PAGE: A Partition Aware Graph Computation Engine

Yingxia Shao, Junjie Yao, Bin Cui, Lin Ma
Department of Computer Science
Key Lab of High Confidence Software Technologies (Ministry of Education)
Peking University
{simon0227, junjie.yao, bin.cui, malin1993ml}@pku.edu.cn

ABSTRACT

Graph partitioning is one of the key components in parallel graph computation, and the partition quality significantly affects the overall computing performance. In the existing graph computing systems, "good" partition schemes are preferred as they have smaller edge cut ratio and hence reduce the communication cost among working nodes. However, in an empirical study on Giraph[1], we found that the performance over well partitioned graph might be even two times worse than simple partitions. The cause is that the local message processing cost in graph computing systems may surpass the communication cost in several cases.

In this paper, we analyse the cost of parallel graph computing systems as well as the relationship between the cost and underlying graph partitioning. Based on these observation, we propose a novel Partition Aware Graph computation Engine named PAGE. PAGE is equipped with two newly designed modules, i.e., the communication module with a dual concurrent message processor, and a partition aware one to monitor the system's status. The monitored information can be utilized to dynamically adjust the concurrency of dual concurrent message processor with a novel Dynamic Concurrency Control Model (DCCM). The DCCM applies several heuristic rules to determine the optimal concurrency for the message processor.

We have implemented a prototype of PAGE and conducted extensive studies on a moderate size of cluster. The experimental results clearly demonstrate the PAGE's robustness under different graph partition qualities and show its advantages over existing systems with up to 59% improvement.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.1.3 [**Programming Techniques**]: Concurrent Programming—*Parallel Programming*

Keywords

Graph Computing; Graph Partition; Message Processing

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2505617.

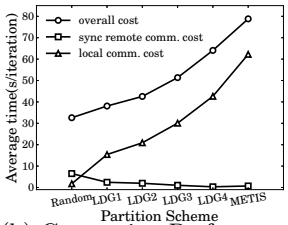
1. INTRODUCTION

With the stunning growth of graph data and related applications, parallel processing becomes the de facto graph computing paradigm for large scale graph analysis tasks. A lot of parallel graph computation systems are introduced, e.g. Pregel, Giraph, GPS, GraphLab and PowerGraph [2, 1, 3, 4, 5]. In these parallel systems, graph partitioning is one of the key components that affect the computing performance. It usually splits an input graph into several balanced subgraphs,and then each subgraph is processed by an individual worker in parallel. The general partition objective is to let these subgraphs have minimum cross edges between different ones, thus diminishing the communication cost among parallel workers [6, 7].

However, integrating a suitable graph partitioning method into parallel graph computation systems is not a trivial task. In most of the current systems, a good balanced graph partition even leads to a decrease in the overall computing performance. Figure 1 displays a PageRank experiment result on six different partition schemes of a large web graph dataset. We apparently notice that the overall cost of PageRank per iteration increases with the quality improving across different graph partitioning schemes. It indicates that current parallel graph systems cannot benefit from high quality graph partitioning.

Scheme	Edge Cut
Random	98.52%
LDG1	82.88%
LDG2	75.69%
LDG3	66.37%
LDG4	56.34%
METIS	3.48%

(a) Partition Quality



(b) Computing Performance

Figure 1: Web Graph[1]Partition

Figure 1 also reveals that with the decrease of edge cut, the *sync remote communication cost*[2] is reduced as expected. However, the *local communication cost* increases fast unexpectedly, and directly leads to the downgrade of overall performance. The processing of local messages becomes

[1]uk-2007-05-u, http://law.di.unimi.it/datasets.php,please refer to the detailed experiment setup in Section 4
[2]refer to Figure 5(a) for the illustration of this cost

a bottleneck in the system and dominates the *overall cost* when the workload of local message processing increases.

Lots of current parallel graph system design does not take the underlying partitioned subgraphs into consideration, and can not detect the increasing workload of local message processing. Though there are some recent tentative work by introducing the centralized message buffer to process local and remote incoming messages[3], most of current graph systems can not effectively utilize the benefit of high quality graph partitioning.

