A BSP-based Parallel Iterative Processing System with Multiple Partition Strategies for Big Graphs

Zhigang Wang, Yubin Bao
Yu Gu, Fangling Leng, Ge Yu
College of Information Science & Engineering
Northeastern University
Shenyang, China
wangzhigang1210@163.com
{*baoyb, guyu, lengfl, yuge*}*@mail.neu.edu.cn*

Chao Deng, Leitao Guo
China Mobile Institute
China Mobile Corp.
Beijing, China
{*dengchao, guolt*}*@chinamobile.com*

***Abstract*—Many applications in real life can produce a large amount of data which can be modeled by a graph. A large graph usually has millions of vertices and billions of edges. This paper presents a BSP-based system, called BC-BSP+, to process large graphs iteratively in parallel. It has the flexibility to configure policies (i.e., disk management parameters) and extend functions (i.e., programming interfaces), and to compute large-scale graphs with fault tolerance and load balance. Especially, three graph partition strategies are proposed to support large graph processing: Randomized Hash Partition (RHP), Balanced Hash Partition (BHP) and Vertex-Cut based on the Range Partition method (VCRP). Lots of experiments are conducted to evaluate BC-BSP+. The experimental results show that the performance of VCRP is better than that of BHP, but the raw graph of the former must be crawled by the bread first search algorithm and vertex IDs must be numbered consecutively. We compare BC-BSP+ with Hadoop, a system based on MapReduce, and the speedup is roughly 8. Moreover, compared with the BSP-based systems, Hama and Giraph, the speedup is also 2 to 6 benefitting from VCRP.**

***Keywords*-BSP; MapReduce; graph process; load balance; graph partition;**

I. Introduction

Graph is an abstract data structure with numerous applications. It can express the real world effectively, such as the road network, the reference among technological literature, the links among web pages, and the relationship in social networks. The theory and algorithms on graphs have been well-studied over the last decades. However, most of them are only suitable for small graphs. With the development of the information technology, the scale of data is increasing drastically, which leads to many large-scale graphs. For instance, Google and Yahoo need to evaluate the importance of billions of web pages, which are modeled as a graph. Given that the graph of web pages is organized by the adjacency list and the size of one whole record is 100 bytes, for a graph with 10 billion vertices and 60 billion edges, it will occupy storage space with more than 1 TB. The cost of time and space of processing so large-scale graph usually exceeds the ability of a concentrated computing system. Therefore, it has become a new challenge to process large-scale graphs efficiently on distributed computing environments.

At present, Hadoop [1], a platform based on MapReduce [2] can process massive data with better fault-tolerance and scalability. While, most graph algorithms need iterative computations, which needs a chain of Hadoop jobs. The cost of warm-up and transferring the static data (i.e., the topology information of a graph) between two consecutive jobs is considerably large. In order to solve this problem, Google developed a parallel graph processing system based on the BSP model [3], called Pregel [4]. Another two open source projects based on BSP are also developed, Giraph [5] and Hama [6]. Giraph is developed by Yahoo and implemented on Hadoop. An application on Giraph is a special MapReduce job without the reduce stage. An inbuilt loop in the map task is used to simulate the iterative computing. Hama is being developed by Apache and also good at processing big data iteratively, especially for the matrix. Pregel and Giraph assumes that all data including graph data and intermediate data (i.e., messages) are resident in memory during iterations. Apparently, the scalability is limited by the memory capacity of the given cluster. Hama supports spilling message data on the local disk by a simple mechanism but the efficiency is rather low. We have developed an open source system whose architecture is similar to Hama, called BC-BSP [7]. BC-BSP uses disk as an assistant device to buffer data when the main memory does not have enough space.

In this paper, we design an enhanced system based on BC-BSP, called BC-BSP+, which supports a flexible configuration, efficient disk buffer management and multiple graph partition strategies. The features of BC-BSP+ are as follows. (a) BC-BSP+ provides flexible configuration and scalability. (b) It takes load-balance into consideration. BC-BSP+ schedules tasks to workers with the consideration of data locality and aims to keep the load-balance. (c) BC-BSP+ can handle very large scale graph data with limited resources benefitting from the disk buffer. (d) BC-BSP+

978-0-7695-5006-0/13 $26.00 © 2013 IEEE
DOI 10.1109/BigData.Congress.2013.31

supports several strategies to partition the raw graph: a Randomized Hash Partition (RHP) method, a Balanced Hash Partition (BHP) [8] method and a Vertex-Cut based on Range Partition (VCRP) method.

The remainder of this paper is organized as follows. Section II gives the overview of BC-BSP+. Section III describes the disk management mechanism. Graph partition strategies are presented in Section IV. Section V shows the experimental results and the analysis. The last section draws the conclusions.

