PAGE: A Partition Aware Engine for Parallel Graph Computation

Yingxia Shao, Bin Cui, *Senior Member, IEEE* and Lin Ma

Abstract—Graph partition quality affects the overall performance of parallel graph computation systems. The quality of a graph partition is measured by the balance factor and edge cut ratio. A balanced graph partition with small edge cut ratio is generally preferred since it reduces the expensive network communication cost. However, according to an empirical study on Giraph, the performance over well partitioned graph might be even two times worse than simple random partitions. This is because these systems only optimize for the simple partition strategies and cannot efficiently handle the increasing workload of local message processing when a high quality graph partition is used. In this paper, we propose a novel partition aware graph computation engine named PAGE, which equips a new message processor and a dynamic concurrency control model. The new message processor concurrently processes local and remote messages in a unified way. The dynamic model adaptively adjusts the concurrency of the processor based on the online statistics. The experimental evaluation demonstrates the superiority of PAGE over the graph partitions with various qualities.

Index Terms—Graph Computation, Graph Partition, Message Processing

1 Introduction

Massive big graphs are prevalent nowadays. Prominent examples include web graphs, social networks and other interactive networks in bioinformatics. The up to date web graph contains billions of nodes and trillions of edges. Graph structure can represent various relationships between objects, and better models complex data scenarios. The graph-based processing can facilitate lots of important applications, such as linkage analysis [8], [18], community discovery [20], pattern matching [22] and machine learning factorization models [3].

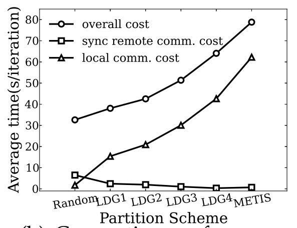
With these stunning growths of a variety of large graphs and diverse applications, parallel processing becomes the de facto graph computing paradigm for current large scale graph analysis. A lot of parallel graph computation systems have been introduced, e.g., Pregel, Giraph, GPS and GraphLab [23], [1], [27], [21]. These systems follow the vertex-centric programming model. The graph algorithms in them are split into several supersteps by synchronization barriers. In a superstep, each active vertex simultaneously updates its status based on the neighbors' messages from previous superstep, and then sends the new status as a message to its neighbors. With the limited workers (computing nodes) in practice, a worker usually stores a subgraph, not a vertex, at local, and sequentially executes the local active vertices. The computations of these workers are in parallel.

• *Yingxia Shao, Bin Cui (corresponding author) and Lin Ma are with the Key Lab of High Confidence Software Technologies (MOE), School of EECS, Peking University, China*
E-mail: {simon0227, bin.cui, malin1993ml}@pku.edu.cn

Therefore, graph partition is one of key components that affect the graph computing performance. It splits the original graph into several subgraphs, such that these subgraphs are of about the same size and there are few edges between separated subgraphs. A graph partition with high quality indicates there are few edges connecting different subgraphs while each subgraph is in similar size. The ratio of the edges crossing different subgraphs to the total edges is called **edge cut (ratio)**. A good balanced partition (or high quality partition) usually has a small edge cut and helps improve the performance of systems. Because the small edge cut reduces the expensive communication cost between different subgraphs, and the balance property generally guarantees that each subgraph has similar computation workload.

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

Fig. 1. PageRank on various web graph[1] partitions

However, in practice, a good balanced graph partition even leads to a decrease of the overall performance in existing systems. Figure 1 shows the performance of PageRank algorithm on six different

1. uk-2007-05-u, http://law.di.unimi.it/datasets.php,please refer to the detailed experiment setup in Section 6.

partition schemes of a large web graph dataset, and apparently the overall cost of PageRank per iteration increases with the quality improvement of different graph partitions. As an example, when the edge cut ratio is about 3.48% in METIS, the performance is about two times worse than that in simple random partition scheme where edge cut is 98.52%. It indicates that the parallel graph system may not benefit from the high quality graph partition.

Figure 1(b) also lists local communication cost and sync remote communication cost (explained in Section 2). We can see that, when the edge cut ratio decreases, the sync remote communication cost is reduced as expected. However, the local communication cost increases fast unexpectedly, which directly leads to the downgrade of overall performance. This abnormal outcome implies the local message processing becomes a bottleneck in the system and dominates the overall cost when the workload of local message processing increases.

Lots of existing parallel graph systems are unaware of such effect of the underlying partitioned subgraphs, and ignore the increasing workload of local message processing when the quality of partition scheme is improved. Therefore, these systems handle the local messages and remote messages unequally and only optimize the processing of remote messages. Though there is a simple extension of centralized message buffer used to process local and remote incoming messages all together [27], the existing graph systems still cannot effectively utilize the benefit of high quality graph partitions.

In this paper, we present a novel graph computation engine, Partition Aware Graph computation Engine (PAGE). To efficiently support computation tasks with different partitioning qualities, we develop some unique components in this new framework. First, in PAGE's worker, communication module is extended with a new dual concurrent message processor. The message processor concurrently handles both local and remote incoming messages in a unified way, thus accelerating the message processing. Furthermore, the concurrency of the message processor is tunable according to the online statistics of the system. Second, a partition aware module is added in each worker to monitor the partition-related characters and adjust the concurrency of the message processor adaptively to fit the online workload.

To fulfill the goal of estimating a reasonable concurrency for the dual concurrent message processor, we introduce the Dynamic Concurrency Control Model (DCCM). Since the message processing pipeline satisfied the prodcuer-consumer model, several heuristic rules are proposed by considering the producer-consumer constraints. With the help of DCCM, PAGE provides sufficient message process units to handle current workload and each message process unit has similar workload. Finally, PAGE can adaptively accept various qualities of the integrated graph partition.

A prototype of PAGE has been set up on top of Giraph (version 0.2.0). The experiment results demonstrate that the optimizations in PAGE can enhance the performance of both stationary and non-stationary graph algorithms on graph partitions with various qualities.

The main contributions of our work are summarized as follows.

- We propose the problem that existing graph computation systems cannot efficiently exploit the benefit of high quality graph partitions.
- We design a new partition aware graph computation engine, called PAGE. It can effectively harness the partition information to guide parallel processing resource allocation, and improve the computation performance.
- We introduce a dual concurrent message processor. The message processor concurrently processes incoming messages in a unified way and is the cornerstone in PAGE.
- We present a dynamic concurrency control model. The model estimates concurrency for dual concurrent message processor by satisfying the producer-consumer constraints. The model always generate proper configurations for PAGE when the graph applications or underlying graph partitions change.
- We implement a prototype of PAGE and test it with real-world graphs and various graph algorithms. The results clearly demonstrate that PAGE performs efficiently over various graph partitioning qualities.

This paper extends a preliminary work [32] in the following aspects. First, we detailed analyze the relationship among the workload of message processing, graph algorithms and graph partition. Second, technical specifics behind the dynamic concurrency control model are analyzed clearly. Third, the practical dynamic concurrency control model, which estimates the concurrency in incremental fashion, is discussed. Fourth, to show the effectiveness of DCCM, the comparison experiment between DCCM and manual tuning are conducted. Fifth, to show the advantage and generality of PAGE, more graph algorithms, such as diameter estimator, breadth first search, single source shortest path, are ran on PAGE.

The remaining paper is organized as follows. Section 2 discusses the workload of message processing in graph computation systems. We introduce PAGE's framework in Section 3. Sections 4 and 5 elaborate the dual concurrent message processor and dynamic concurrency control model respectively. The experimental results are shown in Section 6. Finally, we review the related work and conclude the paper in Section 7 and Section 8.

