Parallel Shortest Path Graph Computations of United States Road Network Data on Apache Spark

Yasir Arfat[1(✉)], Rashid Mehmood[2], and Aiiad Albeshri[1]

[1] Department of Computer Science, FCIT, King Abdulaziz University, Jeddah 21589, Saudi Arabia
yasirarfat08l@gmail.com, aaalbeshri@kau.edu.sa
[2] High Performance Computing Center, King Abdulaziz University, Jeddah 21589, Saudi Arabia
RMehmood@kau.edu.sa

Abstract. Big data is being generated from various sources such as Internet of Things (IoT) and social media. Big data cannot be processed by traditional tools and technologies due to their properties, volume, velocity, veracity, and variety. Graphs are becoming increasingly popular to model real-world problems; the problems are typically large and, hence, give rise to large graphs, which could be analysed and solved using big data technologies. This paper explores the performance of single source shortest path graph computations using the Apache Spark big data platform. We use the United States road network data, modelled as graphs, and calculate shortest paths between vertices. The experiments are performed on the Aziz supercomputer (a Top500 machine). We solve problems of varying graph sizes, i.e. various states of the US, and analyse Spark's parallelization behavior. As expected, the speedup is dependent on both the size of the data and the number of parallel nodes.

Keywords: Big data · Apache spark · Apache hadoop · Graph analytics
Apache GraphX · Shortest paths · Road networks

1 Introduction

Graphs are becoming increasingly popular to model real-world problems [1]. Graph analytics play an important role in information discovery and problem solving. A graph can be any real-life application that can be used to find a relation, routing, and a path. Graphs have many applications such as image analysis [2], social network analysis [3, 4], smart cities [5, 6], scientific and high performance computing [7–9], transportation systems [10], Web analyses [11], and biological analyses [12]. In these applications, a large amount of data is being generated every second, known as big data. Big Data refers to the emerging technologies that are designed to extract value from data having four Vs characteristics; volume, variety, velocity and veracity [13, 14]. Volume defines the generation and collection of the vast amount of data. Variety defines the type of the data stored or generated. Types include structured, semi-structured and unstructured data. Velocity describes the timeline related to the generation and processing of big data.

R. Mehmood et al. (Eds.): SCITA 2017, LNICST 224, pp. 323–336, 2018.
https://doi.org/10.1007/978-3-319-94180-6_30

Veracity refers to the challenges related to the lack of uncertainty in data. Big Data V's and Graphs have a close relationship. For example, volume could represent the number of edges and nodes, and velocity could be considered as the graph's streaming edges. A graph could be uncertain (veracity) and has the variety characteristics because data sources could vary.

The processing of graphs in a distributed environment is a great challenge due to the size of the graph. Typically, a large graph is partitioned for processing. A graph can be partitioned to balance the load on the various machine in a cluster. These partitions are processed in a parallel distributed environment. For the computation of the graph data on the distributed platform, there is a need for scalability and efficiency. These are the two key elements to achieve the good performance. We also need to move our data closer to computation to minimize the overhead of data transfer among the nodes in the cluster. Load balancing and data locality plays a major role in achieving this purpose. It can utilize the whole resource of the system during processing. Moreover, big data is so huge that cannot be able to process by traditional tools and technologies. There are many platforms for graph processing, but these platforms have many issues. Parallel computation of large graphs is a common problem. Therefore, in this scenario parallel distributed platforms are suitable for processing large graphs. In this work, we have used the Graphx [15] for parallel distributed graph processing which is an attractive framework for the graph processing.

In this work, we explore the performance of single source shortest path graph computations using the Apache Spark big data platform. We use the United States road network data, modelled as graphs, and calculate shortest paths between vertices. The experiments are performed on the Aziz supercomputer (a Top500 machine). We solve problems of varying graph sizes, i.e. various states of the US, and analyse Spark's parallelization behaviour. As expected, the speedup is dependent on both the size of the data and the number of parallel nodes.

The rest of the paper is organized as follows. Section 2 gives background material and literature review. Section 3 discusses design and methodology. Section 4 analyses the result. The conclusions and future directions are given in Sect. 5.

2 Background Material and Literature Review

Graph computation has great importance for analysis in various applications. In this section, we explored the state of the art work that already has been done for the graph analysis.

Apache Spark [16] is an open source tool for processing the large data set. It is also next generation of big data applications and alternate for Hadoop. To overcome the issues like disk I/O and performance improvement of Hadoop they introduced the spark. It has several features like memory computation that make it unique. It provides facility like cashing the data in memory. Spark supports the several programming languages, i.e., Python, Java, and Scala. Graphx [15] is an open source platform for the processing graph data. It has various characteristics such as flexibility, speed, parallel graph computation, extends the spark RDD. Hadoop [17] is an open source software to process the big data. It has several characteristics scalability, reliability, fault tolerance,

high availability, local processing & storage, distributed and parallel computing faster and cost effective. Hadoop as many components but most important components are MapReduce and HDFS.

