



Available online at <http://scik.org>

J. Math. Comput. Sci. 2023, 13:15

<https://doi.org/10.28919/jmcs/8300>

ISSN: 1927-5307

THE MAPREDUCE-BASED APPROACH TO IMPROVE THE SHORTEST PATH COMPUTATION

NGUYEN DINH LAU^{1,*}, TRAN THUY TRANG¹, LE THANH TUAN²

¹University of Science and Education, The University of Da Nang, Vietnam

²Information Technology - Cipher Department, Communist Party of Vietnam Central Committee's Office, Vietnam

Copyright © 2023 the author(s). This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract: When building sequential algorithms for problems on the graphic network, the algorithms themselves are not only very complex but the complexity of the algorithms also is very considerab. Thus, sequential algorithms must be parallel to share work and reduce computation time. For above reasons, it is crucial to build parallelization of algorithms in extended graph to find the shortest path. Therefore, a study of algorithm finding the shortest path from a source node to all nodes in the MapReduce architectures is essential to deal with many real problems with huge input data in our daily life. MapReduce architectures processes on (Key, Value) pairs are independent between processes, so multiple processes can be assigned to execute simultaneously on the Hodoop system to reduce calculation time.

Keywords: shortest path; graph; algorithm; Hadoop; MapReduce.

2020 AMS Subject Classification: 68M14, 68Q10.

1. INTRODUCTION

Given extended graph $G = (V, E)$ with a set of vertices V and a set of edges E , where edges can be directed or undirected. Each edge $(u, v) \in E$ is weighted $w(u, v)$. Problem finding the shortest path there are 3 cases:

(a) Problem finding the shortest path from a source node to all nodes (1-n)

*Corresponding author

E-mail address: ndlau@ued.udn.vn

Received October 23, 2023

- (b) Problem finding the shortest path from a source node to destination node (1-1);
- (c) Problem finding the shortest path between every pair of vertices (n-n)

To deal with the problems effectively in computers, it is crucial to build parallel algorithms and the common way we do is to convert the sequential algorithms into parallel algorithms, or convert parallel algorithms into other suitable parallel algorithms which are totally equal to the original algorithms

In paper [3], [4], authors construct parallel all-pairs shortest path algorithm with a MapReduce architecture. Parallel shortest path of an A* algorithm with a MapReduce architecture are implemented in [5], [6], [7], [8]. In paper [9], [10], [11], [12], authors perform parallel data-processing paradigm with Hadoop

In this paper, we operate a parallel algorithm of shortest path algorithm (1-n): the adjacency list based algorithm on MapReduce architecture. In addition to abstract, introduction, conclusion, references, the paper has five algorithms: Algorithm 1: Find the shortest path algorithm; Algorithm 2: BFS algorithm; Algorithm 3: Mapper algorithm; Algorithm 4: Reducer algorithm; Algorithm 5: Random graph create algorithm.

We develop experimental program on Hadoop systems, then offer specific data to evaluate and compare the results of new parallel algorithms with sequential algorithm.

2. HADOOP AND MAPREDUCE

Hadoop has four modules:

- Hadoop Common: These are necessary Java libraries and utilities for other modules to use. These libraries provide file system and OS layer abstraction, and contain Java code to start Hadoop.
- Hadoop YARN: This is framework for managing processes and resources of clusters.
- Hadoop Distributed File System (HDFS): This is distributed file system that provides high-throughput access for data mining applications.
- Hadoop MapReduce: This is YARN-based system for parallel processing of large data sets.

MapReduce:

MapReduce works on a simple principle. Operations take as input a set of key/value pairs and give a set of key/value as output. MapReduce represents computation using just two functions: Map and Reduce. The Map function, takes as input a key/value pair and outputs a set of intermediate key/value pairs. These intermediate key/value pairs are then combined, and intermediate key/value pairs with the same key are passed to the Reduce function. From there, the

IMPROVE THE SHORTEST PATH COMPUTATION

Reduce function calculates on these pairs to give general values rather than the final result. The Map task is performed distributed across storage nodes. The distributed process is done automatically through the input data being broken down. The Reduce task is also distributed through intermediate key/value pairs being grouped into pairs with the same key. MapReduce cluster basically consists of a Master node and Worker nodes. The Master node is responsible for managing and regulating Workers. [9-12, 14,15]

According to Hadoop documentation [6, 9], Hadoop is an Apache open source framework inspired by Google File System [6] [10]. It allows parallel processing on distributed data sets across a cluster of multiple nodes connected under a master-slaves architecture. Hadoop consists of two main components: HDFS [6], [7], [11] and MapReduce [11], [12].

The first component is the Hadoop Distributed File System (HDFS). HDFS is designed to support very large file of data sets. It is also distributed, scalable and fault-tolerant. The Big Data file uploaded into the HDFS is split into block file with specific size defined by the client and replicated across the cluster nodes. The master node (NameNode) manages the distributed file system, namespace and metadata. While the slave nodes (DataNode) manage the storage of block files and periodically report the status to NameNode.