2 The Workload of Message Processing

In Pregel-like graph computation systems, vertices exchange their status through message passing. When the vertex sends a message, the worker first determines whether the destination vertex of the message is owned by a remote worker or the local worker. In the remote case, the message is buffered first. When the buffer size exceeds a certain threshold, the largest one is asynchronously flushed, delivering each to the destination as a single message. In the local case, the message is directly placed in the destination vertex's incoming message queue [23].

Therefore, the communication cost in a single worker is split into local communication cost and remote communication cost. Combining the computation cost, the overall cost of a worker has three components. Computation cost, denoted by t_{comp}, is the cost of execution of vertex programs. Local communication cost, denoted by t_{comml}, represents the cost of processing messages generated by the worker itself. Remote communication cost, denoted by t_{commr}, includes the cost of sending messages to other workers and waiting for them processed. In this paper, we use the cost of processing remote incoming messages at local to approximate the remote communication cost. There are two reasons for adopting such an approximation. First, the difference between two costs is the network transferring cost, which is relatively small compared to remote message processing cost. Second, the waiting cost of the remote communication cost is dominated by the remote message processing cost.

The workload of local (remote) message processing determines the local (remote) communication cost. The graph partition influences the workload distribution of local and remote message processing. A high quality graph partition, which is balanced and has small edge cut ratio, usually leads to the local message processing workload is higher than the remote message processing workload, and vice versa.

2.1 The Influence of Graph Algorithms

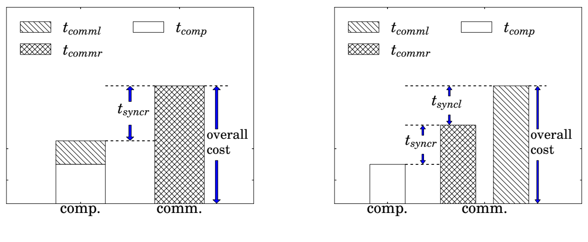
In reality, besides the graph partition, the actual workload of message processing in an execution instance is related to the characteristics of graph algorithms as well.

Here we follow the graph algorithm category introduced in [17]. On basis of the communication characteristics of graph algorithms when running on a vertex-centric graph computation system, they are classified into stationary graph algorithms and non-stationary graph algorithms. The **stationary graph algorithms** have the feature that all vertices send and receive the same distribution of messages between supersteps, like PageRank, Diameter Estimation [14]. In contrast, the destination or size of the outgoing messages changes across supersteps in the **non-stationary graph algorithms**. For example, traversal-based graph algorithms, e.g., Breadth First Search and Single Source Shortest Path, are the non-stationary ones.

In the stationary graph algorithms, every vertex has the same behavior during the execution, so the workload of message processing only depends on the underlying graph partition. When a high quality graph partition is applied, the local message processing workload is higher than the remote one, and vice versa. The high quality graph partition helps improve the overall performance of stationary graph algorithms, since processing local messages is cheaper than processing remote messages, which has a network transferring cost.

For the traversal-based graph algorithms belonging to the non-stationary category, it is also true that the local message processing workload is higher than the remote one when a high quality graph partition is applied. Because the high quality graph partition always clusters a dense subgraph together, which is traversed in successive supersteps. However, the high quality graph partition cannot guarantee a better overall performance for the non-stationary ones, because of the workload imbalance of the algorithm itself. This problem can be solved by techniques in [17], [33].

In this paper, we focus on the efficiency of a worker when different quality graph partitions are applied. The systems finally achieve better performance by improving the performance of each worker and leave the workload imbalance to the dynamic repartition solutions. The next subsection will reveal the drawback in the existing systems when handling different quality graph partitions.

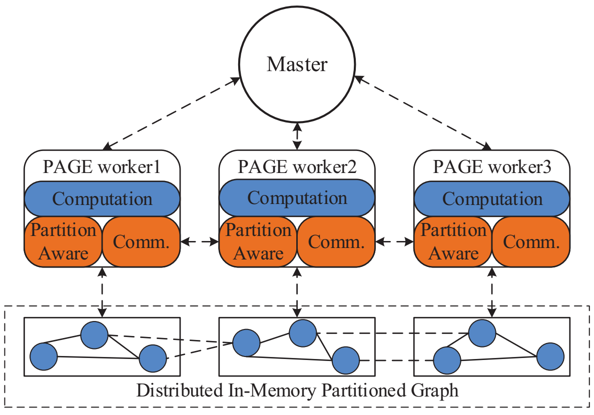


(a) Combination in Giraph (b) Combination in PAGE

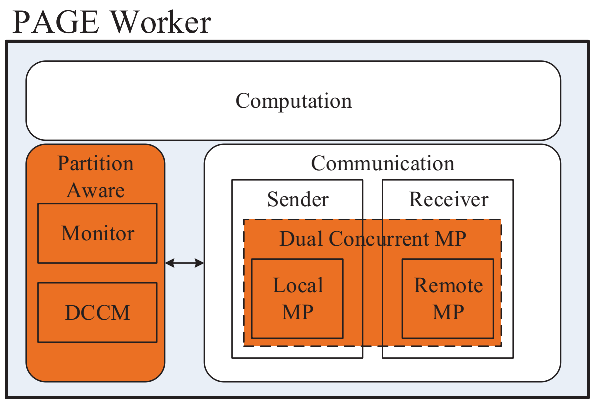
Fig. 2. Different combinations of computation cost, local/remote communication cost. The arrows indicate the ingredients of overall cost.

2.2 The Cost of a Worker in Pregel-like Systems

As mentioned before, the cost of a worker has three components. Under different designs of the communication module, there are several combinations of above three components to determine the overall cost



(a) Architecture



(b) Partition aware worker

Fig. 3. The framework of PAGE

of a worker. Figure 2 lists two possible combinations and illustrates fine-grained cost ingredients as well. Components in a single bar mean that their costs are additive because of the sequential processing. The overall cost equals the highest one among these independent bars.

The cost combination in Giraph is illustrated in Figure 2(a). The computation cost and local communication cost are in the same bar, as Giraph directly processes local message processing during the computation. The sync remote communication cost, $t_{syncr} = t_{commr} - t_{comp} - t_{comml}$, is the cost of waiting for the remote incoming message processing to be accomplished after the computation and local message processing finished. This type of the combination processes local incoming messages and remote incoming messages unequally, and the computation might be blocked by processing local incoming messages. When the workload of processing local incoming messages increases, the performance of a worker degrades severely. This is the main cause that Giraph suffers from a good balanced graph partition which is presented in Section 1.

3 PAGE

PAGE, which stands for **P**artition **A**ware **G**raph computation **E**ngine, is designed to support different graph partition qualities and maintain high performance by an adaptively tuning mechanism and new cooperation methods. Figure 3(a) illustrates the architecture of PAGE. Similar to the majority of existing parallel graph computation systems, PAGE follows the master-worker paradigm. The computing graph is partitioned and distributively stored among workers' memory. The master is responsible for aggregating global statistics and coordinating global synchronization. The novel worker in Figure 3(b) is equipped with an enhanced communication module and a newly introduced partition aware module. Thus the workers in PAGE can be aware of the underlying graph partition information and optimize the graph computation task.

3.1 Overview of two modules

The enhanced communication module in PAGE integrates a dual concurrent message processor, which separately processes local and remote incoming messages, and allows the system to concurrently process the incoming messages in a unified way. The concurrency of dual concurrent message processor can be adjusted by the partition aware model online, to fit the realtime incoming message processing workload. The detailed mechanism of the dual concurrent message processor will be described in Section 4.