Kajdanowicz et al. [18] analyzed three parallel large graph processing approaches such as Bulk Synchronous Parallel (BSP) MapReduce and map-side join. These strategies implemented for the calculation of single source shortest path (SSSP) and relational influence propagation (RIP) of graph nodes for collective classification. They find out that iterative graph processing performs well as compared to MapReduce using the BSP. Liu et al. [19] have described graph partitioning is a major issue for parallel large graph processing. These challenges are a replication of vertices, unbalanced partitioning, and communication between partitions. To solve these problems, they proposed a new graph partitioning framework for the parallel processing of a large graph. The primary goal of this framework was to minimize the bandwidth, balance the load and memory. It has three greedy graph partitioning algorithms. They run theses algorithms using different dataset and find out these algorithms can solve the issue of graph partitioning based on the specification and needs. Wang et al. [20] presented a new approach for maximal clique and k-plex enumeration. It finds the dense subgraph using the binary graph partitioning. It divides the graph in such a way that enables each partition of a graph to process it parallel. It was implanted using the MapReduce. The presented approach has smaller search space and more parallelizable. Braun et al. [21] proposed a technique for the analysis social network using a knowledge based system. The main goal of this approach was mine the interests of the social network represent as graphs. It analyses the relationship between directed graphs and captures the mutual friends using the undirected graph. To analyse the performance of the chosen approach, they have used the Facebook and Twitter dataset.

Laboshin et al. [22] have presented a framework based on the MapReduce for the analysis of web traffic. The primary objective of using this framework was to scale the storage and computing resources for the extensive network. Liu et al. [23] have presented a clustering algorithm for the distributed density peaks to overcome the issue in distance based algorithms. It also calculates the distance between all pairs of vertices. Using this algorithm, the computational cost will be decreased. This algorithm is based on the Apache GraphX [15]. Aridhi et al. [24] analysed various big graph mining frameworks. The primary focus was on the pattern mining that consists of the discovering useful and exciting information from the large graph using mining algorithms. They did detailed analysis on the various mining approaches for the big graphs. Drosou et al. [25] presented a new framework called enhanced Graph Analytical Platform (GAP). It uses the top down approach for the mining the large volume of data. It gives strength many other key features such as HR clustering. It works efficiently for the big data acquiring useful insights. Zhao et al. [26] analysed of different graph processing platforms. They compared these platforms regarding data parallel and graph parallel. For the computation of graph and resource utilization graph platforms works well as compared to data parallel. On the other hand, they find out that regarding size, data parallel graph platforms are superior in performance.

Mohan et al. [27] did a comparison on the parallel graph processing big data platforms. They compared features and performance of these platforms. Pollard et al. [28] presented a new approach for the analysis of scalability and performance of the parallel

graph processing platforms. They analysed the power consumption and performance of the most commonly used algorithm (BFS, SSSP and Page Rank) on the graph processing packages (Graph-Mat, Graph500 Graph Benchmark Suite, and PowerGraph) using different datasets. Suma et al. [29] have also done an important on smarter societies for logistics and planning. They evaluated the proposed approach using parallel distributed framework. Miller et al. [30] presented the graph analytics for query processing to find the shortest path for specific patterns. They introduced the algorithms which show that vertex centric and graph centric algorithms are easily parallelizable. They also argue that MapReduce is not effective for the computation of iterative algorithms. Chakaravarthy et al. [31] have presented a new algorithm that was originated from Delta-stepping and Bellman-Ford algorithms. The main goal of this algorithms was to classify the edges, reducing the inner node traffic and optimization of direction. They have used the SSSP for unweighted graph find out the paths among all other nodes as destination.

Yinglong et al. [32] have described that big data analytics are important to discover for such entities that can easily represent in the form of a graph. It is the primary challenge for the processing of computation of graph-based patterns. They proposed a new system that allows the user to organize the data for the architecture of parallel computing. It also consists of visualization, graph storage, and analytics. They also analyze the data locality regarding graph processing and its effects on the performance of cache memory on a processor. Zhang et al. [33] proposed a new algorithm for the fast graph search that it transforms the complex graphs into vectorial representations based on the prototype in the database. After this, it also accelerates the query efficiency in the Euclidean space by employing locality sensitive hashing. They evaluated their approach against the real datasets, which achieves the high performance in accuracy and efficiency.