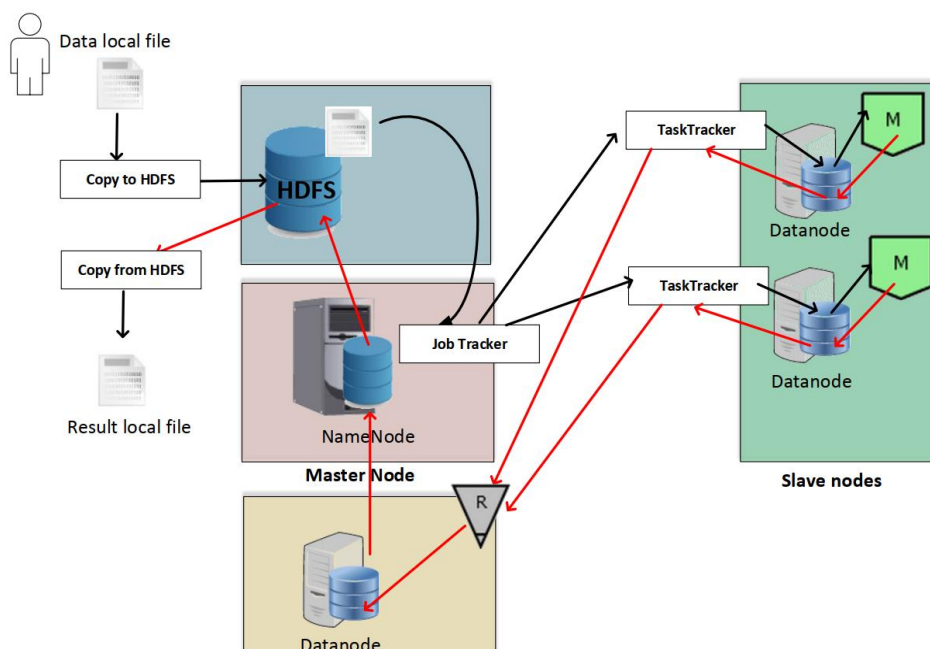


Figure 1. Typical component of one hadoop cluster

As stressed in the figure 1 [5], a hadoop system consists of a master and several slave nodes. The master should consist a NameNode and a JobTracker to manage respectively data storage and

computation jobs scheduling. A slave node could be a DataNode for data storage or/and a TaskTracker for processing computation jobs. A secondary NameNode can be used to replicate data of the NameNode to provide a high quality of service.

The second one is the MapReduce programming model for intensive computation on large data sets in parallel way. To ensure good parallelism, the data input/output needs to be uploaded into the HDFS. In MapReduce framework, the master node works as JobTracker and the slave nodes as TaskTracker. The JobTracker assumes the responsibility and coordinates the job execution. The TaskTracker runs all tasks submitted by the JobTracker.

Figure 2 [8] shows an execution workflow of a MapReduce. The execution workflow is made-up of two main phases:

(a) The *Map phase*, which contains the following steps:

1. The input file is split into several pieces of data. Each piece is called a split or a chunk.
2. Each slave node hosts a map task, called a mapper, reads the content of the corresponding input split from the distributed file system.
3. Each mapper converts the content of its input split into a sequence of key-value pairs and calls the user-defined Map procedure for each key-value pair. The produced intermediate pairs are buffered in memory.
4. Periodically, the buffered intermediate key-value pairs are written to a local intermediate file, called segment file. In each file, the data items are sorted by keys. A mapper node could host several segment files and its number depends on the number of reducer nodes. The intermediate data should be written into different files if they are destined to different reducer nodes. A partitioning function ensures that pairs with the same key are always allocated to the same segment file.

(b) The *Reduce phase*, made of the following steps:

1. On the completion of a map task, the reducer node will pull over its corresponding segments.
2. When a reducer reads all intermediate data, it sorts the data by keys. All occurrences of the same key are grouped together. If the amount of intermediate data is too large to fit in memory, an external sort will be used. The reducer then merges the data to produce for each key a single value.
3. Each reducer iterates over the sorted intermediate data and passes each key-value pair to the reduce function.
4. Each reducer writes its result to the distributed file system

IMPROVE THE SHORTEST PATH COMPUTATION

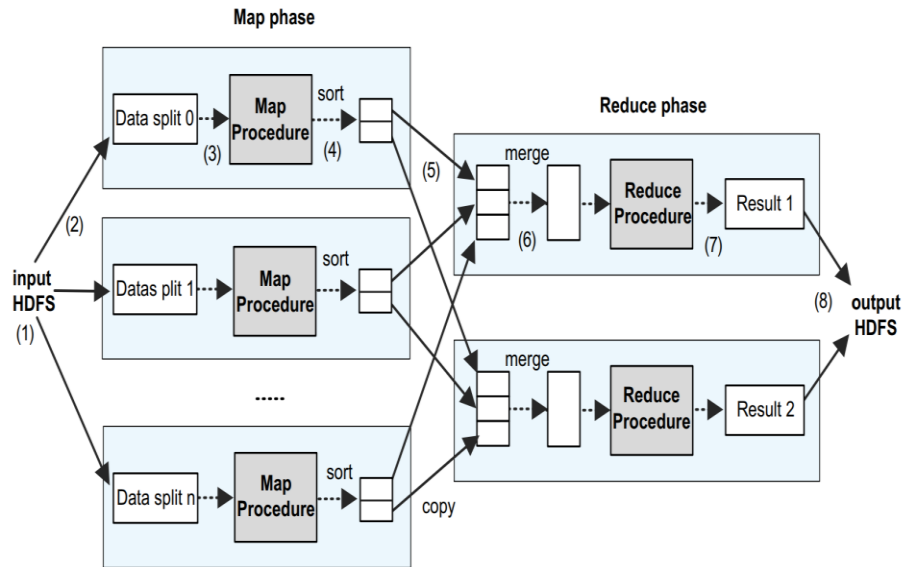


Figure 2. MapReduce execution overview

3. FIND THE SHORTEST PATH ALGORITHM

3.1. Adjacency – List

In many application problems of graph theory, the graph representation as an adjacency list is the most appropriate representation. In this representation, for each vertex v of the graph we store a list of its neighbors, which we will denote by $\{Node\ v, u \in V \mid (v,u) \in E, w(v,u)\}$ with $G=(V,E, w)$.