The partition aware module contains two key components: a monitor and a Dynamic Concurrency Control Model (DCCM). The monitor is used to maintain necessary metrics and provide these information to the DCCM. The DCCM generates appropriate parameters through an estimation model and changes the concurrency of dual concurrent message processor. The details of monitor and DCCM will be presented in Section 5.

With the help of the enhanced communication module and the partition aware module, PAGE can dynamically tune the concurrency of message processor for local and remote message processing with lightweight overhead, and provide enough message process units to run itself fluently on graph partition with different qualities.

3.2 Graph Algorithm Execution in PAGE

The main execution flow of graph computation in PAGE is similar to the other Pregel-like systems. However, since PAGE integrates the partition aware module, there exist some extra works in each superstep and the modified procedure is illustrated in Algorithm 1. At the beginning, the DCCM in partition aware module calculates suitable parameters based on metrics from previous superstep, and then updates the configurations (e.g., concurrency, assignment strategy) of dual concurrent message processor. During a superstep, the monitor tracks the related statistics of key metrics in the background. The monitor updates key metrics according to these collected statistics and feeds up to date values of the metrics to the DCCM at the end of each superstep.

Note that PAGE will reconfigure the message processor in every superstep in case of the non-stationary graph algorithms. As discussed in Section 2, the quality of underlying graph partition may change between supersteps for the non-stationary graph algorithms with dynamic workload balance strategy. Even if the static graph partition strategy is used, the variable workload characteristics of non-stationary graph algorithms require the reconfigurations across supersteps. In summary, PAGE can adapt to different workloads caused by the underlying graph partition and the graph algorithm itself.

Algorithm 1 Procedure of a superstep in PAGE

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

4 Dual Concurrent Message Processor

The dual concurrent message processor is the core of the enhanced communication model, and it concurrently processes local and remote incoming messages in a unified way. With proper configurations for this new message processor, PAGE can efficiently deal with incoming messages over various graph partitions with different qualities.

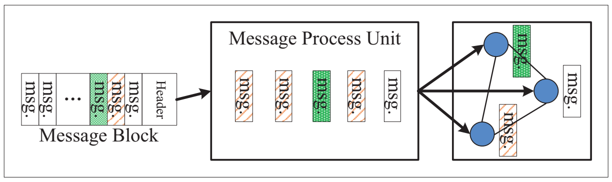


Fig. 4. Message processing pipeline in PAGE

As discussed in Section 2, messages are delivered in block, because the network communication is an expensive operation [10]. But this optimization raises extra overhead in terms that when a worker receives incoming message blocks, it needs to parse them and dispatches extracted messages to the specific vertex's message queue. In PAGE, the **message process unit** is responsible for this extra overhead, and it is a minimal independent process unit in the communication module. A remote (local) message process unit only processes remote (local) incoming message blocks. The **message processor** is a collection of message process units. The remote (local) message processor only consists of remote (local) message process units. Figure 4 illustrates the pipeline that the message process unit handles the overhead.

According to the cost analysis in Section 2.2, we can see that a good solution is to decouple the local communication cost from the computation cost, as the computation will not be blocked by any communication operation. Besides, the communication module can take over both local and remote communications, which makes it possible to process local and remote messages in a unified way. Furthermore, the incoming message blocks are concurrently received from both local and remote sources. It is better to process the local and remote incoming messages separately. These two observations help us design a novel message processor, which consists of a local message processor and a remote message processor. This novel message processor design leads to the cost combination in Figure 2(b). The sync local communication cost, $t_{syncl} = t_{comml} - t_{commr}$, is similar to the sync remote communication cost. It is the cost of waiting for the local incoming message processing to be accomplished after all the remote messages have been processed.

Moreover, in parallel graph computation systems, the incoming messages are finally appended to the vertex's message queue, so different vertices can be easily updated concurrently. Taking this factor into consideration, we deploy the concurrent message process units at the internal of local and remote message processor. Therefore, both local and remote message processors can concurrently process incoming message blocks, and local and remote incoming messages are processed in a unified way.

In summary, the new message processor consists of a local and a remote message processor respectively. This is the first type of the concurrency in the message processor. The second type of the concurrency is at the internal of local and remote message processor. This explains the reason we name this new message processor as dual concurrent message processor.

5 Dynamic Concurrency Control Model

The concurrency of dual concurrent message processor heavily affects the performance of PAGE. But it is expensive and also challenging to determine a reasonable concurrency ahead of real execution without any assumption [25]. Therefore, PAGE needs a mechanism to adaptively tune the concurrency of the dual concurrent message processor. The mechanism is named Dynamic Concurrency Control Model, DCCM for short.

In PAGE, the concurrency control problem can be modeled as a typical producer-consumer scheduling problem, where the computation phase generates messages as a producer, and message process units in the dual concurrent message processor are the consumers. Therefore, the producer-consumer constraints [29] should be satisfied when solving the concurrency control problem.

For the PAGE situation, the concurrency control problem arises consumer constraints. Since the behavior of producers is determined by the graph algorithms, PAGE only requires to adjust the consumers to satisfy the constraints (behavior of graph algorithms), which are stated as follows.

First, PAGE provides sufficient message process units to make sure that new incoming message blocks can be processed immediately and do not block the whole system. Meanwhile, no message process unit is idle.

Second, the assignment strategy of these message process units ensures that each local/remote message process unit has balanced workload since the disparity can seriously destroy the overall performance of parallel processing.

Above requirements derive two heuristic rules:

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

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

Following subsections first introduce the mathematical formulation of DCCM, and then discuss how DCCM mitigates the influences of various graph partition qualities. At last, we present the implementation of DCCM. Table 1 summaries the frequently used notations in the following analysis.

5.1 Mathematical Formulation of DCCM

In PAGE, DCCM uses a set of general heuristic rules to determine the concurrency of dual concurrent message processor. The workload of message processing is the number of incoming messages, and it can be estimated by the incoming speed of messages. Here we use s_l and s_r to denote the incoming speed of local messages and the incoming speed of remote messages, respectively. The ability of a single message processing unit is the speed of processing incoming messages, s_p.

Symbols	Description
er	Edge cut ratio of a local partition.
p	Quality of network transfer.
s_p	Message processing speed.
s_g	Message generating speed.
s_{rg}	Speed of remote messages generation
s_l	Incoming speed of local messages.
s_r	Incoming speed of remote messages.
n_{mp}	Number of message process units.
n_{rmp}	Number of remote message process units.
n_{lmp}	Number of local message process units.
t_{comp}	Computation cost.
t_{comm}	Communication cost.
t_{sycnr}	Cost of syncing remote communication.
t_{sycnl}	Cost of syncing local communication.

TABLE 1
Frequently used notations

On basis of aforementioned two heuristic rules, the following equations must hold.

$$\begin{aligned} n_{mp} \times s_p \geq s_l + s_r, & \qquad \textit{Rule 1} \\ \frac{n_{lmp}}{n_{rmp}} = \frac{s_l}{s_r}. & \qquad \textit{Rule 2} \end{aligned} \tag{1}$$

where n_{mp} stands for the total number of message process units, n_{lmp} represents the number of local message process units, and n_{rmp} is the number of remote message process unit. Meanwhile, n_{mp}=n_{lmp}+n_{rmp}.