In this paper, we present a novel graph computation engine, i.e. Partition Aware Graph computation Engine(PAGE). It is designed to support computation tasks with different partitioning qualities. There are some unique features in its new framework. First, in PAGE's worker unit, communication module is extended with a new dual concurrent message processor. The concurrent message processor allows PAGE to concurrently process both local and remote incoming messages and thus speed up message processing. Second, a partition aware module is added in worker unit to monitor the characters of partitioning and adjust the resources adaptively.

To fulfill the goal of efficient concurrency control, we introduce a dynamic model(Dynamic Concurrency Control Model) to better capture the information. Based on the online metrics provided by the monitor, DCCM sets near optimal parameters for PAGE's message processor. Several heuristic rules are also derived from our empirical study, which we will present the details later in this paper. PAGE has sufficient message process units to handle the current workload and each message process unit has balanced workload.

A prototype of PAGE has been set up on top of Giraph(version 0.2.0). Extensive experiments demonstrate its superb performance and effectiveness. The results show that PAGE provides up to 59% improvement over Giraph, and it can also achieve 14% enhancement when we use a state of the art graph partitioning method METIS instead of random partitioning.

The main contributions of our work can be summarized as follows:

- First, we identify the problem of partition unaware inefficiency in current state of the art graph computation systems. Most of them can not obtain the benefits of graph partitioning, which severely affects the performance of many graph applications.

- Second, we set up a new partition aware graph computation engine. It can effectively harness the online statistics of underlying graph partitioning results, guide parallel processing resources and improve computation performance.

- Third, we design a Dynamic Concurrency Control Model based on several heuristic rules to better profile the characters of graph partition. The DCCM can produce optimal configurations for the graph computation systems.

- At last, we test PAGE on a real large web graph dataset with a moderate size of cluster. Experiments obviously demonstrate that PAGE performs efficiently under a variety of graph partitioning schemes with different qualities.

The remaining of this paper is organized as follows: in Section 2, we first review related literatures. Then in Section 3 we elaborate on PAGE's framework design, details of message processor, partition aware module and the DCCM model. Section 4 reveals the experiment results. At last, we conclude this work and outline future research directions.

2. RELATED WORK

Here we briefly discuss the related research directions as follows.

Graph Computation Systems: Parallel graph computation is a popular technique to process and analyze large scale graphs. Unlike traditional parallel process framework(i.e. MapReduce), most of the current graph systems store graph data in memory instead of file and worker nodes exchange data in a message passing paradigm [8].

For example, Pregel [2] and its open source implementation, Giraph, both follow the Bulk Synchronous Parallel(BSP) model [9] to execute graph algorithms in parallel. A graph computation task in Pregel is divided into several *supersteps* by the global synchronization barriers. However, the default partition scheme in Pregel is hash partition, and it can not utilize the benefit brought by high quality graph partitioning.

Graph Processing System(GPS) [3], another open source implementation of Pregel, has three additional enhancements and applies several other optimizations to improve performance. One optimization is that GPS uses a centralized message buffer in a worker to utilize high quality graph partitioning and decrease the number of synchronization. But this optimization is still preliminary and can not extend to support a variety of graph computation systems.

Graph Partitioning Algorithms: As a well studied research area, graph partition attracts lots of attention, especially in recent parallel graph processing era. METIS [10] is a state of the art off-line graph partitioning package, and it can bring off high quality graph partitioning subject to a variety of requirements.

More recently, streaming graph partitioning models become appealing [11, 12]. They usually partition the large input graph in distributed loading stage for incoming graph or dynamic ones. [11] identifies ten heuristic rules, among which the Linear Deterministic Greedy(LDG) performs best.