Shao et al. [34] have proposed a new approach partitioning aware graph computation engine (PAGE). The benefit of this method is that it controls the online graph partitioning statistics of under lying results of a graph. Second, it monitors the parallel processing resources and enhances the computation resources. Third, it was also designed to support the various graph partitioning qualities. For evaluation of chosen schemes, they showed that it performs well under various partitioning approaches with different qualities. Chen et al. [35] have proposed a new framework of graph partitioning to enhance the performance of the network for graph partitioning itself, storage of partitioned graph and vertex oriented processing of graph. They have developed all these optimizations for the cloud network environments. They also have used the two models such as partition sketch and machine graph. The basic purpose of using these two graphs was to capture the features of graph partitioning process and network performance. Zeng et al. [36] have proposed new parallel multi-level stepwise partitioning algorithm. They divided this algorithm into two phases; one is an aggregate phase and second is partition phase. In aggregate phase, it uses the multilevel weighted label propagation for aggregation of the large graph into the small graph. But in a second phase: It has the K-way balance-partitioning performs on the weighted based on the stepwise mining RatioCut method. It reduces the RatioCut step by step. In each step, sets of vertices are extracted by reducing the part of RatioCut, and these vertices are removed from the graph. In this, they have obtained the k-way balanced

partitioning by this algorithm. In experiments, they have made the comparison with various other existing partitioning approaches using the dataset of a graph.

Lee et al. [37] have introduced vertex block (VBs) partitioner. It is a distributed model for the data partitioner for the large-scale graphs in the cloud. It has three features. First, It has the vertex bock (VBs), and it also extends the extended vertex block (EVBs) as building blocks for the semantic large-scale graphs. Second, vertex block partitioner uses vertex block grouping algorithm to place the high correlation in the graph into the same partition. Third, the VB partitioner speed up the parallel processing of graph pattern queries by minimizing the inter-partition query processing. In results, they showed that proposed approach has higher query latency and scalability over large-scale graphs. Xu et al. [38] have proposed a log based dynamic graph partitioning method. This method uses the recodes and reuses the historical statistical information to refine the partitioning result. It can be used as middleware and deployed to many existing parallel graph-processing systems. It also uses the historical partitioning results for the creation of a hyper graph, and it also uses a new hyper graph streaming strategy to generate the better stream graph partitioning result. Moreover, it also dynamically partitions the huge graph and also uses the system to optimize the graph partitioning to enhance the performance. Yang et al. [39] proposed a new approach called self-evolving distributed graph management environment (sedge). It reduces the communication during the processing of the graph query on the multiple machines. It also has two level of partitioning such as primary partition and dynamic secondary partitioning. These two types of partitions can adapt any kind of real environment. Results show that it enhances the distributed graph processing on the commodity clusters.

3 Design and Methodology

In this section, we discuss the design methodology of our work. We have used three big data tools Apache Hadoop [17], Apache Spark [16] and Apache Graphx [15]. We also setup an Apache Spark cluster. The Hadoop HDFS is used for input and output storage. We process the data using Apache Graphx on Spark cluster and store the output in HDFS. We have also used the Apache Spark's data locality and load balancing techniques. Data locality is of great importance while processing Spark jobs. Computing the jobs and data together have significant effects on performance processing. If the data and code are not together, to improve the job performance we have to move them together. Usually, the data size is greater than the code. Hence it is easy to move the code in serialized form than data in the form of chunks. Spark has inbuilt scheduler for achieving data locality. Data locality is all about how to place the data close to each other to process faster. There are various types [40] of data locality in the spark. These are Process Local, Node Local, No Pref, Rack local, and any. Spark prefers to schedule all tasks at the best locality level, but this is not always possible. On the other hand, to balance the load among all nodes in the cluster, there are two types [41] of partitioning scheme in Apache GraphX. These are vertex-cut and edges-cut. However, GraphX only uses vertex-cut scheme for graph partitioning. The vertex-cut scheme has different types partition strategy, but in our work, we have used the "2D Edge Partitioning" technique.

In Algorithm 1, we have developed a new approach for the computing the SSSP using load balancing and maintaining data locality. In our technique, first, we set the data locality either as the process level, node level or rack level. If any node needs the task or data from another node, it will check the data initially in its process and then check the node. Before requesting from the other node, it will wait for 3 s. If data or task is not available on the node then it will check within the rack, similarly again it will wait for 3 s.

For the Load balancing, we applied graph based partitioning scheme called the 2D edges partitioning which equally partitioning the graph and distributes among all the nodes in the cluster. Next step, we input the edges list and nodes list. It will be map into edges RDDs and vertices RDDs, from these two RDDs we shall draw a graph. After this, we shall input the source vertex to find the shortest path between all other vertices from given vertex. Next step we shall apply Dijkstra algorithm to find shortest paths. In the end, we shall print all the shortest path determined by the Dijkstra algorithm with their total distance.