II. Overview of BC-BSP+

A. BSP Model

BSP (Bulk Synchronous Parallel) is a bulk synchronous model [3]. There is a master to coordinate all tasks in the cluster for storing data and running programs. BSP is a parallel computing model which is well-suited for iterations. A BSP-based application consists of a series of iterations (i.e., super-steps in Pregel). In each iteration, all tasks are executed in asynchronously parallel, and they can send messages to other tasks to exchange the intermediate results. The next iteration can start until the computing and message operations (i.e., sending and receiving) of each task has been completed. The two consecutive iterations is separated by the barrier synchronization. All tasks keep alive until the graph algorithm terminates, which avoids the shortcomings of MapReduce-based systems. Pregel, Hama and Giraph are three typical systems based on BSP.

B. Architecture of BC-BSP+

BC-BSP+ is a BSP-based system and its architecture is shown in Figure 1. It is similar to BC-BSP [7] except the graph partition strategies.

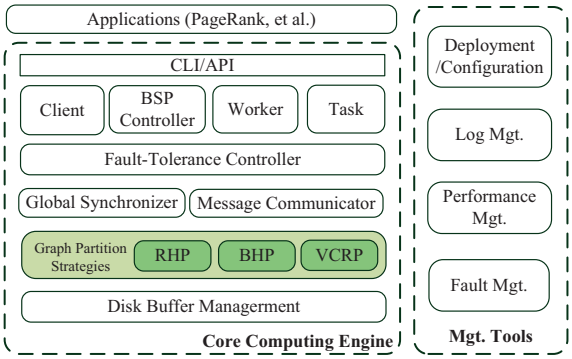

Figure 1: The entire system framework of BC-BSP+

Limited by the manuscript space, this paper only describes the partition strategies, the details of other components are referred to [7]. Users interact with the BC-BSP+ system by Client. All configuration parameters of this job will be delivered to BSPController by Client. For BC-BSP+, a new parameter is the type of graph partition strategies which can be RHP, BHP and VCRP. RHP is the default strategy because it is simple and no other additional conditions are needed. For BHP, users must set the number of $Virtual\ Buckets$ (see in Section IV). And for VCRP, the qualification is that each vertex is assigned a unique ID which is numbered consecutively. Therefore, users must choose a reasonable strategy according to their situations.

C. Interfaces of BC-BSP+

BC-BSP+ provides simple application program interfaces (APIs) for users to implement a complex graph algorithm. The following is a brief introduction to these APIs.

$compute()$: Users implements this interface to express the function of their applications. It will be invoked for every vertex at every super-step if it is necessary.

$VertexContextInterface$: BC-BSP+ is a vertex-centric system. When invoking $compute()$, the context of the current vertex must be visible for users. The context of a vertex includes its ID, value, aggregation results of the previous super-step, and received messages.

$Combiners$: It is used to merge messages sent to the same vertex to reduce the communication overhead.

$Aggregators$: Some statistic information, such as the rank errors for PageRank, can be collected by this interface.

$Partitioner$: Before processing the data, the raw data should be assigned to each task by a certain principle. Now, we provide three strategies for users to choose: Randomized Hash Partition (RHP), Balanced Hash Partition (BHP) and Vertex-Cut based on Range Partition (VCRP).

$Input$ and $Output$: Different Input and Output interfaces are suited for different data source, such as HDFS and HBase.

D. Implementation of BC-BSP+

1) BSPController: BSPController is the center of the BC-BSP+ cluster. It is responsible for managing all worker nodes, such as monitoring the status of the cluster, processing periodical heartbeat information reported by each worker, and controlling the global synchronization for each job. When a user submits a job, it assigns a unique job ID, and then prepares to schedule it. The scheduler selects one job to run by the priority and FIFO strategy. Then the task scheduler assigns tasks to workers considering load balance and data locality.

2) Worker Manager: WorkerManager manages tasks which run on this worker node and maintains statistic information. WorkerManager first registers itself to BSPController to join the BC-BSP+ cluster. Then, it sends heartbeat signals to BSPController to report its status periodically. When a new task arrives at a worker, WorkerManager will create a task process to run it. Tasks belonging to the same job and running on the same worker node will be managed by WorkerAgentForJob which is created by WorkerManager. WorkerAgentForJob is responsible for the local synchronization and aggregation.

3) Task: A BC-BSP+ job consists of several tasks. After all tasks have been assigned to worker nodes, they first load the raw graph data and then partition the data according to the partition strategy. After that, tasks can go into iterative computations.

III. Disk Buffer Management

Pregel and other open source systems, such as Hama and Giraph, suppose that there are enough worker nodes and resources in the cluster to hold all graph data and intermediate data (such as messages) in main memory during computing. However, considering the economic cost and the bottleneck of Master (i.e., BSPController) for a large Master-Slave infrastructure, it is an economic solution if we can spill some data on the disk.

