A Tree Algorithm Based on Parallel Cloud Computing Model

Yuqiang Sun, Xianmei Chang, Huanhuan Cai, Xin Gao, Yuwan Gu

Abstract

Cloud computing is the development of parallel computing, distributed computing and grid computing, and with the advancement of cloud computing, how to design efficient distributed tree algorithm is receiving more and more attention. Constrained by parallel assumption, Parallel tree algorithm are not easy to express in MapReduce. Inspired by Bulk Synchronous Parallel model, we propose an enhanced version of Hadoop MapReduce using MPI, which facilitate interactive data communication between supersteps of parallel tasks. Therefore, this paper provides a new idea for tree algorithm with data transition and iterative computation.

Keywords: Tree Algorithms, Cloud Computing, MapReduce, BSP Model, MPI

1. Introduction

At present, the most simple cloud computing technology can find everywhere in network services, such as search engine, net mail and so on. The user need only input simple instruction and they can get a lot of information, under certain conditions, they can even abandoned U disk and other mobile devices, only enter the online office software such as GoogleDocs, officelivespace and so on to create new document, edit content, and directly share URL of document to their friends or bosses, they can directly open a browser to access the URL, and no longer have to worry about data loss event occurs due to damage to the hard disk of the PC[1]. However, the issues of cloud computing is the large-scale data hidden in the background, as the internet cloud computing platform, it will be more widely involved in the massive data processing tasks, and usually the data scale can reach TB or even PB level, therefore, how to deal with such a large amount of data is one of the major problems of cloud computing to face. Due to the very large amount of data, a single machine can not meet the requirements of massive data processing performance and reliability. Large-scale data processing requires a very high reliability for cloud computing, this paper focuses on the optimization of large-scale data processing programming model MapReduce running on tree algorithm.

In recent years, many research institutions and corporate have developed their own massive data parallel processing system based on MapReduce design specification, the Apache Hadoop MapReduce is an open source[2], and is also the massive data parallel processing standards at current academia and industry, hadoop can be easily deployed in general commercial machine in the cluster, to simplify parallel programming environment, the high abstraction hadoop only provides limited enforcement strategy to users, therefore, in some applications especially tree algorithm, it can only take the high versatility of low efficiency method to trade-off the usability and the execution performance. Most of the distributed tree algorithms include the obvious iterative process and there are dependencies within the data, in the original MapReduce, tree algorithm can only through multi-trip external chain call MapReduce jobs[3-6] to support iterative and data interaction, which is not only needed the developers actively intervene in the execution process but also introduced inevitably to a lot of unnecessary duplication. For some jobs with external chain call, each iterative inevitably will produce job start-up cost, the same data serialization and network transmission cost, iteration intermediate result HDPS persistence cost and so on. Some evolutionary system or based on Hadoop MarReduce system, such as Haloop[7], Twister[8]etc attempt to perform iteration in the job internally, and expect to reduce the persistent cost of several rounds of intermediate results, but most of them store the topology of tree with distributed memory and local cache in the realization, and this design strategy has limitations to

* Corresponding Author: Yuwan Gu(Email: gywss0724@126.com)
the tree with massive data processing.

In this paper, parallel computing framework is designed based on Hadoop and support bulk synchronous programming specification\[^{9}\], the improved framework not only enhances the MapReduce existing programming specifications but also ensure compatible with older MapReduce jobs without modifying to run in the new parallel environment, the new framework will make Map(Reduce) stage decomposed into multiple synchronized supersteps, and superstep tasks are highly asynchronous parallel, which will more efficient support the tree algorithm with information exchange and iterative process, and reduce greatly the unnecessary cost of tree algorithm in MapReduce previous processing mode.

2. Parallel Cloud Computing Model — MapReduce

Tree algorithms are designed based on the serial model in the original computer single-core processor time, however, as the semiconductor device enhancement, computer technology and communication network rapid development, high-grade machines with double CPU or 4 CPU can be seen everywhere, especially in the last years, with the emergence of multi-core processors, algorithm researches based on the parallel model have began to sweep the world, PRAM model, APRAM model, BSP model and Log model etc\[^{11}\] have been used in many aspects. In recent years, with the quickly development of cloud computing, its simplicity and convenience bring to people hope and light, MapReduce is just the programming model proposed by Google to simplify distributed system based on cloud computing. Our tree algorithm is just based on this model.

2.1. MapReduce model implementation mechanism

Hadoop’s MapReduce computing architecture realizes the MapReduce programming model proposed by Google engineers, it makes the complex, large-scale cluster parallel computing process highly abstract to the Map and Reduce functions, and the set of data can be broken down into a number of small data sets, which can be handled completely in parallel, the implementation process is showed in Figure 1\[^{12}\] when the MapReduce functions are called by the user program.

