International Journal of Advanced Science and Computer Applications (2024); 3(1): 55-64
DOI: 10.47679/ijasca.v3i1.44
https://ijasca.org/index.php/ijasca

RESEARCH ARTICLE

The MapReduce-based approach to improve the shortest path computation

Nguyen Dinh Lau[1], Tran Thuy Trang[2], Le Thanh Tuan[3]

Published online: 20 November 2023

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 big. Thus, sequential algorithms must be parralled 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.

Keyword: Shortest path, Graph, Algorithm, Hadoop, MapReduce

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)∈ 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)

(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 our computer, the key here is we have to build parallel algorithms. 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 this paper, we are a parallel algorithm of shortest path algorithm (1-n): the adjacency list based algorithm on MapReduce architecture. Besides Abstract, Introduction, conclusion, references, the paper has four algorithms**:** *Algorithm 1: Find the shortest path algorithm*; *Algorithm 2: BFS Algorithm; Algorithm 3: Mapper Algorithm*; *Algorithm 4: Reducer Algorithm.*

We developed experimental program on Hadoop systems, then offering specific data to evaluate and compare the results of new parallel algorithms with sequential algorithm [1,2]

Hadoop and MapReduce

Hadoop has four module:

- **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

[1,2] Department of Information Technology, University of Education and Science, University of Da Nang, Vietnam
[3] Information Techonology - Cipher Department, Communist Party of Vietnam Central

*) corresponding author

Le Thanh Tuan

Email: lethanhtuan.ict@gmail.com

Received 8 November 2023 | Received in revised form 28 January 2024 | Accepted 21 February 2024
Available online 20 March 2024

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 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. [6-12]

Find the shortest path algorithm

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 | (*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		

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) ≥ 0 ∀(i, j) ∈E and a specified strating vertex a, z.
Output: L(z) is the length of the shortest path from a to z and the shortest path (if L(z) <+∞)

Step 1. *Initialize:*
Assign L(a):= 0. ∀ *x* ≠ *a Assign L(x) :=* ∞. *Assign T:= V.*

Assign P(*x) :=* ∅, ∀*x*∈*V (P(x)is before x vertex on shortest path from a to x).*

Step 2. Calculate *m := min{L(u) | u* ∈ *T}.*

If *m = +* ∞, *return "Not have shortest path from a to z". Finish.*

Else if m < + ∞, *select v* ∈ *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*→z_k →z_{k-1}→ ... → z_1→*z Finish.*

- Else, if *z*≠ *v then go to step 4.*

Step 4. *For any x* ∈ *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 2. 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*

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,∅))*
2. *While ("queue" is not empty AND T is not* spanning all vertex) do
3. *x <= remove the first vertex from queue*
4. *for each vertex y adjacent x do*
5. *if y ∉* 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 r*eturn T is* spanning tree of G
12. *Else if queue is empty return* graph G not connect

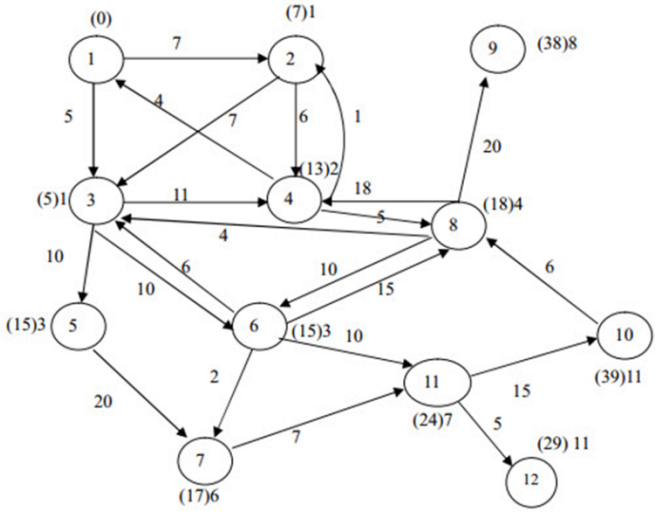


Figure 1. Results finding the shortest path algorithm

Example 3: The graph is showed in figure 2. Applying BFS algorithm finding spanning tree of G