Algorithm 1: Single Source Shortest Path (SSSP) using GraphX

Input: List of vertices, List of edges, source vertex
Output: list of shortest paths

```
1: Function main (vertices, edges)
2:        Locality (local execution, true)
3:        Locality (locality wait for process, 3S)
4:        Locality (locality wait for node, 3S)
5:        locality (locality wait for rack, 3S)
6:        Nodes List ← path for the input vertices file.
7:        Edges List ← path for the input vertices file.
8:        Vertices ← map nodes List
9:        Edges ← map Edges List
10:       Graph ← create graph from vertices and edges
11:       Graph Partition ← partition graph by partitioning strategy edge
          partitioning2D
12:       Finding the shortest distance formula from using shortest path algorithm
13:       Source Vertex: ← Vertex
14:       Computed Dijkstra Algorithms
15:       Prints the shortest paths
```

3.1 The Road Network Dataset

In this section, we shall present the description of the dataset that we have for the performance analysis of graph computations. We have used the DIMACS [42] dataset. This dataset has whole USA road network in the form of graph data. We took the entire USA and Five states of the USA, District of Columbia (DC), Rhode Island (RI), Colorado (CO), Florida (FL), California (CA). The graph data that we have chosen is undirected it has the billions of edges and vertices. Following Table 1. shows the no of edges and vertices in different states and the complete USA.

Table 1. USA road network dataset

Name of road network	Vertices	Edges	Type
District of Columbia (DC)	9559	14909	Undirected
Rhode Island (RI)	53658	69213	Undirected
Colorado (CO)	435,666	1,057,066	Undirected
Florida (FL)	1,070,376	2,712,798	Undirected
California (CA)	1,890,815	4,657,742	Undirected
USA (whole country)	23,947,347	58,333,344	Undirected

We also have visualized road network dataset using Gephi [43]. We have only visualized the DC and RI state data set as shown in Figs. 1 and 2 respectively. We could not visualize the other states data due to the large size which cannot be handled

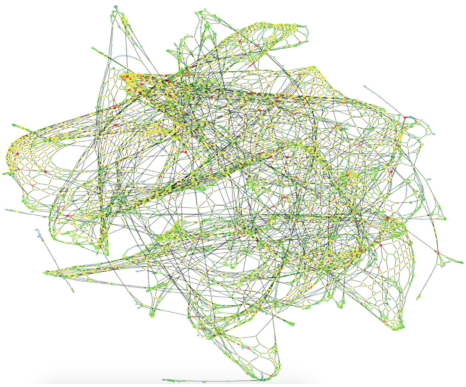

Fig. 1. District of Columbia road network

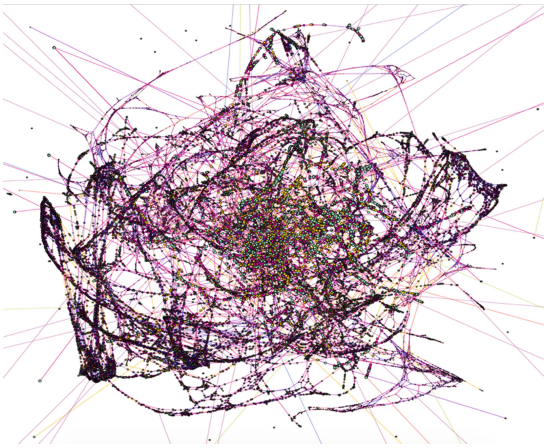

Fig. 2. Rhodes Island road network

on a single PC. We have only visualized two states to perceive the structure of road network datasets.

We have plotted the degree distribution and the histogram of vertex degrees of different states and full USA road network dataset. The primary purpose of this plot was to see the nature of dataset that we are using in the research. In Fig. 3(i), it shows the DC road network dataset visualization and degree of distribution. However, has four thousand vertices with degree 3, and Fig. 3(ii) forty-nine thousand vertices have the degree 6 in FL. In CO road network dataset Fig. 3(iii) shows more no of vertices with degree 6 and over thirteen thousand vertices have degree 4. On the other hand, in Fig. 3(iv) whole USA road network dataset majority of vertices has degree 6.

