International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 12 (2015) pp. 30203-30217

http://www.ripublication.com

Shortest Path Algorithms Using Map Reduce For Automatic Vehicle Location Systems

N. Minni
Department of Computer Science
Avvaiyar Govt College for Women, Karaikal, Puducherry, India
minnimca@yahoo.co.in
N. Rehna
Department of Computer Science
SSS Shasun Jain College, T. Nagar, Chennai, India
rehnamca@yahoo.com

Abstract

Map Reduce is a simple data-parallel programming model designed for scalability and fault-tolerance and for processing and generating large data sets. It was initially created by Google [1] for simplifying the development of large scale web search applications in data centers and has been proposed to form the basis of a 'Data center computer' [5]. Many real world tasks are expressible in this model. In this paper, we introduce a Shortest Path Algorithm for Automatic Vehicle Location Systems (AVL) using MapReduce technique. This algorithm computes the shortest route from the current location to all other cities in the global network. Since the data are distributed in the cloud We've proposed to develop the shortest path algorithm using map reduce technique which processes the data parallely. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

The implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable. A typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use.

Keywords: Cloud Computing, Map Reduce, Map Reduce Shortest Path (MRSP) Algorithm , Adjacency List, Automatic Vehicle Location (AVL) systems. Global Positioning System (GPS)

Introduction

A) Automatic Vehicle Location System (AVL)

Every wonder what the world would be like with little to no congestion, improved safety and efficiency on all roads? Automated Vehicle Location systems and shortest path algorithms are steps in this direction. Vehicles equipped with shortest path algorithms along with AVL technologies are faster, safer, and more efficient than those without [4]. In the not to distant future most vehicles will be equipped with GPS tracking technology, but the real market change will be in the access and permission to use shortest path algorithms and AVL systems [6].

A current debate is starting on who should be allowed to track, what to track, and access to shortest path algorithms

B) Shortest Path Algorithms

A shortest path algorithm is a program, or set of directions that can be executed to provide the shortest path between locations given certain conditions and paths. Conditions such as traffic density, speed of travel, and others, as well as geographic obstacles can be factored in to help the algorithm execute and display the shortest path[9].

The latest algorithms being developed adjust to conditions and dynamically execute to give new shortest paths based not only on distance, but also on time. Shortest path algorithms are essential to improving efficiency and safety in many areas network navigation such as highway systems and disaster clean-up.

Algorithms are used for determining time dependent shortest paths in highway systems given variations in traffic density. Algorithms are used in connection with GPS tracking technology to provide shortest paths for trucks to follow to specific dumping areas when cleaning up after a natural or terrorism related disaster [6].

C) Map Reduce Technique

The MapReduce distributed data analysis framework model introduced by Google [1] provides an easy-to-use programming model that features fault tolerance, automatic parallelization, scalability and data locality-based optimizations. Due to their excellent fault tolerance features, MapReduce frameworks are well-suited for the execution of large distributed jobs in brittle environments such as commodity clusters and cloud infrastructures [1].

The MapReduce framework is a programming paradigm for designing parallel and distributed algorithms [2]. It provides a simple programming interface that is specifically designed to make it easy for a programmer to design a parallel program that can efficiently perform a data-intensive computation. Moreover, it is a framework that allows for parallel programs to be directly translated into computations for cloud computing environments and server clusters [8].

Using this model we've developed a MapReduce algorithm for finding shortest path[3]. Our algorithm computes the shortest paths to all the cities from the current location.

Section 2 describes the basics and working of cloud computing. Section 3 tells the need for cloud computing and MapReduce. In Section 4, we describe the MapReduce framework on cloud computing environment. Section 5 briefly explains the graph representations. Section 6 introduces the concept of parallel breadth first search technique. Section 7 clearly explains the usage of MapReduce in the proposed work. In section 8, MRSP algorithm is developed using MapReduce which finds the shortest paths for AVL systems. Section 9 discusses about the advantages of MapReduce. Section 10 discusses the conclusions and future works.

Cloud Computing

Cloud computing refers to the provision of computational resources on demand via a computer network. Cloud computing offers computer application developers and users an abstract view of services that simplifies and ignores much of the details and inner workings. A provider's offering of abstracted Internet services is often called "The Cloud" [5]. Cloud computing is computation, software, data access, and storage services that do not require end-user knowledge of the physical location and configuration of the system that delivers the services.

