Chapter 8
Parallel Shortest Path Big Data Graph Computations of US Road Network Using Apache Spark: Survey, Architecture, and Evaluation

Yasir Arfat, Sugimiyanto Suma, Rashid Mehmood, and Aiiad Albeshri

8.1 Introduction

Smart applications and infrastructures are increasingly relying on graph computations. We are witnessing a continuous increase in the use of graphs to model real-world problems [1]. The emergence of many graph-based software, programming languages, graph databases, and benchmarks—such as ArangoDB, Neo4j, Sparksee, Gremlin, and Graph 500—provide the evidence for the increasing popularity of graph-based computing. Graph analytics plays 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, route, or a path. Graphs have many applications such as image analysis [2], social network analysis [3, 4], smart cities [5–7], communication networks [8–14], scientific and high performance computing [15–20], transportation systems [21], Web analyses [22], healthcare [23–25], and biological analyses [26]. In these applications, a large amount of data is being generated every second, commonly referred to as big data.

Big Data refers to the "emerging technologies that are designed to extract value from data having four V's characteristics; volume, variety, velocity and veracity" [27, 28]. Volume defines the generation and collection of the vast amount of data.

Y. Arfat · A. Albeshri
Department of Computer Science, FCIT, King Abdulaziz University, Jeddah, Saudi Arabia
e-mail: yqasim@stu.kau.edu.sa; aaalbeshri@kau.edu.sa

S. Suma
Division of Data, Department of Engineering, Kumparan, Jakarta Selatan, Indonesia
e-mail: sugimiyanto.sugimiyanto@kumparan.com

R. Mehmood (✉)
High Performance Computing Center, King Abdulaziz University, Jeddah, Saudi Arabia
e-mail: RMehmood@kau.edu.sa

R. Mehmood et al. (eds.), *Smart Infrastructure and Applications*,
EAI/Springer Innovations in Communication and Computing,
https://doi.org/10.1007/978-3-030-13705-2_8

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. Veracity refers to the challenges related to the 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 machines 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 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, as mentioned earlier, big data cannot be processed by traditional tools and technologies. There are many platforms for graph processing, but these platforms have performance 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 [29–31] for parallel distributed graph processing which is a widely used framework for the graph processing. The big data platform that we have used for distributed graph computing of shortest paths is Apache Spark [32].

This chapter extends our earlier work on single source shortest path computations of big data road network graphs using Apache Spark. In our earlier work [33], we had used the US road network data, modelled as graphs, and calculated shortest paths between two vertices over a varying number of up to 368 compute cores. The experiments were performed on the Aziz supercomputer (a former Top500 machine [34]). We had analyzed Spark's parallelization behavior by solving problems of varying graph sizes, i.e., various states of the USA with up to over 23 million vertices and 58 million edges.

We focus in this chapter on computing a set of large varying number of shortest path queries on a (source, destination) vertex pair. The number of queries used are 10, 100, 1 K, 10 K, 100 K, and 1 M queries executed over up to 230 CPU cores. We achieve good performance, and as expected, the speedup is dependent on both the size of the data and the number of parallel nodes. In addition to the extended results, this chapter provides a detailed literature on shortest path graph computations. The system architecture for graph computing in Spark is explained with additional details using the architecture depiction and elaborated algorithms. We call our system, the Big Data Shortest Path Graph Computing (BDSPG) system.

The rest of the chapter is organized as follows. Section 8.2 gives background and literature review. Section 8.3 describes the design and methodology of the BDSPG system. Section 8.4 presents the analysis of result. The conclusions and future directions are given in Sect. 8.5.

8.2 Literature Review

Smart urban infrastructure greatly replies on smart mobility designs. Many approaches have been proposed to address smart mobility-related challenges [35]. These include, among many others, modelling and simulation-based approaches [36, 37], location-based services [38], telematics [39], social media-based approaches [40–42], approaches based on vehicular networks (VANETs) and systems [43–45], autonomic mobility management [46, 47], autonomous driving [48], mobility in emergency situations [49–54], approaches to improve urban logistics [40, 55], and big data-based approaches [40–42, 56]. A recent book discusses several smart society proposals on infrastructure and applications including smart mobility [7]. Many mobility problems naturally map to graph-based computations; shortest path computations are one of them and are of great significance in smart mobility infrastructure designs. In this section, we discuss state-of-the-art work from the literature on graph-based road network shortest path computations.

Quddus and Washington developed an algorithm to find the shortest path between two points called weight-based shortest path and vehicle trajectory aided map-matching (stMM) [57]. It improves the map-matching of low-frequency positioning data on a roadmap. They exploit a well-known A* search algorithm. They tested the performance of proposed approach with collected data from rural, suburban, and urban areas in Nottingham and Birmingham, UK. Szucs designed and implemented a model and an algorithm for route planning in road network [58]. They proposed a solution that also aims to find the equilibrium in the path optimization problem. The proposed approach takes the uncertainty of state information of roads, their uncertainty and influencing factors into account. The system is based on the Dempster-Shafer theory, which helps to model the uncertainty and Dijkstra's algorithm which allows finding the best route. Feng et al. proposed an improvement of alternative route calculation, based on alternatives figures [59]. They exploit a bidirectional Dijkstra algorithm to explore the route. They introduced three quotas to measure the quality of an Alternative Figures (AG). They introduce the concept of pheromones into the Plateau method and enhance the ability of Plateau method to find a meaningful alternative road.