Solving Equation 1 yields

$$\begin{aligned} n_{lmp} \geq \frac{s_l}{s_p}, \\ n_{rmp} \geq \frac{s_r}{s_p}. \end{aligned} \tag{2}$$

Finally, DCCM can estimate the concurrency of local message processor and the concurrency of remote message processor separately, if the metrics s_l, s_r, s_p are known. The optimal concurrency reaches when DCCM sets n_{lmp}=$\lceil \frac{s_l}{s_p} \rceil$ and n_{rmp}=$\lceil \frac{s_r}{s_p} \rceil$. Because, at this point, PAGE can provide sufficient message process units, and it consumes minimal resources as well.

5.2 Adaptiveness on Various Graph Partitions

Section 2 has analyzed that the workload of message processing is related to both graph partition and graph algorithms. In this section, we explain the reason that previous DCCM can adaptively tune the concurrency of dual concurrent message processor when the underlying graph partition quality is changed.

Before the detailed discussion, we first give three metrics.

1) **Edge cut ratio of a local partition.** It is the ratio between cross edges and total edges in a local graph partition. It is denoted as er. This metric judges the quality of graph partitioning

in a worker. The higher ratio indicates the lower quality.

2) **Message generating speed.** It represents the overall generating velocity of all outgoing messages in a worker. This metric implies the total workload for a worker. We denote it as s_g.
3) **Quality of network transfer.** This reflects the degree of network influence to the message generation speed s_g. When the generated messages are sent to a remote worker, the speed of generated messages is cut-off in the view of the remote worker. This is caused by the factor that network I/O operation is slower than local operation. The quality is denoted as $p \in (0, 1)$. The larger p indicates the better network environment. In addition, we can define the equivalent speed of remote messages' generation as $s_{rg} = s_g \times p$.

Now we proceed to reveal the relationship between DCCM and the underlying graph partition. Following analysis is based on the assumption that the stationary graph algorithms are ran on a certain partition. Because stationary graph algorithms have predictable communication feature.

The local incoming messages are the one whose source vertex and destination vertex belong to the same partition. Thus, the incoming speed of local message, s_l, is the same as $s_g \times (1 - er)$, which stands for the generating speed of local messages. Similarly, s_r equals $s_{rg} \times er$. Then Equation 1 can be rewrited as

$$\begin{aligned} n_{mp} \times s_p &= s_g \times (1 - er) + s_{rg} \times er, \\ \frac{n_{lmp}}{n_{rmp}} &= \frac{s_g \times (1 - er)}{s_{rg} \times er} = \frac{1 - er}{p \times er}. \end{aligned} \tag{3}$$

Solving n_{mp}, n_{lmp}, n_{rmp} from Equations 3, we can have the following results

$$n_{mp} = \frac{s_g}{s_p}(1 - (1 - p) \times er), \tag{4}$$

$$n_{lmp} = n_{mp} \times \frac{1 - er}{p \times er + (1 - er)}, \tag{5}$$

$$n_{rmp} = n_{mp} \times \frac{p \times er}{p \times er + (1 - er)}. \tag{6}$$

From Equations 4 5 6, we have following observations that indicate correlated relationships between graph partition and the behavior of DCCM when running stationary graph algorithms on PAGE.

First, PAGE needs more message process units with the quality growth of graph partitioning, but the upper bound still exists. This is derived from the fact that, in Equation 4, the n_{mp} increases while er decreases, since the p is fixed in a certain environment. However, the conditions, $0 < p < 1$ and $0 < er < 1$, always hold, so that n_{mp} will not exceed s_g/s_p. Actually, not only the parameters s_g, s_p dominate the upper bound of total message process units, but also p heavily affects the accurate total number of message process units under various partitioning quality during execution.

The accurate total number of message process units is mainly affected by s_g and s_p, while er only matters when the network is really in low quality. Usually in a high-end network environment where p is large, the term $(1 - p) \times er$ is negligible in spite of er, and then the status of whole system (s_g, s_p) determines the total number of message process units. Only in some specific low-end network environments, the graph partitioning quality will severely affect the decision of total number of message process units.

Unlike the total number of message process units, the assignment strategy is really sensitive to the parameter er. From Equation 3, we can see that the assignment strategy is heavily affected by (1-er)/er, as the value of p is generally fixed for a certain network. Lots of existing systems, like Giraph, do not pay enough attention to this phenomenon and suffer from high quality graph partitions. Our DCCM can easily avoid the problem by handling online assignment based on Equations 5 and 6.

Finally, when the non-stationary graph algorithms are ran on PAGE, the graph partition influence to the DCCM is similar as before. The difference is that the edge cut ratio of a local partition is only a hint, not the exact ratio for local and remote incoming message distribution. Because the unpredictable communication features of non-stationary graph algorithms cannot guarantee that a lower er leads to higher workload of local message processing. However it does for many applications in reality, such as traversal-based graph algorithms.

5.3 Implementation of DCCM

Given the heuristic rules and characteristic discussion of the DCCM, we proceed to present its implementation issues within the PAGE framework. To incorporate the DCCM's estimation model, PAGE is required to equip a monitor to collect necessary information in an online way. Generally, the monitor needs to maintain three high-level key metrics: s_p, s_l, s_r. However, there is a problem that it is difficult to measure accurate s_p. Because the incoming message blocks are not continuous and the granularity of time in the operating system is not precise enough, it is hard for the DCCM to obtain an accurate time cost of processing these message blocks. In the end, it leads to an inaccurate s_p.

Therefore, we introduce a DCCM in incremental fashion based on the original DCCM. Recall the cost analysis in Section 2, we can materialize the workload through multiplying the speed and the corresponding cost. Equation 2 can be represented as follows (use '=' instead of '$\geq$').

$$s_l \times t_{comp} = s_p \times n'_{lmp} \times (t_{comp} + t_{syncr} + t_{syncl}) \tag{7}$$

$$s_r \times t_{comp} = s_p \times n^{'}_{rmp} \times (t_{comp} + t_{syncr}) \quad (8)$$

where $n^{'}_{lmp}$ and $n^{'}_{rmp}$ are the concurrency of local and remote message processor in current superstep, respectively.

The estimated concurrency of the local and remote message processor for the next superstep can be

$$n_{lmp} = \frac{s_l}{s_p} = n^{'}_{lmp} \times (1 + \frac{t_{syncr} + t_{syncl}}{t_{comp}}) \quad (9)$$

$$n_{rmp} = \frac{s_r}{s_p} = n^{'}_{rmp} \times (1 + \frac{t_{syncr}}{t_{comp}}) \quad (10)$$

Finally, the new DCCM can estimate the latest n_{lmp} and n_{rmp} based on the previous values and the corresponding time cost t_{comp}, t_{syncl}, t_{syncr}. The monitor is only responsible to track three cost-related metrics. As the monitor just records the time cost without any additional data structures or complicated statistics, it brings the negligible overhead. In our PAGE prototype system implementation, we apply the DCCM with incremental fashion to automatically determine the concurrency of the system.

5.4 Extension to Other Data Processing Scenarios

Although the initial goal of dual concurrent message processor and dynamic concurrency control model is to help graph computation system efficiently handle messages and be partition aware, we found that the techniques can be beneficial to other data processing scenarios as well.

First, as a graph computation system, PAGE can also efficiently handle other structured data on which the problem can be remodeled as graph. For example sparse matrix-vector multiplication can be remodeled as a vertex-centric aggregation operator in a graph whose adjacent matrix equals the given sparse matrix and vertex values are from the given vector [26], then the techniques in PAGE will improve the performance of sparse matrix-vector multiplication under the corresponding graph model.