How it works

Instead of installing a suite of software for each computer, we'd only have to load one application. That application would allow workers to log into a Web-based service which hosts all the programs the user would need for his or her job. Remote machines owned by another company would run everything from e-mail to word processing to complex data analysis programs. It's called cloud computing, and it could change the entire computer industry [5].

In a cloud computing system, there's a significant workload shift. Local computers no longer have to do all the heavy lifting when it comes to running applications. The network of computers that make up the cloud handles them instead. Hardware and software demands on the user's side decrease. The only thing the user's computer needs to be able to run is the cloud computing system's interface software, which can be as simple as a Web browser, and the cloud's network takes care of the rest.

Need For Cloud Computing and Mapreduce

Cloud computing has been driven fundamentally by the need to process an exploding quantity of data in terms of exabytes as we are reaching the Zetta Bytes Era. One critical trend shines through the cloud is Big Data.

Indeed, it’s the core driver in cloud computing and will define the future of IT. When a company needed to store and access more data they had one of two choices. One option could be to buy a bigger machine with more CPU, RAM, disk space etc. This is known as scaling vertically. Of Course, there is a limit to how big of a machine we can actually buy and this does not work when we start talking about internet scale. The other option would be to scale horizontally. This usually meant contacting some database vendor to buy a bigger solution. These solutions do not come cheap and

therefore required a significant investment. Today the source of data generated not only by the users and applications but also "machine- generated" and such data is exponentially leading the change in the big data space. Dealing with big datasets in the order of terabytes or even petabytes is a challenging. In cloud computing environment a popular data processing engine for big data is Hadoop-MapReduce due to ease-of-use, scalability and failover properties. [10]

Basics of Map Reduce

MapReduce is a framework for processing huge datasets on certain kinds of distributable problems using a large number of computers, collectively referred to as a cluster or as a grid [7].

A. Programming Model

The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: Map and Reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function. The Reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory [1].

In Fig. 1, mappers are applied to all input key-value pairs, which generate an arbitrary number of intermediate key-value pairs. Reducers are applied to all values associated with the same key. Between the map and reduce phases lies a barrier that involves a large distributed sort and group by [3].

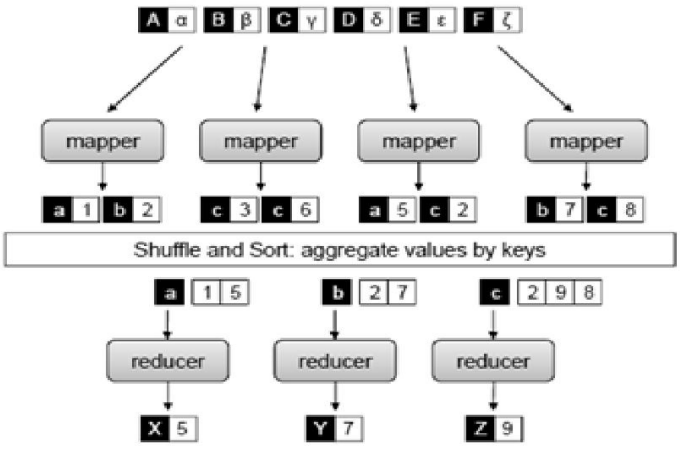


Figure 1: Simplified View of Map Reducee

B. The MapReduce Framework

The Map and Reduce functions of MapReduce are both defined with respect to data structured in (key, value) pairs. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain.

$$Map(k1,v1) \rightarrow list(k2,v2)$$

The Map function is applied in parallel to every item in the input dataset. This produces a list of (k2,v2) pairs for each call. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together, thus creating one group for each one of the different generated keys.

The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain.

$$Reduce(k2, list\ (v2)) \rightarrow list(v3)$$

Each Reduce call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list.

Thus the MapReduce framework transforms a list of (key, value) pairs into a list of values. This behavior is different from the functional programming map and reduce combination, which accepts a list of arbitrary values and returns one single value that combines all the values returned by map [1].

C. Parallel map reduce computations

The parallelism of the MapReduce framework comes from the fact that each map or reduce operation can be executed on a separate processor independently of others [7]. The system automatically schedules map-shuffle-reduce steps and routes data to available processors, including provisions for fault tolerance.