Zeng and Church demonstrated the relative value of A* algorithm to solve simple point-to-point shortest path problems on real road networks [60]. It is applied to road networks from two counties of California, USA. They state that Dijkstra algorithm can be improved by taking advantage of network properties associated with GIS-source data. Whereupon, Dijkstra does not take advantage of the spatial attributes which are available in a GIS setting, while A* can take the advantage of spatial coordinates in trimming the search to find the shortest path. Malewicz et al. proposed Pregel, a framework for large-scale graph processing [61]. The framework is similar in concept to MapReduce. It provides users with a natural API for programming graph algorithms while managing the details of distribution invisibly, including messaging and fault tolerance. It contributes providing a suitable system for large-scale graph computing. They deployed dozens of Pregel applications. The users report that the API is intuitive, easy to use, and flexible. The experiment shows that the performance, scalability, and fault tolerance of proposed framework are satisfactory for computing graph jobs with billions of vertices.

Yan et al. proposed a framework called Graphine for graph-parallel computation in multicore clusters [62]. It addresses the problem of existing distributed graph-parallel frameworks which cannot scale well with the increasing number of cores per node. They implemented the proposed framework and evaluated it. The experiment result shows that their proposed framework achieves sublinear scalability with the number of nodes, a number of cores per node, and graph size up to one billion vertices, as well as achieves 2∼15 times faster than the state-of-the-art Power Graph on a cluster with 16 multicore nodes. Selim and Zhan proposed an algorithm and data reduction technique based on data nodes in large networks dataset [63]. It is done by computing similarity computation, maximum similarity clique (MSC), and then finding the shortest path due to the data reduction in the graph. The technique aims to reduce the network that will have a significant impact regarding performance (shortest time and faster analysis) on calculating the shortest path. The proposed technique takes into account shortest path problem between two nodes in a large undirected network graph. The result shows that their proposed technique beats up Dijkstra's shortest path algorithm with large datasets with respect to execution time. Zhou et al. presented a new graph processing framework based on Google's Pregel called P++ [64]. The proposed framework aims to reduce the system overhead for algorithms that require many iterations in Pregel. It extends Pregel by some new terms such as introducing a new data structure, internal compute, super-vertex, and new API. Their proposed approach has been evaluated by using real datasets with cases Shortest Path and PageRank. The result shows that their proposed technique demonstrate its superior performance.

Cao et al. proposed an approach for solving the stochastic shortest path problem in vehicle routing [65]. It aims to find the optimal path that maximizes the probability of arriving at specified destination before the given deadline. Their approach is data-driven which explores big data generated in traffic. They evaluated the performance using a real traffic data extracted from real GPS trajectories of vehicles in road network of Munich city, which consists of 170 nodes and 277 edges. The experiment result shows that the proposed approach outperforms traditional methods. Hou U et al. developed a framework to solve online shortest path problem called live traffic index (LTI) [66]. The proposed framework aims for computing the shortest path according to live traffic conditions. It enables drivers to effectively and quickly get the live traffic information on the broadcasting channel. There is no existing efficient solution that can offer affordable costs for online shortest path computation at both client and server sides. The conventional architecture scales poorly with the number of clients. Their approach is that the server collects live traffic information and distributes it over radio or wireless network. They evaluated their approach with four different road maps, including New York City, San Francisco bay area road map, San Joaquin road map, and Oldenburg road map.

The result shows that their proposed method reach optimal solution in terms of four performance factors for online path computation.

Strehler et al. developed a model called fully polynomial-time approximation scheme (FPTAS) for finding shortest energy-efficient routes for electric and hybrid vehicles [67]. It aims to resolve the problem of electric and hybrid vehicles regarding the shortest path problem and planning of the trip, whereupon recharging an electric car takes longer than refilling fossil fuels car. Their contribution is introducing a general model for the routing of hybrid and electric vehicles with intermediate stops at charging station and convertible resources. They are using Matlab to represent and test their model. The used dataset are engine model, topographical information, and road data of German. They are in improvement phase that the running time of the proposed algorithms may not be suitable for practical purposes, particularly, when it is running on a mobile phone or on an in-car device. Hong et al. developed a multicore computing approach to find shortest route from single source and single destination while avoiding obstacles [68]. Whereupon, the existing approaches have limited ability in dealing with real-time analysis in big data environments. They use multicore computing to speed up the computation and analysis using Python's official Multiprocessing library. Thus, the parallelization is core based. The approach itself exploits the notion of a convex hull for evaluating obstacles and constructing pathways iteratively. The experiment result shows their proposed approach for parallel processing has significant improvements over sequential computing for wayfinding and navigation tasks with a large number of obstacles in complex urban area. Mozes et al. developed an algorithm by combining two techniques for computing shortest paths in directed planar graphs [69]. The two combined techniques are STOC'94 and FOCS'01. It aims to remove the log n dependency of the shortest path algorithm in the running time, in order to have better and optimal performance. The theoretical proving shows that their proposed technique obtains a speedup over previous algorithms for solving shortest-path problem.

