threads

We evaluated the impact of the number of I/O threads on the performance of CSMqGraph. Figure 18 shows the execution time when CSMqGraph runs the PageRank algorithm on Twitter as the number of I/O threads increases(2 external storage devices). It can be seen from the Fig. 18, when the number of I/O threads is small, the bandwidth

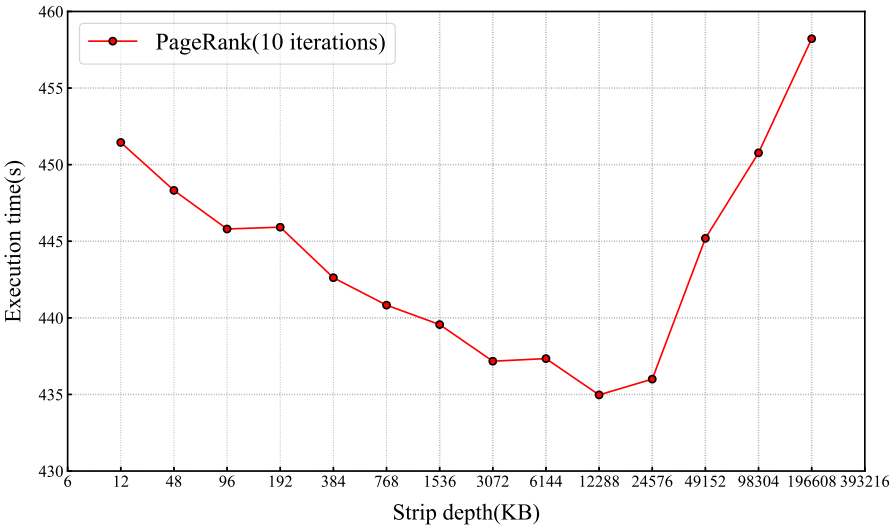

Fig. 17 Execution time of PageRank on Twitter at different stripe depths

of multiple external storage devices cannot be saturated, thus CPU is the bottleneck of performance. As the number of I/O threads increases, the execution time of CSMq-Graph is significantly reduced. When the number of I/O threads increases to 6, the execution time doesn't continue to decrease as the number of I/O threads increases. The performance is due to the shift of the bottleneck from CPU to external storage devices. Therefore, for different numbers of external storage devices, the total number of I/O threads is set to the product of the number of external storage devices and the number of I/O threads per external storage device when the saturation point is reached.

5 Related Work

With the explosion of graph scale, many systems [12,18,23] have focused on achieving high efficiency for iterative graph analysis. They improve the efficiency either by fully utilizing the sequential bandwidth of external storage devices or by achieving a better data locality to reduce the redundant data accesses.

GraphChi [5] eliminates random data access from disks by scanning the entire graph data in each iteration. By using a novel parallel sliding windows method to reduce random I/O accesses, GraphChi [6] is able to process large-scale graphs in reasonable time. GraphChi also supports selective scheduling, and its shard representation can have even better effect than GridGraph on I/O reduction. Though I/O amount required by GraphChi is rather small, it has to issue many fragmented reads across shards. Thus the performance is not ideal enough due to limited bandwidth usage.

X-Stream [21] introduces an edge-centric scatter-gather processing model. Accesses to vertices are random and happen on a high level of storage hierarchy which is small but fast. And accesses to edges and updates fall into a low level of storage hierarchy

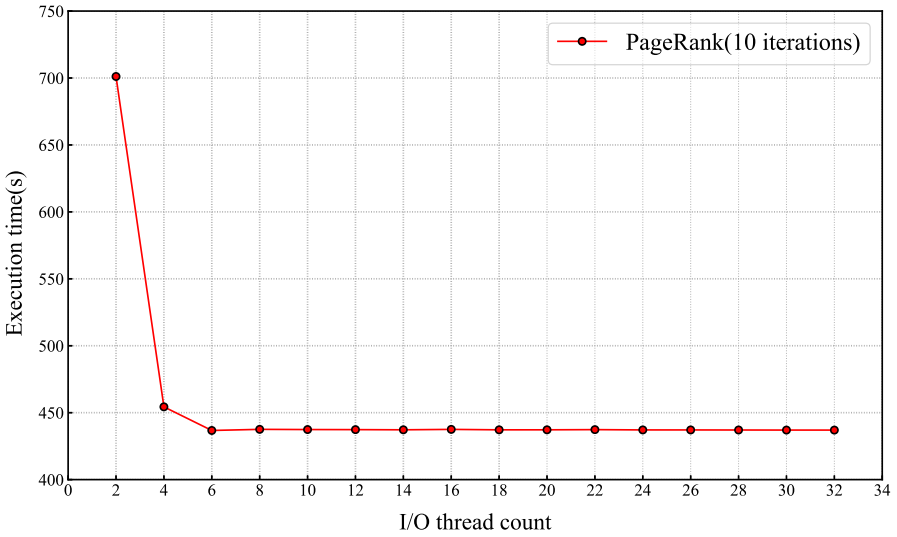

Fig. 18 Execution time of PageRank on Twitter with different numbers of I/O threads

which is large but slow. However, these accesses are sequential so that maximum throughput can be achieved. Although X-Stream can leverage high disk bandwidth by sequential accessing, it needs to generate updates which could be in the same magnitude as edges, and its lack of support on selective scheduling could also be a critical problem when dealing with graphs of large diameters.