The outputs from a reduce step can, in general, be used as inputs to another round of map-shuffle-reduce steps. Thus, a typical MapReduce computation is described as a sequence of map-shuffle-reduce steps that perform a desired action in a series of rounds that produce the algorithm's output after the last reduce step.

D. Execution Overview

Fig. 2 shows the overall flow of a MapReduce operation in our implementation [1]. When the user program calls the MapReduce function, the following sequence of actions occurs.

1. The MapReduce library in the user program first splits the input files into M pieces. It then starts up many copies of the program on a cluster of machines.
2. One of the copies of the program—the master— is special. 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.

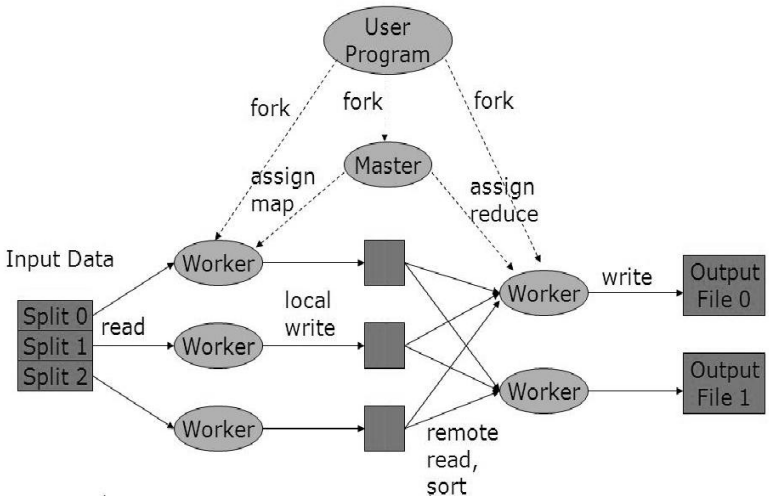


Figure 2: Execution Overview

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 for its partition, 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.

After successful completion, the output of the MapReduce execution is available in the R output files.

Graph Representations

A graph with n nodes can be represented as an n x n square matrix M, where a value in cell mij indicates an edge from node ni to node nj . In the case of graphs with weighted edges, the matrix cells contain edge weights; otherwise, each cell contains either a one (indicating an edge), or a zero (indicating none). With undirected graphs, only half the

matrix is used (e.g., cells above the diagonal). For graphs that allow self loops [3], the diagonal might be populated; otherwise, the diagonal remains empty.

Fig 3 provides an example of a simple directed graph and its adjacency matrix representation.

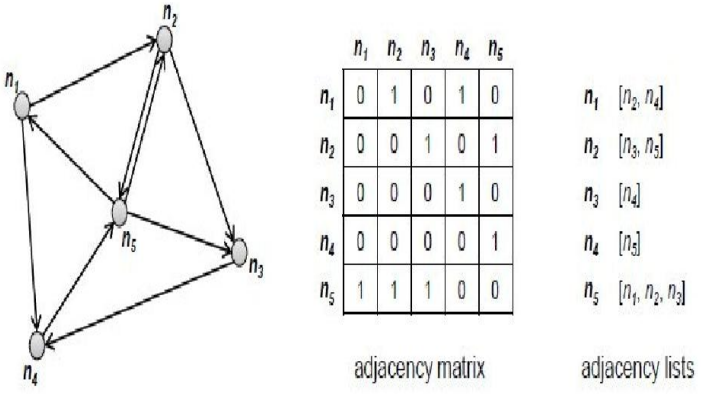


Figure 3: Sample Graph and Its Adjacency Matrix

Parallel Breadth-First Search

One of the most common and well-studied problems in graph theory is the single-source shortest path problem, where the task is to find shortest paths from a source node to all other nodes in the graph. The edges can be associated with costs or weights, in which case the task is to compute lowest-cost or lowest-weight paths. The solution to these kind of problems are usually solved using the Dijkstra's algorithm. this famous algorithm assumes sequential processing [9]. This paper deals with how to solve this problem in parallel, and more specifically, with MapReduce.

It is apparent that parallel breadth-first search is an iterative algorithm, where each iteration corresponds to a MapReduce job. The first time we run the algorithm, we discover all nodes that are connected to the source. The second iteration, we discover all nodes connected to those, and so on. Each iteration of the algorithm expands the "search frontier" by one hop, and, eventually, all nodes will be discovered with their shortest distances [3].