In this work, Abraham et al. [70] have worked on the point to point the shortest path computations on the road network data. They modelled the road network as a graph having highways with low dimension. The algorithm they named Hub labels for computation of shortest path. The authors claim that it works faster for all types of queries. However, they have not used the parallel implementation of an algorithm. The performance this might suffer from significant data computation of this algorithm. In this chapter, they have not used the US road network dataset. It uses a general algorithm for the computation of road network graph data. Sanders et al. [71] have presented the real-world road network processing algorithms. They claim that algorithm takes less as compared to the Dijkstra. In this work, they have not used the parallel implementation of the algorithm. They also did not use the big data computation. Peng et al. [72] has presented a framework for the computation of the road network distance using a single source-target pair. In presented algorithms, they mapped the distance into a distributed structure of hash. For the implementation, they used the Apache Spark and in memory computation for the distance of road network computation. They experimented their algorithm using US and NYC road network dataset. Zhu et al. [73] have proposed an index structure called Arterial Hierarchy (AH) for the shortest and distance queries in a road network. They argue that existing work concentrates on the practical or asymptotic performance. The problem with state of the art was worst regarding space and time. The primary objective of this chapter was to minimize the gap between theory and practice for shortest path queries on road network. For the evaluation, they have used the 20 million nodes. The proposed technique performs better than existing approaches for road network dataset. Moreover, in this work, they have not used the weighted road network graph data.

Zheng et al. [74] have presented all pair shortest path algorithm. The proposed algorithm was an alternative to the Floyd-Warshall. They implemented their algorithm using Apache Spark and analyzed the performance of their algorithm. They argue that the performance of Floyd-Warshall algorithm suffered using Apache Spark due to a large number of global updates. To solve this issue, they have used the fewer global update steps based on computation that has been done on each iteration. As a result, they showed that their algorithm performs better than Floyd-Warshall algorithm. However, their work is different as compared to our work. We are parallelizing the shortest path between two vertices source and target. Djidjev et al. [75] have presented all pair shortest path algorithm using GPU cluster. They have used both centralized and decentralized computation for the all pair shortest path algorithm. They have presented the two algorithms that use the Floyd-Warshall method. For implementation, they have used the multi-GPU cluster. They have also used the California state road network dataset that consists of 1.9 million vertices and five million edges. Aridhi et al. [76] have presented the shortest path algorithm on the base of MapReduce. To solve the shortest path problem in an efficient way, they have partitioned the graph into subgraphs then they process it parallel. The algorithm they have proposed is an iterative whose performance will suffer when these are large of input data due to its iterative nature. For an experiment, they have used the French road network dataset from the OpenStreetMap. For the computation, they have used the Hadoop and MapReduce. Faro et al. [77] have presented a shortest path all pair algorithm with and without traffic congestions on the road network. The main objective of this chapter was to find the fastest shortest path on road network. They implemented the proposed all pair shortest algorithm parallel. First, they tried to find the shortest path then tried to find the alternate shortest path in case of traffic congestion. They implemented their algorithm using the GPU. They have not used any road network dataset, neither Spark nor Hadoop.

Kajdanowicz et al. [78] used the Bulk Synchronous Parallel (BSP), map-side join, and MapReduce for the graph computation. They applied these approaches for the single source shortest path (SSSP) and relational influence propagation (RIP) for collective classification of graph vertices. They stated that using BSP iterative graph processing perform better as compared to MapReduce. Liu et al. [79] have proposed a framework for parallel processing of large graph to solve the issue of communication between partitions, unbalanced partition, and replication of vertices. This framework uses three different greedy graph partitioning algorithms. They run these algorithms using the various dataset and observed that whether these algorithms can solve the issues of graph partitioning based on the specific needs. The major objective of this framework was to balance the load and reduce the bandwidth. Wang et al. [80] proposed a technique for k-plex enumeration and maximal clique approach. Using the binary graph partitioning approach, find the dense subgraph from the graph. It parallel process each partition of the graph by dividing the graph. MapReduce was used for implementation. Braun et al. [81] presented a new approach for social network analysis for knowledge-based systems. The major objective of this technique is to mine the interests of social network and represent as graph. The directed graph has been used for relationship analysis and undirected graph has been used to capture mutual friends. They have used the Facebook and Twitter dataset to analyze the performance of the proposed approach.

Laboshin et al. [82] proposed a new framework based on MapReduce to analyze the web traffic. The major objective of proposed framework was to scale the storage and computing resources for the extensive network. Liu et al. [83] proposed a clustering algorithm for the distributed density. This algorithm solves the issues in distance-based algorithms. This algorithm calculates the distance among all pairs of vertices. The authors claim that using this algorithm computational cost will be reduced. They implemented their algorithm using Apache GraphX [29, 30]. Aridhi et al. [84] investigated different frameworks for mining of big graph. The major focus was to use the mining algorithm for pattern mining that consists of the discovering useful information from the huge graph dataset. They analyzed comprehensively different mining techniques for the large graphs. Drosou et al. [85] proposed an enhanced Graph Analytical Platform (GAP) framework for processing of large graph dataset. This framework uses the top-down approach for mining of huge graph dataset. It provides the strength to features like HR clustering. It is an effective framework for the big data getting useful insights.

Zhao et al. [86] evaluated various graph computation platforms. They did comparison between graph- and data-parallel platform for processing of large dataset. They found out that graph-parallel platforms perform better for resource utilization and graph computation as compared to data-parallel platforms. However, data-parallel platform for graph processing is superior in performance regarding size. Mohan [87] et al. compared the graph computation platforms for large data processing using the key features and performance. Miller et al. [88] investigated the graph analytics from perspective of query processing. There are issues in finding the interesting information from the graph whether it's a shortest path or pattern matching from the graph. They also introduced algorithms which show that vertex centric and graph centric algorithms are easily parallelizable. They stated that MapReduce is not an ideal platform for the iterative algorithm.