Example 1: A undirected graph in figure 1 with 12 vertices and its adjacency list are shown in Table 1.

Table 1. Adjacency list

1	2,7	3,5		
2	3,7	4,6		
3	4,11	5,10	6,10	
4	1,4	2,1	8,5	
5	7,20			
6	3,6	7,2	8,15	11,10
7	11,7			
8	3,4	4,18	6,10	9,20
10	8,6			
11	10,15	12,5		

3.2. Find the shortest path algorithm

Find the shortest path algorithm from a vertex to z vertex in the graph as following:

Algorithm 1: Find the shortest path algorithm

Input: A connected graph $G=(V, E, w)$, $w(i, j) \geq 0 \quad \forall (i, j) \in E$ and a specified starting vertex a, z .

Output: $L(z)$ is the length of the shortest path from a to z and the shortest path (if $L(z) < +\infty$)

Step 1. *Initialize:*

Assign $L(a) := 0. \quad \forall x \neq a$ Assign $L(x) := \infty$. Assign $T := V$.

Assign $P(x) := \emptyset, \quad \forall x \in V$ ($P(x)$ is before x vertex on shortest path from a to x).

Step 2. Calculate $m := \min\{L(u) \mid u \in T\}$.

If $m = +\infty$, return "Not have shortest path from a to z ". Finish.

Else if $m < +\infty$, select $v \in T$ so that $L(v) = m$, and assign $T := T - \{v\}$ go to Step 3.

Step 3.

- If $z = v$ then $L(z)$ is shortest distance from a to z . From z tracing back the front vertex-edge, we receive the shortest path as following: assign $z_1 = P(z)$, $z_2 = P(z_1)$, ..., $z_k = P(z_{k-1})$, $a = P(z_k)$. It is inferred that the shortest path is: $a \rightarrow z_k \rightarrow z_{k-1} \rightarrow \dots \rightarrow z_1 \rightarrow z$ Finish.

- Else, if $z \neq v$ then go to step 4.

Step 4. For any $x \in T$ adjacent (post-adjacent) v

If $(L(x) > L(v) + w(v, x))$ then assign $L(x) := L(v) + w(v, x)$ and $P(x) := v$. Back to step 2.

Theorem 1. The complexity of the algorithm is $O(n^3)$ [13]

Example 2: The graph is showed in figure 1. Applying finding the shortest path algorithm, at each vertex symbol $(v, L(v), P(x))$, v is vertex, $L(v)$ is the length of the shortest path to v , $P(x)$ is before x vertex on shortest path to x .

3.3. BFS algorithm

Given the graph $G= (V, E)$. A tree T is called a covering tree or spanning tree of G , if T is a covering subgraph of G . In this algorithm, we denote Q as the queue of vertices, *adjacent x* as the list of vertices adjacent to vertex x . The BFS is described follows.

Algorithm 2: BFS algorithm

Input: A connected graph $G = (V, E)$ and a specified starting vertex v

Output: The breadth first indices of the vertices of G (spanning tree of G) starting with $v=1$, or graph G not connect

1. Initialize: "queue" with the start vertex v , T is graph ($T=(v, \emptyset)$)
2. While ("queue" is not empty AND T is not spanning all vertex) do
3. $x \leftarrow$ remove the first vertex from queue
4. for each vertex y adjacent x do
5. if $y \notin T$ then
6. Begin
7. place edge (x,y) and vertex y on T
8. place y on queue
9. End
10. Endwhile
11. If T is spanning all vertex of G then return T is spanning tree of G
12. Else if queue is empty return graph G not connect

Example 3: The graph is showed in figure 3. Applying BFS algorithm finding spanning tree of G

