Research Article

Based Approach to Improve the All-Pair Shortest Path Computation

[1]Lau Nguyen Dinh and [2]Le Thanh Tuan

[1]Department of Information Technology, University of Education and Science, University of Danang, Vietnam
[2]Department of IT-Cipher, Communist Party of Vietnam Central Committee's Office, Vietnam

Article history
Received: 29-07-2023
Revised: 04-12-2023
Accepted: 25-05-2025

Corresponding Author:
Lau Nguyen Dinh
Department of Information Technology, University of Education and Science, University of Danang, Vietnam
Email: ndlau@ued.udn.vn

Abstract: This paper presents the development of a centralized algorithm for finding the shortest path between all pairs of vertices on a graph, utilizing the basic structure of the Floyd-Warshall sequential algorithm. The proposed algorithm leverages multiple processors to compute the shortest path between all vertex pairs, with one processor taking the central role in data management. This central processor divides the workload, assigning tasks to other processors for parallel computation of the shortest paths. The algorithm is implemented in Java and tested on both MapReduce and MPI architectures, running on the computer system at Hanoi National University of Education (ccs1.hnue.edu.vn). The study utilizes traffic road data collected from Da Nang City, Vietnam, providing real-world input for evaluating the algorithm's performance on large-scale datasets. The results demonstrate the efficiency of the algorithm in a distributed environment, highlighting its potential for solving shortest path problems in real-world urban networks.

Keywords: All Pairs, Floyd-Warshall System, MapReduce

Introduction

The Floyd-Warshall algorithm, which finds the shortest paths, has garnered significant attention due to its wide range of practical applications. Given a graph $G(V, E, w)$ with $w(i, j) > 0$ for all edges (i, j), the problem of finding the shortest path between all vertices is addressed by the Floyd-Warshall algorithm (APSP), which is one of the optimization problems on graph networks with broad real-world applications as well as interesting implications in discrete mathematics. Currently, the parallel processing model has been strongly developed to address issues such as deadlock, which the sequential processing model must automatically resolve, including challenges like program execution time, processing speed, memory storage capacity, and large-scale data processing (Roosta, 2000).

The Floyd-Warshall algorithm has a sequential complexity of (n^3), which can be time-consuming when applied to graphs with a large number of vertices and edges. To address this issue, we implement the Floyd-Warshall algorithm on MapReduce and MPI structures to leverage multi-core, multi-processor, and cluster systems. In this study, we conduct experiments on both MapReduce and MPI structures using the cluster at Hanoi National University of Education. Several studies have explored the Floyd-Warshall algorithm in different parallel environments. The emergence of MapReduce and MPI architectures is particularly suitable for handling large-scale data problems.

There are several publications related to this algorithm in various libraries mentioned (Wang *et al.*, 2008; Adoni *et al.*, 2017-2018; Dean & Ghemawat, 2008; Ghemawat *et al.*, 2003; Han *et al.*, 2006; Hena & Jeyanthi, 2021; Kalia & Gupta, 2021; Khezr & Navimipour, 2017; Kulkarni *et al.*, 2015; Lau *et al.*, 2014; Naik *et al.*, 2014; Petrosyan & Astsatryan, 2022; Raj, 2018; Singh & Bawa, 2017; MPI Forum, 1993; Vavilapalli *et al.*, 2013; Wadkar & Siddalingaiah, 2014; Wasi-ur-Rahman *et al.*, 2018; Xiaobing *et al.*, 2013; Yan & Song, 2012). We utilize MapReduce and MPI models to experiment with real datasets.

With the advent of MapReduce and MPI architecture. The MapReduce and MPI architectures are suitable for problems with large datasets. In the study by Dragomir (2016), a parallel algorithm to solve the problem has been proposed, which has proven to reduce parallel computing time compared to sequential computing. However, in the papers by Dragomir *et al.* (2019), the authors require the number of elements that each processor receives to be $\frac{n}{\sqrt{p}} x \frac{n}{\sqrt{p}}$, where p is a numbers slave. This leads to the number of vertices of the input graph and the number of processors must match to divide the elements of the weight matrix for p processors.

To the best of my knowledge, there are very few studies that have specifically tested the shortest path distance calculation between two points using data from Da Nang City on MapReduce and MPI architectures. In this context, the authors propose a new algorithm to evaluate the shortest path problem on a traffic road dataset from Da Nang. The paper aims to address this gap by conducting experiments using MapReduce and MPI architectures on traffic data from Da Nang, leveraging the potential of multi-core, multi-processor, and cluster systems to improve performance and scalability. The structure of the paper will include an introduction to the problem, background of the methodology, framework, then the proposed algorithm on different frameworks, and the experimental setup and results based on real-world traffic data.