BC-BSP+ applies disk space to store some graph data and intermediate messages temporally in order to process large-scale data by utilizing limited resources. As illustrated in Figure 2, BC-BSP+ divides the JVM heap space into three parts to cache graph data objects, messages and other objects. The space percentages occupied by the three parts are α, β and γ, which can be configured by users.

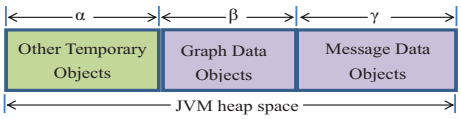

Figure 2: The management model for JVM heap space

In order to avoid the overflow of memory, the data should be spilled onto disk when the memory occupied by the data exceeds the given threshold. For BC-BSP+, graph data and message data are swapped by using hash bucket techniques. Thus, graph records and messages will be hashed into the corresponding bucket by the same hash function. Therefore, we can ensure that messages sent to the same vertex will be hashed into the same bucket. Thus, the local computing is divided into several stages. At every stage, one bucket is processed. For the ith stage, we first load messages sent to the ith hash bucket from the disk into the main memory. Then, graph data of the ith bucket are also put into memory. At last, every record will be processed by invoking $compute()$. Consequently, BC-BSP+ can match the graph data with the messages sent to them efficiently even though they are spilled on the disk.

For messages, each task maintains three queues: (1) ReceivedQueue: it manages messages which are sent from the last super-step, processed in the current super-step, and kept in memory as far as possible; (2) ReceivingQueue: it manages messages which will be processed in the next super-step, and have the highest priority to be spilled onto disk; (3) SendingQueue: manages messages which are produced during the computing, and will be combined by $Combiner$ when the length of the Queue exceeds a given threshold and sent to destination tasks immediately.

IV. Graph Data Partition Strategy

The overhead of communication is a heavy bottleneck for the distributed computation based on BSP. The existing works focus on the combination function, such as Pregel, Giraph, Hama and Hadoop. The communication cost is affected by the graph data partition strategy greatly [9]. Thus, we have designed three graph partition strategies for BC-BSP+: (1) Randomized Hash Partition (RHP), which is simple; (2) Balanced Hash Partition (BHP), which considers load balance; (3) Vertex-Cut based on the Range Partition (VCRP), which efficiently utilizes the locality of the raw graph. The process of loading data and partitioning a graph is called $Preprocessing$.

A. RHP and BHP Mechanisms

Randomized Hash Partition (RHP) is a simple partition strategy which has been adopted by Pregel, Hama and other platforms. RHP partitions a graph by the hash code of one vertex ID. It is simple but may lead to heavy load skew. Furthermore, RHP does not consider the locality of graph, so the communication cost among different partitions will be considerably large.

Traditional graph partition technology usually requires many iterations, so the time complexity is too high [10], [11], [12], [13], [14], and the partition results don't have the mapping information from vertices to partitions. Thus, we propose a Balanced Hash Partition (BHP) [8] method by extending RHP. The metric of one task's load is defined as $Computation\ Load$.

Definition 1: Computation Load

The computation load of one task includes two parts: processing local graph data and sending new messages. It can be evaluated as: $|E| \cdot (Cost_{disk} + Cost_{cpu} + Cost_{network})$, where, $|E|$ denotes the number of vertices and edges of one task. Considering that the scale of edges is rather larger than that of vertices, we ignore the overhead of processing vertices.

As illustrated in Figure 3, for a graph processing job with P tasks, in order to achieve the balance of each partition as much as possible, we first partition the raw graph into M buckets ($M > P$) and collect the metadata of every bucket, such as the number of records and outgoing edges. We call these M buckets as $Virtual\ Buckets$. After that, we re-organize the M $Virtual\ Buckets$ into P partitions by a greedy algorithm to balance the $Computation\ Load$ of each partition. In addition, we also consider the locality of different $Virtual\ Buckets$ to reduce the communication cost during re-organizing.

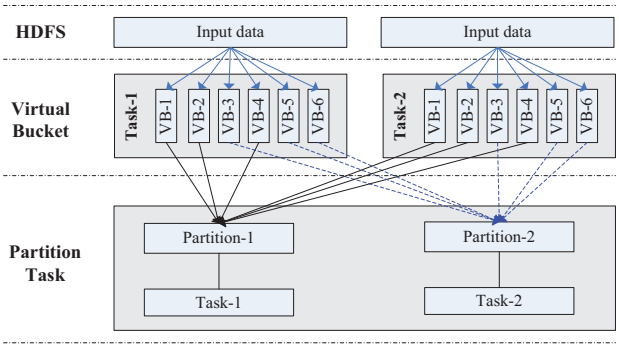

Figure 3: The illustration of the BHP method

B. Vertex-Cut Based on the Range Partition Strategy

PowerGraph [9] has proposed a new mechanism, named Vertex-Cut, to reduce the message scale passed among tasks during the iteration. Its effectiveness is validated by experiments. However, the $Preprocessing$ is rather time-consuming. PowerGraph designed three partition strategies: RHP, Oblivious and Coordinated. Oblivious and Coordinated are both heuristic algorithms. PowerGraph evaluated their replication factor of vertices and the running time of $Preprocessing$. The experiment results show that the strategy with a lower replication factor is more time-consuming. Thus, Oblivious is a compromised method.