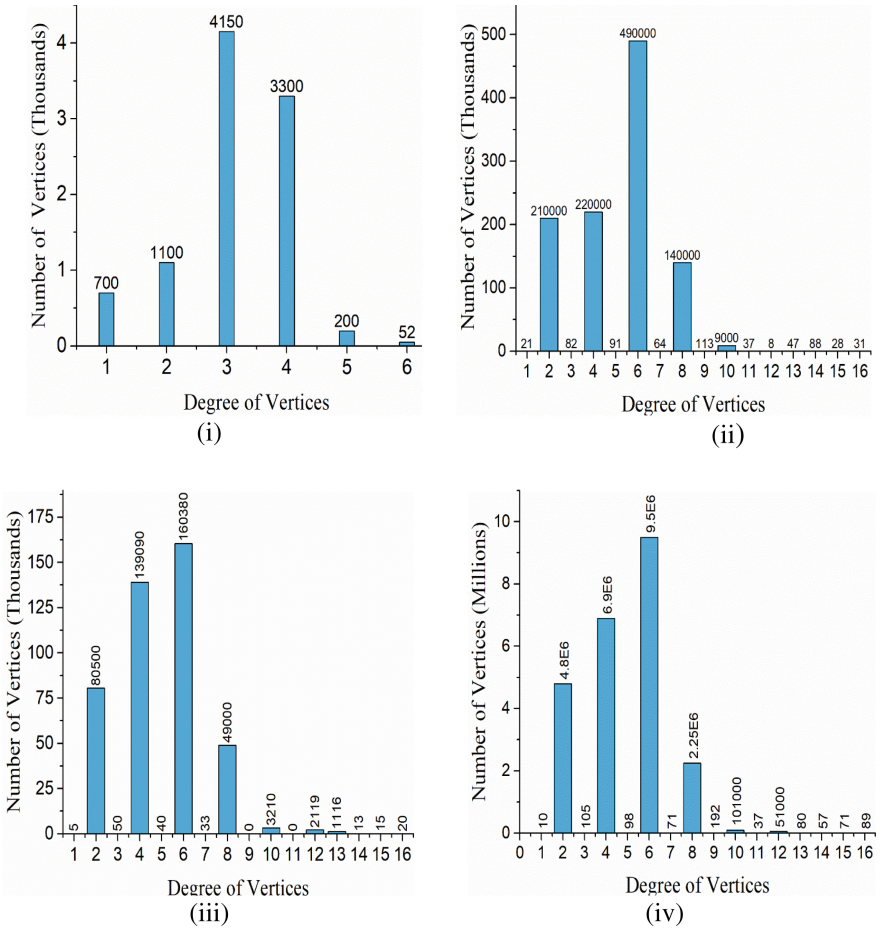

Fig. 3. Visualization of (i) District of Columbia road network (ii) Florida road network (iii) Colorado road network (iv) Whole USA road network

3.2 Experimental Setup

For experimental setup, we have built the spark cluster setup using the Aziz supercomputer. In this configuration, we have used the different Aziz nodes, as it varies for the same dataset for 1, 2, 4, 8 and 16 Aziz nodes. We have used Apache Hadoop HDFS to store input and output data where as for the processing of the data we used the Apache spark. Rest of software and hardware configuration is given below in Table 2.

Table 2. Configuration environment

The node type	Master	Slave
Software and Hardware environment	Linux centOS, JDK 1.7, Processor 2.4 GHz, Apache Spark 2.0.2, GraphX, 24 cores, Apache Hadoop HDFS	Linux centOS, JDK 1.7, Processor 2.4 GHz, 24 cores, Apache Spark 2.0.1, GraphX Apache Hadoop HDFS
Memory	94 G	94 G per slave
Quantity	1	Different slave as 1, 2, 4, 8 and 16

4 Results and Analysis

We implemented the parallel and sequential code on the spark cluster using the Aziz supercomputer. The difference between the results for the sequential version and for a single node is that the single node provides a parallel Spark execution of the shortest path algorithm using multiple cores. We have run the code sequentially on the Aziz supercomputer for all different USA states DC, RI, CO, FL, CA and the whole USA. We found that sequentially on Apache Spark it takes 4.07 s for DC road network and 5.05 s for RI road network. In CO road network it took 7.54 s, whereas 17.76 s taken by FAL road network. In CA road network, it takes 37.92 s. These are all timing has been shown in Fig. 4, which we run sequentially on Apache Spark. On the other hand, we also run the whole USA road network sequentially that took 476.33 s for the processing as shown in Fig. 5.

Similarly, we run the data for the 1, 2, 4, 8 and 16 Aziz nodes using the Apache Spark cluster parallel using and note down the timing of each as shown in Fig. 4. For 1 Aziz node, it took 2.70,2.90, 6.67, 10.77 and 24.62 s for the processing of DC, RI, CO, FL and CA states road network dataset respectively. Using 2 Aziz nodes, it also takes 2.42, 2.49, 4.70, 9.04 and 21.90 s for the processing of DC, RI, CO, FL and CA states road network dataset using our approach respectively. Using our technique, the running of 4 Aziz nodes, a processing time of different states as follows: 2.39, 2.41, 4.01, 8.89 and 19.49 s for DC, RI, CO, FL, and CA road network. For 8 Aziz nodes, it takes 2.30, 2.34, 3.90, 8.48 and 18.89 s for DC, RI, CO, FL, and CA road network. Again, we double the Aziz nodes up to 16, run the code using our developed approach it takes 2.22, 2.28, 3.62, 7.99 and 16.18 s for DC, RI, CO, FL, and CA road network. In this case, we can compare our results with 8 Aziz nodes, but there is not much speed up due to a small dataset and transfer time. We also run the whole USA road network dataset using our approach as we found that on 1, 2, 4, 8, 16 Aziz nodes it takes 155.63,