![Figure 1. Execution Overview](image)

The whole process is simplified two steps:

1. map( k1, v1) \rightarrow list( k2, v2) ;
2. reduce( k2, list( v2) ) \rightarrow list( v2) .
Figure 1 shows the overall flow of a MapReduce operation. When the user program calls the MapReduce function, the following sequence of action occurs (the numbered labels in Figure 1 correspond to the number in the list below):

1. The MapReduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece (controllable by the user via an optional parameter). It then starts up many copies of the program on a cluster of machines.
2. One of the copies of the program is special—the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.
3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user—defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.
4. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.
5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. The sorting is needed because typically many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used.
6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for this reduce partition.
7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

2.2. Improved MapReduce model

In this paper, map and reduce stage in MapReduce parallel computing framework support message passing, and which helps completing data interaction of parallel tasks. In view of this, the BSP model provides a higher level of parallel abstract.

The design method of BSP computing is: in the BSP model, a parallel operation is composed by a series of supersteps, and each superstep constitutes a phase parallel. The BSP is mainly composed by three sequential parts: (1) each task in the superstep is implemented independently, asynchronous and parallel; (2) communication, parallel task completes data interaction with message passing before the end of superstep; (3) when all the parallel tasks in the same superstep complete data interaction, the whole parallel can move into the next superstep and the next round of phase parallel begin.

Comparing with the figure 2(a) and figure 2(b) we can see that the MapReduce parallel processing framework supporting by BSP model is logical consistency in the whole. From the macroscopic point of view, in a MR process, if we view the whole map process as a superstep and the whole reduce process as another superstep, and the shuffle process between the map and reduce is viewed as synchronization and communication of supersteps, in fact, the MapReduce itself is suitable for the bulk synchronous programming, that is the reality basis why chain MapReduce jobs can meet with the iterative computing with data interaction.

The improved parallel processing framework attempts to support supersteps in the map or reduce, based on this design model, the iterative computing can be completed in a MR processing using multiple supersteps’ synchronous execution in the map (or reduce) stage. However it needs multiple MapReduce jobs to external chain call before. With the improved framework, only using the original MapReduce programming experience, the parallel program developer can prepare to more efficient parallel application, what is more, it effectively reduces the time when carrying out the external iteration, moreover, with the original program experience, people can prepare to the more efficient parallel applications in the improved parallel framework.
3. Tree Algorithm Realization for Cloud Computing Mode

3.1. Computing mode of tree algorithm in BSP model

If we want to carry out the tree operation into the improved parallel framework, the tree must be partitioned into key-value pair sets, so as to suit for the MapReduce processing, usually the storage structure of tree has parents representation, child representation and child-brother representation. Here, we adopt to child representation as the basic representation of tree storage structure, and construct key-value pair for tree in order to suit for the improved parallel framework.

Hypothesis tree $T=(V, E)$ consist of vertex set $V(T)=\{v_1, v_2, v_3, \ldots v_n\}$ and edge (connecting vertices) set $E(T)=\{(v_i, v_j) | i, j=1,2, \ldots n\}$, local topology of tree directly adjacent to a vertex can be represented by the parent node of this vertex and child node set of this vertex.

Definition 1 (child node set) if node $v_i \in V$ is belong to the tree $T=(V, E)$, and if $v_j$ is the direct precursor of $v_i$, then $v_j$ is the child node of $v_i$, then $Cd(v_i) = \{v_j | (v_j, v_i) \in E\}$ is the children node set of $v_i$.

Definition 2 (parent node) if node $v_i \in V$ is belong to the tree $T=(V, E)$, and if $v_j$ is the direct successor of $v_i$, then $v_j$ is the parent node, and expressing $P(v_i)$.

The parent node and child node represent the dependencies between parallel tasks, and it is also the path of the message passing. Base on the child representation, we change properly, and abstract the tree’s key-value pair format in the improved framework. As showed in figure 3.

<table>
<thead>
<tr>
<th>Key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key name</td>
<td>Node identifier (vi)</td>
</tr>
</tbody>
</table>

Figure 3. The tree’s key-value pair expression in the improved parallel computing framework
is a updating state value, and it represents the node intermediate state of this iteration, when the iteration convergence appears, the intermediate state value will be the result state of tree operation.

As following is the main processing steps of tree computing model:

1. Node-driven functions activate. As mentioned above, the tree is prepared to partition into key-value pair set, every key-value pair presents the computing unit which centre around this node, user designs Map processing logic according to application requirement and using the information contained in the key-value pair computes the intermediate state value of the first round iterative node.

2. Node intermediate state interactive communication. With the messing passing mechanism of MPI, intermediate state value of node will be passed according to the node’s adjacency relation in graph, adjacency relation expresses in the node’s parent and child node set.