The three partition strategies of PowerGraph ignore the locality of the raw graph data. The raw graph data are usually crawled from Web pages or social networks with different strategies. For big data, the breadth-first search (BFS) algorithm can yield high-quality pages [15] and is easy to be implemented in parallel [16]. We notice that the locality of the raw graph data generated by BFS can be kept by the range partition strategy. Furthermore, the range partition strategy has the flexibility to balance the $Computation\ Load$ of different tasks. Therefore, we propose the vertex-cut method based on the range partition mechanism, called VCRP, to obtain a lower replication factor and reduce the running time of $Preprocessing$.

To introduce VCRP better, we suppose that: (1) The raw graph data are crawled with the BFS algorithm. (2) Each vertex is assigned a unique ID which is numbered consecutively according to the order of BFS. (3) For the raw graph, a whole record, which includes the vertex data and its outgoing edge data, is maintained by only one task.

1) Range Partition Strategy: For a graph processing job, P tasks will be started to execute the computation in parallel, which means the raw graph must be divided into P partitions. Before the iterative computation, every task loads the partial graph data and organizes them on the local disk. Every partial graph has the approximate byte size in order to ensure the load balance. After that, every task collects the statistic information of its partial graph data, which is denoted by I. The information of the ith task is:

$$I_i =< ID[task], Min, R, |V|, |E| > \quad (1)$$

Where, Min is the minimal vertex ID in this partition, R denotes the range of vertex IDs, and $|V|$ and $|E|$ are the number of records and outgoing edges separately in this partition. I is reported to the BSPController and then the whole graph will be partitioned into P partitions. Every task handles only one partition. The map table between tasks and partitions is called the global route table G. G will be sent to every task to provide the addressing service for communication. For the ith partition P_i, its metadata is:

$$P_i =< ID[task], Min^{'}, R^{'}, |V|^{'}, |E|^{'} > \quad (2)$$

and the vertex ID is continuous between two consecutive partitions:

$$GetMin(P_{i+1}) = GetMin(P_i) + GetRange(P_i) \quad (3)$$

Many optimization strategies can be implemented during partitioning. As illustrated in Figure 4, the balance of $Computation\ Load$ can be controlled by adjusting the record distribution among different partitions. Consequently, the elements of P metadata are similar with I but their values may be different.

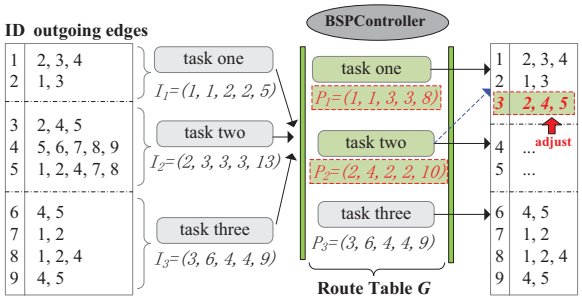

Figure 4: The illustration of the Range Partition method

2) Vertex-Cut based on Range Partition (VCRP): We integrate the Vertex-Cut technique with the Range Partition strategy and then propose the new VCRP method. It partitions the input graph with the Range Partition strategy first. After exchanging records among tasks, the next stage is $Shuffle$, which will be used to complete the Vertex-Cut. During $Shuffle$, every outgoing edge in P_i is sent to $P_i^{'}$ which contains its destination vertex ID. For $P_i^{'}$, edges belonging to the same source vertex v will be re-organized by the adjacency list. The source vertex in $P_i^{'}$ is a replication of v in P_i (as shown in Figure 5). After $Shuffle$, the system begins to compute graph iteratively.

During iterations, every vertex receives messages from the previous iteration, updates its value and then computes the value of its replications. Notice that its "outgoing edge

list" is now "partition distribution list" which records the locations of its replications. Then it sends "replication value" to all replication vertices by networks. In $P_i^{'}$, the replication vertex receives "replication value", generates messages, and then sends them to the destination vertices by local processing (as shown in Figure 5).