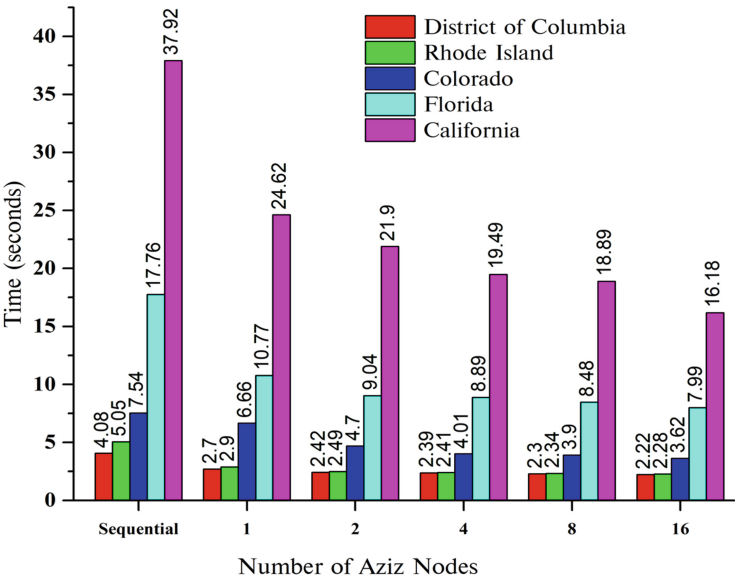

Fig. 4. Performance comparison of different Aziz nodes using states of USA road network dataset.

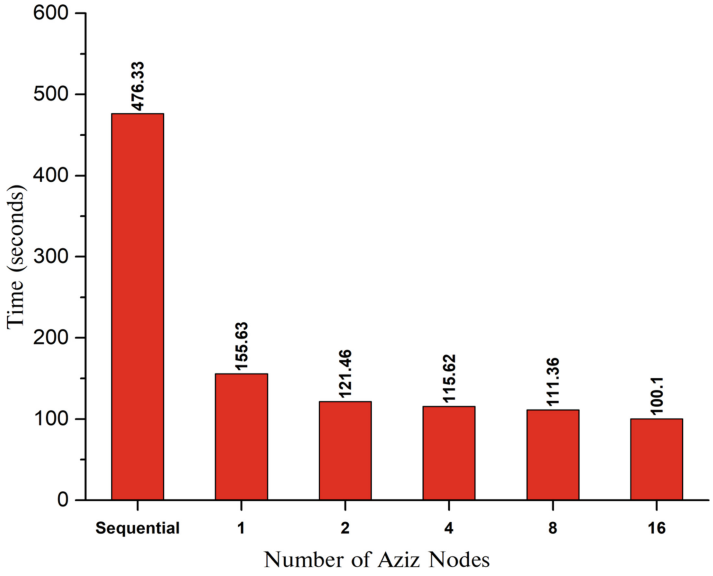

Fig. 5. Performance comparison of different Aziz nodes using entire USA road network dataset.

121.46, 115.62, 111.36, and 100.10 s for processing of data parallel on Apache Spark as shown in Fig. 5.

We have executed our code for a different number of USA states, as well as the whole USA road network dataset. We have achieved a good speed up as given in Figs. 4 and 5. In Fig. 4, there is a comparison of different USA states road network dataset, each state has different size of the dataset. Smaller data affects the scale of parallelism that can be achieved. Moreover, it is computationally expensive. However, when we move from a small data set (i.e., DC to CA) to a higher size of dataset get more speed up and consumes less time using different Aziz nodes. However, there is also increase in the execution time when we shall use the more nodes in the cluster, and our data is small. It happens due to the transfer of data among the nodes in the cluster. Therefore, we can only get the parallelism with certain no of nodes, or we have to see that how many numbers of nodes are needed to process the particular dataset to achieve good speedup.

5 Conclusion and Future Work

Graph analytics plays an important role in discovering and understanding the useful information. Graphs also have many applications such smart cities, social media, biological networks, etc. these applications have a significant amount data that cannot be processed on the traditional tools and technologies. So, parallel distributed platforms are suitable for the processing of large size of data.