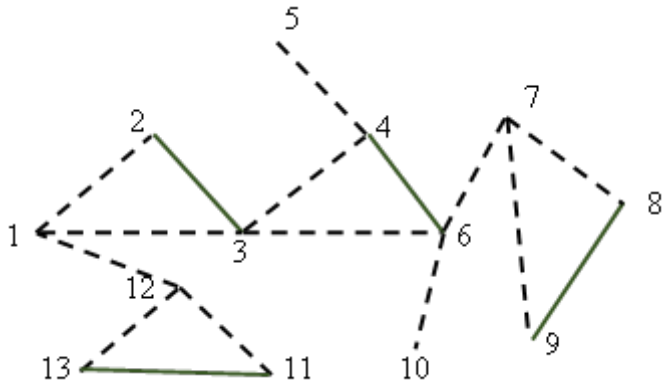


Figure 2. Result is *spanning tree of G (dashed edges)*

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 based on Adjacent list

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) ≥ 0 ∀(i, j) ∈E and a specified source node v.

Initialize: Adjacency – ***List*** **representation as follows:**

{Node i|∀i ∈V*,Node Label,Node Status} TAB {Node j |*∀*j Adjacent i,w(i,j) }*

There in:

- *Node Label: is L, Assign L(v): = 0.* ∀ $x \neq a$ *Assign L(x) :=* ∞ (INF)
- *Node Status: Assign Node Status=Unmarked* ∀ $x \in V$

Example 4: The graph is showed in figure 1, Adjacency–List representation as follows (Table 2)

Table 2: Initialize: Adjacency – List

{ *Node i|*∀i ∈V*,Node Label,Node Marked* }	{Node j | ∀j 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|*∀i ∈V*,Node label,Node Node status, [Path]}*

Path = The path from the source node to the visiting node

- Value: *{Node j |*∀*j 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**|** ∀j *Adjacen* 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∈ *V*

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

- Maper:

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\{v} in step 2 in algorithm 1.

In iterations next, the Maper keeps re-computing paths for vertices whose shortest path was already found by statement Path=Path+"-"+node i in step 1.2 of algorithm 1.

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 ∀j=si}. So after each iteration of the Redecer 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 Hodoop system to reduce calculation time

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

How to perform MapReduce on a specific graph

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)				
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 output emitted by the reducer next iteration is				
Key	**Value**			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1}	{3,7}	{4,6}		
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{4,13,UNMARKED,1-2}	{1,4}	{2,1}	{8,5}	
{5,15,UNMARKED,1-3}	{7,20}			
{6,15,UNMARKED,1-3}	{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)				
Key	**Value**			
{1,0,MARKED,1}	{2,7}	{3,5}		
{1,17,UNMARKED,1-2-4}				
{2,7,MARKED,1}	{3,7}	{4,6}		
{2,14,UNMARKED,1-2-4}				
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{3,21,UNMARKED,1-3-6}				
{4,13,MARKED,1-2}	{1,4}	{2,1}	{8,5}	
{5,15,MARKED,1-3}	{7,20}			
{6,15,MARKED,1-3}	{3,6}	{7,2}	{8,15}	{11,10}
{7,35,UNMARKED,1-3-5}				
{7,17,UNMARKED,1-3-6}				
{7,INF,UNMARKED}	{11,7}			
{8,18,UNMARKED,1-2-4}				
{8,INF,UNMARKED}	{3,4}	{4,18}	{6,10}	{9,20}
{8,30,UNMARKED,1-3-6}				
{10,INF,UNMARKED}	{8,6}			
{11,25,UNMARKED,1-3-6}				
{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 next iteration is				
Key	**Value**			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1}	{3,7}	{4,6}		
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{4,13,MARKED,1-2}	{1,4}	{2,1}	{8,5}	
{5,15,MARKED,1-3}	{7,20}			
{6,15,MARKED,1-3}	{3,6}	{7,2}	{8,15}	{11,10}
{7,17,UNMARKED,1-3-6}	{11,7}			
{8,18,UNMARKED,1-2-4}	{3,4}	{4,18}	{6,10}	{9,20}
{10,INF,UNMARKED}	{8,6}			
{11,25,UNMARKED,1-3-6}	{10,15}	{12,5}		
The output of the reducer will be the input to the mapper class Mapper Output (sorted)				
Key	**Value**			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1}	{3,7}	{4,6}		
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{3,21,UNMARKED,1-3-6}				
{3,22,UNMARKED,1-2-4-8}				
{4,13,MARKED,1-2}	{1,4}	{2,1}	{8,5}	
{4,36,UNMARKED,1-2-4-8}				
{5,15,MARKED,1-3}	{7,20}			
{6,15,MARKED,1-3}	{3,6}	{7,2}	{8,15}	{11,10}
{6,28,UNMARKED,1-2-4-8}				
{7,17,UNMARKED,1-3-6}				
{7,17,MARKED,1-3-6}	{11,7}			