For example, in Figure 5, there are three partitions (i.e., tasks) of a graph job and the raw graph data are organized by the adjacency list. After $Shuffle$, outgoing edges are distributed to partitions (i.e., tasks) which contain the destination vertices (replications) and the raw outgoing edge list is used to record the distribution of partitions. And then, during the computation, tasks only send "replication value" to "replications" via network and local messages will be generated for the destination vertices. The "replication value" scale is rather less than that of "local message", so the overhead of communication will be reduced greatly.

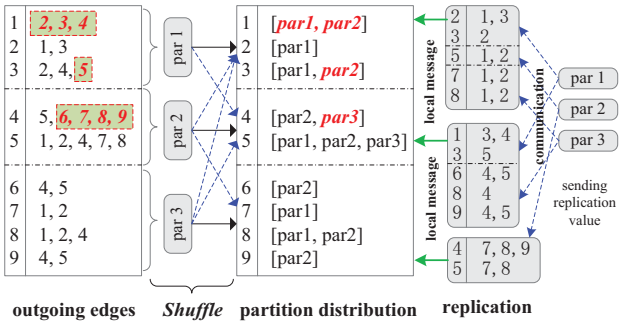

Figure 5: The illustration of the VCRP method

C. The Locality of VCRP Analysis

For the randomized Vertex-Cut, J.E. Gonzalez and Y. Low et al. [9] evaluated the expected replications for a vertex v as follows:

$$\Im[|H(v)|] = |P|(1 - (1 - \frac{1}{|P|})^{|E(v)|}) \quad (4)$$

Where, $|P|$ is the number of tasks (i.e., partitions) and $E(v)$ is the edge collection of v. And the expected replications for the whole graph is [9]:

$$\Im[\frac{1}{|V|}\sum_{v \in V}|H(v)|] = \frac{|P|}{|V|}\sum_{v \in V}(1 - (1 - \frac{1}{|P|})^{|E(v)|}) \quad (5)$$

Before analyzing the locality of VCRP, we fist give the notations throughout this section.

Definition 2: Maximal Destination Vertex ID (MDVID)

Before $Shuffle$, the partition which contains vertex v can be obtained from the route table G by $P_i = getPartition(G, ID[v])$. Then, the precursor partition collection is defined as: $P_{pre} = \{P_0, P_1, P_2, ..., P_i\}$. MDVID is the maximal destination vertex ID which satisfies: $\forall e \in P_{pre}, MDVID \geq DstID[e]$.

Definition 3: Randomized Distributed Edges ($E_{rde}(v)$)

Before $Shuffle$, for the vertex v, its outgoing edge collection is $E(v)$, then $E_{rde}(v) = \{e|DstID[e] \leq MDVID \wedge e \in E(v)\}$.

Definition 4: Cluster Distributed Edges ($E_{cde}(v)$)

Before $Shuffle$, for the vertex v, its outgoing edge collection is $E(v)$, then $E_{cde}(v) = \{e|DstID[e] > MDVID \wedge e \in E(v)\}$.

For example, in Figure 5, the outgoing edges in shadow boxes are all $Cluster\ Distributed\ Edges$. Then, for vertex v, we can evaluate the expected replications of VCRP as follows:

$$\Im[|R(v)|] = |P_{pre}|(1 - (1 - \frac{1}{|P_{pre}|})^{|E_{rde}(v)|}) + \chi \quad (6)$$

Where, χ is a variable which is equal to 1 if $|E_{cde}| > 0$ and $|P_{pre}| < |P|$. Otherwise, its value is zero.

Since the graph is generated by BFS and is partitioned with the Range Partition strategy, for v, if its $|E_{cde}| > 0$ and $|P_{pre}| < |P|$, then, the $DstID[e]$ must be in the same partition. Notice that, for some special vertices, the $DstID[e]$ may be across two consecutive partitions. While, for the whole graph, this situation only happens $(|P| - 1)$ times at most. So we ignore it for one vertex v and consider it when computing the maximal expected replications for the whole graph:

$$\Im[\frac{1}{|V|}\sum_{v \in V}|R(v)|] = \frac{1}{|V|}((|P| - 1) + \sum_{v \in V}\Im[|R(v)|]) \quad (7)$$

By analyzing the derivative of Formula 4 and Formula 6, we can conclude that for vertex v, the expected replications of VCRP is fewer than that of randomized Vertex-Cut, but their overheads of $Preprocessing$ are similar. And experiments in Section V show that the expected replications approximate to Coordinated, the complex heuristic algorithm of PowerGraph.

V. Experiments and Performance Evaluation

We evaluate BC-BSP+ prototype system by the PageRank algorithm over real data sets [17], [18], [19] and synthetic data sets. Data sets are listed in Table I. We compare BC-BSP+ with Hadoop, Giraph and Hama. The cluster is composed of 33 nodes which are connected by gigabit Ethernet to a switch. Each node has 2 Intel Core i3-2100 CPUs, 8GB RAM (but 2GB for one task), and a 500GB disk with 7,200 RMP.