In this work, we processed parallel distributed graph based SSSP using GraphX and Apache Spark cluster on Aziz. We applied the load balancing and data locality based approach to compute the SSSP. We have used the DIMACS road network dataset for a research experiment. This dataset contains different states of road network dataset and whole USA road network. We found that our approach takes less time for the execution and increase the speedup. However, we also find out that if data is small, there is speed up but not much. On the other hand, it is observed that if the data is significant, we can achieve high performance. However, there are certain constraints such as reduction in speed when the number of nodes in a cluster increase. We obtain speed up and high performance at a particular number of nodes. For future work, we shall apply this approach our health care transport application for computation of SSSP.

Acknowledgments. The authors acknowledge with thanks the technical and financial support from the Deanship of Scientific Research (DSR) at the King Abdulaziz University (KAU), Jeddah, Saudi Arabia, under the grant number G-661-611-38. The experiments reported in this paper were performed on the Aziz supercomputer at King Abdulaziz University.

References

1. Lu, Y., Cheng, J., Yan, D., Wu, H.: Large-scale distributed graph computing systems. Proc. VLDB Endow. **8**, 281–292 (2014)
2. Sanfeliu, A., Alquézar, R., Andrade, J., Climent, J., Serratosa, F., Vergés, J.: Graph-based representations and techniques for image processing and image analysis. Pattern Recognit. **35**, 639–650 (2002)
3. Ding, Y., Yan, S., Zhang, Y.B., Dai, W., Dong, L.: Predicting the attributes of social network users using a graph-based machine learning method. Comput. Commun. **73**, 3–11 (2016)
4. Khan, A., Uddin, S., Srinivasan, U.: Adapting graph theory and social network measures on healthcare data. In: Proceedings of the Australasian Computer Science Week Multiconference on - ACSW 2016, pp. 1–7. ACM Press, New York (2016)
5. Mehmood, R., Graham, G.: Big data logistics: a health-care transport capacity sharing model. Procedia Comput. Sci. **64**, 1107–1114 (2015)
6. Mehmood, R., Meriton, R., Graham, G., Hennelly, P., Kumar, M.: Exploring the influence of big data on city transport operations: a Markovian approach. Int. J. Oper. Prod. Manag. (2016). Forthcoming
7. Hendrickson, B., Kolda, T.G.: Graph partitioning models for parallel computing. Parallel Comput. **26**, 1519–1534 (2000)
8. Mehmood, R., Crowcroft, J.: Parallel iterative solution method for large sparse linear equation systems. Comput. Lab. Univ. (2005)
9. Kwiatkowska, M., Parker, D., Zhang, Y., Mehmood, R.: Dual-processor parallelisation of symbolic probabilistic model checking. In: DeGroot, D., Harrison, P. (eds.) Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS, pp. 123–130. IEEE, Volendam (2004)
10. Alazawi, Z., Abdljabar, M.B., Altowaijri, S., Vegni, A.M., Mehmood, R.: ICDMS: an intelligent cloud based disaster management system for vehicular networks. In: Vinel, A., Mehmood, R., Berbineau, M., Garcia, C.R., Huang, C.-M., Chilamkurti, N. (eds.) Nets4Cars/Nets4Trains 2012. LNCS, vol. 7266, pp. 40–56. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29667-3_4
11. Junghanns, M., Petermann, A., Neumann, M., Rahm, E.: Management and analysis of big graph data: current systems and open challenges. In: Zomaya, A., Sakr, S. (eds.) Handbook of Big Data Technologies, pp. 457–505. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-49340-4_14
12. Oh, S., Ha, J., Lee, K., Oh, S.: DegoViz: an interactive visualization tool for a differentially expressed genes heatmap and gene ontology graph. Appl. Sci. **7**, 543 (2017)
13. Mehmood, R., Faisal, M.A., Altowaijri, S.: Future networked healthcare systems: a review and case study. In: Big Data: Concepts, Methodologies, Tools, and Applications, pp. 2429–2457. IGI Global (2016)
14. Arfat, Y., Aqib, M., Mehmood, R., Albeshri, A., Katib, I., Albogami, N., Alzahrani, A.: Enabling smarter societies through mobile big data fogs and clouds. Procedia Comput. Sci. **109**, 1128–1133 (2017)
15. GraphX | Apache Spark
16. Apache Spark: Apache Spark™ - Lightning-Fast Cluster Computing
17. Welcome to Apache™ Hadoop®! - index.pdf. https://hadoop.apache.org/index.pdf
18. Kajdanowicz, T., Kazienko, P., Indyk, W.: Parallel processing of large graphs. Futur. Gener. Comput. Syst. **32**, 324–337 (2014)

