Mapreduce Iterative Computation Model Based on Non-Global Parallel and Heartbeat Synchronization

Jun Yu\textsuperscript{a}, Lin Wang\textsuperscript{b}, Mingjie Xu\textsuperscript{c}, Guangxin Zhu\textsuperscript{d}, Min Bu\textsuperscript{e}, Yang Ling\textsuperscript{f} and Haiyang Chen\textsuperscript{g}

State Grid Electric Power Research Institute (SGEPI) Nanjing, China

\textsuperscript{a}yujun@sgepri.sgcc.com.cn, \textsuperscript{b}wanglin18@sgepri.sgcc.com.cn, \textsuperscript{c}xumingjie@sgepri.sgcc.com.cn, \textsuperscript{d}zhuguangxin@sgepri.sgcc.com.cn, \textsuperscript{e}bumin@sgepri.sgcc.com.cn, \textsuperscript{f}842587420@qq.com, \textsuperscript{g}chenhaiyang@sgepri.sgcc.com.cn

Abstract. As the core computing mode of cloud computing, MapReduce is limited by the non-global parallel computing model that it is not easy to perform iterative synchronization algorithm. So it is impossible to perform fine-grained task adjustment based on semantics. This paper breaks through the original computational model of MapReduce, ensures that the existing computational model is compatible with the old MapReduce jobs, and introduces the heartbeat synchronization mechanism that allows the changed state data to interact between Parallel Layers of parallel tasks. This system provides a highly flexible message custom interface, and an adaptive message passing mechanism is designed for different application requirements, which supports algorithms with iterative processing and requirements of data interaction more efficiently. The experimental results on the real large-scale graph dataset showed that compared to the original MapReduce job external processing, the internal Parallel Layer iterative computation model proposed in this paper greatly reduces the processing time of the Mapreduce processing iterative algorithm.

1. Introduction

The rapid development of technologies such as cloud computing, Internet of Things and social networking has greatly enriched the production channels of various massive heterogeneous data. Parallel processing highly scalable massive data is one of the key technologies. MapReduce [1] is a distributed parallel programming model proposed by Google to deal with very massive data sets. It is also the core computing mode of cloud computing. Many research institutes and companies have developed their own massive data parallel processing system based on MapReduce design specification. Hadoop developed by Apache is an open source implementation of MapReduce, and it is also the de facto standard of parallel processing massive data in academia and industry. Hadoop can be easily deployed in a common commercial machine cluster. To simplify the user's parallel programming environment, the highly abstract Hadoop only provides users with limited execution strategies. So in some applications, only high universality and low efficiency method can be adopted. The intention is to make a compromise between ease of use and performance. A large number of algorithms contain obvious iterative processes and there are certain dependencies within the data. In the original
MapReduce, iteration and data interaction can only be supported by invoking MapReduce jobs [2–3] with multiple external chains. Not only does the developer need to actively intervene in the execution process, but it will inevitably introduce a lot of unnecessary duplication costs. Some Hadoop-based systems or MapReduce-like systems, such as HaLoop [4], Twister [5], etc. attempt to complete iterations within the job, in order to reduce the cost of persistence of multiple rounds of intermediate results. But in the implementation of storing the topology of the graph with distributed memory and local cache, this design strategy has certain limitations for large graph processing with massive data.

This inelastic way of solving problems stems from a very important proposition in the MapReduce programming specification: there is no dependency between Mapper and Reducer, and it can be executed independently on different data slices without interaction. This is a non-global "hypothesis" of parallel computing. Parallel problems which can be solved based on this pattern can be decomposed into multiple completely independent parts that can be executed independently and asynchronously. In this ideal mode, there is no communication between asynchronous parallel Mapper and Reducer, and data interaction only depends on the Shuffle processing between Mapper and Reducer. Therefore, some parallel algorithms with iterative process which need the intermediate data interaction can only be invoked by means of a chain of MapReduce jobs to meet the data interaction requirements, and decide when to terminate the call of chain according to the iterative convergence conditions of the application. This paper breaks through the assumption of MapReduce based on non-global parallel computing, and designs a parallel computing calculation model of heartbeat synchronization based on Hadoop. The new computing model decomposes the Map (Reduce) phase into multiple synchronized Parallel Layers. The tasks in the Parallel Layer are asynchronous and highly parallel. The Parallel Layer uses the message passing mechanism to complete the data interaction of tasks. The new computational model uses the approach of graph node-driven to support a wide range of iterative algorithms with information interaction requirements and iterative processing more efficiently, and greatly reduces the unnecessary cost of iterative algorithms in MapReduce original processing mode.