A. Evaluation of RHP, BHP and VCRP

This suit of experiments is used to evaluate the performance of RHP, BHP and VCRP. First, we set 9 tasks and 1GB JVM memory to complete the graph $Preprocessing$ job.

Table I: Characteristics of datasets

Dataset	Vertices	Edges	Avg. Degree	File Size
Wiki-talk	2,394,385	5,021,410	2.097	45.4MB
Skitter	1,696,415	11,095,298	6.54	116MB
Patent	3,774,768	16,518,948	4.38	203MB
Live-J	4,847,571	68,993,773	14.23	700MB
Wiki-pp	5,716,808	130,160,393	22.77	1.5GB
Twitter	41,700,000	1,470,000,000	35.25	12.59GB

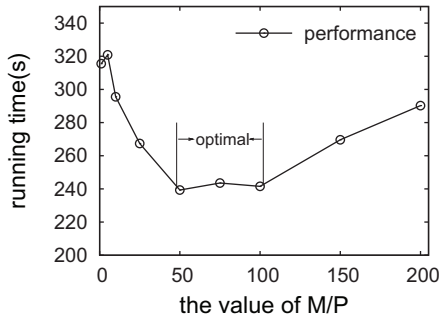

Figure 6: The effect of different M/P for PageRank

We validate the effect of BHP by comparing it with RHP. Considering that the Range Partition method can accurately adjust records to ensure the load balance, we do not evaluate its effect. Obviously, the number of $Virtual\ Buckets$ will affect the performance of BHP. As shown in Figure 6, for Live-J, when the number of M/P (M is the number of $Virtual\ Buckets$ and P is the number of partitions) is between 50 and 100, the effect is better. Thus, in the following experiments, we set the number of $Virtual\ Buckets$ as 50.

Since the $Computation\ Load$ is affected by the number of edges, we describe the load unbalance as the variance of outgoing edges among different partitions. We define the Maximal Unbalance Degree (MUD) as the D-value between the maximal volume of edges and the minimal volume of edges among partitions. Figure 7 shows that MUD of BHP has been reduced by roughly 30% than that of RHP. Figure 8 shows that, compared with RHP, the communication scale of $Preprocess$ of BHP can be reduced by up to 41% (for Live-J) because BHP considers the data locality when re-organizing the $Virtual\ Buckets$.

Figure 9 shows that running time of $Preprocessing$ for RHP, BHP and VCRP. We find that the cost of BHP is more than RHP because the former needs to re-organize $Virtual\ Buckets$ and construct the global route table. However, the performance of VCRP is similar to that of RHP, even better, which benefits from utilizing the locality of Range Partition.

Then we compare the communication scale per iteration and the overall running time of one graph job. Figure 10 shows that the communication scale of BHP can be reduced by up to 36% than that of RHP (for Live-J). For VCRP, the

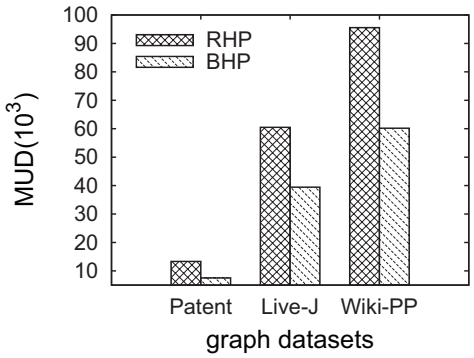

Figure 7: MUD of outgoing edges

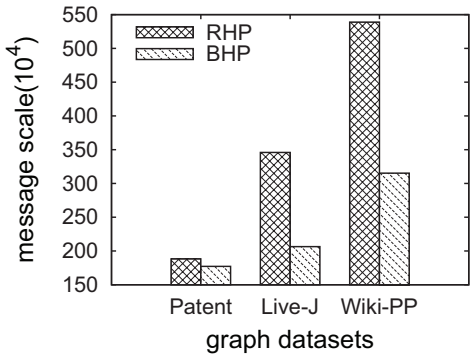

Figure 8: The message scale of $Preprocessing$

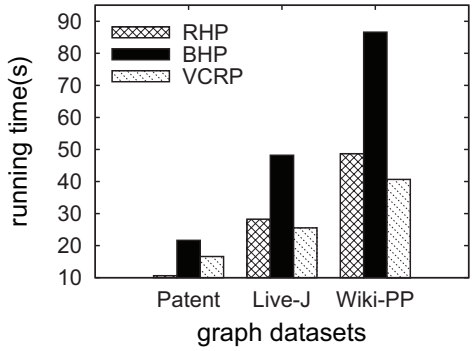

Figure 9: The running time of $Preprocessing$

value is even 87.3%. Figure 11 shows the overall running time. Obviously, the performance of BHP and VCRP is better than that of RHP. Especially, the speedup of VCRP compared to RHP is roughly a factor of 2 to 6. The reason is that VCRP reduces the communication scale effectively.