Usage of Map Reduce in Proposed work

As this paper is dealt with finding shortest route from one city to all other cities in the global network, the data we are going to consider is certainly big. The global network consists of big data sets in terabytes or even petabytes. Hence it becomes necessary to use MapReduce technique which is a common tool for problem solving in cloud computing environment.

The basic idea is to partition a large problem into sub problems. To the extent that the sub problems are independent they can be tackled in parallel by different workers-

threads in a processor core, cores in a multi-core processor, multiple processors in a machine, or many machines in a cluster. Intermediate results from each individual worker are then combined to yield the final output. [3]

In this paper, we deal with a global network considering a large number of cities and we apply MapReduce technique in the shortest path algorithm for AVL systems. The data is given in the form of graph and parallel breadth first search is used as mentioned earlier. The data may be structured or unstructured. The approach is as follows.

1. The global network graph is divided into components that are relatively disjoint. This is known as graph clustering. [3]
2. The divided components (subgraph) are inserted into buckets.
3. Apply the function map() which emits shortest distances to all reachable nodes on a single bucket and repeats it concurrently for all the buckets in parallel, storing the result (processing of each bucket) in another set of buckets called result buckets.
4. Apply the function reduce() on each of these result buckets. This function takes the input from the result buckets, finds the shortest distant node from the source node and made permanent. This functionality once again takes place concurrently.
5. The new node made permanent is considered as source node for the next iteration.
6. The steps 3, 4 and 5 above are iteratively performed until the shortest routes are found from source node to all other nodes.
7. The parallel results obtained are combined to yield the final shortest routes in the global network.

Map Reduce Shortest Path Algorithm

(MRSP Algorithm)

The cloud store billions of cities (nodes) and their distances from their adjacent cities in the form of adjacency matrices. So, there is a need for an algorithm that gives the shortest route to the required locations specific to a query.

A. Introduction to MRSP Algorithm

The input to the algorithm is a directed, connected graph G = (V,E) represented with adjacency lists, w containing edge distances such that $w(u, v) \geq 0$, and the source nodes.

The algorithm begins by first setting distances to all vertices d[v], $v \in V$ to ∞, except for the source node, whose distance to itself is zero. The algorithm maintains Q, a global priority queue of vertices with priorities equal to their distance values d.

The algorithm operates by iteratively selecting the node with the lowest current distance from the priority queue (initially, this is the source node). At each iteration, the algorithm “expands" that node by traversing the adjacency list of the selected node to see if any of those nodes can be reached with a path of a shorter distance. The algorithm terminates when the priority queue Q is empty, or equivalently, when all nodes have been considered. The algorithm discussed here only computes the shortest

distances. The actual paths can be recovered by storing “backpointers" for every node indicating a fragment of the shortest path.

B. Performance Evaluation of MRSPAlgorithm

We need to compute shortest paths for a large set of data that is distributed on several clusters. Shortest Path can be calculated even faster by using the MapReduce method which in turn reduces the computation time. This MapReduce algorithm is executed on all clusters parallely.

Our MapReduce algorithm is framed using parallel breadth first search technique. Since Shortest Path is recursively defined, the MapReduce algorithm is given as iterative procedure. The algorithm begins by first setting distances to all vertices d[v], v ϵ V to ∞, except for the source node, whose distance to itself is zero. The algorithm maintains Q, a global priority queue of vertices with priorities equal to their distance values d.

Figure 4 provides the pseudo code of the shortest path algorithm using map reduce

Pseudo Code:

```
class Mapper
method   Map(nid n , node N)
   d  ← N.Distance
   Emit(nid n ,N)
   for all nodeid m ϵ N.AdjacencyList do Emit (nid m , d + 1)
class Reducer
method Reduce(nid m , [d1, d2,…])
dmin    ← ∞
M   ← Ø
for all d ϵ counts [d1,d2,…] do
   if IsNode(d) then
M ← d
else if d < dmin then
            dmin ← d
M:Distance ← dmin
Emit(nid m; node M)
```

Figure 4: Pseudo-code for parallel breath-first search in MapReduce

In the pseudo code, we use n to denote the node id as integer and N to denote the node’s corresponding data structure (adjacency list and distance). The algorithm works by mapping over all nodes and emitting a key-value pair for each neighbor on the node’s adjacency list. The key is the node id of the adjacent node and value is the distance.