19. Liu, X., Zhou, Y., Guan, X., Sun, X.: A feasible graph partition framework for random walks implemented by parallel computing in big graph. In: Chinese Control Conference, CCC 2015, pp. 4986–4991, September 2015
20. Wang, Z., Chen, Q., Hou, B., Suo, B., Li, Z., Pan, W., Ives, Z.G.: Parallelizing maximal clique and k-plex enumeration over graph data. J. Parallel Distrib. Comput. **106**, 79–91 (2017)
21. Braun, P., Cuzzocrea, A., Leung, C.K., Pazdor, A.G.M., Tran, K.: Knowledge Discovery from Social Graph Data. Procedia Comput. Sci. **96**, 682–691 (2016)
22. Laboshin, L.U., Lukashin, A.A., Zaborovsky, V.S.: The big data approach to collecting and analyzing traffic data in large scale networks. Procedia Comput. Sci. **103**, 536–542 (2017)
23. Liu, R., Li, X., Du, L., Zhi, S., Wei, M.: Parallel implementation of density peaks clustering algorithm based on spark. Procedia Comput. Sci. **107**, 442–447 (2017)
24. Aridhi, S., Mephu Nguifo, E.: Big graph mining: frameworks and techniques. Big Data Res. **6**, 1–10 (2016)
25. Drosou, A., Kalamaras, I., Papadopoulos, S., Tzovaras, D.: An enhanced Graph Analytics Platform (GAP) providing insight in Big Network Data. J. Innov. Digit. Ecosyst. **3**, 83–97 (2016)
26. Zhao, Y., Yoshigoe, K., Xie, M., Zhou, S., Seker, R., Bian, J.: Evaluation and analysis of distributed graph-parallel processing frameworks. J. Cyber Secur. Mobil. **3**, 289–316 (2014)
27. Mohan, A., Remya, R.: A review on large scale graph processing using big data based parallel programming models. Int. J. Intell. Syst. Appl. **9**, 49–57 (2017)
28. Pollard, S., Norris, B.: A Comparison of Parallel Graph Processing Benchmarks (2017)
29. Suma, S., Mehmood, R., Albugami, N., Katib, I., Albeshri, A.: Enabling next generation logistics and planning for smarter societies. Procedia Comput. Sci. **109**, 1122–1127 (2017)
30. Miller, J.A., Ramaswamy, L., Kochut, K.J., Fard, A.: Research directions for big data graph analytics. In: Proceedings of the 2015 IEEE International Congress on Big Data, BigData Congress 2015, pp. 785–794 (2015)
31. Chakaravarthy, V.T., Checconi, F., Petrini, F., Sabharwal, Y.: Scalable single source shortest path algorithms for massively parallel systems. In: Proceedings of the 28th International Parallel and Distributed Processing Symposium, IPDPS, pp. 889–901 (2014)
32. Xia, Y., Tanase, I.G., Nai, L., Tan, W., Liu, Y., Crawford, J., Lin, C.: Explore efficient data organization for large scale graph analytics and storage. In: Proceedings of the 2014 IEEE BigData Conference, pp. 942–951 (2014)
33. Zhang, B., Liu, X., Lang, B.: Fast Graph Similarity Search via Locality Sensitive Hashing. In: Ho, Y.-S., Sang, J., Ro, Y.M., Kim, J., Wu, F. (eds.) PCM 2015. LNCS, vol. 9314, pp. 623–633. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24075-6_60
34. Shao, Y., Cui, B., Ma, L.: PAGE: A partition aware engine for parallel graph computation. IEEE Trans. Knowl. Data Eng. **27**, 518–530 (2015)
35. Chen, R., Yang, M., Weng, X., Choi, B., He, B., Li, X.: Improving large graph processing on partitioned graphs in the cloud. In: Proceedings of the Third ACM Symposium on Cloud Computing, SoCC 2012, pp. 1–13 (2012)
36. Zeng, Z., Wu, B., Wang, H.: A parallel graph partitioning algorithm to speed up the large-scale distributed graph mining. In: Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications - BigMine 2012, pp. 61–68 (2012)
37. Lee, K., Liu, L.: Efficient data partitioning model for heterogeneous graphs in the cloud. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2013)
38. Xu, N., Chen, L., Cui, B.: LogGP: a log-based dynamic graph partitioning method. Proc. VLDB Endow. **7**, 1917–1928 (2014)

39. Yang, S., Yan, X., Zong, B., Khan, A.: Towards effective partition management for large graphs. In: Proceedings of the 2012 International Conference on Management Data - SIGMOD 2012, pp. 517–528 (2012)
40. Spark Data Locality Documentation. https://spark.apache.org/docs/latest/tuning.html#data-locality
41. GraphX Partitioning Scheme Documentation. https://spark.apache.org/docs/latest/graphx-programming-guide.html#optimized-representation
42. DIMACS Implementation Challenge. http://www.dis.uniroma1.it/challenge9/download.shtml
43. Gephi - The Open Graph Viz Platform. https://gephi.org/