3. Iterative operation about the node driving function. Receive and parse the messages which are passed by the last supersteps, then obtain the new intermediate state value of adjacency node. With the intermediate state value set of new adjacency node, we can use the map processing logic which is designed by the user to compute the intermediate state value of the current round iteration.

4. Iterative terminated detection. According to the specific iterative termination condition in application decides to return the step 2 to continue to implement the iterative processing or stop iteration to return results. The system can specify two iterative termination conditions, one is if the error between adjacent supersteps is less than the specified threshold, two is if the number of iterations reaches the setting upper limit.

3.2. An example of the tree

In our life, there are many problems can be transformed into a tree to handle, such as the current reality tree used by business management[14], the genealogy of the human society[15], the file folder in our computer and the Baidu search and so on, all problems with hierarchical relationship can be described by tree. Bellowing is a simple example—zoo tour area’ divided as the example to introduce tree’s application in the improved framework.

Here we assume that tree T(V,E) represents zoo tour area, where the node vi∈V represents the area, TP(vi) is the possibility of the node being visited, it is relate to the tree’s topology, for example, its area status(i.e. the child node set), that is to say, the TP value of one area is determined by the other pointing to area TP value, a basic formula can be expressed as:

\[
TP(v_i) = \begin{cases} 
  m_0, & r = 0 \\
  1 - q + q \sum_{v_j \in Cd(v_i)} \frac{PR(v_j)^{r-1} TP(v_j)}{P(v_j)}, & r = 0
\end{cases}
\]

In the formula, \( r \) represents the iterative round, \( m_0 \) is the initial value of node, \( q \) represents the damping coefficient (the detailed description is available in the literature [16]). In the case of given each node to a random initial vale \( m_0 \), TP value of node will tend to converge after multi-round iteration.

First of all, the tree should be transformed as key-value pair set in the MapReduce parallel processing framework, just like two-tuples <From Vertex ID, To Vertex ID>, element of tuple represents the end nodes associated edge, this form requires only two simple MR process which can transform into the general form shown in figure 3, where the state represents the TP value.

Algorithm 1    tree algorithm based on the original MapReduce

Map step:
//the tree is decomposed into key-value pairs as shown in Figure 3  
//\( T=(V, E) \rightarrow (\text{key, value})1 \ldots \) 
//parse:  function of analyzing the key-value pairs 
//count:  function of computing the node’s TP value 
MAP(Key key, Value value) 
vi=parse(value); P(vi)=parse(value);
TP(vi) = count(value);
For vj ∈ P(vi) do
    Output (Key vj, value TP(vi));
End for
Output (Key vi, value value);  // passing topology of the original tree

Reduce step:
REDUCE (Key vj, Value [w1, w2,...])
    neweset ← ∅;
    for wi ∈ [w1, w2,...] do
        if wi ∈ Cd(vj) then
            neweset = neweset + wi;
        else
            value = wi
        end if
    end for
    TP(vj) = count(value, neweset);
    Value = update(value, TP(vj), newset);
Output (key vj, Value value).

Algorithm 2  MapReduce tree algorithm based on supporting BSP model

Map step—setup operation:
i = superstepCounter++;
   // counter for supersteps
   Msgi = new userdefineMessage();    // creating blank user-defined message
   resetInputDataOffset();            // resetting input data access offset

Map step—map operation:
MAP (Key key, Value value)
MPI_Init();
MPI_Comm_size();
MPI_Comm_rank();
MPI_Alltoallv();
vj = parse(value); P(vj) = parse(value);
MPI_Send();
If (stopflag == false)
    If (i == 0)          // the first iteration, execution start processing
        TP(vj) = count(value);
    Else          // performing iterative processing
        TP(vj) = count(value, newValueSeti-1);
    end if
    Msgi.setStateValue(PR(vj));
    else         // terminating iteration, output
        value = update(value, newValueSeti-1);
        MPI_Recv();
    end if

Map step—cleanup operation:
If (compare(Msgi-1, Msgi) > threshold)          // detecting iterative termination
    MPI_BARRIER();
    MPI_Send();        // sending messages, performing interactive processing
    MPI_Recv();
    MPI_Wait();        // in a wait state
    MsgSeti = receive();  // receiving to collect messages after waking up
    NewValueSeti = getNewValue(MsgSeti);      // getting new IP value set carried by the messages
Else

A Tree Algorithm Based on Parallel Cloud Computing Model
Yuqiang Sun, Xianmei Chang, Huanhuan Cai, Xin Gao, Yuwan Gu
Stopflag=true;       //setting iterative termination sign
MPI_Finalize()
End if

Setup and cleanup operations implement only before and after each supersteps computing, map operation deals with the key-value pair of input data slice in cycle.