We should emphasize that though there are remarkable progresses in these above areas, the parallel graph processing and foremost partition are usually isolated. In this paper, we are not aiming to invent new graph partition algorithms, but instead we exploit the information from graph partition to guide the parallel graph computing. It guarantees that these systems can efficiently execute the tasks on various graph partitioning schemes with different qualities.

3. DESIGN OF PARTITION AWARE GRAPH COMPUTATION

PAGE stands for a novel **P**artition **A**ware **G**raph computation **E**ngine. In this section, we first outline its overall framework and then introduce two new modules to support partition aware processing.

3.1 Framework of PAGE

PAGE is designed to support different graph partition qualities and maintain high performance by the online ad-

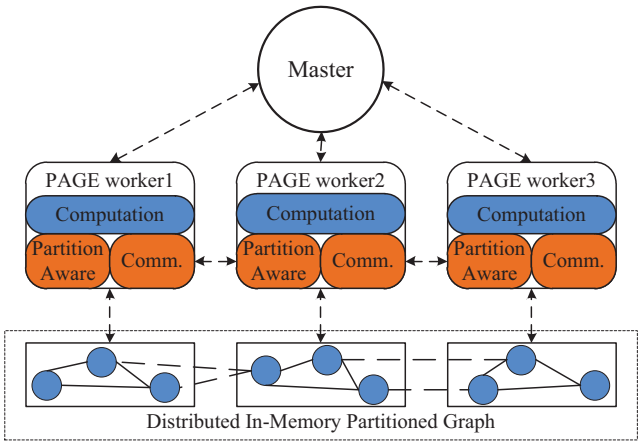


Figure 2: Architecture of PAGE

justing mechanism and some new cooperation methods. Figure 2 illustrates the architecture of PAGE. Similar to most of the current parallel graph computation systems, PAGE follows the Master-Worker paradigm. The master is responsible for aggregating global statistics and coordinating global synchronization. The new worker in Figure 3 is equipped with an enhanced communication module and a newly introduced partition aware module. Now workers in PAGE take on the graph computation task aware of underlying graph partitioning information.

In our novel design, the enhanced communication module integrates a dual concurrent message processor, which concurrently processes local and remote incoming messages. The partition aware module monitors several online metrics and adjusts the concurrency of dual concurrent message processor through a dynamic estimation model.

The computation in PAGE still consists of several supersteps separated by global synchronization barriers. In each superstep, each vertex runs a vertex-program with messages from the previous superstep concurrently, and then sends messages to other vertices if necessary. The computation finishes when no vertexes send out messages.

The simplest way to support the dual concurrent message processor is to add a sufficiently large number of message process units and distribute them into local and remote message processor at the begin of running the system. However, it is costly and also challenging to determine a reasonable number of message process units ahead of actual execution without any reasonable assumption [13]. In PAGE, we dynamically adjust the concurrency of message processor through a partition aware module so that the system can run fluently and efficiently.

3.2 Communication Module

The traditional communication module is unable to scale well with the high quality graph partitioning. Here, we design a dual concurrent message processor to accept local and remote incoming message blocks. With proper configurations for this new message processor, PAGE can efficiently deal with incoming messages over a variety of graph partitioning schemes with different partitioning qualities. Here we first introduce the basic concepts used in message processor, and then explain its mechanism in detail.

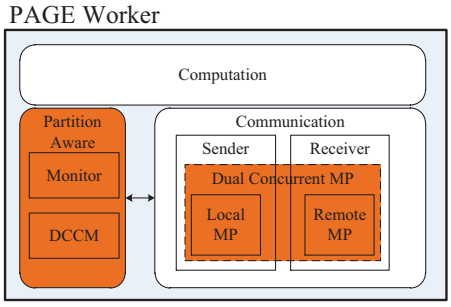


Figure 3: New Designed PAGE worker

3.2.1 *Message Processor*

It is well known that network communication is a costly operation [14], and we always try to reduce the number of network IO operations. A useful optimization is to combine several outgoing messages with the same destination into a single *message block* and thus we can reduce several network IO operations into one. Most of the current parallel graph computation systems apply this method in communication module.