After shuffle and sort, reducers will receive keys corresponding to the adjacent nodeids and distances corresponding to all edges leaving that node. The reducer will find the minimum of these distances and make that corresponding node permanent. From this node the next iteration will start.

We implement our MRSP Algorithm for the graph shown in Fig 5. The adjacency matrix is given in Fig 6.

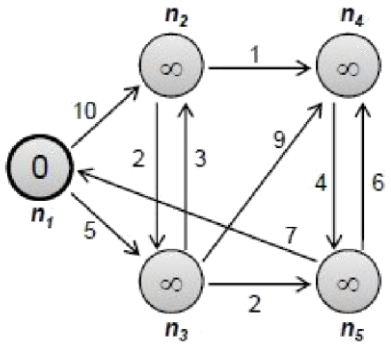


Figure 5: Interconnected Graph For 5 Cities (Nodes)

	n1	n2	n3	n4	n5
n1	0	10	5	0	0
n2	0	0	2	1	0
n3	0	3	0	9	2
n4	0	0	0	0	4
n5	7	0	0	6	0

Figure 6: Adjacency matrix for fig 5

1) The Map Phase

In the map phase, the mappers emit distances to reachable nodes. The map phase receives as input the source node and their node's structure (adjacency list , distance). It computes the distances for adjacent nodes and produces the intermediate values, the adjacent nodes and their distances from the source node.

Initially all nodes have a distance of ∞. The distance of source node is set to 0.

In iteration1 node n1 emit distances to its adjacent nodes n2 and n3. The Key value pairs generated by the map phase are (n2,10) and (n3,5). Similarly all the other nodes distributes their distances evenly to their outgoing nodes.

2) The Reduce Phase

The reduce phase receives as input the adjacent nodes and their distances The reducers select the minimum of those nodes currently emitted by the map phase and those which are not yet made permanant .

The output of reduce phase is the node with minimum distance. It makes this node permanent from which the next iteration starts.

In Iteration 1 of the reduce phase n3 is the node with minimum distance 5. Hence the node n3 is made permanent which serves as the source node for the map phase in the next iteration.

The nodes that will be expanded next, in order, are n5, n2, and n4. The algorithm terminates with the end state shown in iteration 4, where we've discovered the shortest distance to all nodes.

We iterate the algorithm until there are no more node distances that are ∞. Since the graph is connected, all nodes are reachable, and all discovered nodes are guaranteed to have the shortest distances.

The following fig 7 shows the iterations of MapReduce in computing the Shortest paths.

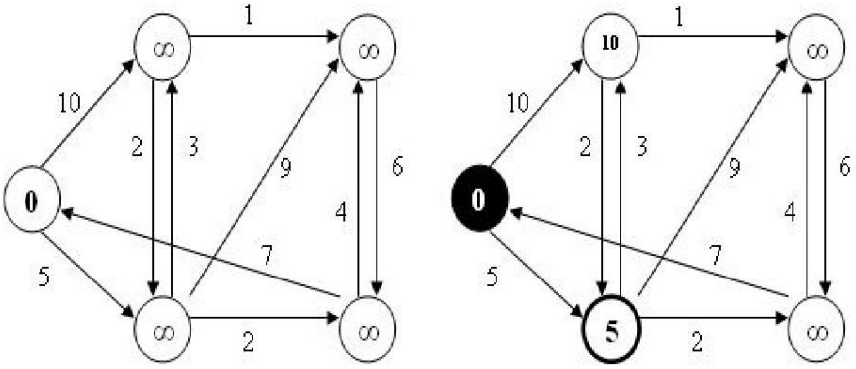
Iteration 1:

Map

Source node is n1
Emits distances to adjacent nodes n2,n3

Reduce

Choose minimum node n3 and make it permanent.
n3.distance = 5
n3.path = n1 → n3



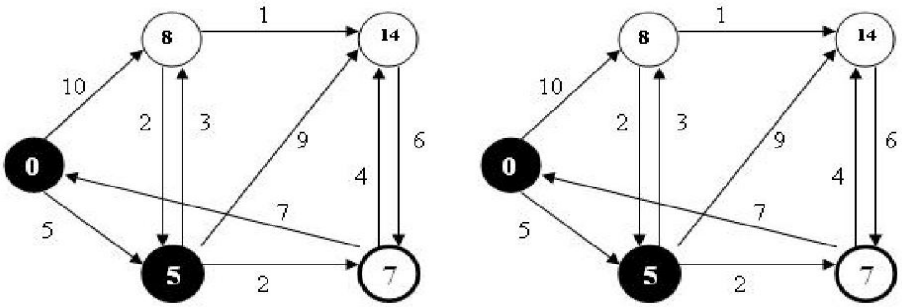
Iteration 2:

Map

Source node is n3
Emits distances to adjacent nodes n2,n4,n5

Reduce

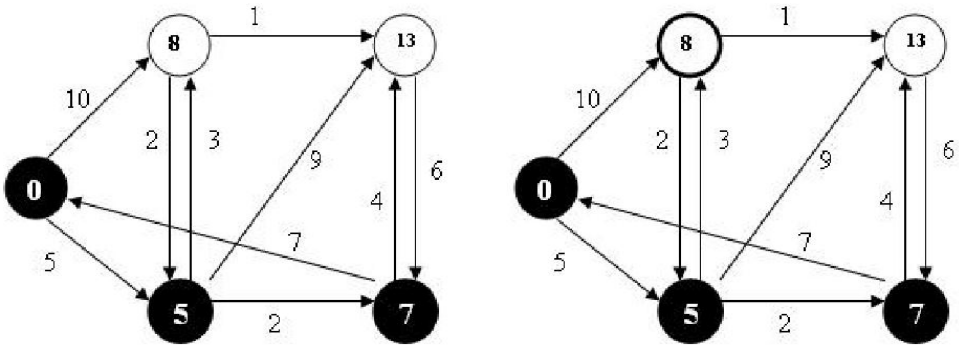
Choose minimum node n5 and make it permanent.
n5.distance = 7
n5.path = n1 → n3→n5



Iteration 3:

Map
Source node is n5
Emits distances to adjacent nodes n4

Reduce
Choose minimum node n2 and make it permanent.
n2.distance = 8
n2.path = n1 → n3→n2



Iteration 4:

Map
Source node is n5
Emits distances to adjacent nodes n4

Reduce
Choose minimum node n4 and make it permanent.
n4.distance = 9
n4.path = n1 → n3→n2→n4

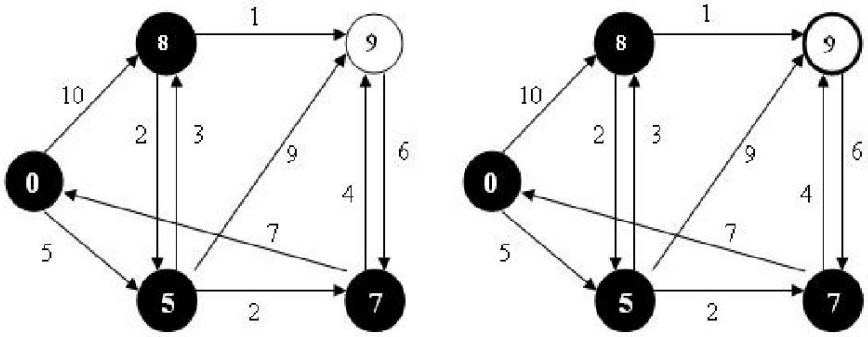


Figure 7: Shows The Four Iterations For The Graph Of Fig 5

The algorithm ends here since there are no more nodes with distances ∞. We have found the shortest path to all the other nodes n2,n3,n4,n5 from the source node n1.

The shortest distances calculated to all adjacent nodes by the map phase are illustrated in table 1.

Table 1: Map Phase Emitting Distances To Reachable Nodes

Iteration	**Source node**	**n2.dist**	**n3.dist**	**n4.dist**	**n5.dist**
1	n1	10	5	∞	∞
2	n3	8	5	14	7
3	n5	8	5	13	7
4	n2	8	5	9	7
5	n4	No more nodes to visit			

The minimum distance node which is to be made permanent next computed by the reduce phase is illustrated in table 2.

Table 2: Reduce Phase Choosing Node With Minimum Distance

Iteration	**n2.dist**	**n3.dist**	**n4.dist**	**n4.dist**	**Node chosen**
1	10	5	∞	∞	n3
2	8	5	14	14	n5
3	8	5	13	13	n2
4	8	5	9	9	n4

The shortest distances and their shortest paths of all the nodes(cities) from the source node is depicted in table3.