Chakaravarthy et al. [89] proposed an algorithm that is derived from the Delta-stepping and Bellman-Ford algorithms. The primary objective was to categorize the edges, minimize the traffic of inner vertices, and optimize the directions. They applied the single source shortest path (SSSP) to get the shortest path between the vertices. Yinglong et al. [90] stated that big data analytics are essential for the entities that can be represented as graph. It is the main challenge for the computation of graph bases patterns. They presented a new architecture that allow users to organize the data for parallel computation. This architecture has three components: graph storage, analytics, and visualization. They evaluated the data locality for the processing and effects on the performance of cache memory on a processor. Zhang et al. [91] presented an algorithm for the fast graph search. This algorithm converts the completed graphs into vectorial representations on the basis of prototype in the database. So, it accelerates the query efficiency in a Euclidean space by using locality-sensitive hashing. They examined their proposed approach using real dataset that gets higher performance regarding accuracy and efficiency. Pollard et al. [92] proposed a new technique for the parallel graph processing platforms analysis based on the performance and scalability. They used the breadth first search, page rank, and single source shortest for the analysis of power consumption and performance with packages of graph processing with various datasets.

Table 8.1 provides a comparison of various shortest path graph computation approaches. The table includes information for each work regarding aim and objectives, the approach used, the dataset sources, the type of datasets, the platforms used, research gap, and comments. We would have preferred to include the names of authors of the respective works in a separate column in the table but these were omitted to save space and fit the table in as few pages as possible.

8.3 Methodology and Design

This section details the methodology and design of our Big Data Shortest Path Graph Computing (BDSPG) System.

Figure 8.1 shows the architecture for shortest path computations. First, it will take the graph data as input for the computation of shortest path. This data can be directed or undirected graph data but in our work, we are using undirected graph dataset. Once we have data we have uploaded any distributed file system, so nodes in the cluster can easily access this data. There can be any distributed file system such as FEFS, NEFS, and HDFS. But in our work, we are using HDFS. After keeping input graph data, we build the graph and perform the one pair shortest path (OPSP) using GraphX [31]. After computation of OPSP, we shall get the shortest path having total distance and vertices in the source and target vertex.

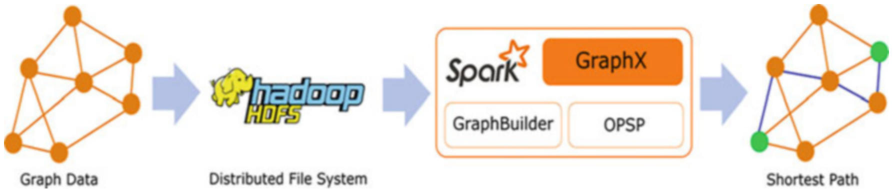

Fig. 8.1 The Big Data Shortest Path Graph Computing (BDSPG) System Architecture

Table 8.1 Comparison of various shortest path graph computation approaches

Aim and objectives [Source]	Approach	Dataset	Dataset type	Platforms	Research gap/comment
Shortest path between two points [57]	Find the shortest path between two points called weight-based shortest path	Collected data from rural, suburban, and urban areas in Nottingham and Birmingham, UK	Road network	Not clear	Big data platform, e.g., apache spark not used
Find the equilibrium in the path optimization problem [58]	An algorithm for route planning in road network	Transport network Vertices: 4111 Edges: 5443	Road network	C programming language	Spark not used
Enhance the ability of plateau method to find a meaningful alternative road [59]	Bidirectional Dijkstra algorithm to explore the route	Beijing China Vertices: 153275 Edges: 433719	Road network	Java	Not distributed, only one node used
Point-to-point shortest path [60]	A* algorithm to solve simple point-to-point shortest path problems on real road networks	Road network data from two counties in California. Santa Barbara: Vertices 33,074, edges: 78144, Los Angeles: Vertices: 195233, edges: 532178	Road network	C programming with visual studio	Apache spark is not used
Large SSSP graph processing [61]	A natural API for programming graph algorithms while managing the details of distribution invisibly	Claims to use a framework for billions of edges and vertices	Any type of graph data	C++	

(continued)

Table 8.1 (continued)

Aim and objectives [Source]	Approach	Dataset	Dataset type	Platforms	Research gap/comment

To solve problem of existing distributed graph-parallel frameworks which cannot scale well with the increasing number of cores per node [62]	Graphine for graph-parallel computation in multicore clusters	LiveJournal:(5,363,260 vertices, and 79,023,142 edges) Hollywood (2,180,759 vertices and 228,985,632 edges)
		
		Arabic:(22,744,080 vertices, and 639,999,458 edges)
		
		Twitter: (41,652,230 vertices, and 1,468,365,182 edges. SK: 50,636,154 vertices, and 1,949,412,601 edges.
		