But the above mentioned optimization also brings up side-effects. When one worker receives some incoming message blocks, it needs to parse the message blocks and dispatch extracted message to the specific worker's message queue. The overhead brought by this operation depends on the specific implementation. The following of this section first gives two basic concepts used in the paper, and then discuss methods to identify the cost and strategies to reduce it:

Message Process Unit: It is a minimal independent process unit in communication module that is responsible to extract messages of each vertex from incoming message blocks and update the corresponding vertex's message queue. The message process unit that only processes remote(local) incoming message blocks is called remote(local) message process unit.

Message Processor: It is a collection of message process units. The remote(local) message processor only contains remote(local) message process units.

Figure 4 illustrates the pipeline that the message process unit receives incoming message blocks, parses them into separate messages, and appends the message to corresponding vertex's message queue.

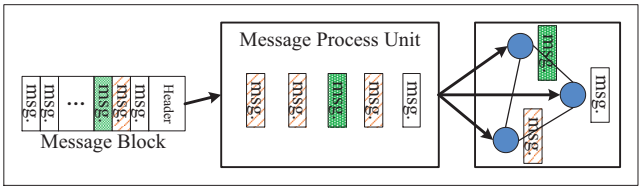
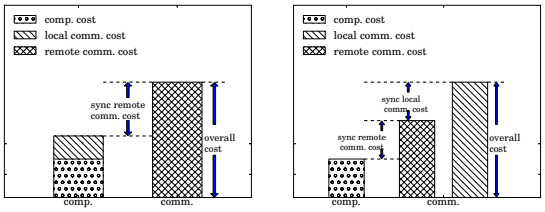


Figure 4: Message Processing Pipeline in PAGE

A worker in most of the current parallel graph computation systems deals with incoming message blocks from both local and remote sources. In general, we can use a centralized buffer to store both kinds of incoming message blocks, and concurrent message process units get message blocks to be parsed from the centralized buffer. But the two sources of incoming message blocks have been naturally concurrent



(a) Combination in Giraph (b) Combination in PAGE

Figure 5: Different Combinations of Computation Cost, Local/Remote Communication Cost. The Arrows Indicate The Components of Overall Cost.

when they receive message blocks. If we merge them into a centralized buffer, the centralized buffer would mandatorily increase the length of message block's parsing path and may become the bottleneck.

3.2.2 Dual Concurrent Message Processor

First, let's take a look at different types of cost in a graph computation system. The cost of a communication module consists of *local communication cost* which is caused by processing local incoming message blocks and *remote communication cost* from processing remote incoming message blocks. With the addition of the *computation cost* in computation module, there are kinds of combinations among the above three types of cost.

Figure 5 lists two kinds of combination and illustrates the other components of the overall performance under each case. From it, we can find that the combination (b) is better at separately processing local and remote incoming messages. The separated processing design can improve the overall performance when the *local communication cost* is high.

Moreover, the incoming messages are finally appended to the vertex's message queue, so different vertices can easily be updated concurrently. Based on this observation, we apply the concurrent message process units at the internal of local and remote message processor. Therefore, both local and remote message processors can concurrently process the incoming message blocks if the workload exceeds the ability of a single message process unit.

Dual concurrent message processor consists of a local and a remote message processor respectively. The first concurrency of message processor is that local and remote incoming messages are concurrently processed. The second concurrency is at the internal of the local and remote message processors.

3.3 Partition Aware Module

To guarantee the high performance, PAGE needs to provide a mechanism to satisfy the following two requirements. First, PAGE should have sufficient message process units to make sure that the new incoming message blocks can be processed immediately, and at the same time do not block the whole system. Second, the assignment strategy of these message process units should ensure that each local or remote message process unit has balanced workload since disparity seriously destroys the overall performance of any parallel processing.