Moreover, we also compare the effect of VCRP with the original Vertex-Cut method of PowerGraph over Twitter. The number of tasks varies from 8 to 32. Oblivious and Coordinated are two heuristic methods of PowerGraph. Figure 12 shows that the replication factor of VCRP is similar to that of Coordinated. However, the running time of $Preprocessing$

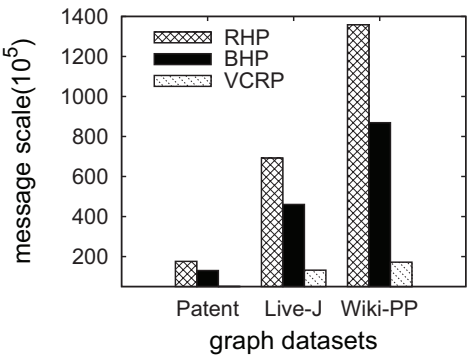

Figure 10: The message scale per iteration

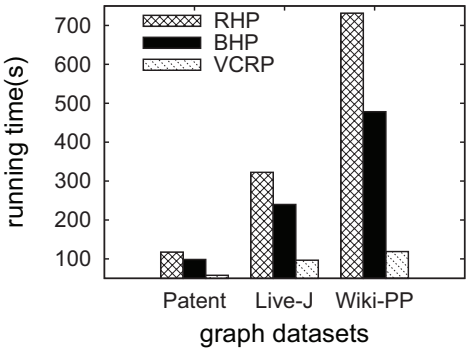

Figure 11: The overall performance

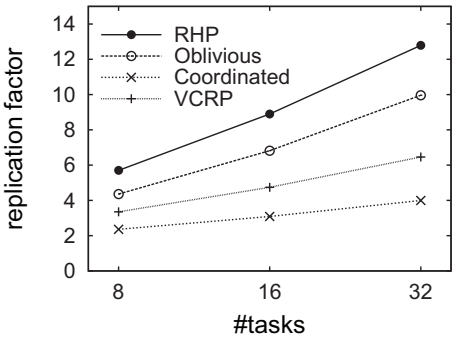

Figure 12: Replication factor (Twitter)

of VCRP is further less than that of Coordinated (as shown in Figure 13). Therefore, the overall performance of VCRP is better than that of other three methods.

Considering the wonderful performance of VCRP, we adopt it as the default partition strategy in the following experiments.

B. Evaluation of Overall Performance of BC-BSP+ and Hadoop

We use 9 nodes of the cluster to run PageRank for BC-BSP+ and Hadoop on real datasets listed in Table I. The JVM memory is set to 1GB. For BC-BSP+, 9 tasks

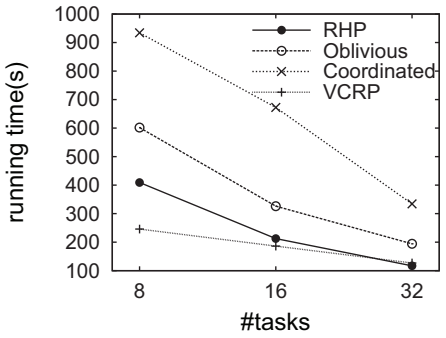

Figure 13: The running time of $Preprocessing$ (Twitter)

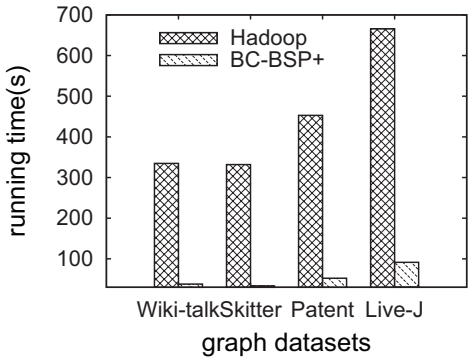

Figure 14: The overall performance of Hadoop and BC-BSP+

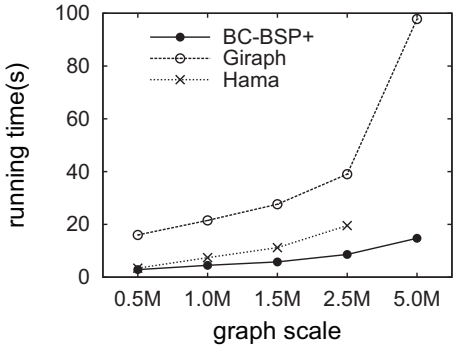

Figure 15: Comparison among BC-BSP+, Giraph and Hama over synthetic datasets, where 0.5M denotes 0.5 million vertices and the average degree is always 21.5

are started to compute in parallel. While, for Hadoop, the number of Mapper tasks is determined by the raw data block size of HDFS, but the number of Reducer tasks is setup as 9 (i.e., 9 nodes). Based on the above environment and configurations, the experiment results show that the speedup of BC-BSP+ compared to Hadoop is roughly a factor of 8 (shown in Figure 14).