Second, the techniques in PAGE can also be extended to enhance other big data processing platform which satisfies the producer-consumer model. Distributed stream computing platform (e.g., Yahoo S4 [24], Twitter Storm [2]) is one kind of popular platforms to process real-time big data. In Storm, the basic primitives for doing stream transformations are "spouts" and "bolts". A spout is a source of streams. A bolt consumes any number of input streams, conducts some processing, and possibly emits new streams. Given an example of word count for a large document set, the core bolt keeps a map in memory from word to count. Each time it receives a word, it updates its state and emits the new word count. In practice, the parallelism of a blot is specified by user and this is inflexible. Since the logic of a blot satisfies the producer-consumer model, by designing an estimation model that is similar to DCCM, the stream computing platform can also automatically adjust the proper number of processing elements according to the workload.

6 EMPIRICAL STUDIES

We have implemented the PAGE prototype on top of an open source Pregel-like graph computation system, Giraph [1]. To test its performance, we conducted extensive experiments and demonstrated the superiority of our proposal. The following section describes the experimental environment, datasets, baselines and evaluation metrics. The detailed experiments evaluate the effectiveness of DCCM, the advantages of PAGE compared with other methods and the performance of PGAE on various graph algorithms.

6.1 Experimental setup

Graph	Vertices	Edges	Directed
uk-2007-05	105,896,555	3,738,733,648	yes
uk-2007-05-u	105,153,952	6,603,753,128	no
livejournal	4,847,571	68,993,773	yes
livejournal-u	4,846,609	85,702,474	no

TABLE 2
Graph dataset information

All experiments are ran on a cluster of 24 nodes, where each physical node has an AMD Opteron 4180 2.6Ghz CPU, 48GB memory and a 10TB disk RAID. Nodes are connected by 1Gbt routers. Two graph datasets are used: uk-2007-05-u [7], [6] and livejournal-u [5], [19]. The uk-2007-05-u is a web graph, while livejournal-u is a social graph. Both graphs are undirected ones created from the original release by adding reciprocal edges and eliminating loops and isolated nodes. Table 2 summarizes the meta-data of these datasets with both directed and undirected versions.

Graph partition scheme. We partition the large graphs with three strategies: Random, METIS and Linear Deterministic Greedy (LDG). METIS [16] is a popular off-line graph partition packages, and LDG [30] is a well-known stream-based graph partition solution. The uk-2007-05-u graph is partitioned into 60 subgraphs, and livejournal-u graph is partitioned into 2, 4, 8, 16, 32, 64 partitions, respectively. Balance factors of all these partitions do not exceed 1%, and edge cut ratios are list in Figure 1(a) and Table 3. The parameter setting of METIS is the same as METIS-balanced approach in GPS [27].

Furthermore, in order to generate various partition qualities of a graph, we extend the original LDG algorithm to an iterative version. The iterative LDG partitions the graph based on previous partition result, and gradually improves the partition quality in every following iteration. We name the partition result from iterative LDG as LDGid, where a larger id indicates the higher quality of graph partitioning and the more iterations executed.

Baselines. Throughout all experiments, we use two baselines for the comparison with PAGE.

The first one is Giraph. However, as we notice from Figure 2(a) that the local message processing and computation run serially in the default Giraph. This cost combination model is inconsistent with our evaluation. We modify it to asynchronously process the local messages, so that Giraph can concurrently run computation, local message processing and remote message processing. In the following experiments, **Giraph** refers to this modified Giraph version. Note that, this modification will not decrease the performance of Giraph.

The other one is derived from the technique used in GPS. One optimization in GPS, applies a centralized message buffer and sequentially processes incoming messages without synchronizing operations, which decouples the local message processing from the computation and treats the local and remote message equivalently. We implement this optimization on the original Giraph and denote it as **Giraph-GPSop**.

Metrics for evaluation. We use the following metrics to evaluate the performance of a graph algorithm on a graph computation system.

- **Overall cost.** It indicates the whole execution time of a graph algorithm when running on a computation system. Due to the property of concurrent computation and communication model, this metric is generally determined by the slower one between the computation and communication.
- **Sync remote communication cost.** It presents the cost of waiting for all I/O operations to be accomplished successfully after the computation finished. This metric reveals the performance of remote message processing.
- **Sync local communication cost.** It means the cost of waiting for all local messages to be processed successfully after syncing remote communication. This metric indicates the performance of local message processing.

All three metrics are measured by the average time cost per iteration. The relationship among these metrics can be referred to Figure 2.

6.2 Evaluation on DCCM

Dynamic Concurrency Control Model (DCCM) is the key component of PAGE to determine the concurrency for dual concurrent message processor, balance the workload for both remote and local message processing as well, and hence improve the overall performance. In this section, we demonstrate the effectiveness of DCCM through first presenting the concurrency automatically chosen by DCCM based on its estimation model, and then showing that these chosen values are close to the manually tuned good parameters. Finally, we also show DCCM converges efficiently to estimate a good concurrency for dual concurrent message processor.

6.2.1 *Concurrency chosen by DCCM*

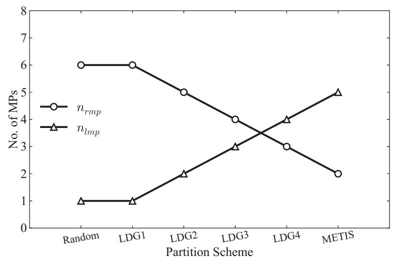


Fig. 5. Assignment strategy on different partition schemes in PAGE.

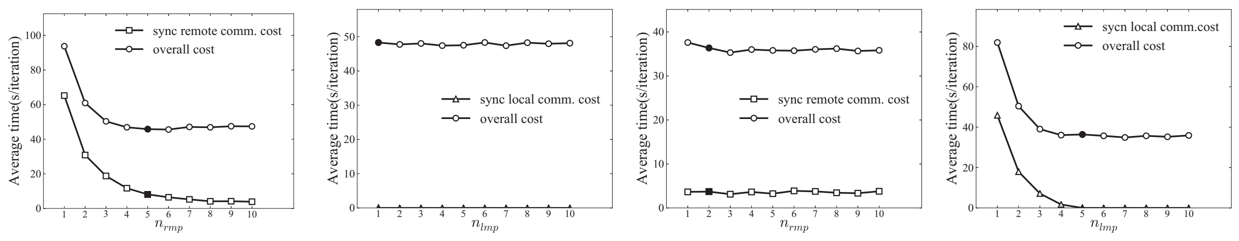
Figure 5 shows the concurrency of dual concurrent message processor automatically determined by the DCCM, when running PageRank on uk-2007-05-u graph. The variation of concurrency on various graph partition schemes are consistent with the analysis in Section 5.2.

In Figure 5, there are two meaningful observations. First, the total number of message process units, n_{mp}, is always equal to seven across six different partition schemes. This is because the cluster has a high-speed network and the quality of network transfer, p, is high as well. Second, with the decrease of edge cut ratio (from left to right in Figure 5), the n_{lmp} decided by the DCCM increases smoothly to handle the growing workload of local messages processing, and the selected n_{rmp} goes oppositely. According to above parameters' variations across different graph partition schemes, we also conclude that the assignment strategy is more sensitive to the edge cut ratio than the total message process units n_{mp}.

6.2.2 *Results by manual tuning*

Here we conduct a series of manual parameter tuning experiments to discover the best configurations for the dual concurrent message processor when running PageRank on uk-2007-05-u in practice. Hence, it helps verify that the parameters determined by DCCM are effective.