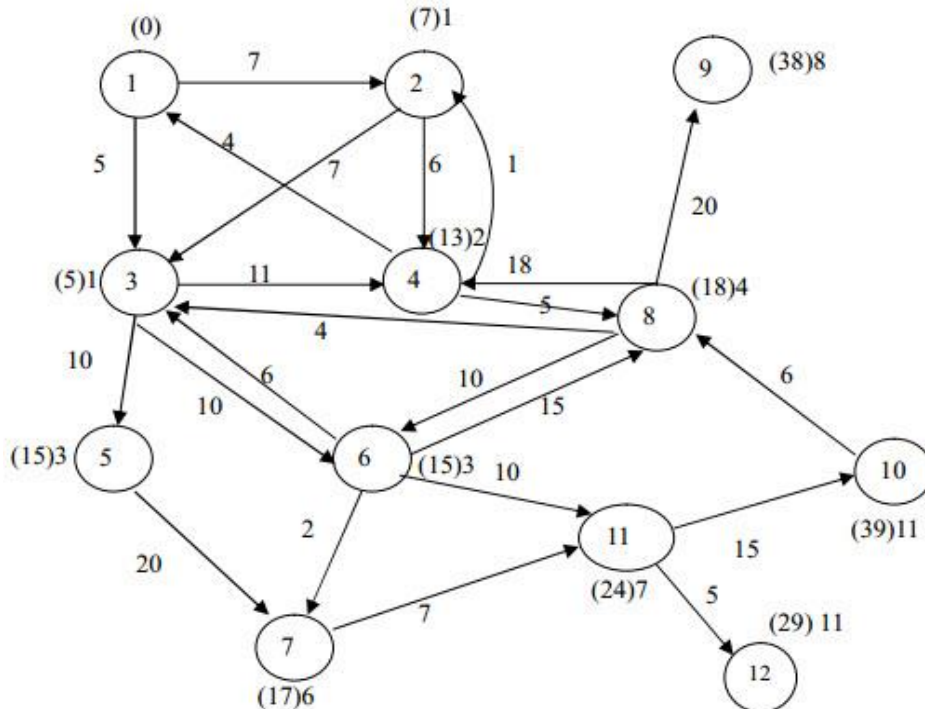


Figure 3. Results finding the shortest path algorithm

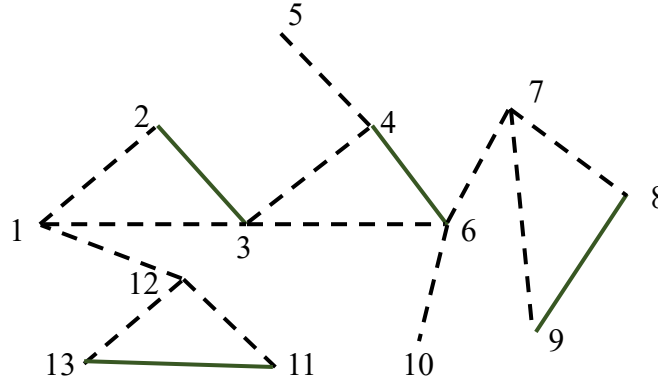


Figure 4. Result is *spanning tree of G* (dashed edges)

4. FIND THE SHORTEST PATH ALGORITHM ON MAPREDUCE

So far, parallel algorithms finding shortest path have been implemented on multi-core processors, with shared external memory. What is new in this approach is to implement the parallel shortest path algorithm on Map Reduce structure. In case of Hadoop framework, to make the algorithm efficient for running it parallel on several machines. Algorithm to find the shortest path on MapReduce bases on Adjacent list

4.1. Proposed MapReduce of find shortest path algorithms

The problem investigated in this section the set of shortest paths from the source node to all other nodes in the graph on MapReduce architecture.

Map stage:

The mapper class takes the entire file an input and parses it line by line.

Reduce stage:

The output of the mapper will be the input to the reducer class. The reducer class takes the minimum of all the path weights and adds it to the adjacency list of the keyId node.

Data representation:

A connected graph $G=(V, E, w)$, $w(i, j) \geq 0 \quad \forall (i, j) \in E$ and a specified source node v .

Initialize: Adjacency – List representation as follows:

$\{Node\ i | \forall i \in V, Node\ Label, Node\ Status\} \text{ TAB } \{Node\ j | \forall j\ Adjacent\ i, w(i, j)\}$

There in:

- *Node Label:* is L , Assign $L(v) := 0. \quad \forall x \neq a\ Assign \quad L(x) := \infty$ (INF)

- *Node Status:* Assign $Node\ Status=Unmarked \quad \forall x \in V$

Example 4: The graph is showed in figure 3, Adjacency–List representation as follows (Table 2).

IMPROVE THE SHORTEST PATH COMPUTATION

Table 2. Initialize: Adjacency – List

$\{Node\ i \forall i \in V, Node\ Label, Node\ Marked\ \}$	$\{Node\ j\ \ \forall j\ \text{adjacent}\ i, w(i,j)\ \}$			
{1,0,UNMARKED}	{2,7}	{3,5}		
{2,INF,UNMARKED}	{3,7}	{4,6}		
{3,INF,UNMARKED}	{4,11}	{5,10}	{6,10}	
{4,INF,UNMARKED}	{1,4}	{2,1}	{8,5}	
{5,INF,UNMARKED}	{7,20}			
{6,INF,UNMARKED}	{3,6}	{7,2}	{8,15}	{11,10}
{7,INF,UNMARKED}	{11,7}			
{8,INF,UNMARKED}	{3,4}	{4,18}	{6,10}	{9,20}
{10,INF,UNMARKED}	{8,6}			
{11,INF,UNMARKED}	{10,15}	{12,5}		

Algorithm 3: Mapper algorithm

Input: (Key, Value) is Adjacent-list

- **Key:** $\{Node\ i|\forall i \in V, Node\ label, Node\ Node\ status, [Path]\}$ Path = The path from the source node to the visiting node

- **Value:** $\{Node\ j\ |\forall j\ \text{Adjacent}\ i, w(i,j)$

Output: (Key, Value)

BEGIN

1. // Find Adjacency – List representation

For (int i=1, i<=|V|, i++)

If (Node label <> inf) and (Node status =Unmarked)

Begin

1.1. Emit (Key, value) = (Key, value),

//But assign Node status=marked in field Node status

1.2. Emit (Key), Key= {Node j| $\forall j$ Adjacent i, Node label, Node status, Path}

/* There in

- Node label= node label+w(i,j)

- Node Status=Unmarked

- Path=Path+"-"+node i

*/

End

Else Emit (key, value) = (key, value)

2. Sort: Sort (Key, Value) with field Node i| $i \in V$

END.

Algorithm 4: Reducer algorithm

Input: (Key, Value), the output of the mapper will be the input to the reducer class

Output: (Key, Value), the output of the reducer will be the input to the mapper class, The MapReduce Job is repeatedly run until all (Key, Value) have Node status = Marked

BEGIN

1. Assign set $S = \{s_1, s_2, \dots, s_n\} = \{1, 2, \dots, |V|\}$

\forall (Key, Value) pairs

Begin

For (int i=1, i<=|V|, i++)

Begin

\forall Node j | j = s_i

Emit (Key, Value)

\\There in

- Node label_{min} = min {Node label_j \forall j = s_i }

- (Key, Value) = {Node k | k = j, Node label_{min}, Node status, [path]}

- If Value = Null then

Value = { \forall l | l adjacent k, w(k, l)}

Else Value = Value

End;

End;

2. If (Node status = Marked) for all nodes then stop, return **final Output**

Else return Output, the output of the reducer will be the input to the mapper class and the MapReduce Job

END.