Materials and Methods

We surveyed all the actual traffic routes in Da Nang City, Vietnam. The data is very enormous and is always updated regularly. They are attached to the weights at T-junctions and internet sections and that results in the massive actual data to interpret so we tested the algorithm on many different libraries and on many blocks for comparison. The general algorithm in the distributed environment is tested on MPI with the computer cluster of Hanoi National University of Education. The MapReduce algorithm has been tested on many different blocks and Table (1) compares the results of the computation time among the blocks.

Hadoop MapReduce

The Hadoop ecosystem consists of four main modules:

- Hadoop Common: These are the Java libraries and utilities that other modules rely on. They provide the file system and OS abstraction layer and contain the necessary Java code to initiate Hadoop
- Hadoop Yet Another Resource Negotiator (YARN): This is a framework for managing the processes and resources within a cluster
- Hadoop Distributed File System (HDFS)
- Hadoop MapReduce

Input and output data are stored in HDFS. The algorithm is divided into blocks and mapped into a MapReduce function for execution. These two functions will take input as keys and values, then calculate, reduce, and rearrange for further processing.

MPIs

The Message Passing Interface (MPI) is a library of functions designed for use in programs written in C, Fortran, and C++. As its name implies, MPI facilitates communication between processors by passing messages. The MPI program operates with independent processors executing their own instructions in a Multiple Instruction Multiple Data (MIMD) model. These processors do not need to execute identical instructions and communication occurs through MPI communication functions. The MPI application runs as a set of concurrent tasks, with the program consisting of user-written code linked with MPI library functions.

APSP Algorithm

```
Input: Graph G = (V, E, w), V = {1, 2, …, n}, with weight w
Output: Matrix D = [d(l, m)], Matrix P = [p(l, m)]
Method:
    Let $D_0 = [d_0(l, m)]$, with $d_0(l, m) = w(l, m)$
    Let $P_0 = [p_0(l, m)]$, with $p_0(l, m) = m$
        and $p_0(l, m)$ is undefined if no edge from node l to node m.
    Assign t = 0
    For t = 0, …, n
        if t = n, stop. $D = D_n$, $P = P_n$. Else
        t := t + 1
        For l in |V|
            For m in |V|
                if $d_{t-1}[l, m] > d_{t-1}[l, t] + d_{t-1}[t, m]$
                    $d_t[l, m] := d_{t-1}[l, t] + d_{t-1}[t, m]$
                    $p_t[l, m] := p_{t-1}[l, t]$
```

APSP Algorithm on MapReduce

```
Algorithm APSP Mapper

Input: (Key, Value): {i, { ∀j in V, w(i, j) }}
Output: (Key, Value)
For i = 1 to n do
    (Key, Value) = (Key, Value)
For i = 1 to n do
    M = |A|, A = {j | (i, j) ∈ E}
        For j = 1 to m do
            Emit (Key, Value)
            Key: i
            Value_1: {$i_1$, w(m, $j_1$), [Num]}
            Value_{t>1}: {m, w(k, m) + w(j, m), [Num]}
```

“Num” is used to record the number of times the path is received. Num=1,2,3,4,… or NULL

```
Algorithm APSP Reducer

Input: (Key, Value) from Mapper
Output: (Key, Value)
For i = 1 to n ∀ ID ∈ Key
    If (ID == i)
        Key = i;
        Value = {a | (a, i) ∈ E,
            Min{w(i, a)}, [Num]}
        Emit (Key, Value)
Sort by key
```

APSP Algorithm in General Distributed Environment

In the initial matrix D_0, there are always n rows and n columns. The parallel algorithm will divide the number of elements of D_0 by m processors to recalculate the new elements. Specifically, divide n rows equally by m processors.

For example, we have a matrix D_0 consisting of 8 rows and 8 columns. If we recalculate the matrix on 3 processors, the first processor (B_0) will receive 3 rows, the second processor (B_1) will receive 3 rows and the third processor (B_2) will receive 2 rows.