We tune the parameters n_{lmp} and n_{rmp} one by one on a specific partition scheme. When one variable is tuned, the other one is guaranteed to be sufficiently large that does not seriously affect the overall performance. When we conduct the tuning experiment on



(a) n_{rmp} tuning on Random (b) n_{lmp} tuning on Random (c) n_{rmp} tuning on METIS (d) n_{lmp} tuning on METIS

Fig. 6. Tuning on Random and METIS partition schemes. Black points are the optimal choices.

METIS scheme to get the best number of local message process units (n_{lmp}), for example, we manually provide sufficient remote message process units (n_{rmp}) with the DCCM feature off, and then increase n_{lmp} for each PageRank execution instance until the overall performance becomes stable.

Figure 6 shows the tuning results of running PageRank on Random and METIS partition schemes respectively. We do not list results about LDG1 to LDG4, as they all lead to the same conclusions as above two schemes. The basic rule of determining the proper configurations for a manual tuning is choosing the earliest points where the overall performance is close to the stable one as the best configuration.

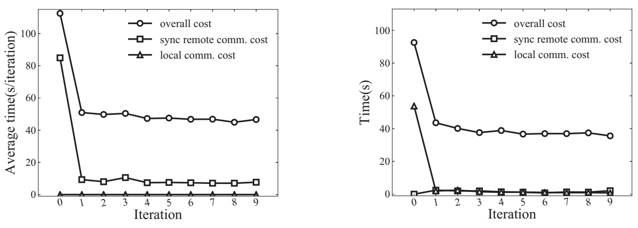
As shown in Figure 6(a), when the number of remote message process units exceeds five, the overall performance is converged and changes slightly. This indicates the remote message processor with five message process units inside is sufficient for this workload, which is running PageRank on random partitioned uk-2007-05-u. Though, the sync remote communication cost can still decrease a bit with continuously increasing n_{rmp}. But the large number of message process units will also affect other parts of the system (e.g., consuming more computation resources), the overall performance remains stable because of the tradeoff between two factors (i.e., number of message processor units and influence on other parts of the systems).

From Figure 6(b), we can easily figure out one local message process unit is sufficient to handle the remained local message processing workload in Random scheme. More message process units do not bring any significant improvement. Based on this tuning experiment, we can see that totally six message process units are enough. Among them, one is for the local message processor, and the other five are for remote message processor.

In Figure 5, the parameters chosen by the DCCM are six message process units for the remote message processor and one for local message processor, when the Random partition scheme is applied. Though they do not exactly match the off-line tuned values, but they fall into the range where the overall performance has been converged and the difference is small. So the DCCM generates a set of parameters almost as good as the best ones acquired from the off-line tuning.

By analyzing Figures 6(c) and 6(d), we can see seven message process units are sufficient, in which five belong to local message processor and the remained are for remote message processor. This time the DCCM also comes out the similar results as the manually tuned parameters.

Through a bunch of parameter tuning experiments, we verify the effectiveness of the DCCM and find that the DCCM can choose appropriate parameters.



(a) Random partition scheme (b) METIS partition scheme

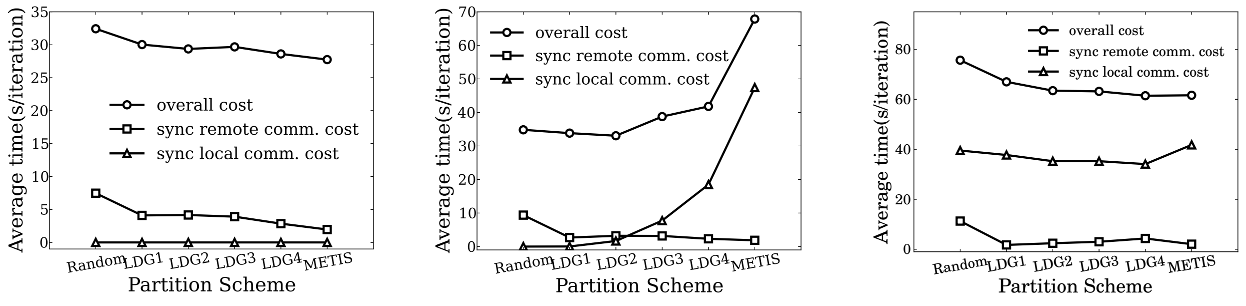
Fig. 7. Performance on adaptively tuning by DCCM

6.2.3 *Adaptivity of DCCM*

In this subsection, we show the results about the adaptivity on various graph partition scheme in PAGE. All the experimental results show that the DCCM is sensitive to both partition quality and initial concurrency setting. It responses fast and can adjust PAGE to a better status within a few iterations.

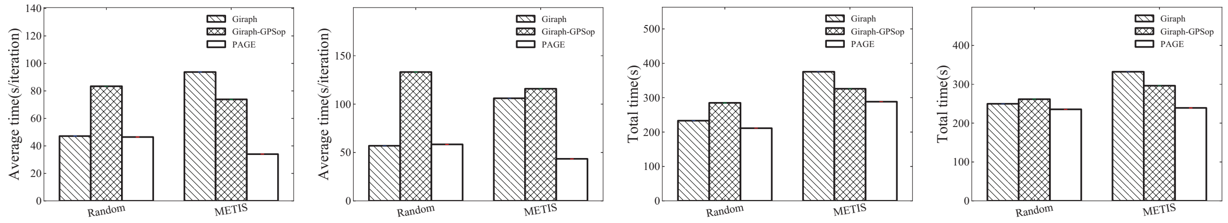
We run the PageRank on Random and METIS graph partition schemes, with randomly setting the concurrency of the remote message processor and local message processor at the beginning, and let DCCM automatically adjust the concurrency.

Figure 7(b) illustrates each iteration's performance of PageRank running on the METIS partition scheme of uk-2007-05-u. It clearly shows that PAGE can rapidly adjust itself to achieve better performance for the task. It costs about 93 seconds to finish the first iteration, where the sync local communication cost is around 54 seconds. After first iteration, PAGE reconfigures the concurrency of dual concurrent message processor, and achieves better overall cost in successive iterations by speeding up the process of local messages. The second iteration takes 44 seconds, and the sync local communication cost is close to



(a) Performance on PAGE (b) Performance on Giraph (c) Performance on Giraph-GPSop

Fig. 8. PageRank performance on different systems



(a) PageRank (b) Diameter Estimation (c) BFS (d) SSSP

Fig. 9. The performance of various graph algorithms

zero already. Similar results can be obtained for the Random partitioning scheme, shown in Figure 7(a).

6.3 Comparison with Other Methods

In this section, we compare the performance of PAGE with the Pregel-like baselines, i.e., Giraph and Giraph-GPSop. We first present the advantage of PAGE by profiling PageRank execution instance, followed by the evaluation on various graph algorithms. In the end, we show that PAGE can also handle the situation where varying the number of partitions leads to the change of graph partition quality.

6.3.1 Advantage of PAGE

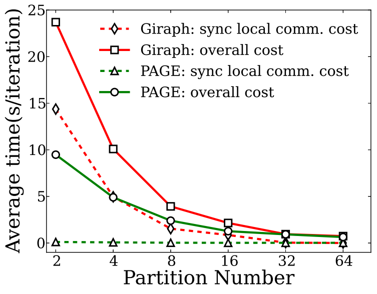
We demonstrate that PAGE can maintain high performance along with the various graph partition qualities. Figures 8(a) and 8(b) describe the PageRank performance across various partition schemes on PAGE and Giraph. We find that, with the increasing quality of graph partitioning, Giraph suffers from the workload growth of local message processing and the sync local communication cost rises fast. In contrast, PAGE can scalably handle the upsurging workload of local message processing, and maintain the sync local communication cost close to zero. Moreover, the overall performance of PAGE is actually improved along with increasing the quality of graph partitioning. In Figure 8(a), when the edge cut ratio decreases from 98.52% to 3.48%, the performance is improved by 14% in PAGE. However, in Giraph, the performance is even downgraded about 100% at the same time.