Theorem 2. The algorithm finding the shortest path from a vertex to many vertices in MapReduce is true.

Proof:

The entire graph is read from the HDFS, transferred from Mappers to Reducers, and then, with updated distance values, written to the HDFS

- Mapper:

Mapper processes a single vertex u , emitting Key have weight Node label $+w(u,v)$ for each vertex v in u 's adjacency list is computed and sent to the Reducers. (step 1.2 of algorithm 3). Once a vertex v has been tested and mapped (mapper), that vertex v is marked as Marked and will not be mapped in subsequent iterations. The same task is assigned $T=T\setminus\{v\}$ in step 2 in algorithm 1.

In iterations next, the Mapper keeps re-computing paths for vertices whose shortest path was already found by statement $Path=Path+"-"+node\ i$ in step 1.2 of algorithm 3.

Like BFS, the program distinguishes between "Marked" and "Unmarked" vertices. Marked vertices are those that could potentially help reduce the distance for another vertex. I define a vertex to be marked if and only if its distance value changed in the previous iteration. The only exception to this rule is source vertex a , which is set to "Marked" before the first iteration. Note that a vertex that was marked in one iteration could become unmarked in the next, and vice versa.

- Reducer:

For every vertex v , no matter if its shortest path was already found in previous iterations or not in Reducer, the statement $(Key, Value) = \{Node\ k|k=j, Node\ labelmin, Node\ status, [path]\}$ in algorithm 4 creates a $(Key, Value)$ pair with a Node label having the smallest value and this is reflected in formula $Node\ labelmin = \min \{Node\ labelj\ \forall j=s_i\}$. So after each iteration of the Reducer function, algorithm 4 will update the shortest path of the vertices for every vertex v , no matter if its shortest path was already found in previous iterations or not, the Reduce function is executed to recompute the shortest distance.

From the above analysis, it can be seen through each iteration, the algorithms will update the longest path, shortest path and mark high-level points as Marked or Unmarked. Thus, Mapreduce will execute iteratively until all the ranks have been marked as "Marked". The final output is the end of the problem.

Mapping and reducing processes on $(Key, Value)$ are independent between processes, so multiple processes can be assigned to execute simultaneously on the Hadoop system to reduce calculation time. ■

The next part is the implementation of Mapper and Reducer for a specific graph.

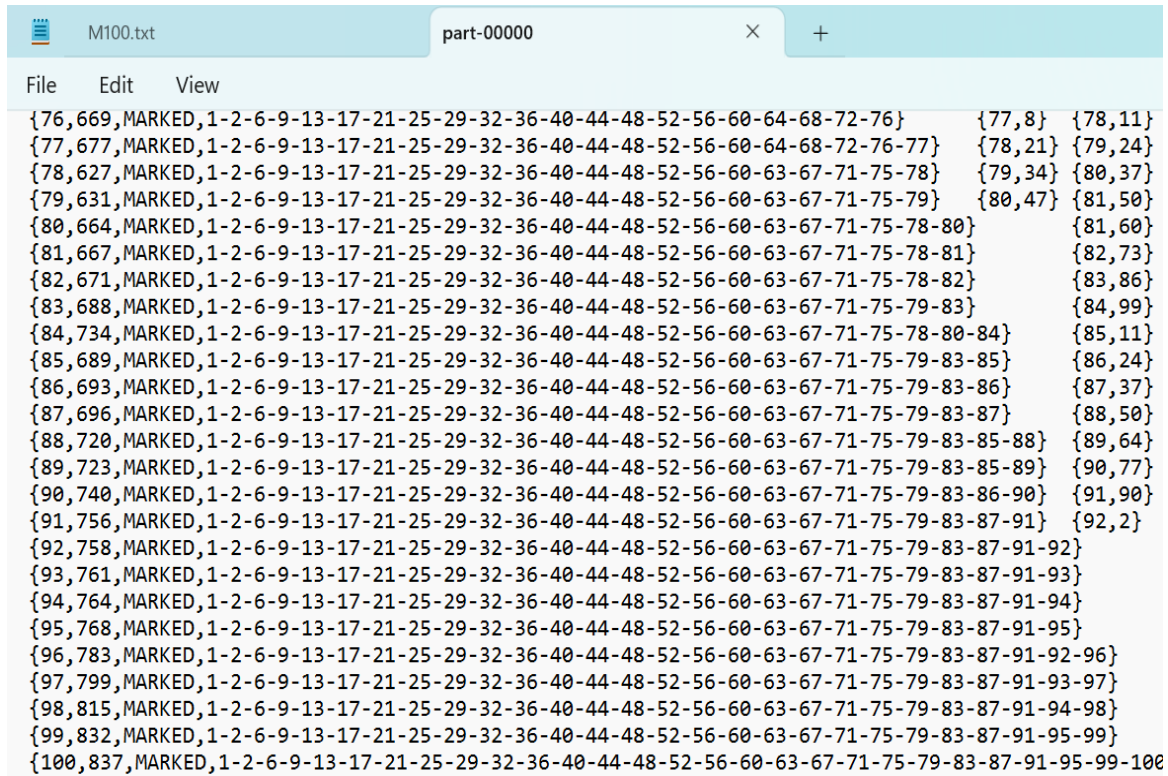
4.2. How to perform MapReduce on a specific graph