The above requirements indicate the following two heuristic rules:

Algorithm 1 Superstep Processing in PAGE

```
1: if DCCM detects key metrics changed significantly then
2:     DCCM reconfigures dual concurrent message processor parameter.
3: end if
4: for each active vertex v in partition P do
5:     call vertex program of v;
6:     send message to neighborhood;
7:     /*monitor tracks related statistics in the background.*/
8: end for
9: synchronization barrier
10: monitor updates key metrics, and feeds to DCCM
```

1. **Ability Lower-bound:** the message processing ability of the total message process units should be no less than the total workload of message processing.

2. **Workload Balance Ratio:** the assignment of total message process units should satisfy the ratio between local and remote workload.

Partition aware module monitors the underlying graph partitioning and adjusts PAGE's runtime parameters. It contains two key components: a monitor and a **D**ynamic **C**oncurrency **C**ontrol **M**odel. The monitor is used to maintain several necessary metrics and provide these information to DCCM. The DCCM generates the optimal parameters for dual concurrent message processor according to the current metrics.

In Algorithm 1, we illustrate the new procedure of a superstep in PAGE with integrating the partition aware module. At the beginning, the DCCM in partition aware module calculates the optimal parameters based on metrics from previous superstep, and then updates the configurations(i.e. concurrency, assignment strategy) of dual concurrent message processor. During this superstep, the monitor tracks the related statistics of key metrics in the background. The monitor updates key metrics based on these collected statistics and feeds up to date values of the metrics to the DCCM at the end of the superstep.

The DCCM quantifies the above two rules and calculates the optimal parameters, such as the total number of the message process units, the number of local message process units and the number of the remote message process units, based on metrics provided by the monitor.

Meanwhile, the monitor maintains the following three high-level metrics to assist the DCCM:

1. **Incoming Speed of Local Messages:** represents the workload of local message processing. It has integrated the message generation speed and edge cut of the local partition. This metric can be obtained by tracking the number of local incoming messages and the corresponding time cost.

2. **Incoming Speed of Remote Messages:** represents the workload of remote message processing. It not only encodes the message generation speed and edge cut of the local partition, but also encodes the penalty of network influence. It can be calculated similarly to the speed of local incoming messages.

3. **Message Processing Speed:** defines the velocity of message processing, and reflects the ability of a message process unit to process messages. The number of messages processed by a message process unit and the corresponding time cost can derive the this metrics.

4. EMPIRICAL STUDY

We have developed a PAGE prototype on top of the open source project, Giraph[1]. To demonstrate its performance, we conducted extensive experiments on an in-house cluster of 24 nodes and proved its advantages. Here we first introduce the environment, datasets, baselines and the metrics used. Then wereveal the partition awareness performance and counterpart comparison experiments.

4.1 Experimental Setup

In our in-house cluster, each processing node has an AMD Opteron 4180 2.6Ghz CPU, 48GB memory and a 10TB disk RAID. Nodes are connected by 1Gbt routers.

Datasets: The graph dataset is uk-2007-05-u [15, 16]. It is an undirected one created from the original release by adding reciprocal edges as well as eliminating loop circles and isolated nodes. Table 1 lists the overview information of the dataset with both directed and undirected versions.

Graph	Vertices	Edges	Type
uk-2007-05	105,896,555	3,738,733,648	directed
uk-2007-05-u	105,153,952	6,603,753,128	undirected

Table 1: Graph Dataset Information

Graph Partitioning Scheme : We partitioned the graph with three strategies: Random, METIS and Linear Deterministic Greedy(LDG). The uk-2007-05-u graph is partitioned into 60 subgraphs. The balance factors of all these partitions do not exceed 1%. The parameter setting of METIS is the same with METIS-balanced approach in GPS[3].

In order to produce a variety of partition qualities of a graph, we extend the original LDG algorithm to an iterative version. Here, LDG partitions the graph based on the previous partition result, and gradually improves the partition quality in each following iteration. We name the partition result from iterative LDG as LDG*id*. A larger *id* indicates the higher quality of graph partitioning and that the more iterations are executed.

Baselines :Here we select two baselines to compare the advantages of the proposed PAGE.