From Figure 8(c), we notice the Giraph-GPSop achieves better performance with the improving quality of graph partition as PAGE does. But PAGE is more efficient than Giraph-GPSop over various graph partitions with different qualities. Comparing with Giraph, PAGE always wins for various graph partitions, and the improvement ranges from 10% to 120%. However, Giraph-GPSop only beats Giraph and gains around 10% improvement over METIS partition scheme which produces a really well partitioned graph. For Random partition, Giraph-GPSop is even about 2.2 times worse than Giraph. It is easy to figure out that the central message buffer in Giraph-GPSop leads to this phenomena, as Figure 8(c) illustrates that the sync local communication cost[2] is around 40 seconds, though it is stable across six partition schemes. Overall, as a partition aware system, PAGE can well balance the workloads for local and remote message processing with different graph partitions.

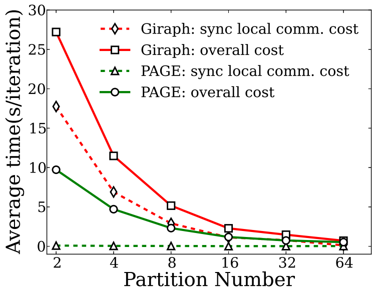
6.3.2 Performance on Various Graph Algorithms

In this section, we show that PAGE helps improve the performances of both stationary graph algorithms and non-stationary graph algorithms. The experiments are ran on PAGE, Giraph and Giraph-GPSop with two stationary graph algorithms and two non-stationary graph algorithms. The two stationary graph algorithms are PageRank and diameter estimation, and

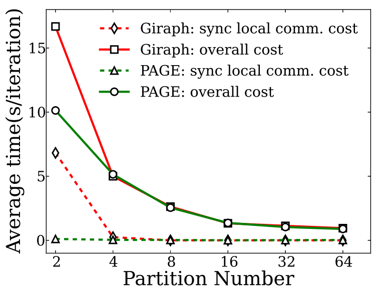
2. Due to the central message buffer, we treat all the messages as local messages and count its cost into the local communication cost in Giraph-GPSop.



(a) LDG partition scheme



(b) METIS partition scheme



(c) Random partition scheme

Fig. 10. Result of various partition numbers on social graph.

the two non-stationary ones are breadth first search (BFS) and single source shortest path (SSSP).

Figures 9(a) and 9(b) illustrate the performance of stationary graph algorithms on Random and METIS partition schemes of uk-2007-05-u graph dataset. We find that for the stationary graph algorithms, when the quality of graph partition is improved, PAGE can effectively use the benefit from a high quality graph partition and improve the overall performance. Compared with the Giraph and Giraph-GPSop, PAGE outperforms them because PAGE concurrently processes incoming messages in a unified way.

Figures 9(c) and 9(d) present the performance of non-stationary graph algorithms. Similar to the stationary graph algorithms, the performance of PAGE surpasses those of Giraph and Giraph-GPSop. This implies PAGE's architecture can also facilitate the non-stationary graph algorithms. However, the performance is not improved when the quality of partition scheme is increased. The reason has been discussed in Section 2. The non-stationary graph algorithms have a workload imbalance problem, which can be solved by the dynamic partition strategy [17], [33].

6.3.3 *Performance by varying numbers of partitions*

Previous experiments are all conducted on a web graph partitioned into fixed number of subgraphs, i.e., 60 partitions for uk-2007-05-u. In practice, the number of graph partitions can be changed, and different numbers of partitions will result into different partition qualities. We run a series of experiments to demonstrate that PAGE can also efficiently handle this situation. Here we present the results of running PageRank on a social graph, livejournal-u. Table 3 lists the edge cut ratios of livejournal-u partitioned into 2, 4, 8, 16, 32, 64 partitions by LDG, Random and METIS respectively.

First, Figures 10(a), 10(b) and 10(c) all show that both PAGE and Giraph perform better when the partition number increases across three partition schemes, which is obvious as parallel graph computing systems can benefit more from higher parallelism. When the graph is partitioned into more subgraphs, each subgraph has smaller size, and hence the overall performance will be improved with each worker having less workload. On the other hand, the large number of subgraphs brings heavy communication, so when the partition number reaches a certain threshold (e.g., sixteen in the experiment), the improvement becomes less significant. This phenomenon reveals parallel processing large-scale graph is a good choice, and it will improve the performance.

Partition Scheme	LDG(%)	Random(%)	METIS(%)
2 Partitions	20.50	50.34	6.46
4 Partitions	34.24	75.40	15.65
8 Partitions	47.54	87.86	23.54
16 Partitions	52.34	94.04	28.83
32 Partitions	55.55	97.08	32.93
64 Partitions	57.36	98.56	36.14

TABLE 3
Partition quality of livejournal-u

Second, the improvement between PAGE and Giraph decreases with the increasing number of partitions. As the number of partitions increases, the quality of graph partitioning decreases which means the local message processing workload decreases. Since Giraph performs well over the low quality graph partitioning, the performance gap between PAGE and Giraph is small when the number of partitions is large. Besides, the point where PAGE and Giraph have close performance varies with different graph partitioning algorithms. In METIS scheme, PAGE and Giraph have similar performance around 64 partitions, while in Random scheme, it is about four partitions when they are close. The reason is that different graph partition algorithms produce different partition quality, and the bad algorithms will generate low quality graph partition even the number of partitions is small.

Third, PAGE always performs better than Giraph across three partition schemes for any fixed number of partitions and the reason has been discussed in Section 6.3.1. But with the increasing number of partitions, the improvement of PAGE decreases. This is because the workload for each node becomes low when the partition number is large. For a relatively small graph like livejournal, when the partition number is around 64, each subgraph only contains tens of thousand vertices. However, the results are sufficient to show the robustness of PAGE for various graph partitions.

7 Related work

This work is related to several research areas. Not only graph computation systems are touched, but also the graph partition techniques and effective integration of them are essential to push forward current parallel graph computation systems. Here we briefly discuss these related research directions as follows.

Graph computation systems. Parallel graph computation is a popular technique to process and analyze large scale graphs. Different from traditional big data analysis frameworks (e.g., MapReduce [11]), most of graph computation systems store graph data in memory and cooperate with other computing nodes via message passing interface [13]. Besides, these systems adopt the vertex-centric programming model and release users from the tedious communication protocol. Such systems can also provide fault-tolerant and high scalability compared to the traditional graph processing libraries, such as Parallel BGL [12] and CGMgraph [9]. There exist several excellent systems, like Pregel, Giraph, GPS, Trinity.

Since messages are the key intermediate results in graph computation systems, all systems apply optimization techniques for the message processing. Pregel and Giraph handle local message in computation component but concurrently process remote messages. This is only optimized for the simple random partition, and cannot efficiently use the well partitioned graph. Based on Pregel and Giraph, GPS [27] applies several other optimizations for the performance improvement. One for message processing is that GPS uses a centralized message buffer in a worker to decrease the times of synchronization. This optimization enables GPS to utilize high quality graph partition. But it is still very preliminary and cannot extend to a variety of graph computation systems. Trinity [28] optimizes the global message distribution with bipartite graph partition techniques to reduce the memory usage, but it does not discuss the message processing of a single computing node. In this paper, PAGE focuses on the efficiency of a worker processing messages.