		UK: 105,896,555 vertices, and 3,738,733,648 edges
			Social media, web, journal	Apache spark, OpenMPI, C	Do not use road network dataset

Reduce the network that will have a significant impact on performance (shortest time and faster analysis) on calculating the shortest path [63]	Computing similarity computation, maximum similarity clique (MSC), and then finding the shortest path due to the data reduction in the graph	XML-based network graph dataset	Network graph		Do not use road network dataset

Aims to reduce the system overhead for algorithms that require many iterations in Pregel [64]	A new graph processing framework based on Google's Pregel called P++. It extends Pregel by some new terms such as introducing a new data structure, internal compute, super-vertex, and new API.	Social network dataset twitter (40 million vertices, > 1 billion edges), and synthetic dataset by using first K vertices and their edges to evaluate the scalability of their proposed framework	Dataset: Social network dataset	P++ framework	They have used the unweighted graph dataset
Find the optimal path that maximizes the probability of arriving at a specified destination before the given deadline [65]	Solving the stochastic shortest path problem in vehicle routing	Road network of Munich city, which consists of 170 nodes and 277 edges. And also weighted graph of travel time, extracted from real GPS trajectories of BMW vehicles	Synthetic and road network dataset	Machine learning	Small dataset, only on single node, not parallel
Compute the shortest path according to the live traffic conditions [66]	A framework to solve online shortest path problem called live traffic index (LTI)	New York City (NYC) (264 k nodes, 733 k edges), San Francisco bay area road map (SF) (174 k nodes, 443 k edges), San Joaquin road map (SJ) (18 k nodes, 48 k edges), and Oldenburg road map (OB) (6 k nodes, 14 k edges)	Road network	Java	Small dataset, not distributed, single node only

(continued)

Table 8.1 (continued)

Aim and objectives [Source]	Approach	Dataset	Dataset type	Platforms	Research gap gap/comment
Resolve the problem of electric and hybrid vehicles regarding the shortest path problem and planning of the trip [67]	Developed a model called fully polynomial-time approximation scheme (FPTAS) for finding shortest energy-efficient routes for electric and hybrid vehicles	Synthetic for simple evaluation. Real dataset topographical information, and road data from Germany. Data size not clear	Synthetic and road network dataset.	They used Matlab to represent and test their model	No distributed implementation, single node only
Remove limitations of existing approaches and limited ability in dealing with real-time analysis in big data environments [68]	Developed a multicore computing approach to find shortest route from single source and single destination while avoiding obstacles	The initial area of focus is the Arizona State University campus in Tempe, Arizona. The campus has 179 buildings that represent obstacles to direct travel between an origin and destination. With total 2688 pairs of origin-destination	Not clear	Sequential and parallel process in Python	Road network dataset not used, small dataset, apache spark not used
Remove the log n dependency of the shortest-path algorithm in the running time, in order to have better and optimal performance [69]	An algorithm by combining two techniques for computing shortest paths in directed planar graphs	Theoretical proof, no dataset	N/A	N/A	Not distributed, only one node

Exact point-to-point shortest paths in road networks [70]	Hub labels (HL), a labeling algorithm to compute shortest path	DIMACS Road network dataset (USA road network and Europe road network)	Road network	C++ and Microsoft visual C++	Not parallel
Find shortest path [71]	Presented a new speedup technique for route planning that exploits the hierarchy inherent in real-world road networks	DIMACS Road network (USA, Europe, Germany)	Road network	C++	No parallel or big data implementation
Speed up complex spatial analytical queries and evaluate a large number of network distance queries which are posed as a large set containing N source-target pairs [72]	Presented a framework for the computation of the road network distance using a single source-target pair	US (V:23 M, E:58 M), Bay Area (V: 758,104,E:1 M) and NYC (V: 264,346, E: 733,846) road network dataset	Road network	Apache spark	
Minimize the gap between theory and practice for shortest path queries on road network [73]	Proposed an index structure called arterial hierarchy (AH) for the shortest and distance queries in a road network	Delaware (48,812 V 120,489E) New Hampshire (115,055 V, 264218E) Maine (187,315 V, 422998 E) Colorado (435,666 V, 1057066E) Florida (1,070,376 V,2712798E) California and Nevada (1,890,815 V, 4657742E) Eastern US 3598623 V, 8778114E) Western US (6,262,104 V, 15248146) Central US (14,081,816 V,34292496E) United States (23 M, 58E)	Road network	C++	

(continued)

Table 8.1 (continued)

Aim and objectives [Source]	Approach	Dataset	Dataset type	Platforms	Research gap gap/comment
Improve the performance APSP using Apache Spark [74]	All-pairs-shortest-paths (APSP)	Synthetic distance matrices	Synthetic dataset	Apache spark	No real dataset
To use both centralized and decentralized computation for the all pair shortest path algorithm [75]	All pair shortest path algorithm using GPU cluster	Graph dataset 1.9 M vertices, 5 M edges	Road network	GPU cluster	
Solve the shortest path problem in an efficient way, partitioned the graph into subgraphs then process it parallel [76]	Presented the shortest path algorithm using MapReduce	French road network dataset from the OpenStreetMap	Road network	Hadoop and MapReduce	MapReduce cannot give good performance under iterative environment
Find the fastest shortest path on road network [77]	Presented a shortest path all pair algorithm with and without traffic congestions on the road network	N/A	N/A	GPU	No road network dataset, not spark or Hadoop
Use BSP iterative graph processing for better performance [78]	The bulk synchronous parallel (BSP), map-side join and MapReduce for the graph computation	Twitter, YouTube, and tele with millions of edges	Social media	MapReduce	MapReduce used
Solve the issues of communication between partitions, unbalanced partition, and replication of vertices [79]	Proposed a framework for parallel processing of large graphs	There are several datasets but largest one is Facebook: 1000001 V, 23728298 E: Live journal: 3997962 V, 34681189 E. (Facebook, twitter, live media)	Real dataset, synthetic dataset, used the SNAP dataset	MapReduce	MapReduce