Table 3: Shortest Distances and Paths To All The Other Nodes From Source Node n1

Nodes	**Shortest distances From the source node n1**	**Shortest path From the source node n1**
n2	8	n1 → n3→n2
n3	5	n1 → n3
n4	9	n1 → n3→n2→n4
n5	7	n1 → n3→n5

Advantages of Mapreduce

1) Fault tolerance

Fault tolerance is handled via re-execution. The master pings every worker periodically. If no response is received from a worker in a certain amount of time, the master marks the worker as failed [1].

On worker failure, it will detect failure via periodic heartbeats, re-execute completed and in-progress map tasks, re-execute in progress reduce tasks and task completion is committed through master.

When a map task is executed first by worker A and then later executed by worker B (because A failed), all workers executing reduce tasks are notified of the re-execution. Any reduce task that has not already read the data from worker A will read the data from worker B.

The master writes periodic checkpoints of the master data structures. If the master task dies, a new copy can be started from the last check pointed state.

2) Parallelism

The map() functions runs in parallel, creating different intermediate values from different input data sets. The reduce() functions also run in parallel, each working on a different output key. All values are processed independently[7].

3) Optimizations

Reduce phase can't start until map phase is completely finished. A single slow disk controller can rate-limit the whole process. Master redundantly executes slow-moving map tasks and uses results of first copy to finish. The Combiner functions can run on same machine as a mapper. It causes a mini-reduce phase to occur before the real reduce phase, to save the bandwidth[7].

4) Scalability

The MapReduce programming model and distributed execution platform is described for easily scaling systems to run efficiently across thousands of commodity machines on a dedicated high-speed network. The scalability achieved using MapReduce to implement data processing across a large volume of CPUs with low implementation costs, whether on a single server or multiple machines, is an attractive proposition [8].To achieve good scalability, a computer system needs to be able to run many simultaneous threads of execution in isolation of each other, or at least with minimal dependency between executions.

Conclusions and Future Work

In this paper, we've presented and analyzed the performance of a MapReduce application in Shortest Path computation for Automatic vehicle location systems. This enables us to find out the shortest route from any location to all other cities using parallel breadth first search technique. This approach helps to make use of the global data which are provided in the cloud and to process them parallely. So this MRSP algorithm saves computation time and guarantees quality. We also presented the fundamentals of MapReduce Programming with the open source framework. This framework accelerates the processing of large amounts of data through distributed processes, delivering very fast responses. It can be adopted and customized to meet the various development requirements and can be scaled by increasing number of nodes available for processing. The extensibility and simplicity of the frame work are the key differentiators that make it a promising tool for data processing.

References

[1] Dean, J., & Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters (2004). Google, Inc. Gottfrid, D., 2004.

[2] Google's MapReduce programming model Revisited,Ralf Lammel, July 2007.

[3] Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer University of Maryland, College Park Manuscript prepared April 11, 2010

[4] Automatic Vehicle Location Systems, S.A. Mahdavifar, G.R. Sotudeh., Heydari, World Academy of Science, Engineering and Technology,2009

[5] Above the Clouds: A Berkeley View of Cloud Computing, Michael Armbrust,Armando Fox, Rean, Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia, Electrical Engineering and Computer Sciences, University of California at Berkeley,February 10, 2009

[6] Automated Vehicle Location using Global Positioning Systems for First Responders,Mr. Daniel Portillo, Institute for Information Technology Applications,Technical Report Series February 2008

[7] MapReduce in the Clouds *for.* Science. Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, Geoffrey Fox, CloudCom 2010.

[8] Cluster Computing at a Glance Mark Bakery and Rajkumar Buyyaz, July 2010

[9] Boris V. Cherkassky, Loukas Georgiadis, Andrew V. Goldberg, Robert E. Tarjan, and Renato F. Werneck, Shortest-Path Feasibility Algorithms: An Experimental Evaluation, in ACM Journal of Experimental Algorithmics, vol. 14, no. 2, pp. 2.7:1-2.7:37, Association for Computing Machinery, Inc., 2009

[10] Rabi Prasad Padhy , Big Data Processing with Hadoop-MapReduce in Cloud Systems , IJ-CLOSER, Feb 2013

[11] http://research.microsoft.com/users/goldberg