Suppose that at the time of calculating D_4 (k = 4), B_1 will broadcast the data of the 4th row to processors B_0 and B_2 to recalculate new elements. At the kth iteration, the processors will receive the kth row to calculate new elements based on the old elements they hold and repeat k times (Aridhi *et al.*, 2014).

```
Input: Graph G, n = |V|, with weight w for edge (i, j), 1
master and m-1 slaves
Output: Matrix D = [d[i, j]] is the length of all pairs (i, j).
Matrix P = [p(i, j)] used to determine the shortest path.
Step 1: Master executes.
    Master receives the starting matrices $D_0$, $P_0$. The main
    processor divides the initial matrices $D_0$ and $P_0$ for m
    slaves (n rows of $D_0$ and $P_0$ are divided equally).
    We construct $T_i$ (i = 0, …, m-1) as the set of rows that
    slave $B_i$ will receive:
        k = 0;
        For i = 0 to m-1 do
            $T_i$ = {k+1, …, k+$n_i$}
            k = $n_i$ + k;
    Let $D_{0i}$, $P_{0i}$ (i = 0, …, m-1) be the matrices sent by the
    master to slave $B_i$.
    The master sends $T_i$, $D_{0i}$, and $P_{0i}$ to each slave $B_i$.
Step 2:
    m slave receive $D_{0i}$ (i=1, …, m-1>), $P_{0i}$ (i=1, …, m-1)
    and $T_i$ (i=1,..,m-1). So master $B_0$ with receive $D_{01}$, $P_{01}$,
    $T_0$ and slave $B_1$ receive $D_{01}$, $P_{01}$ and $T_1$, slave $B_2$
    receive $D_{02}$, $P_{02}$ and $T_2$,…, slave $B_{m-1}$ receive $D_{0(m-1)}$,
    $P_{0(m-1)}$ and $T_{m-1}$.
Step 3:
    Let the two matrices recalculated at the kth iteration
    (k=1, … n) on slave $B_i$ be $D_{ki}$ and $P_{ki}$ (i=0, …, m-1). So
    at the kth iteration, master $B_0$ will recalculate $D_{k0}$ and
    $P_{k0}$, $B_1$ will recalculate $D_{k1}$ and $P_{k1}$, $B_2$ will recalculate
    $D_{k2}$ and $P_{k2}$, …, $B_{m-1}$ will recalculate $D_{k(m-1)}$ $P_{k(m-1)}$
    For k := 1 to n do
        3.1. $B_i$(i=0, …, m-1) check if k ∈ $T_i$ then $B_i$ will
        send the kth row data in the matrix $D_{(k-1)i}$ (i=0, …,
        m-1) to $B_0$, $B_1$, …, $B_{i-1}$, $B_{i+1}$, …, $B_{m-1}$.
        3.2. If k ∉ $T_i$ then $B_i$>(i=0, …, m-1) will receive the
        kth row data.
        In each partition, each master and slaves performs
        the computation of the elements of the matrix.
        $D_{kz}$ and $P_{kz}$ by $D_{(k-1)z}$ and $P_{(k-1)z}$ (z=0, …, m-1),
        the mean
        $B_i$ (i=0, …, m-1) perform the following tasks at the
        same time:
        For $v_i$ ∈ $T_i$ (i = 0, …, m-1) do call algorithm APSP
Step 4:
    Let the two matrices recalculated at the k = nth iteration
    on $B_i$ be $D_{ni}$ and $P_{ni}$ (i=1, …, m-1), then m-1 master and
    salves will send the two matrices $D_{ni}$ and $P_{ni}$ (i=1, …,
    m-1) to the master.
Step 5:
    Conclusion, the result of matrix D is the union of the $D_{ni}$
    (i=0, …, m-1) and matrix P is the union of the $P_{ni}$ (i=0,
    …, m-1) that the slave sent in step 4. End.
```

Theorem 1: D is the shortest path length matrix.

Proof:

We prove by induction over k the following proposition:

d_k(l,m) is the length of the shortest path among the paths connecting vertex l to m through intermediate vertices [1…k].

- Basic step: Obviously the proposition is true for k = 0.
- Induction step: Suppose the proposition is true for k-1. Consider d_k[l; m].