Effective load balancing and efficient parallel performance [80]	Proposed a technique for k-plex enumeration and maximal clique approach	Social networks, wikivote, epinions, Slashdot0902, Gowalla_edges, youtube Pokec, WikiTalk, web graphs, uk2005 it2004, BerkStan, WebGoogle, WikiComm, wikipedia2009, miscellaneous, networks, HepPh, EuAll, dblp2012 skitter	The largest dataset they have used is: Pokec (1,632,803 V, 22,301,964 E)	MapReduce	No road network dataset
Mine the interests of social network and represent as graph [81]	Presented a new approach for social network analysis for knowledge-based systems	Ego-Facebook (4039 V, and 88,234E) Ego-Twitter (81,306 V, 1,768,149E)	Social media	Apache Hadoop 0.20.0	They have not used weighted graph
Scale the storage and computing resources for the extensive network [82]	A new framework based on MapReduce to analyze the web traffic	N/A	N/A	N/A	A framework to process large dataset
Solve the issues in distance-based algorithms [83]	Proposed a clustering algorithm for distributed density	News dataset contains 10 topics, and 47,956 texts	News dataset	Apache Spark and GraphX [93]	No road network dataset
Use mining algorithm for pattern mining for discovering useful information from a huge graph dataset [84]	Investigated different frameworks for mining of big graph	N/A	N/A	N/A	It is a review of different graph mining frameworks

(continued)

Table 8.1 (continued)

Aim and objectives [Source]	Approach	Dataset	Dataset type	Platforms	Research gap gap/comment
It provides the strength to features like HR clustering [85]	Proposed an enhanced graph analytical platform (GAP) framework for processing of large graph dataset	Twitter dataset and synthetic data mobile-based dataset. Size of this dataset is not given	Social media and synthetic	Not clear	Path graph analysis not used, no clear mention of anything about graph processing platforms
Comparison between graph and data parallel platform for processing of large dataset [86]	Evaluations of various graph computation platforms	Facebook (61,876,615 V, 336,776,269E). LiveJournal (4,847,571 V, 68,993,773E) and DBLP (317,080 V, 1,049,866E)	Social networking dataset	Spark-0.9.0 PowerGraph-2.2 Giraph 1.0.0 GPS-0.0.1	No road network dataset
Comparison using the key features and performance [87]	Compared the graph computation platforms for large data processing	Collaboration (317,080 V, 1,049,866 E) Communication (36,692 V, 183831 E) CA road network (1,965,206 V, 2,766,607 E)	Real graph dataset and synthetics	Hadoop, Giraph, MapReduce, and BSP	Not directly related to our work. However, it shows how to process large graphs using BSP. They used road unweighted graph dataset
Finding interesting information from the graph whether it's a shortest path or pattern matching from the graph [88]	Investigated graph analytics from the perspective of query processing	N/A	N/A	Theoretical evaluations	This work is related to future trends and direction of graph analytics

Categories the edges, minimize the traffic of inner vertices and optimize the direction [89]	Proposed an algorithm that is derived from the delta-stepping and bellman-ford algorithms	R-MAT graph with 2^{38} vertices and 2^{42} edges on 32,768 blue gene/Q nodes. USA road network with 2.7 M vertices and 6.8 M edges. Friendster 63 M vertices and 1.8 B edges. Twitter 41 M vertices and 1.4B edges	Synthetic R-MAT and real-world graphs	GPU and NUMA multicores	Uses SSSP path
Computation of graph-based patterns [90]	Presented a new architecture that allows users to organize the data for parallel computation	10 billion vertices and 200 billion edges	Not clear	PERCH (POWER7+ cluster)	Uses SSSP
Main aim was to accelerate the query efficiency in a Euclidean space by using locality sensitive hashing [91]	Presented an algorithm for fast graph search	NCI1 and NCI109, mutagenicity dataset	The dataset provided related to cancer and chemicals	Empirical evaluations	
Analysis of performance and scalability [92]	Proposed a new technique for parallel graph processing platforms	Cit-patents (3,774,768 vertices and 16,518,948 edges) Dota-league	Synthetic and real-world datasets	Graph-mat, the Graph500, the graph algorithm platform benchmark suite, GraphBIG, and PowerGraph and C	No weighted graph network dataset, have used shortest path computations

We propose an approach for the parallel shortest path computation with multiple queries of a pair of vertices using Apache Spark. In this approach, we have two functions: The One Pair Shortest Path (OPSP) algorithm to find the best route between a pair of vertices, and the main driver program which builds the graph, constructs and parallelizes the queries, and invokes OPSP function. Algorithm 1 (Fig. 8.2) shows the OPSP algorithm. In this algorithm, we employ the concept of the well-known Dijkstra algorithm to find the optimal route between source and destination in a graph problem. This algorithm first explores the neighbor vertices of the current vertex from *distPaths[0]* (the path of minimum distance from *src* to *dest*), inserts the neighbor's vertex id to a set of explored vertices *exp[],* if the neighbor vertices have not been explored in advance, keeps track of explored paths from source to the neighbors (the list of vertices to reach the neighbors) and its distance (*neboursPath.concat(distPathRest)*), picks the path with minimum distance to be explored further (*sortByDist()* ascending), and calls the OPSP function itself (recursive) until the path with minimum distance meets the destination, then will return the minimum distance and the paths to reach the destination (*dist, paths.reverse()*).