2. Related Work

At present, there are many achievements in the research of distributed parallel programming model and related optimization for large-scale graph algorithms. Both the programming and computational models designed in this paper have been used to improve the processing power of the iterative graph algorithm on the Hadoop platform using the heartbeat synchronization programming specification. But there are many differences in design strategies and implementation details (especially message processing). Compared with the multi-threaded architecture built on Hadoop by Giraph, the computational model designed in this paper adopts an intrusive design pattern, which realizes distributed coordination between working nodes by communication, without manual segmentation and distribution of input data. Therefore, it has advantages in compatibility, ease of use and versatility. The Surfer system provides MapReduce primitives and Propagation primitives, and uses building blocks based on primitive to support large-image algorithms on the cloud. The main goal is providing visual monitoring at runtime, and not involving the specific implementation details of graph features. Lin et al. realized local optimization in the graph algorithm of MapReduce in the literature [6], and proposed optimization techniques such as merging in Mapper, avoided repeated topology transfer and range partitioning, but it is still based on multiple rounds of MapReduce job scheduling. There are also many parallel processing models, such as Apache's HAMA [7] and CMU's GraphLab [8], which both support iteration, but these platforms are oriented to specific problem areas. In addition, there are many researches [9] that hope to improve the processing power of MapReduce by using Message Passing Interface (MPI) in parallel computing, but these studies do not give implementation based on Hadoop platform, and there are defects in features such as fault tolerance, scalability, and robustness [10].
3. Method Design
This paper aims to inherit the original features of Hadoop and introduces the heartbeat synchronization model, uses the message passing mechanism and Parallel Layer synchronization to support the distributed graph algorithm more efficiently.

3.1. Non-global parallel model
Parallel computing can be represented and implemented by using a variety of different parallel programming models. Several parallel tasks can be executed independently on mutually exclusive input set partitions without communication costs. Here is using the non-global parallel formal definition to analyze the MapReduce programming model. As can be seen from Fig.1, the MapReduce job execution plan is divided into two phases, Map and Reduce. It is also called an MR process. The asynchronous parallel tasks (Map1~Map3 or Reduce1~Reduce3) in each phase of the MR process are all running in an ideal mode of non-global parallelism.

In the Map phase, the input data is automatically split into equally sized independent input segments Split0~Split64 (the default value of Split is 64MB, the size is same as the underlying distributed system memory block. This strategy avoids the network cost caused by the data transmission which may be triggered by Split crossing the block boundary). The input fragment is a set of several key-value pairs. The MapReduce parallel processing computing model will distribute Map1~Map3 to the execution node where the input fragment is located according to the data localization optimization strategy. During the execution process, there is no dependency between Map1~Map3 that without communication interaction, which conforms to the non-global parallel formal definition. The intermediate result is a new set of key-value pairs.

In the Reduce phase, the intermediate results generated by Map1~Map3 are input to the intermediate result partitions Part1~Part3 generated by the output key partition operation as Reduce1~Reduce3. During the execution process, there is no dependency between Reduce1~Reduce3, no communication interaction is required, and the non-global parallel formal definition is met. Finally, the output of the Reducer is automatically persisted to the underlying HDFS.

![Non-global parallel model](image)
The Map phase and the Reduce phase are serially synchronized, and there is an implicit synchronization and communication process that is transparent to the user.

The Reducer must wait until the last Mapper has finished executing. However, the Shuffle process in the intermediate data generated by Mapper is executed in an overlapping manner with Mapper. After the end of any Mapper, the Reducer can shuffle the intermediate results, and shorten the length of parallel pipeline processing and improve the processing efficiency. If a parallel application requires multiple MR processes, then the serial process is also synchronized between the MR process and the next MR process. Communication and data interaction also occur only between the Map phase and the Reduce phase in one MR process and between multiple MR processes.