Table 3. Input and Output for MapReduce

Mapper Input: Table 2				
Mapper Output (sorted)				
Key	Value			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,UNMARKED,1}				
{2,INF,UNMARKED}	{3,7}	{4,6}		
{3,5,UNMARKED,1}				
{3,INF,UNMARKED}	{4,11}	{5,10}	{6,10}	
{4,INF,UNMARKED}	{1,4}	{2,1}	{8,5}	
{5,INF,UNMARKED}	{7,20}			
{6,INF,UNMARKED}	{3,6}	{7,2}	{8,15}	{11,10}
{7,INF,UNMARKED}	{11,7}			
{8,INF,UNMARKED}	{3,4}	{4,18}	{6,10}	{9,20}
{10,INF,UNMARKED}	{8,6}			
{11,INF,UNMARKED}	{10,15}	{12,5}		
The output of the mapper will be the input to the reducer class				
The output emitted by the reducer is				
Key	Value			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,UNMARKED,1}	{3,7}	{4,6}		
{3,5,UNMARKED,1}	{4,11}	{5,10}	{6,10}	
{4,INF,UNMARKED}	{1,4}	{2,1}	{8,5}	
{5,INF,UNMARKED}	{7,20}			
{6,INF,UNMARKED}	{3,6}	{7,2}	{8,15}	{11,10}
{7,INF,UNMARKED}	{11,7}			
{8,INF,UNMARKED}	{3,4}	{4,18}	{6,10}	{9,20}
{10,INF,UNMARKED}	{8,6}			
{11,INF,UNMARKED}	{10,15}	{12,5}		
The output of the reducer will be the input to the mapper class				
Mapper Output (sorted)				

IMPROVE THE SHORTEST PATH COMPUTATION

Key	Value			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1}	{3,7}	{4,6}		
{3,14,UNMARKED,1-2}				
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{4,13,UNMARKED,1-2}				
{4,16,UNMARKED,1-3}				
{4,INF,UNMARKED}	{1,4}	{2,1}	{8,5}	
{5,15,UNMARKED,1-3}				
{5,INF,UNMARKED}	{7,20}			
{6,15,UNMARKED,1-3}				
{6,INF,UNMARKED}	{3,6}	{7,2}	{8,15}	{11,10}
{7,INF,UNMARKED}	{11,7}			
{8,INF,UNMARKED}	{3,4}	{4,18}	{6,10}	{9,20}
{10,INF,UNMARKED}	{8,6}			
{11,INF,UNMARKED}	{10,15}	{12,5}		
The output of the mapper will be the input to the reducer class				
The MapReduce Job is looped until all nodes are marked and then stopped				
The output emitted by the reducer next iteration is				
Key	Value			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1-2}	{3,7}	{4,6}		
{3,5,MARKED,1-3}	{4,11}	{5,10}	{6,10}	
{4,13,MARKED,1-2-4}	{1,4}	{2,1}	{8,5}	
{5,15,MARKED,1-3-5}	{7,20}			
{6,15,MARKED,1-3-6}	{3,6}	{7,2}	{8,15}	{11,10}
{7,17,MARKED,1-3-6-7}	{11,7}			
{8,18,MARKED,1-2-4-8}	{3,4}	{4,18}	{6,10}	{9,20}
{9,38,MARKED,1-2-4-8-9}				
{10,39,MARKED,1-3-6-7-11-10}	{8,6}			
{11,24,MARKED,1-3-6-7-11}	{10,15}	{12,5}		
{12,29,MARKED,1-3-6-7-11-12}				



```

M100.txt part-00000
File Edit View
{76,669,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-64-68-72-76} {77,8} {78,11}
{77,677,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-64-68-72-76-77} {78,21} {79,24}
{78,627,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-78} {79,34} {80,37}
{79,631,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79} {80,47} {81,50}
{80,664,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-78-80} {81,60}
{81,667,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-78-81} {82,73}
{82,671,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-78-82} {83,86}
{83,688,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83} {84,99}
{84,734,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-78-80-84} {85,11}
{85,689,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-85} {86,24}
{86,693,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-86} {87,37}
{87,696,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87} {88,50}
{88,720,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-85-88} {89,64}
{89,723,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-85-89} {90,77}
{90,740,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-86-90} {91,90}
{91,756,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91} {92,2}
{92,758,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91-92}
{93,761,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91-93}
{94,764,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91-94}
{95,768,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91-95}
{96,783,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91-92-96}
{97,799,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91-93-97}
{98,815,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91-94-98}
{99,832,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91-95-99}
{100,837,MARKED,1-2-6-9-13-17-21-25-29-32-36-40-44-48-52-56-60-63-67-71-75-79-83-87-91-95-99-100}

```

Figure 5. Result on file part-00000

4.3. Experimental results

We Developing experimental programs on Hadoop 3.3.0 systems, then offering specific data to evaluate and compare the results of sequential algorithms or with the other previous parallel algorithms. The experimental results show that such approach achieves significant gain of computational time. The implementation of Mapper and Reducer for a specific graph (M100), it is implemented (figure 5).

Random graphs (Figure 7) are created as our database to test the algorithms. Input: NumNode, Expansion coefficient.

Example 5: Input: NumNode=5, Expansion coefficient=2

Output: Node 1 adjacent Node 2 and Node 3; Node 2 adjacent Node 3 and Node 4; Node 3 adjacent Node 4 and Node 5; Node 4 adjacent Node 5. With $w(\text{Node } i, \text{Node } j)$ is random (see Algorithm 5)

Algorithm 5: Random graph create algorithm**Input:** NumNode, Expansion coefficient**Output:** Graph (Namefile.txt)**BEGIN**

```

ofstream f ("Namefile.txt");
f<<"{"<<<1<<<","0,UNMARKED}";
for(int i=1;i<=Expansion coefficient;i++)
    Begin
        srand(Number);
        int w = rand() % (100 - 2 + 1) + 2;
        Number=Number+1;
        f<<"    {"<<<i+1<<<","<<<w<<<}";
    end;
    f<<endl;
for(int i=2;i<= NumNode;i++)
    Begin
        f<<"{"<<<j<<<","INFINITY,UNMARKED}";
        for(int j=i+1;j<=i+Expansion coefficient;j++)
            if(j<= NumNode)
                Begin
                    srand(Munber);
                    int w = rand() % (100 - 0 + 1) + 0;
                    Number=Number+1;
                    f<<"    {"<<<j<<<","<<<w<<<}";
                End;
            f<<endl;
        End;
    End;
    f.close();

```

END.

Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Browse Directory

/outputXYZ95369871679300

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	Admin	supergroup	0 B	1	128 MB	_SUCCESS
-rw-r--r--	Admin	supergroup	1.42 KB	1	128 MB	part-00000

Hadoop, 2014.

Figure 6. Result on HDFS (file part-00000)

We experimentally random graphs nodes as follows: The graph corresponds to 10000 nodes, 49985 edges (Expansion coefficient=5); 15000 nodes, 74985 edges (Expansion coefficient=5) and 20000 nodes, 99985 edges (Expansion coefficient=5). The simulation result demonstrates that the runtime of parallel algorithms in the MapReduce architectures is better than sequential algorithm.

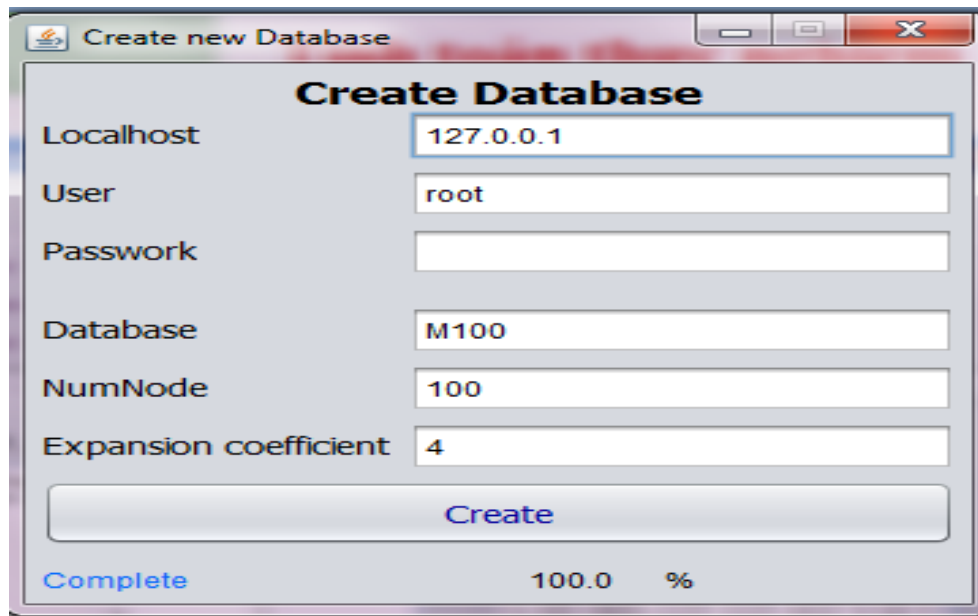


Figure 7. Create database (Random graph create algorithm)

IMPROVE THE SHORTEST PATH COMPUTATION

```

M100.txt
File Edit View
{77,INFINITY,UNMARKED} {80,77} {82,50} {82,57} {85,57}
{80,INFINITY,UNMARKED} {81,60} {82,63} {83,67} {84,70}
{81,INFINITY,UNMARKED} {82,73} {83,76} {84,80} {85,83}
{82,INFINITY,UNMARKED} {83,86} {84,89} {85,93} {86,96}
{83,INFINITY,UNMARKED} {84,99} {85,1} {86,5} {87,8}
{84,INFINITY,UNMARKED} {85,11} {86,15} {87,18} {88,21}
{85,INFINITY,UNMARKED} {86,24} {87,28} {88,31} {89,34}
{86,INFINITY,UNMARKED} {87,37} {88,41} {89,44} {90,47}
{87,INFINITY,UNMARKED} {88,50} {89,54} {90,57} {91,60}
{88,INFINITY,UNMARKED} {89,64} {90,67} {91,70} {92,73}
{89,INFINITY,UNMARKED} {90,77} {91,80} {92,83} {93,86}
{90,INFINITY,UNMARKED} {91,90} {92,93} {93,96} {94,99}
{91,INFINITY,UNMARKED} {92,2} {93,5} {94,8} {95,12}
{92,INFINITY,UNMARKED} {93,15} {94,18} {95,21} {96,25}
{93,INFINITY,UNMARKED} {94,28} {95,31} {96,34} {97,38}
{94,INFINITY,UNMARKED} {95,41} {96,44} {97,47} {98,51}
{95,INFINITY,UNMARKED} {96,54} {97,57} {98,61} {99,64}
{96,INFINITY,UNMARKED} {97,67} {98,70} {99,74} {100,77}
{97,INFINITY,UNMARKED} {98,80} {99,83} {100,87}
{98,INFINITY,UNMARKED} {99,90} {100,93}
{99,INFINITY,UNMARKED} {100,96}
{100,INFINITY,UNMARKED}
Ln 30, Col 14 | 100% | Windows (CRLF) | UTF-8
    
```

Figure 8. Graph with NumNode =100, Expansion coefficient=4

The simulation result demonstrates that the runtime of parallel algorithms on large graph is better than small graph.