Algorithm 2 (Fig. 8.3) shows the main driver program. It builds the graph, the queries, and executes the queries with OPSP algorithm. First, the program builds a graph from the given input of vertices and edges *G(V,E)*. Then, constructs queries *q* from the given input of list of *queries(src,dst)*, which contains multiple pairs of *src* and *dest*. Furthermore, *q* is partitioned with *np* size and becomes *Q(src,dst)*. Afterwards, *Q(src,dst), G(V,E),* and initialization variable *exp[]* as a set of explored vertices, and *distPaths(list(k,v[]))* as an initial step of 0 distance and source vertex are passed to OPSP function in Algorithm 1. Multiple queries of *Q(src,dst)* are executed in parallel by multiple executors in cluster nodes of Spark. Thus, each executor computes different multiple queries at the same time *t*.

8.3.1 Dataset

We have used the DIMACS [94] dataset. The DIMACS is a collection of various datasets. It also has road network dataset containing more than 50 states of the USA and various districts. It is an undirected weighted graph that consists of millions of edges and nodes. We considered in our experiments the entire US dataset. We have also investigated in this chapter results for five different states of the USA. These are District of Columbia (DC), Rhode Island (RI), Colorado (CO), Florida (FL), and California (CA). Each node has node id, latitude, and longitude. Every edge also has source node id, target node id, travel time, distance, and category of road. Table 8.2 shows the number of edges and vertices in different states as well as for the complete US road network. Figure 8.4 graphically displays degree of vertices for selected states and whole US road network.

We also have visualized road network dataset using Gephi [95]. We have only visualized the DC and RI state data set as shown in Figs. 8.5 and 8.6, respectively.

Algorithm 1: One Pair Shortest Path (OPSP)

```
Input  : $G(V, E) \leftarrow$ graph data
         $Q(src, dest) \leftarrow$ list of queries
         $exp[\,] \leftarrow$ init of explored vertices
         $distPaths(list(k, v[\,])) \leftarrow$ init of explored paths with distance
Output: shortest path of $Q(src, dst)$
if $distPaths == null$ then
    return (0, $\emptyset$)
else
    foreach $i \leftarrow distPaths.iterate()$ do
        $minDistPath((k, v[\,])) \leftarrow distPaths[0]$;
        $disPathRest \leftarrow distPaths[1, distPaths.\mathrm{length}()]$;
        $dist \leftarrow minDistPath.k$;
        $paths[\,] \leftarrow minDistPath.v$;
        $head \leftarrow paths[0]$;
        $rest \leftarrow paths[1, paths.\mathrm{length}()]$;
        if $head == Q_i.dst$ then
            return ($dist$, $paths$.reverse())
        else
            $neboursPath \leftarrow list((0, [\,]))$;
            $nb \leftarrow G(head)$;
            $de \leftarrow nb$.distance;
            $ve \leftarrow nb$.vertice;
            foreach $nb.iterate()$ do
                if $exp[\,].contains(ve)$ then
                    continue
                else
                    $neboursPath \leftarrow (dist + de,\ ve.\mathrm{concat}(paths))$
                end
            end
            $combDistPath \leftarrow neboursPath.\mathrm{concat}(distPathRest)$;
            $sortDistPath \leftarrow combDistPath.\mathrm{sortByDist}()$;
            OPSP($G$, $Q$, $exp$.concat($head$), $sortDistPath$);
        end
    end
end
```

Fig. 8.2 The One Pair Shortest Path (OPSP) Algorithm

We could not visualize the other states data due to the large size which cannot be handled on a single PC. We have only visualized two states to perceive the structure of road network datasets. We will look into visualizing larger datasets using Spark in the future.

```
Algorithm 2: Main Function
  Input  : $E \leftarrow$ list of edges
           $V \leftarrow$ list of vertices
           $Q(src, dest) \leftarrow$ list of queries
           $np \leftarrow$ number of partition
  Output: file of shortest path list
1 $G(V, E) \leftarrow$ RDD(V,E);
2 $q \leftarrow$ queries$(src, dst)$ ;
3 $Q(src, dst) \leftarrow$ RDD$(q)$.repartition$(np)$;
4 $SPL \leftarrow OPSP((G(V, E),\ Q(src, dst)), exp[\,], distPaths(list(k, v[\,]))$;
5 $SPL.saveAsFile$;
```

Fig. 8.3 The Master Algorithm

Table 8.2 USA road network dataset

Name of Road Network	Vertices	Edges	Type
District of Columbia (DC)	9559	14,909	Undirected
Rhode Island (RI)	53,658	69,213	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

8.4 Results and Discussion

For experimental setup, we have built a Spark cluster setup on the Aziz supercomputer [34]. In this configuration, we have used different number of nodes, varying from one to sixteen. We have used Apache Hadoop HDFS to store input and output data. Apache Spark has been used for the data processing. The Master and Salve Spark nodes used on the Aziz supercomputer have the following configuration.

- Linux CentOS, JDK 1.7, Dual Socket Intel Xeon E5-2695v2 12-core processor, 2.4 GHz, Total 24 cores, 96GB RAM, Apache Spark 2.0.1, GraphX Apache Hadoop HDFS.