One of the following two cases will occur:

i. Among the paths connecting vertex l to m through intermediate vertices [1…k] with the shortest length, there exists a path p that does not pass through vertex k. Then p is also the shortest path connecting vertex l to m through intermediate vertices [1…k-1], so by the inductive assumption. d_{k-1}[l; m] = d[p]≤ d_{k-1}[l; k]+ d_{k-1}[k; m]
Therefore, according to the calculation of d_k, we have d_k[l; m]:= d_{k-1}[l; m] = d[p] is the length from node l to node m passing the nodes {1, 2, …, k-1, k}

ii. Every path connecting vertex l with m through the intermediate vertices {1, 2, …, k-1, k} with the shortest length passes through vertex k. Let p = (l, …, k, …, m) be the shortest path connecting vertex l with m through the intermediate vertices {1, 2, …, k-1, k}. Then the segments (l, …, k) and (k, …, m) must also be the shortest paths through the intermediate vertices [1…k-1]. We have: d_{k-1}[l; k]+ d_{k-1}[k; m]=d[p] < d_{k-1}[l; m].
(The last inequality follows from the assumption that all paths connecting vertices l and m through intermediate nodes {1, 2, …, k-1, k} with the shortest length pass through vertex k).
Therefore, according to the calculation of d_k, we have d_k[l; m] = d_{k-1}[l; k]+ d_{k-1}[k; m] = d[p] is the length of the shortest path from node l to node m passing intermediate nodes {1, 2, …, k-1, k}.

Theorem 2: APSP algorithms are correct.

Proof:

In Theorem 1, we have shown that D is the length of the shortest path. Now we only need to prove that the method of constructing the path according to the matrix P, as in the real algorithm, really gives the shortest path.

If d_k(l; m) <+∞, assign:

$$\delta_k\left[l,m\right]=\left[l,l_1,\ldots,l_h,l_{h+1},\ldots,l_s,m\right]$$

Is the path from l to m built based on matrix Pk as follows:

$$l_1=p_k\left(l;m\right),l_2=p_k\left(l_1,m\right),...,l_{h+1}=$$
$$p_k\left(l_h,m\right),...,p_k\left(l_s;m\right)=m$$

We prove by induction on k the following lemma:

Lemma: $\delta_k\left[l;m\right]$ has a length is $d_k(l;\ m)$, that is, the shortest path among the paths connecting vertices l to m through intermediate vertices {1,2,…,k}.

- Basic step: Obviously the proposition is true for k=0
- Inductive step: Suppose the proposition is true for k-1. Consider:
  $\delta_k\left[l;m\right]=\left[l,l_1,\dots,l_h,l_{h+1},\dots,l_s,l_{s+1}=m\right]$

We consider the following two possibilities:

i. $d_k[l;\ m] = d_{k-1}[l;\ m]$:
$p_k[l;\ m] = p_{k-1}[l;\ m] \Rightarrow l_1 = p_{k-1}[l;\ m] \in [1\ldots k-1]$ and $(l;\ l_1) \in E$, we prove $d_k[l_1;\ m] = d_{k-1}[l_1;\ m]$.
Suppose otherwise
$d_k[l_1;\ m] < d_{k-1}[l_1;\ m]$. Then we have $d_{k-1}[l_1;\ m] > d_{k-1}[l_1;\ k] + d_{k-1}[k;\ m]$
$d_{k-1}[l;\ k] + d_{k-1}[k;\ m] \le d_{k-1}[l;\ l_1] + d_{k-1}[l_1;\ k] + d_{k-1}[k;\ m] < d_{k-1}[l;\ l_1] + d_{k-1}[l_1;\ m] \le d_{k-1}[l;\ m]$
$\Rightarrow d_k[l;\ m] < d_{k-1}[l;\ m]$, conflict with $d_k[l;\ m] = d_{k-1}[l;\ m]$
So we have $d_k[l_1;\ m] = d_{k-1}[l_1;\ m]$
From that, similar to above, we can deduce that
$p_k[l_1;\ m] = p_{k-1}[l_1;\ m] \Rightarrow l_2 = p_{k-1}[l_1;\ m] \in [1\ldots k-1]$ and $(l_1;\ l_2) \in E$
h=0, …, m, we have $p_k(l_h;\ m) = p_{k-1}[l_h;\ m]$ and $l_{h+1} = p_{k-1}[l_h;\ m] \in [1\ldots.k-1]$ and $(l_h;\ l_{h+1}) \in E \Rightarrow$ $\delta_k\left[l;m\right]=\delta_{k-1}\left[l;m\right]$ is the shortest path among the paths connecting vertex l to m through intermediate vertices [1…k-1] with length equal to dk-1[l; m] and dk[l; m]= dk-1[l; m], so it is also the shortest path among the paths connecting vertex l to m through intermediate vertices [1…k], with length equal to dk[l; m].