C. Evaluation of Overall Performance of BC-BSP+, Giraph and Hama

This suit of experiments are used to test the processing ability of BC-BSP+, Giraph and Hama, by running the PageRank algorithm. Because the difference of the processing capability of the three BSP-based systems and the difference of their expressions on data, we generate the synthetic datasets to complete the comparison. The memory of every JVM is set as 2GB. The experimental results are shown in Figure 15.

As illustrated in Figure 15, the overall performance of BC-BSP+ is always better than that of Giraph and Hama. Hama can not run PageRank when the vertex scale is more than 2.5 million, since the system runs out of memory. The running time of BC-BSP+ is 2 times faster than that of Hama. Compared with Giraph, the speedup is even 6. BC-BSP+ can process large graphs efficiently with limited resources.

VI. Conclusions

This paper describes the BC-BSP+ system, a platform for large-scale graph processing based on the BSP model. It implements the main functions mentioned in Pregel. Furthermore, BC-BSP+ can process large graphs with limited resources because it supports disk buffer management. And we design BHP and VCRP partition strategies to improve the overall performance. The performance of VCRP is better than that of BHP. However, the raw graph of VCRP must be crawled by the bread-first search algorithm and vertex IDs must be numbered consecutively. The experiments show that the overall performance of BC-BSP+ is better than that of Hadoop, Giraph and Hama.

In future work, we will analyze the locality of the depth-first search (DFS) algorithm and validate the effect of VCRP for graphs crawled by the BFS algorithm and the DFS algorithm. And we will improve functions of BC-BSP+.

Acknowledgment

This work is partially supported by the Key National Natural Science Foundation of China under Grant No.61033007, the National Natural Science Foundation of China under Grant No.61173028 and No.61272179, the Fundamental Research Funds for the Central Universities under Grant No.N100704001 and No.N110404006, and the Ministry of Education of China and China Mobile Foundation under Grant No.MCM20125021.

References

[1] Apache Hadoop, http://hadoop.apache.org/

[2] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51(1):107-113, 2008.

[3] L.G. Valiant. A Bridging Model for Parallel Computation. Communications of the ACM, 33(8):103-111, 1990.

[4] G. Malewicz, M.H. Austern, A.J.C. Bik, et al. Pregel: a System for Large-Scale Graph Processing. In Proc. of SIGMOD, pages 135-146, 2010.

[5] Apache Incubator Giraph, http://incubator.apache.org/giraph/

[6] Apache Hama, http://hama.apache.org/

[7] Y.B. Bao, Z.G. Wang, Y. Gu, G. Yu, et al. BC-BSP: A BSP-Based Parallel Iterative Processing System for Big Data on Cloud Architecture. First International DASFAA Workshop on Big Data Management and Analytics, 2013.

[8] S. Zhou, Y.B. Bao, Z.G. Wang, et al. BHP: A BSP Model Oriented Graph Data Partition with Load Balancing. Journal of Frontiers of Computers Science and Technology, unpublished.

[9] J.E. Gonzalez, Y. Low, H.J. Gu, et al. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In Proc. of OSDI, pages 17-30, 2012.

[10] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. Bell Syst. Tech. J., 49(2):291-307, 1970.

[11] C.M. Fiduccia and R.M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In Proc. of the 19th Design Automation Conference, pages 175-181, 1982.

[12] B. Krishnamurthy. An Improved Min-Cut Algonthm for Partitioning VLSI Networks. IEEE Transactions on Computers, 33(5):438-446, 1984.

[13] W.E. Donath and A.J. Hoffman. Lower Bounds for the Partitioning of Graphs. IBM Journal of Research and Develoepement, 17(5):420-425, 1973.

[14] M. Girvan and M.E.J. Newman. Community Structure in Social and Biological Networks. In Proc. of the National Acad. of Sci. of the United States of America, 9(12):7821-7826, 2002.

[15] M. Najork and J.L. Wiener. Breadth-First Search Crawling Yields High-Quality Pages. In Proc. of WWW, pages 114-118, 2001.

[16] P. Boldi, B. Codenotti, M. Santini and S. Vigna. UbiCrawler: a Scalable Fully Distributed Web Crawler. Software-Practice & Experience, 34(8):711-726, 2004.

[17] Stanford Large Network Dataset Collection, http://snap.stanford.edu/data/

[18] Using the Wikipedia link dataset, http://haselgrove.id.au/wikipedia.htm

[19] What is Twitter, http://an.kaist.ac.kr/traces/WWW2010.html