{8,30,UNMARKED,1-3-6}				
{8,18,MARKED,1-2-4}	{3,4}	{4,18}	{6,10}	{9,20}
{9,38,UNMARKED,1-2-4-8}				
{10,INF,UNMARKED}	{8,6}			
{10,40,UNMARKED,1-3-6-11}				
{11,25,UNMARKED,1-3-6}				
{11,24,UNMARKED,1-3-6-7}				
{11,25,MARKED,1-3-6}	{10,15}	{12,5}		
{12,30,UNMARKED,1-3-6-11}				

The output of the mapper will be the input to the reducer class
The output emitted by the reducer next iteration is

Key	**Value**			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1}	{3,7}	{4,6}		
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{4,13,MARKED,1-2}	{1,4}	{2,1}	{8,5}	
{5,15,MARKED,1-3}	{7,20}			
{6,15,MARKED,1-3}	{3,6}	{7,2}	{8,15}	{11,10}
{7,17,MARKED,1-3-6}	{11,7}			
{8,18,MARKED,1-2-4}	{3,4}	{4,18}	{6,10}	{9,20}
{9,38,UNMARKED,1-2-4-8}				
{10,40,UNMARKED,1-3-6-11}	{8,6}			
{11,24,UNMARKED,1-3-6-7}	{10,15}	{12,5}		
{12,30,UNMARKED,1-3-6-11}				

The output of the reducer will be the input to the mapper class
Mapper Output (sorted)

Key	**Value**			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1}	{3,7}	{4,6}		
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{4,13,MARKED,1-2}	{1,4}	{2,1}	{8,5}	
{5,15,MARKED,1-3}	{7,20}			
{6,15,MARKED,1-3}	{3,6}	{7,2}	{8,15}	{11,10}
{7,17,MARKED,1-3-6}	{11,7}			
{8,18,MARKED,1-2-4}	{3,4}	{4,18}	{6,10}	{9,20}
{8,46,UNMARKED,1-3-6-11-10}				
{9,38,MARKED,1-2-4-8}				
{10,40,MARKED,1-3-6-11}	{8,6}			
{10,39,UNMARKED,1-3-6-7-11}				
{11,24,MARKED,1-3-6-7}	{10,15}	{12,5}		
{12,29,UNMARKED,1-3-6-7-11}				
{12,30,MARKED,1-3-6-11}				

The output of the mapper will be the input to the reducer class
The output emitted by the reducer next iteration is

Key	**Value**			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1}	{3,7}	{4,6}		
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{4,13,MARKED,1-2}	{1,4}	{2,1}	{8,5}	
{5,15,MARKED,1-3}	{7,20}			
{6,15,MARKED,1-3}	{3,6}	{7,2}	{8,15}	{11,10}
{7,17,MARKED,1-3-6}	{11,7}			
{8,18,MARKED,1-2-4}	{3,4}	{4,18}	{6,10}	{9,20}
{9,38,MARKED,1-2-4-8}				
{10,39,UNMARKED,1-3-6-7-11}	{8,6}			
{11,24,MARKED,1-3-6-7}	{10,15}	{12,5}		
{12,29,UNMARKED,1-3-6-7-11}				

The output of the reducer will be the input to the mapper class
Mapper Output (sorted)

Key	**Value**			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1}	{3,7}	{4,6}		
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{4,13,MARKED,1-2}	{1,4}	{2,1}	{8,5}	
{5,15,MARKED,1-3}	{7,20}			
{6,15,MARKED,1-3}	{3,6}	{7,2}	{8,15}	{11,10}
{7,17,MARKED,1-3-6}	{11,7}			
{8,18,MARKED,1-2-4}	{3,4}	{4,18}	{6,10}	{9,20}
{8,45,UNMARKED,1-3-6-7-11-10}				
{9,38,MARKED,1-2-4-8}				