First, as we notice from Figure 5(a), the default Giraph executes local message processing directly in computation thread, and it blocks following computations. This means that local message processing and computation are sequentially run in the default Giraph. And this combination model is inconsistent with our evaluation. We modified it to asynchronously process the local messages, so that the Giraph also concurrently runs computation, local message processing and remote message processing. Be aware that, the modification will not decrease the performance. In the following experiments, **Giraph** means the modified Giraph version.

As mentioned in Section 2, one optimization in GPS, applies a centralized message buffer and processes incoming messages sequentially without synchronizing operations. It would also eliminate partition unaware problem in a way. We implemented this optimization on Giraph, noted as **Giraph-GPSop** later.

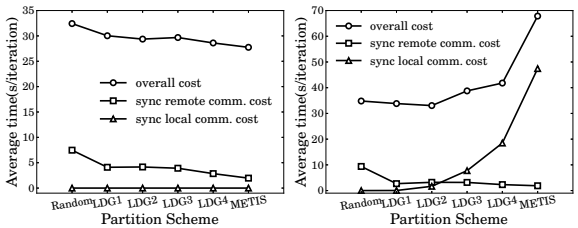
Metrics For Evaluation We ran PageRank algorithm to test the performance, in 10 iterations and used the following metrics to evaluate the performance of different graph computation systems:

- **Overall Cost.** It indicates the overall performance of the graph computation system. Due to the property of concurrent computation and communication model, this metric is determined by the slower between computation and communication. PAGE has three concurrent components as described in Figure 5(b).
- **Sync Remote Communication Cost.** It presents the cost of waiting for all the IO operations to be accomplished successfully after computation finished.
- **Sync Local Communication Cost.** It measures the cost of waiting for all local messages to be processed successfully after syncing remote communication.

All three metrics are measured by average time cost per iteration. We can refer to Figure 5(b) for the relationship among these metrics.

4.2 Partition Awareness in PAGE

Here we demonstrate the PAGE's partition aware advantage over Giraph. Figure 6 reveals the PageRank performance. We find that, with the increasing quality of graph partitioning, Giraph suffers from the workload growth of local message processing and the *sync local communication cost* raises fast. In contrast, PAGE can scalably handle the upsurging workload of local message processing, and maintain the *sync local communication cost* close to 0.



(a) PAGE (b) Giraph

Figure 6: PageRank on Different Implementations

Figure 6(a) also shows that in PAGE the overall performance increases along with the improvement of graph partitioning. When the edge cut decreases from 98.52% to 3.48%, the performance is improved by 14% in PAGE. However, in Giraph, the performance is downgraded about 100% at the same time.

As a partition aware system, PAGE can obtain the benefit brought by high quality graph partitioning, and still efficiently process low quality graph partitioning.

4.3 Optimization Rules Improvement

Here we demonstrate that, Giraph-GPSop is not an efficient approach without any other optimizations. In contrast, PAGE is an efficient partition aware system, and its method is extensible to other graph computation systems.

Figure 7 shows the comparison result of PAGE, Giraph-GPSop and Giraph. We notice that both Giraph-GPSop and PAGE achieve better performance with the quality of graph partitioning improving. But PAGE is more efficient than Giraph-GPSop over various graph partitioning with different qualities. Compared to Giraph, PAGE always wins over various qualities of graph partitioning, and the improvement ranges from 7% to 59%. However, Giraph-GPSop only beats Giraph and gains 8% improvement over METIS partition scheme which produces a pretty well partitioned graph. For Random partition, Giraph-GPSop is about 2.2 times worse than Giraph.

In [3], GPS is around 12 times faster than Giraph(old version). This is because the GPS applied several optimizations to reduce memory usage. These optimizations would speed up message processing, and finally a small number of message process units would satisfy the requirements. But in Giraph, without these optimizations, the speed of message processing is relatively slow, so the Giraph-GPSop suffers.