Comparing algorithm 1 with algorithm 2 we can see that algorithm 1 need multiple iteration in a MR processing, and need add convergence judgment between two MR processing to determine when to terminate iteration, and in addition to the updated node intermediate state value(in this case, the TP value) need to pass among the different nodes through shuffle mechanism, in order to ensure the next iteration implementation, the operation process does not change to the whole tree’s topology(such as P (vi), Cd (vi) and md), and it also needs to repeat transmission and persistence during every MR. Therefore, although MapReduce can perform the tree operation but not easy to express this computing including data interactive dependence and iterative demand. However, the tree computing under the improved parallel framework can make full use of the tree itself topology characteristics and operational characteristics, according to the tree computing intrinsic iterative needs, iteration can be completed by using multiple synchronous supersteps within map step and reduce step, and the changed intermediate state values pass messages between the supersteps. Unchanged tree topology does not need to pass and persistence, thus efficiency is improved effectively.

4. Message Passing based on MPI

4.1. Message passing process of MPI

The MPI standard defines a set of portable programming interface, and designs parallel algorithm of application, parallel computing based on message passing can be achieved when calling these interface, linking the MPI library of corresponding platform. It is because MPI provides a unified interface, the standard is supported widely by various parallel platforms, and it also makes MPI program with good portability At present, MPI has become the most widely used and the most stable parallel programming platform, almost all of the parallel computing environment and popular operating systems (Unix, Linux and Windows NT) are all supported.

MPI parallel programming by calling message passing library function, these functions realize the data exchange between multi-processors, and provide interface between parallel tasks about synchronizing and receiving, sending data\[16\]. Therefore, in this paper, for the computing tasks with data dependence but can not completely independent implement, MPI library can make them pass with internal and order, and make the asynchronous computing task in the same superstep to complete data interactive within supersteps through message passing. The parallel program design flow above the message passing process is shown in figure 4.

![Diagram](image.png)

**Figure 4.** MPI program design flow
First of all, header files mpi.h is included in all MPI programs, then MPI_Init() completes all program initialization work and enter into the system, for the MapReduce parallel computing framework supporting by BSP model, it need to use MPI_Barrier() to divide map stage and reduce stage in MR process into a number of synchronous execution supersets, multiple asynchronous parallel Mapper or Reducer in a superstep will wait each other with MPI_Wait() after sending message, that is to say, through aggregation process, message sent by task in superstep i can be used by task in superstep i+1, namely tasks in supersteps is complete asynchronous, all tasks between supersteps synchronize through MPI_Barrier(), and data interactive with MPI_Semd and MPI_Recv(), finally end of the process with MPI_Finalize().

4.2. MPI cloud computing model

The users refer the procedure or question to cloud computing platform Hadoop, the implementation process as shown in Figure 5.

In the process of cloud computing, Map should execute at the end of data input to avoid moving a large number of data. Map mapping distributes the instructions to multiple workers, Reduce reducing merges the results which computed by Map’s worker, Master regularly detects worker, when a fault occurs, re-execute map, and re-execute reduce if it does not completion. In implementing, task granularity general requires that the small data block is less than or equal to a block size in HDFS, and Map and Reduce are greater than the number of worker so that load balancing and fault recovery.

In the process of cloud computing, Map should execute at the end of data input to avoid moving a large number of data. Map mapping distributes the instructions to multiple workers, Reduce reducing merges the results which computed by Map’s worker, Master regularly detects worker, when a fault occurs, re-execute map, and re-execute reduce if it does not completion. In implementing, task granularity general requires that the small data block is less than or equal to a block size in HDFS, and Map and Reduce are greater than the number of worker so that load balancing and fault recovery.

![Figure5. MPI cloud computing implementation process](image-url)
5. Conclusions

In this paper, aiming at the current problem that tree algorithm based on MapReduce executes in a poor efficiency, through the introduction of BSP model in the open source Hadoop, we achieve an improved parallel computing framework supporting message passing. Through internalizing the iterative process between supersteps in Map or Reduce stage, which effectively reduce the cost as the numerous rounds calling in previous, and provide an efficient computational model for the distributed tree algorithm design.

This paper presents the cloud computing application in the message passing and tree, and gives the cloud computing core algorithm MapReduce and its improved framework.. and verified by example, it provides a new idea for the distributed data of cluster system and massive data processing.

6. Acknowledgment

Supported by The project of general office of Broadcasting and Television (GD10101) and Natural Science Fund in JiangSu (BK2009535) and Natural Science Fund in ZheJiang(Y1100314).

7. References

[16] Chen Yue-yan:Research on Distributed Parallel Computing Based on MPI and MapReduce, Nanjing Normal University, (2010-5-11) .