{10,39,MARKED,1-3-6-7-11}	{8,6}			
{11,24,MARKED,1-3-6-7}	{10,15}	{12,5}		
{12,29,MARKED,1-3-6-7-11}				
The output of the mapper will be the input to the reducer class The output emitted by the reducer next iteration is				
Key	**Value**			
{1,0,MARKED,1}	{2,7}	{3,5}		
{2,7,MARKED,1}	{3,7}	{4,6}		
{3,5,MARKED,1}	{4,11}	{5,10}	{6,10}	
{4,13,MARKED,1-2}	{1,4}	{2,1}	{8,5}	
{5,15,MARKED,1-3}	{7,20}			
{6,15,MARKED,1-3}	{3,6}	{7,2}	{8,15}	{11,10}
{7,17,MARKED,1-3-6}	{11,7}			
{8,18,MARKED,1-2-4}	{3,4}	{4,18}	{6,10}	{9,20}
{9,38,MARKED,1-2-4-8}				
{10,39,MARKED,1-3-6-7-11}	{8,6}			
{11,24,MARKED,1-3-6-7}	{10,15}	{12,5}		
{12,29,MARKED,1-3-6-7-11}				

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, it is implemented detail in section 4.2

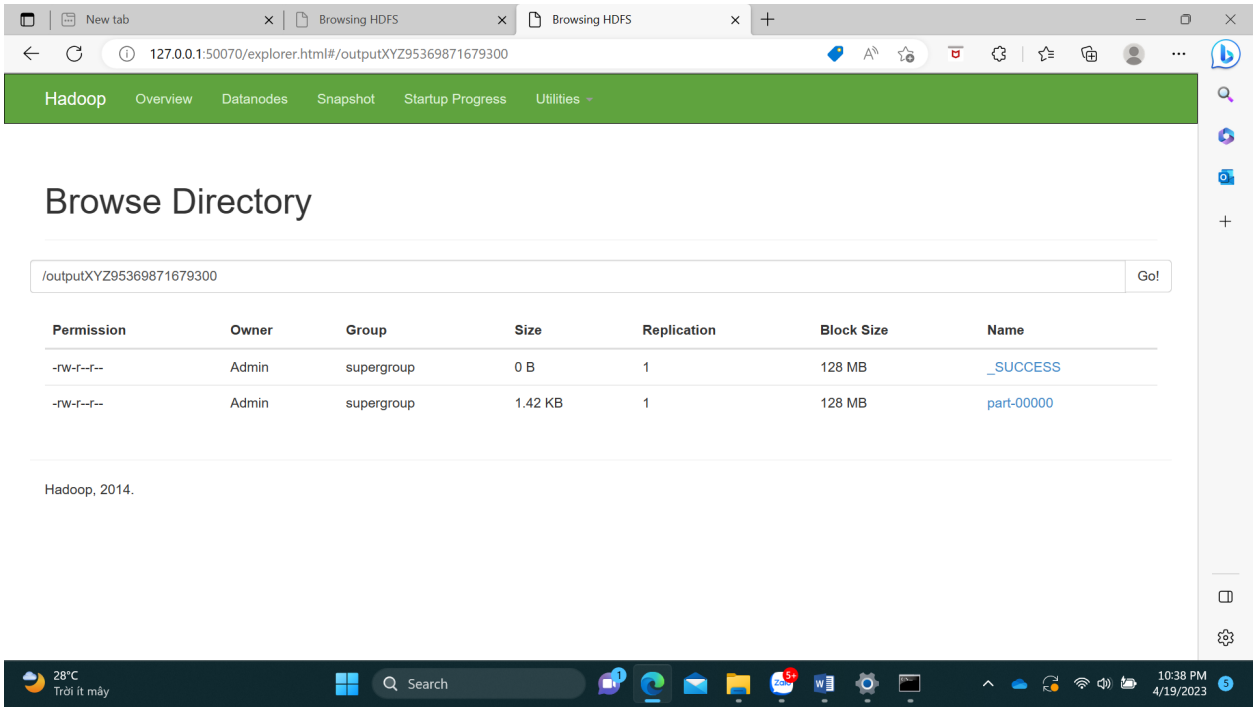


Figure 3. Result on Hadoop

Conclusion and Further Works

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

Developing experimental programs on Hadoop 3.3.0 parallel system, then offering specific data to evaluate and compare the results of new parallel algorithms with sequential algorithms

As part of future work:

- Proving complexity of the algorithms by MapReduce find shortest path algorithm for a given graph size.
- Apply of MapReduce 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.