GridGraph [22] proposes 2-Level hierarchical partitioning scheme to improve the locality and reduce the amount of I/Os. It realizes the sequential access of the external storage graph through the double sliding window. However, GridGraph uses the operating system I/O management method based on striped volume and adopts multi-external-storage single-queue thread pool model to use multiple external storage devices. It limits the sequential bandwidth of each external storage devices and the parallelism of multiple external storage devices.

FlashGraph supports both pulling data from SSDs and pushing data with message passing. FlashGraph does provide asynchronous execution of vertex programs to overlap computing with data access. But it needs user input to sort, merge, submit and poll I/O requests, which is insufficient to the sequential bandwidth of each external storage device. In addition, the semi-external mode naturally lacks the processing power of a large graph that cannot be fully loaded into memory.

Graphene, a semi-external memory processing system that efficiently reads the graph data on SSDs while managing the metadata in DRAM. Graphene incorporates graph data awareness in I/O management behind an I/O centric programming model and performs fine-grained I/Os on flash-based storage devices. It uses a Bitmap-based approach to quickly reorder, deduplicate, and merge the requests and exploits asynchronous I/O to submit as many I/O requests as possible to saturate the I/O bandwidth of flash devices. It relies on expensive SSD arrays and large memory to provide high

I/O bandwidth and cache all vertex data. While most of the out-of-core systems are HDD-friendly and aim to achieve reasonable performance with low hardware costs.

These systems are primarily dedicated to optimize I/O performance. There are two main optimization principles: sequential I/O [14,21,28] and on-demand I/O [16,24]. Sequential I/O inevitably leads to unnecessary I/O, resulting in the external I/O is inefficient. Taking on-demand I/O to ensure I/O efficiency. It inevitably leads to a large amount of random I/O, which is not conducive to the sequential bandwidth of external storage devices. Therefore, CSMqGraph proposes coarse-grained striping method matching sequential large I/O to maximize sequential bandwidth of each external storage device and an I/O management strategy based on multi-external-storage multi-queue making I/O threads dedicated to each external storage device to further improve I/O throughput and fully exploit the parallelism of multiple external storage devices. CSMqGraph shows significant performance improvement compared with state-of-the-art out-of-core graph engines.

6 Conclusion

In this paper, we have designed and developed CSMqGraph that consists of two techniques including a coarse-grained striping method matching sequential large I/O and an I/O management strategy based on multi-external-storage multi-queue. It improves the I/O throughput of each external storage device and fully exploit the parallelism of multiple external storage devices. The experiments show that CSMqGraph achieves significantly better performance than state-of-the-art out-of-core graph systems. The performance of CSMqGraph is mainly restricted by I/O bandwidth. In the future, we plan to further optimize our research for evolving graph analysis and also extend it to the distributed platform.

Acknowledgements This work is supported by NSFC No. 61772216, 61821003, U1705261, Wuhan Application Basic Research Project No. 2017010201010103, Fund from Science, Technology and Innovation Commission of Shenzhen Municipality No. JCYJ20170307172248636, and Fundamental Research Funds for the Central Universities.

References

1. Chi, Y., Dai, G., Wang, Y., Sun, G., Li, G., Yang, H.: Nxgraph: an efficient graph processing system on a single machine. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 409–420. IEEE (2016)
2. Coffman, T., Greenblatt, S., Marcus, S.: Graph-based technologies for intelligence analysis. Commun. ACM **47**(3), 45–47 (2004)
3. Del Sol, A., Fujihashi, H., O'Meara, P.: Topology of small-world networks of protein-protein complex structures. Bioinformatics **21**(8), 1311–1315 (2005)
4. Doerr, C., Blenn, N.: Metric convergence in social network sampling. In: Proceedings of the 5th ACM Workshop on HotPlanet, pp. 45–50. ACM (2013)
5. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: Powergraph: distributed graph-parallel computation on natural graphs. In: Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12), pp. 17–30 (2012)

6. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: Graphx: graph processing in a distributed dataflow framework. In: 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14), pp. 599–613 (2014)
7. Huberman, B.A., Adamic, L.A.: Internet: growth dynamics of the world-wide web. Nature **401**(6749), 131 (1999)
8. Jeong, H., Mason, S.P., Barabási, A.L., Oltvai, Z.N.: Lethality and centrality in protein networks. Nature **411**(6833), 41 (2001)
9. Jeong, H., Tombor, B., Albert, R., Oltvai, Z.N., Barabási, A.L.: The large-scale organization of metabolic networks. Nature **407**(6804), 651 (2000)
10. Kang, U., Tsourakakis, C.E., Faloutsos, C.: Pegasus: a peta-scale graph mining system implementation and observations. In: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, pp. 229–238. Washington, DC, USA (2009)
11. Khayyat, Z., Awara, K., Alonazi, A., Jamjoom, H., Williams, D., Kalnis, P.: Mizan: a system for dynamic load balancing in large-scale graph processing. In: Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys '13, pp. 169–182. ACM, New York, NY, USA (2013). https://doi.org/10.1145/2465351.2465369
12. Kumar, P., Huang, H.H.: G-store: high-performance graph store for trillion-edge processing. In: SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 830–841. IEEE (2016)
13. Kwak, H., Lee, C., Park, H., Moon, S.: What is twitter, a social network or a news media? In: Proceedings of the 19th International Conference on World Wide Web, pp. 591–600. ACM (2010)
14. Kyrola, A., Blelloch, G., Guestrin, C.: Graphchi: large-scale graph computation on just a {PC}. In: Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12), pp. 31–46 (2012)
15. Lee, E.K., Katz, R.H.: An analytic performance model of disk arrays. In: ACM SIGMETRICS Performance Evaluation Review, vol. 21, pp. 98–109. ACM (1993)
16. Liu, H., Huang, H.H.: Graphene: fine-grained IO management for graph computing. In: 15th USENIX Conference on File and Storage Technologies (FAST 17), pp. 285–300. USENIX Association, Santa Clara, CA (2017). https://www.usenix.org/conference/fast17/technical-sessions/presentation/liu
17. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 135–146. ACM (2010)
18. Nguyen, D., Lenharth, A., Pingali, K.: A lightweight infrastructure for graph analytics. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP '13, pp. 456–471. ACM, New York, NY, USA (2013). https://doi.org/10.1145/2517349.2522739
19. Randles, M., Lamb, D., Taleb-Bendiab, A.: A comparative study into distributed load balancing algorithms for cloud computing. In: 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops, pp. 551–556. IEEE (2010)
20. Roy, A., Bindschaedler, L., Malicevic, J., Zwaenepoel, W.: Chaos: scale-out graph processing from secondary storage. In: Proceedings of the 25th Symposium on Operating Systems Principles, pp. 410–424. ACM (2015)
21. Roy, A., Mihailovic, I., Zwaenepoel, W.: X-stream: edge-centric graph processing using streaming partitions. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pp. 472–488. ACM (2013)
22. Shiloach, Y., Vishkin, U.: An o(log n) parallel connectivity algorithm. J. Algorithms **3**, 57–67 (1982)
23. Shun, J., Blelloch, G.E.: Ligra: a lightweight graph processing framework for shared memory. In: ACM Sigplan Notices, vol. 48, pp. 135–146. ACM (2013)
24. Vora, K., Xu, G., Gupta, R.: Load the edges you need: a generic i/o optimization for disk-based graph processing. In: 2016 {USENIX} Annual Technical Conference ({USENIX} {ATC} 16), pp. 507–522 (2016)
25. Wang, P., Zhang, K., Chen, R., Chen, H., Guan, H.: Replication-based fault-tolerance for large-scale graph processing. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 562–573. IEEE (2014)
26. Wang, Z., Gu, Y., Bao, Y., Yu, G., Yu, J.X.: Hybrid pulling/pushing for i/o-efficient distributed and iterative graph computing. In: Proceedings of the 2016 International Conference on Management of Data, pp. 479–494. ACM (2016)

27. Zhao, Y., Yoshigoe, K., Xie, M., Zhou, S., Seker, R., Bian, J.: Lightgraph: lighten communication in distributed graph-parallel processing. In: 2014 IEEE International Congress on Big Data, pp. 717–724. IEEE (2014)
28. Zheng, D., Burns, R., Szalay, A.S.: Toward millions of file system iops on low-cost, commodity hardware. In: SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1–12. IEEE (2013)
29. Zheng, D., Mhembere, D., Burns, R., Vogelstein, J., Priebe, C.E., Szalay, A.S.: Flashgraph: processing billion-node graphs on an array of commodity ssds. In: 13th {USENIX} Conference on File and Storage Technologies ({FAST} 15), pp. 45–58 (2015)
30. Zhu, X., Han, W., Chen, W.: Gridgraph: large-scale graph processing on a single machine using 2-level hierarchical partitioning. In: 2015 {USENIX} Annual Technical Conference ({USENIX} {ATC} 15), pp. 375–386 (2015)

Affiliations

Shuo Chen[1] · Zhan Shi[1] · Dan Feng[1] · Shang Liu[1] · Fang Wang[1] · Lei Yang[1] · Ruili Yu[1]

✉ Zhan Shi
zshi@hust.edu.cn

Shuo Chen
shuochen@hust.edu.cn

Dan Feng
dfeng@hust.edu.cn

Shang Liu
1183358546@qq.com

Fang Wang
wangfang@hust.edu.cn

Lei Yang
1322337157@qq.com

Ruili Yu
2294236515@qq.com

[1] Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China