ii. $d_k[l;\ m] < d_{k-1}[l;\ m]$: We have:
$d_k[l;\ m] = d_{k-1}[l;\ k] + d_{k-1}[k;\ m] < d_{k-1}[l_1;\ m]$ and $p_k[l;\ m] = p_{k-1}[l;\ k]$
With $l_1 = p_k[l;\ m] = p_{k-1}[l;\ k] \in \delta_{k-1}\left[l;k\right]$. We prove $d_{k-1}[l_1;\ k] + d_{k-1}[k;\ m] < d_{k-1}[l_1;\ m]$ (*)
Suppose otherwise
$d_{k-1}[l_1;\ k] + d_{k-1}[k;\ m] \ge d_{k-1}[l_1;\ m]$. We have:
$d_{k-1}[l;\ l_1] + d_{k-1}[l_1;\ k] + d_{k-1}[k;\ m] \ge d_{k-1}[l;\ l_1] + d_{k-1}[l_1;\ m] \ge d_{k-1}[l;\ m]$.
Because $l_1 = p_{k-1}[l;\ k] \in [1\ldots k-1]$ lying on the way $\delta_{k-1}\left[l;k\right]$, so $d_{k-1}[l;\ l_1] + d_{k-1}[l_1;\ k] = d_{k-1}[l;\ k]$
$\Rightarrow d_{k-1}[l;\ k] + d_{k-1}[k;\ m] \ge d_{k-1}[l;\ m]$ conflict.
So (*) is true and we have $d_k[l_1;\ m] = d_{k-1}[l_1;\ k] + d_{k-1}[k;\ m]$ and $p_k[l_1;\ m] = p_{k-1}[l_1;\ k] \Rightarrow l_2 = p_k[l_1;\ m] = p_{k-1}[l_1;\ k]\ \delta_{k-1}\left[l;k\right]$
Recursively, we can prove that there exists h such that:
$\left\{l_3,l_4,\dots,l_h=\mathrm{k}\right\}\delta_{k-1}\left[l;k\right]$
That is:
$\delta_{k-1}\left[l;k\right]=\delta_k\left[l;k\right]\delta_k\left[l;m\right]$
Next, we have
$d_k[l;\ k] + d_k[k;\ m] \ge d_k[l;\ m] = d_{k-1}[l;\ k] + d_{k-1}[k;\ m]$
On the other hand $d_k[l;\ k] \le d_{k-1}[l;\ k]$ and $d_k[k;\ m] \le d_{k-1}[k;\ m] \Rightarrow d_k[l;\ k] = d_{k-1}[l;\ k]$ and $d_k[k;\ m] = d_{k-1}[k;\ m]$

According to case (i) (applied to the pair of vertices (k; m)), $\delta_k\left[k;m\right]=\delta_{k-1}\left[k;m\right]$ is the shortest path among the paths connecting vertices k to m through intermediate vertices {1,2,…,k-1}, with length equal to dk(k; m). On the other hand, according to construction $\delta_k\left[l;m\right]$, then $\delta_k\left[l;m\right]$ is the connection of two paths $\delta_k\left[l;k\right]$ and $\delta_k\left[k;m\right]$:$\delta_k\left[l;m\right]=\delta_k\left[l;k\right]\cup\delta_k\left[k;m\right]$ and has length equal to d_k[i,j]. and has a length equal to $\delta_k\left[l;m\right]$ is the shortest path among the paths connecting vertices i to j through intermediate vertices [1…k].

The lemma is proven.

From the lemma, we deduce the required proof because:

$\delta\left[l;m\right]=\delta_n\left[l;m\right]$

Dataset

The input data is examined from traffic roads in Da Nang City, Vietnam with this traffic network data including 140 main traffic intersections, 302 traffic routes, and 1004 source-destination node pairs.

The nodes at T-junctions and intersections in reality have their weights, so the number of nodes for experimentation surges a lot. We also always update the real data because urban data is increasingly expanding for our experimentation.