Table 4. The run time

Graph	10000 nodes	15000 nodes	20000 nodes
Time	123 mins	141 mins	174 mins

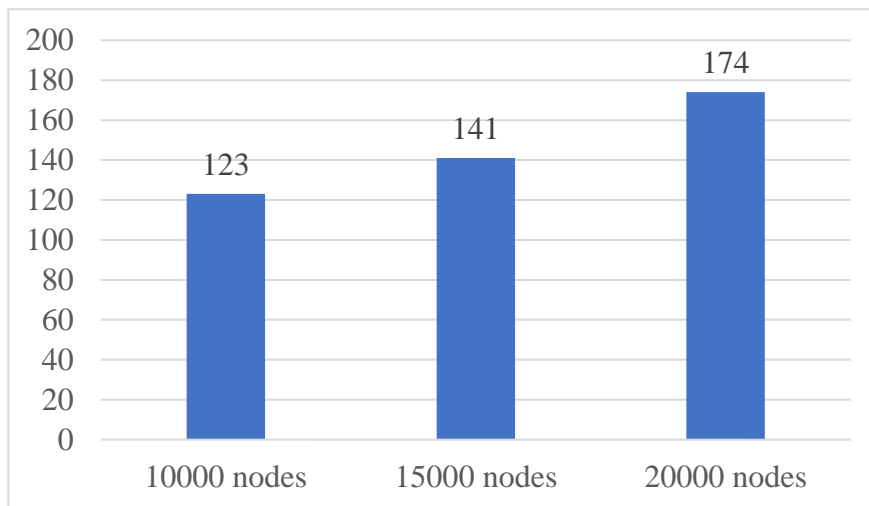


Figure 9. Chart performs the run time of graphs

5. CONCLUSION AND FURTHER WORKS

This paper presents new parallel algorithms (algorithm 3, 4 and 5) based on the actual requirements, proving soundness. In addition, thesis also does parallelization for existing algorithms, then indicates the advantages of the new ones over previous algorithms.

In particularly, this work develops experimental programs on Hadoop 3.3.0 parallel system, then offers specific data to evaluate and compare the results of new parallel algorithms with sequential algorithms

There's a certain novelty value of the algorithms compared to other papers

- The algorithms create a random graph
- The algorithms are generalized
- The algorithms demo on Hadoop 3.3.0 systems
- The algorithms are proven.

As part of future work:

- Proving complexity of the algorithms by MapReduce find shortest path algorithm for a given graph size.
- Applying MapReduce to find shortest path algorithm approach on a real road network.
- Suggesting parallel algorithms on MapReduce architectures for problems: listing combinatorial algorithm or finding the shortest path in extended graph, or find maximum flow on network graph or traveling salesman problem or Genetic Algorithm.

ACKNOWLEDGMENT

We would like to express my sincere thanks to University of Science and Education, The University of Da Nang for their great support for this work through the project code T2023-TN-07.

CONFLICT OF INTERESTS

The authors declare that there is no conflict of interests.

REFERENCES

- [1] S.H. Roosta, Paralell processing and parallel algorithm theory and computation, Springer, (2000).
- [2] R. Sedgewick, Algorithms in C part 5: graph algorithms (third edition), Addison-Wesley, (2000).
- [3] V. Dragomir, All-pair shortest path modified matrix multiplication based algorithm for a one-chip MapReduce

- architecture, U.P.B. Sci. Bull., Ser. C, 78 (2016), 95-108.
- [4] V. Dragomir, G.M. Ştefan, All-pair shortest path on a hybrid Map-Reduce based architecture, Proc. Romanian Acad. Ser. A, 20 (2019), 411–417.
- [5] S. Aridhi, V. Benjamin, P. Lacomme, et al. Shortest path resolution using hadoop, MOSIM'14, Nancy – France, (2014).
- [6] W.Y.H. Adoni, T. Nahhal, B. Aghezzaf, et al. The MapReduce-based approach to improve the shortest path computation in large-scale road networks: the case of A* algorithm, J. Big. Data. 5 (2018), 16.
- [7] W.Y.H. Adoni, T. Nahhal, B. Aghezzaf, et al. MRA*: Parallel and distributed path in large-scale graph using MapReduce-A* based approach, in: E. Sabir, A. García Armada, M. Ghogho, M. Debbah (Eds.), Ubiquitous Networking, Springer International Publishing, Cham, 2017: pp. 390-401.
- [8] S. Aridhi, P. Lacomme, L. Ren, et al. A MapReduce-based approach for shortest path problem in large-scale networks, J. Eng. Appl. Artif. Intell. 41 (2015), 151-165.
- [9] A. Hadoop, Welcome to Apache Hadoop. <http://hadoop.apache.org/>. Accessed 10 Mar 2017.
- [10] S. Ghemawat, H. Gobioff, S.T. Leung, The Google file system, in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, ACM, Bolton Landing NY USA, 2003: pp. 29-43.
- [11] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM. 51 (2008), 107-113.
- [12] V.K. Vavilapalli, A.C. Murthy, C. Douglas, et al. Apache Hadoop YARN: yet another resource negotiator, in: Proceedings of the 4th Annual Symposium on Cloud Computing, ACM, Santa Clara California, 2013: pp. 1-16.
- [13] N.D. Lau, T.Q. Chien, L.M. Thanh, Improved computing performance for algorithm finding the shortest path in extended graph, in: Proceedings of the 2014 international conference on foundations of computer science (FCS'14), USA, (2014), pp. 14-20.
- [14] M. Hena, N. Jeyanthi, A three-tier authentication scheme for kerberized hadoop environment, Cybern. Inform. Technol. 21 (2021), 119-136.
- [15] D. Petrosyan, H. Astsatryan, Serverless high-performance computing over cloud, Cybern. Inform. Technol. 22 (2022), 82-92.