3.2. Heartbeat synchronization

The main challenge of introducing the heartbeat synchronization mechanism is that in the existing MapReduce parallel computing model, message passing between Mapper or Reducer is not supported. In the model of this paper, a parallel job consists of a series of Parallel Layers. Each Parallel Layer constitutes a Phase Parallel. The heartbeat synchronization calculation is mainly composed of three ordered parts: (1) Non-global parallel computing. Tasks within the Parallel Layer execute in independent asynchronous parallelism; (2) Communication. Parallel tasks use message passing mechanism to complete data interaction before the end of Parallel Layer; (3) Synchronize. Synchronize waiting for all parallel tasks in the same Parallel Layer to complete the interaction, then the entire parallel can move to the next Parallel Layer into the next round of parallelism.

It can be seen from Fig.1 that the MapReduce parallel processing computation model supports the BSP model designed in this paper and the original MapReduce parallel processing computation model are consistent in the overall computational model logic. Viewed from a macro perspective, if the entire Map phase of an MR process is regarded as a Parallel Layer. The entire Reduce stage is considered as another Parallel Layer. The Shuffle process between the two phases is considered as the Barrier Grid synchronization and communication process between Parallel Layers. In that way, MapReduce itself is also in line with the heartbeat synchronous calculation that is why the chained MapReduce job scheduling can meet the implementation basis of the iterative operation with interactive requirements.

To ensure that messages can be passed between Parallel Layers in an orderly fashion, the system must rely on an efficient synchronization mechanism between Parallel Layers. In the BSP model, the Synchronous waiting between Parallel Layers is implemented by Barrier Grid [11]. Barrier Grid is a controllable coarse-grained global synchronization mechanism. Barrier Grid can be used to divide a parallel task into multiple consecutive loosely synchronized Parallel Layers, as shown in Figure 1, Parallel Layer0, Parallel Layer1, etc. It ensures that messages are only collected within one Parallel Layer and passed between adjacent subsequent Parallel Layers.

The improved parallel processing computing model in this paper attempts to support Parallel Layer inside the Map or Reduce phase. Based on this design pattern, iterative calculations that previously required external chain calls through multiple MapReduce jobs can now be performed in one MR process. Synchronous execution of multiple Parallel Layers inside the Map phase (or inside the Reduce phase) can be done. Complex message passing control is handled by the new runtime system. Parallel program developers only need to use the original MapReduce program development experience to write more efficient parallel applications under the improved parallel computing model. The improved parallel computing model effectively reduces the cost of introducing external iterations that take up a lot of processing time. However, compared to the cost of the original Hadoop parallel computing model [12], the improved parallel computing model which supporting the BSP model also introduces some new costs.

First, coarse-grained Barrier Grid synchronization makes the overall execution time of a single Parallel Layer sensitive to a single slowest completion task. The problem of abnormal task completion time in non-consistent state can be effectively mitigated by using the Speculative Execution mechanism of Hadoop, using redundant tasks performed by redundant. And because the computation and communication of multiple tasks concurrent in a Parallel Layer are performed in an overlapping
manner, the cost can be further amortized by multiple tasks asynchronously parallel in a Parallel Layer. Second, Barrier Grid synchronization in the new cost model is a potential bottleneck that can cause performance degradation. But, in fact, the Barrier Grid synchronization in the improved model only synchronous transfers the implicit Barrier Grid for Map and Reduce phases between the multiple MR processes in the original model to multiple Barrier Grid synchronization in Map and Reduce phases in one MR process. So in essence there is no new synchronization cost added.

4. Method Implementation

The Precursor collection and the Successor collection represent the dependencies between node-based parallel computing tasks and they are also the path of message passing. Based on the adjacency list, we abstract the transformation of the input key-value pairs of the directed graphs used in the improved computational model by appropriately expanding the deformation, as shown in Table 1.

| Key   | Value       |
|-------|-------------|
| ID    | NodeID      |
|       | Precursor   |
|       | Successor   |
|       | Metadata    |
|       | NodeState   |

Table 1. Key-Value pair for the new computational model

The input key in Fig.2 can be any content (the default key is the offset at the beginning of the line of text). The input value is divided into 5 basic parts: node identifier, direct precursor set, direct successor set, metadata, and current node state. The metadata is the key information of the entity represented by the graph element, including the metadata of the node and the metadata of the edge. For example, in a traffic network diagram, a node represents a road intersection. And the node metadata may include information such as location coordinates, the region, and the node name. The edge represents a connected road segment between nodes, and the edge metadata may include the link length, the section capacity and one-way traffic or not, etc. The current node state is the current state value that is continuously updated with the iterative operation, which represents the intermediate state of the node of this iteration. When the iteration converges, the intermediate state value becomes the result state value of the graph operation.