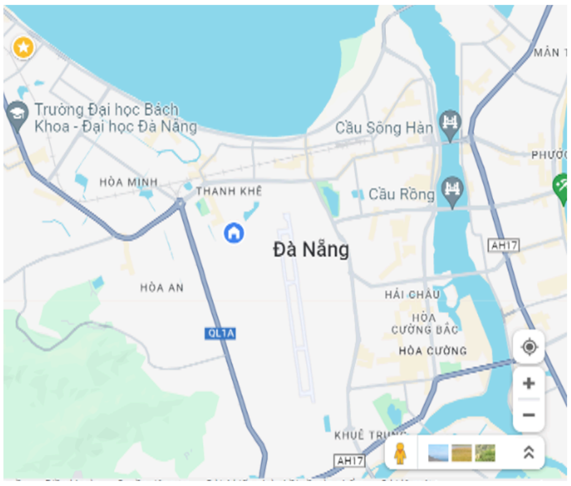
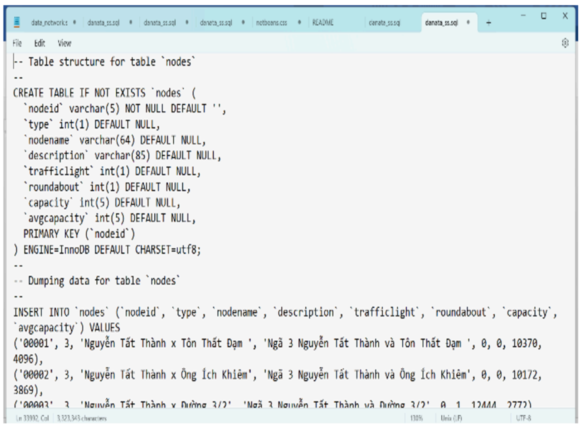


Fig. 1: Map of Da Nang city

The control area is shown on the map in Figure (1) and is modeled using an expanded traffic network as follows:

G (V, E,w). (Table NODES contains V, w)

Figure (2) is the structure of the 'node' table stored in the file danata_ss.sql. Table 'node' contains V (set nodes) and w (capacity of nodes).

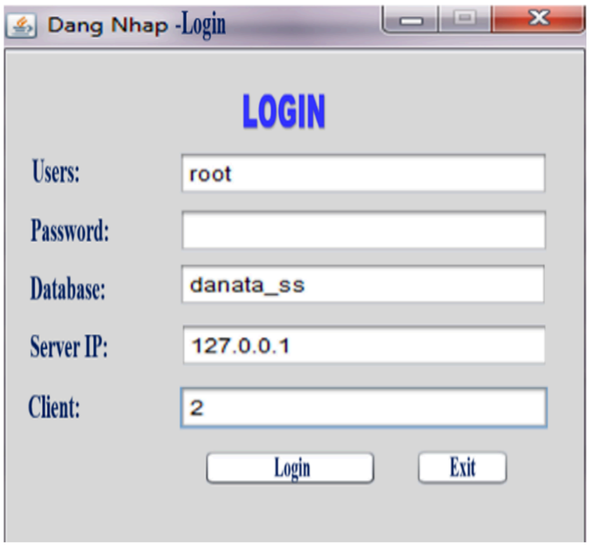


```
-- Table structure for table `nodes`
--
CREATE TABLE IF NOT EXISTS `nodes` (
  `nodeid` varchar(5) NOT NULL DEFAULT '',
  `type` int(1) DEFAULT NULL,
  `nodename` varchar(64) DEFAULT NULL,
  `description` varchar(85) DEFAULT NULL,
  `trafficlight` int(1) DEFAULT NULL,
  `roundabout` int(1) DEFAULT NULL,
  `capacity` int(5) DEFAULT NULL,
  `avgcapacity` int(5) DEFAULT NULL,
  PRIMARY KEY (`nodeid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
--
-- Dumping data for table `nodes`
--
INSERT INTO `nodes` (`nodeid`, `type`, `nodename`, `description`, `trafficlight`, `roundabout`, `capacity`,
`avgcapacity`) VALUES
('00001', 3, 'Nguyễn Tất Thành x Tôn Thất Đạm ', 'Ngã 3 Nguyễn Tất Thành và Tôn Thất Đạm ', 0, 0, 10370,
4096),
('00002', 3, 'Nguyễn Tất Thành x Ông Ích Khiêm', 'Ngã 3 Nguyễn Tất Thành và Ông Ích Khiêm', 0, 0, 10172,
3869),
('00003', 3, 'Nguyễn Tất Thành x Đường 3/2', 'Ngã 3 Nguyễn Tất Thành và Đường 3/2', 0, 1, 12444, 2772)
```

Fig. 2: Table structure for table `nodes`

Results and Discussion

The login interface (Figure 3) allows users to enter their credentials to access the system. The user authentication data is stored in a SQL database named danata_ss. Upon successful login, the system retrieves the relevant records from this database and inputs them into the program for further testing and analysis.