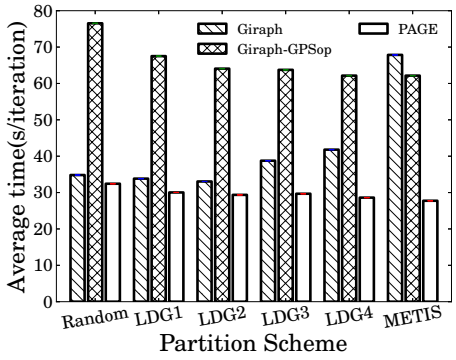


Figure 7: Performance Comparison among Giraph, Giraph-GPSop, PAGE.

The above results imply that the optimization in GPS for partition unaware problem is heavily dependent on other optimizations to extend to different graph computation systems. In contrast, PAGE performs well on top of Giraph without sophisticated optimizations and is more versatile.

5. CONCLUSION

In this paper, we identify the partition unaware problem in current graph computation systems and its severe drawbacks for efficient parallel large scale graphs processing . Then we introduce PAGE, a partition aware graph computation engine that monitors three high-level key running metrics and dynamically adjusts the system configurations. In the adjusting model, we discuss several heuristic rules to effectively cover the system characters and get near optimal parameters. We have successfully set up a prototype and conducted extensive experiments to prove that PAGE is an efficient and general parallel graph computation engine.

There are some principles inspired from our PAGE practice, which can serve as guidelines for lots of future work:

- **Concurrency is indispensable.** Message processors in the parallel graph system should be concurrent, not only because of the hardware support, but also due to the requirement of the natural vertex-centric update model.
- **Dynamic is elastic.** The system should dynamically determine the degree of concurrency for the message processor. The fixed number of message process units is limited.
- **Runtime statistics are valuable.** Since the local and remote message processing workloads are correlated to the quality of graph partitioning to some extent, the system should choose proper metrics and efficiently gather related statistics of the local partition.

ACKNOWLEDGMENTS

This research was supported by the National Natural Science foundation of China under Grant Grant No. 61272155, 60933004 and 61073019. Besides, thanks that Semih Salihoglu provides the dataset of partitioned web graph uk-2007-05-u.

6. REFERENCES

[1] *Giraph*. https://github.com/apache/giraph.
[2] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proc. of SIGMOD*, pages 135–146, 2010.
[3] Semih Salihoglu and Jennifer Widom. Gps: A graph processing system. Technical report, Stanford University, 2012.
[4] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *PVLDB*, 5(8):716–727, April 2012.
[5] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: distributed graph-parallel computation on natural graphs. In *Proc. of OSDI*, pages 17–30, 2012.
[6] Zuhair Khayyat, Karim Awara, Amani Alonazi, Hani Jamjoom, Dan Williams, and Panos Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. In *Proc. of EuroSys*, pages 169–182, 2013.
[7] Shang Zechao and Yu Jeffrey Xu. Catch the wind: Graph workload balancing on cloud. In *Proc. of ICDE*, pages 553–564, 2013.
[8] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978.
[9] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.
[10] George Karypis and Vipin Kumar. Parallel multilevel graph partitioning. In *Proc. of IPPS*, pages 314–319, 1996.
[11] Isabelle Stanton and Gabriel Kliot. Streaming graph partitioning for large distributed graphs. In *Proc. of KDD*, pages 1222–1230, 2012.
[12] Charalampos E. Tsourakakis, Christos Gkantsidis, Božidar Radunović, and Milan Vojnović. Fennel: Streaming graph partitioning for massive scale graphs. Technical report, Microsoft, 2012.
[13] John Ousterhout. Why threads are a bad idea (for most purposes). In *USENIX Winter Technical Conference*, January 1996.
[14] Guojing Cong, George Almasi, and Vijay Saraswat. Fast pgas implementation of distributed graph algorithms. In *Proc. of SC*, pages 1–11, 2010.
[15] P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *Proc. of WWW*, pages 595–602, 2004.
[16] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In *Proc. of WWW*, pages 587–596, 2011.