Each iteration process contains the same processing logic, which includes the following main processing steps:

- Node-driven performance function startup processing. As mentioned above, the graph is divided into input key-value pairs by pre-processing. Each key-value pair represents node-centered computing elements. The user-designed Map processing logic uses the information contained in the key-value pairs according to the application requirements to calculate the intermediate state value of the node in the first iteration.
- Node-driven performance function iterative processing. Accepting and parsing the message passed by the previous Parallel Layer to obtain the new intermediate state value of the neighboring node, applying the user-designed Map processing logic to calculate the intermediate state value of the node in this iteration that on the set of state values in the middle of the new neighboring node and the set of original input key-value pairs representing the graph topology.
- Iterative termination detection. According to the iterative termination condition of the specific application to decide that return to step 2 to continue the iterative process, or stop the iteration to return the calculation result. The system can specify two iteration termination conditions, one is to compare whether the result error between adjacent Parallel Layer is less than the specified threshold, the other is the number of iterations reached the upper limit or not.

It can be seen from the above description that the core content of the computational model of MapReduce-based graphs is node-based asynchronous parallel computing and neighbor-based synchronous messaging.
5. Conclusion
This paper addresses the problem of poor execution performance of current MapReduce-based graph algorithms, an improved parallel computing model supporting Barrier Grid message passing is introduced by introducing a heartbeat synchronization model on the basis of open source Hadoop. By internalizing the iterative process into the Parallel Layer of the Map or Reduce stage, the overhead of multiple rounds of job scheduling is effectively reduced, and provides an efficient computing mode for the design of the distributed large graph algorithm. However, how to solve the problem of degraded dense graph performance with frequent information interaction and the implementation of a wider range of machine learning and clustering algorithms under this platform remains to be further studied.

Acknowledgments
This work was financially supported by the State Grid Corporation of Science and Technology (WBS number: 521104170019).

References
[1] Dean J, Ghemawat S. “MapReduce: Simplified data processing on large clusters” Operating System Design and Implementation (OSDI’04), 2004, pp. 137-150
[2] Bu Y, Howe B, Balazinska M, Ernst MD. HaLoop: Efficient iterative data processing on large clusters // Proceedings of the Conference on Very Large Databases (VLDB’10). Singapore, 2010: 285–296
[3] Ekanayake J, Li H,Zhang B, Gunarathne T, Bae SH, Qiu J, Fox G. “Twister: A runtime for iterative MapReduce” International Symposium on High Performance Distributed Computing, 2010, pp. 810-818.
[4] Chen R, Weng X, He B, Yang M. ”Large graph processing in the cloud” International Conference on Management of Data (SIGMOD’10), 2010, pp.1123–1126.
[5] Lin J,Schatz M. “Design patterns for efficient graph algorithms in MapReduce” Mining and Learning with Graphs (MLG’10), 2010, pp. 78-85
[6] Lin J, Dyer C. “Data Intensive Text Processing with MapReduce”, 2010.
[7] Seo S, Yoon EJ, Kim J, Jin S, Kim JS, Maeng S. “HAMA: An efficient matrix computation with the MapReduce framework” Cloud Computing Technology and Science Conference (CloudCom’10), 2010, pp. 721-726.
[8] Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein J. “GraphLab: A new framework for parallel machine learning” Uncertainty in Artificial Intelligence (UAI’10), 2010.
[9] Hoefler T, Lumsdaine A, Dongarra J. “Towards efficient MapReduce using MPI” European PVM/MPI Users’ Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, 2009, pp. 240-249.
[10] Plimpton Steven J,Devine Karen D. “MapReduce in MPI for large scale graph algorithms”, Parallel Computing, 2011, 37 (9), pp. 610-632.
[11] Albrecht Jeannie, Tuttle Christopher, Snoeren Alex C, Vahdat Amin. “Loose synchronization for large scale networked systems” the Annual Conference on USENIX’06 Annual Technical Conference (ATEC’06), 2006, pp. 28.
[12] Karloff H, Suri S, Vassilvitskii S. “A model of computation for MapReduce” the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’10), 2010, pp. 938-948.