Users:	root
Password:	
Database:	danata_ss
Server IP:	127.0.0.1
Client:	2

Fig. 3: Login interface

For the traffic network in Da Nang city, the running time on the 1, 2, 4, 6, and 8 blocks gives results shown in Table (1), and the level of Speedup acceleration is shown in Figure (4) (T_s is the sequential running time, T_p is the parallel running time, T_s/T_p is the speedup level). This result shows

In Table (1), some blocks are provided to experiment with the program on the MapReduce structure. The results reveal that the experiment on the huge number of blocks needs a shorter time on the same input data set, this is shown in Figure (4). In Figure (4), the speedup is greater when the blocks are larger. This means that the algorithm implemented on the MapReduce structure is much better than the sequential algorithm. The time is significantly reduced when the input data is enormous. In this study, we also experimented with the general algorithm in the MPI structure. The experimental results on MPI also reduce the computing time when the number of processors is larger.

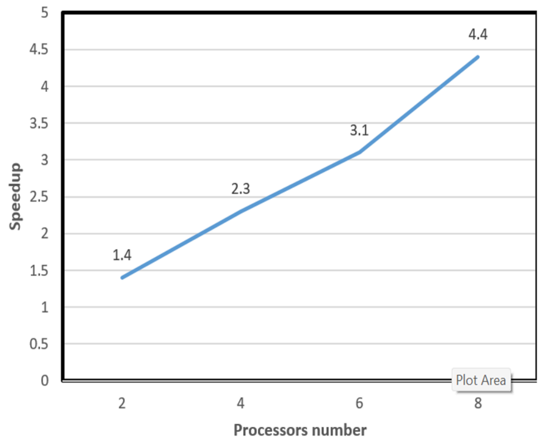


Fig. 4 : The speedup on blocks with data from Da Nang city

Table 1: Time and Speedup

Number block in MapReduce	Run time (minute)	T_s/T_p (Speedup)
1	90.6848	
2	65.5342	1.4
4	40.1279	2.3
6	29.3527	3.1
8	20.5134	4.4

Conclusion

The MapReduce APSP algorithm is presented in detail. We have built a mathematical model for the problem of an expanded traffic network, thereby designing the data structure and collecting data for a large traffic network in Da Nang City. When performing on data with 1 block, the algorithm execution time is too large. Therefore, we build a parallel algorithm to improve the computational performance of the traffic distribution problem for extended traffic networks. The results in Figure (4) show that the parallel algorithm reduces time significantly compared to the sequential algorithm.

Funding Information

The authors have not obtained any financial support or funding to report.

Author's Contributions

Nguyen Dinh Lau: Research algorithms, discrete mathematics, graphs, and parallel algorithms on libraries such as MPI, CUDA, and MapReduce.

Le Thanh Tuan: Research on AI algorithms, machine learning, and practical applications.

Ethics

The authors affirm that there are no conflicts of interest and that the work complies with the ethical policies of the Journal of Computer Science.

References

Adoni, W. Y. H., Nahhal, T., Aghezzaf, B., & Elbyed, A. (2017). MRA*: Parallel and Distributed Path in Large-Scale Graph Using MapReduce-A* Based Approach. *Ubiquitous Networking*, 390-401. https://doi.org/10.1007/978-3-319-68179-5_34

Adoni, W. Y. H., Nahhal, T., Aghezzaf, B., & Elbyed, A. (2018). The MapReduce-based approach to improve the shortest path computation in large-scale road networks: the case of A* algorithm. *Journal of Big Data*, *5*(1), 16. https://doi.org/10.1186/s40537-018-0125-8

Aridhi, S., Lacomme, P., Ren, L., & Vincent, B. (2015). A MapReduce-based approach for shortest path problem in large-scale networks. *Engineering Applications of Artificial Intelligence*, *41*, 151-165. https://doi.org/10.1016/j.engappai.2015.02.008

Dean, J., & Ghemawat, S. (2008). MapReduce. *Communications of the ACM*, *51*(1), 107-113. https://doi.org/10.1145/1327452.1327492

Dragomir, V. (2016). All-pair shortest path modified matrix multiplication based algorithm for a one-chip MapReduce architecture. *UPB Scientific Bulletin Series*, *78*(4), 95-108.

Dragomir, V., & Ștefan, G. M. (2019). All-pair shortest path on a hybrid Map-Reduce based architecture. *Proceedings Of The Romanian Academy*, *20*(4), 411-417.