Acknowledgments. We would like to express my sincere thanks to University of Education and Science, The University of Danang for their great support about this work.

References

Seyed H. Roosta, *Paralell processing and parallel algorithm theory and computation*, Springer, 2000.

Robert Sedgewick, *Algorithms in C part 5: graph algorithms (third edition)*, Addison-Wesley, 2000.

V. Dragomir, *All-pair shortest path modified matrix multiplication based algorithm for a one-chip MapReduce architecture*, U.P.B. Sci. Bull., Series C, *78*, 4, pp. 95-108, 2016.

Voichiţa DRAGOMIR, Gheorghe M. ŞTEFAN, *All-pair shortest path on a hybrid Map-Reduce based architecture,* Proceedings of The Romanian Academy, Series A, the publishing House of the Romanian Academy, Volume 20, Number 4/2019, pp. 411–417.

Sabeur Aridhi, Vincent Benjamin, Philippe Lacomme, Libo Ren, *Shortest path resolutionusing hadoop*, MOSIM'14, Nancy – France, 7/11/2014.

Wilfried Yves Hamilton Adoni, Tarik Nahhal, Brahim Aghezzaf* and Abdeltif Elbyed, *The MapReduce-based approach to improve the shortest path computation in large-scale road networks*: the case of A* algorithm, journal of Big Data, Springer, open access, 2018

Wilfried Yves Hamilton Adoni, Tarik Nahhal, Brahim Aghezzaf, and Abdeltif Elbyed, MRA*: *Parallel and Distributed Path in Large-Scale Graph Using MapReduce-A* Based Approach*, Springer International Publishing AG 2017, pp. 390–401, 2017

Sabeur Aridhi, Philippe Lacomme, Libo Ren, Benjamin Vincent, *A MapReduce-based approach for shortest path problem in large-scale networks,* Elsevier, *Journal of* Engineering Applications of Artificial Intelligence 41 (2015) 151–165

Hadoop, A. Welcome to Apache Hadoop. http://hadoop.apache.org/. Accessed 10 Mar 2017.

Ghemawat S, Gobioff H, Leung ST. *The google file system*. In: ACM SIGOPS operating systems review, vol. 37. New York: ACM; 2003. p. 29–43.

https://doi.org/10.1145/945445.945450.

Dean J, Ghemawat S. *Mapreduce: simplified data processing on large clusters*. Commun ACM. 2008;51(1):107–13.

Vavilapalli VK, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B, Baldeschwieler E, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H. *Apache hadoop YARN*: yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing. Santa Clara: ACM Press; 2013. p. 1– 16.

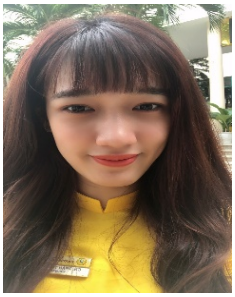
Nguyen Dinh Lau, Tran Quoc Chien, Le Manh Thanh, *Improved Computing Performance for Algorithm Finding the Shortest Path in Extended Graph,* proceedings of the 2014 international conference on foundations of computer science (FCS'14), USA, pp 14-20, 2014

LAU NGUYEN DINH

Born in 1978 in Dien Ban, Quangnam, Vietnam. He graduated from Maths_IT faculty of Hue university of science in 2000. He got master of science (IT) at Danang university of technology and hold Ph.D Degree in 2015 at Danang university of technology. His main major: Applicable mathematics in transport, parallel and distributed process, discrete mathemetics, graph theory, grid Computing and distributed programming.

TRAN THUY TRAN

Born in 1993 in Lien Chieu, Danang, Vietnam. She graduated from IT faculty of University of Education and Science, University of Da Nang, Vietnam in 2015. Her main major: Applicable mathematics, discrete mathemetics, graph theory and linear optimization.



LE THANH TUAN

Born in 1978 in Hai Chau, Danang, Vietnam. He graduated from Maths_IT faculty of Hue university of science in 2002. He got master of science (IT) at Danang university of technology. His main major: Applicable