At the aspect of message processing techniques, the real-time stream processing systems are also related. Usually the stream processing systems are almost equal to message processing systems, since streams (or events) are delivered by message passing. N. Backman et al. [4] introduced a system-wide mechanism to automatically determine the parallelism of a stream processing operator and the mechanism was built on simulation-based search heuristics. In this paper, PAGE applies a node-level dynamic control model, but the basic idea is able to guide the design of system-wide solution.

Graph partitioning algorithms. To evaluate the performance of distributed graph algorithm, Ma et al. introduced three measures, which are visit times, makespan and data shipment. As efficiency (makespan) remains the dominant factor, they suggested to sacrifice visit times and data shipment for makespan, which advocates a well-balanced graph partition strategy when designing distributed algorithms [22]. Actually, various graph partitioning algorithms focused on this object as well. METIS [16], [15] is an off-line graph partitioning package which can bring off high quality graph partitioning subject to a variety of requirements. But it is expensive to use METIS partitioning large graphs. More recently, streaming graph partitioning models became appealing [30], [31]. In the streaming model, a vertex arrives with its neighborhood, and its partition id is decided based on the current partial graph information. The model is suitable for partitioning the large input graph in distributed loading context, especially for the state-of-the-art parallel graph computation systems. [30] described the difficulty of the problem and identified ten heuristic rules, in which the Linear Deterministic Greedy (LDG) rule performs best. The LDG assigns a vertex to a partition where the vertex has the maximal neighbors. In addition, it applies a linear penalty function to balance the workload.

Besides, several studies [17], [33], [27] focus on dynamic graph repartition strategies to achieve the well-balanced workload. The work of dynamic graph repartition and PAGE are orthogonal. To balance the workload, the proposed strategies repartition the graph according to the online workload. Thus the quality of underlying graph partition changes along with repartitioning. The existing graph computation systems may suffer from the high-quality graph partition at a certain point, but PAGE can mitigate this drawback and improve the performance by decreasing the cost of a worker further.

8 Conclusion

In this paper, we have identified the partition unaware problem in current graph computation systems and its severe drawbacks for efficient parallel large scale graphs processing. To address this problem, we proposed a partition aware graph computation engine named PAGE that monitors three high-level key running metrics and dynamically adjusts the system configurations. In the adjusting model, we elaborated two heuristic rules to effectively extract the system characters and generate proper parameters. We have successfully implemented a prototype system and conducted extensive experiments to prove that PAGE is an efficient and general parallel graph computation engine.

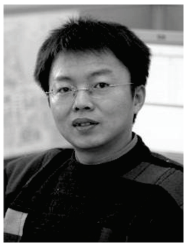
ACKNOWLEDGMENTS

The research is supported by the National Natural Science Foundation of China under Grant No. 61272155 and 973 program under No. 2014CB340405.

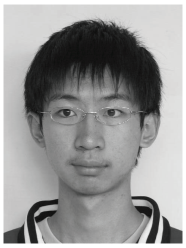
REFERENCES

[1] *Giraph*. https://github.com/apache/giraph.
[2] *Storm*. http://storm.incubator.apache.org/.
[3] A. Amr, S. Nino, N. Shravan, J. Vanja, and S. Alexander J. Distributed largescale natural graph factorization. In *WWW*, pages 37–48, 2013.
[4] N. Backman, R. Fonseca, and U. Çetintemel. Managing parallelism for stream processing in the cloud. In *HotCDP*, pages 1:1–1:5, 2012.
[5] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *SIGKDD*, pages 44–54, 2006.
[6] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In *WWW*, pages 587–596, 2011.
[7] P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *WWW*, pages 595–602, 2004.
[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW*, pages 107–117, 1998.
[9] A. Chan, F. Dehne, and R. Taylor. Cgmgraph/cgmlib: Implementing and testing cgm graph algorithms on pc clusters and shared memory machines. In *Journal of HPCA*, pages 81–97, 2005.
[10] G. Cong, G. Almasi, and V. Saraswat. Fast pgas implementation of distributed graph algorithms. In *SC*, pages 1–11, 2010.
[11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI*, pages 107–113, 2004.
[12] D. Gregor and A. Lumsdaine. The parallel bgl: A generic library for distributed graph computations. In *POOSC*, 2005.
[13] C. A. R. Hoare. Communicating sequential processes. In *CACM*, pages 666–677, 1978.
[14] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *ICDM*, pages 229–238, 2009.
[15] G. Karypis and V. Kumar. Parallel multilevel graph partitioning. In *IPPS*, pages 314–319, 1996.
[16] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *CDROM*, pages 1–13, 1998.
[17] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. In *EuroSys*, pages 169–182, 2013.
[18] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Journal of the ACM*, pages 604–632, 1999.
[19] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, pages 695–704, 2008.
[20] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640, 2010.
[21] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. In *PVLDB*, pages 716–727, 2012.
[22] S. Ma, Y. Cao, J. Huai, and T. Wo. Distributed graph pattern matching. In *WWW*, pages 949–958, 2012.
[23] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.
[24] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *ICDMW*, pages 170–177, 2010.
[25] J. Ousterhout. Why threads are a bad idea (for most purposes). In *USENIX Winter Technical Conference*, 1996.
[26] A. Roy, I. Mihailovic, and W. Zwaenepoel. X-stream: Edge-centric graph processing using streaming partitions. In *SOSP*, pages 472–488, 2013.
[27] S. Salihoglu and J. Widom. Gps: a graph processing system. In *SSDBM*, pages 22:1–22:12, 2013.
[28] B. Shao, H. Wang, and Y. Li. Trinity: A distributed graph engine on a memory cloud. In *SIGMOD*, pages 505–516, 2013.
[29] H. Simonis and T. Cornelissens. Modelling producer/consumer constraints. In *CP*, pages 449–462, 1995.
[30] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs. In *SIGKDD*, pages 1222–1230, 2012.
[31] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. Fennel: Streaming graph partitioning for massive scale graphs. In *WSDM*, pages 333–342, 2014.
[32] S. Yingxia, Y. Junjie, C. Bin, and M. Lin. Page: A partition aware graph computation engine. In *CIKM*, pages 823–828, 2013.
[33] S. Zechao and Y. Jeffrey Xu. Catch the wind: Graph workload balancing on cloud. In *ICDE*, pages 553–564, 2013.

Yingxia Shao received his BSc in Information Security from Beijing University of Posts and Telecommunications in 2011. Now he is a third year PhD candidate in the School of EECS, Peking University. His research interests include large-scale graph analysis, parallel computing framework and scalable data processing.



Bin Cui is a Professor in the School of EECS and Vice Director of Institute of Network Computing and Information Systems, at Peking University. His research interests include database performance issues, query and index techniques, multimedia databases, Web data management, data mining. He has served in the Technical Program Committee of various international conferences including SIGMOD, VLDB and ICDE, and as Vice PC Chair of ICDE 2011, Demo CO-Chair for ICDE 2014, Area Chair of VLDB 2014. He is currently in the Editorial Board of VLDB Journal, IEEE Transactions on Knowledge and Data Engineering (TKDE), Distributed and Parallel Databases Journal (DAPD), and Information Systems. Prof. Cui has published more than 100 research papers, and is the winner of Microsoft Young Professorship award (MSRA 2008), and CCF Young Scientist award (2009).



Lin Ma is a junior undergraduate student in the School of EECS, Peking University. He is taking internship in the group of Data and Information Management. Topics he has been working on include large graph processing, parallel graph computing and graph algorithms.