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. *ACM SIGOPS Operating Systems Review*, *37*(5), 29-43. https://doi.org/10.1145/1165389.945450

Han, S.-C., Franchetti, F., & Püschel, M. (2006). Program generation for the all-pairs shortest path problem. *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, 222-232. https://doi.org/10.1145/1152154.1152189

Hena, M., & Jeyanthi, N. (2021). A Three-Tier Authentication Scheme for Kerberized Hadoop Environment. *Cybernetics and Information Technologies*, *21*(4), 119-136. https://doi.org/10.2478/cait-2021-0046

Kalia, K., & Gupta, N. (2021). Analysis of hadoop MapReduce scheduling in heterogeneous environment. *Ain Shams Engineering Journal*, *12*(1), 1101-1110. https://doi.org/10.1016/j.asej.2020.06.009

Khezr, S. N., & Navimipour, N. J. (2017). MapReduce and Its Applications, Challenges, and Architecture: a Comprehensive Review and Directions for Future Research. *Journal of Grid Computing*, *15*(3), 295-321. https://doi.org/10.1007/s10723-017-9408-0

Kulkarni, D., Sharma, N., Shinde, P., & Varma, V. (2015). Parallelization of Shortest Path Finder on GPU: Floyd-Warshall. *International Journal of Computer Applications*, *118*(20), 1-4. https://doi.org/10.5120/20858-3547

Lau, N. D., Thanh, L. M., & Quoc, C. T. (2014). Improved Computing Performance for Algorithm Finding the Shortest Path in Extended Graph. *Proceedings of the International Conference on Foundations of Computer Science (FCS)*, 1-7.

MPI Forum. (1993). MPI: a message passing interface. *Supercomputing '93: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, 878-883. https://doi.org/10.1145/169627.169855

Naik, N. S., Negi, A., & Sastry, V. N. (2014). A review of adaptive approaches to MapReduce scheduling in heterogeneous environments. *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 677-683. https://doi.org/10.1109/icacci.2014.6968497

Petrosyan, D., & Astsatryan, H. (2022). Serverless High-Performance Computing over Cloud. *Cybernetics and Information Technologies*, *22*(3), 82-92. https://doi.org/10.2478/cait-2022-0029

Raj, P. (2018). Chapter Seven - The Hadoop Ecosystem Technologies and Tools. *A Deep Dive into NoSQL Databases: The Use Cases and Applications*, *109*, 279-320. https://doi.org/10.1016/bs.adcom.2017.09.002

Roosta, S. H. (2000). Data Flow and Functional Programming. *Parallel Processing and Parallel Algorithms: Theory and Computation*, 411-437. https://doi.org/10.1007/978-1-4612-1220-1_9

Singh, H., & Bawa, S. (2017). A MapReduce-based scalable discovery and indexing of structured big data. *Future Generation Computer Systems*, *73*, 32-43. https://doi.org/10.1016/j.future.2017.03.028

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., & Baldeschwieler, E. (2013). Apache Hadoop YARN. *SOCC '13: Proceedings of the 4th Annual Symposium on Cloud Computing*, 1-16. https://doi.org/10.1145/2523616.2523633

Wadkar, S., & Siddalingaiah, M. (2014). Monitoring Hadoop. *Pro Apache Hadoop*, 203-215. https://doi.org/10.1007/978-1-4302-4864-4_9

Wang, H., Tian, L., & Jiang, C.-H. (2008). Practical Parallel Algorithm for All-Pair Shortest Path Based on Pipelining. *Journal of Electronic Science and Technology*, *6*(3), 329-333.

Wasi-ur- Rahman, Md., Islam, N. S., Lu, X., Shankar, D., & Panda, D. K. (DK). (2018). MR-Advisor: A comprehensive tuning, profiling, and prediction tool for MapReduce execution frameworks on HPC clusters. *Journal of Parallel and Distributed Computing*, *120*, 237-250. https://doi.org/10.1016/j.jpdc.2017.11.004

Xiaobing, X., Chao, B., & Feng, C. (2013). An Insight into Traffic Safety Management System Platform based on Cloud Computing. *Procedia - Social and Behavioral Sciences*, *96*, 2643-2646. https://doi.org/10.1016/j.sbspro.2013.08.295

Yan, Z., & Song, Q. (2012). An Implementation of Parallel Floyd-Warshall Algorithm Based on Hybrid MPI and OpenMP. *ICECC '12: Proceedings of the 2012 International Conference on Electronics, Communications and Control*, 2461-2466. https://doi.org/10.5555/2417502.2418392