8.4.1 Single Shortest Path Query Results

In our earlier work [33], we had presented results for a single shortest path query on up to 16 nodes (368 cores) for the USA states DC, RI, CO, FL, CA, and the whole US road network with up to over 23 million vertices and 58 million edges (see Table 8.2). See [33] for the detailed results and analysis.

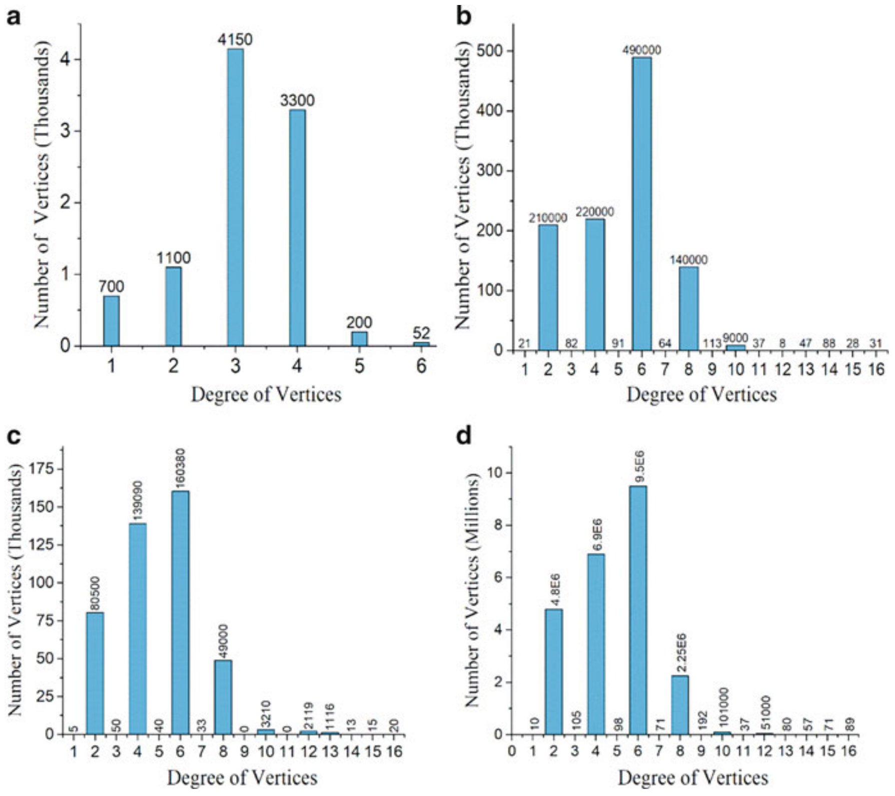

Fig. 8.4 Visualization of (**a**) District of Columbia road network (**b**) Florida road network (**c**) Colorado road network (**d**) Whole US road network

Fig. 8.5 District of Columbia road network

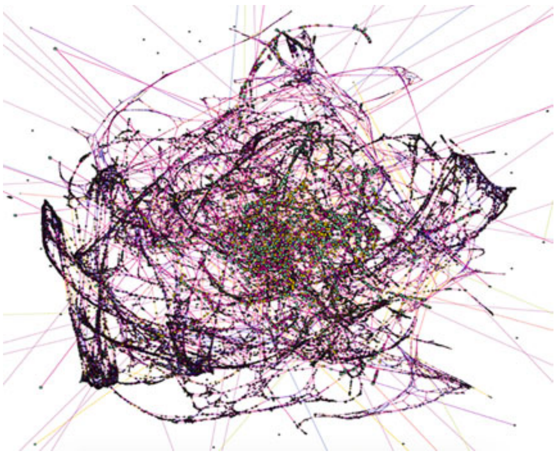
Fig. 8.6 Rhode Island road network

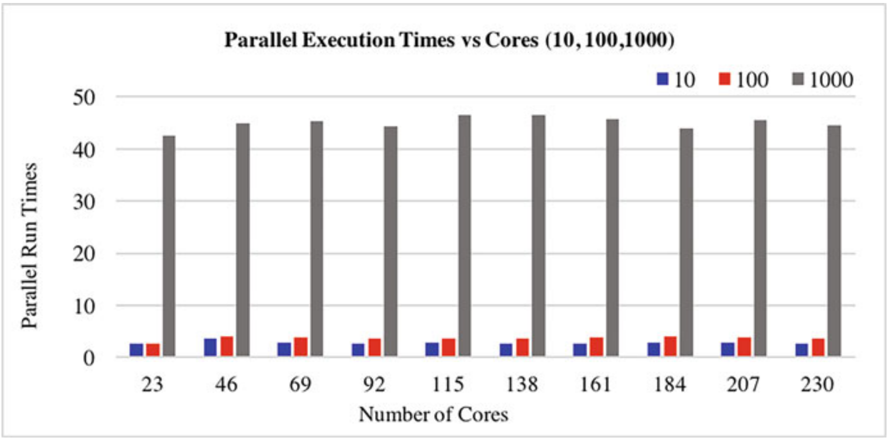

Fig. 8.7 Parallel execution time of varying number of cores

8.4.2 *Multiple Shortest Path Query Results*

The aim here is to investigate and achieve high performance in finding the shortest path of multiple queries with our proposed parallel-shortest path algorithm between the source and the target. Using Spark, we run in parallel a varying number of queries, each computing shortest path between a (source, destination) pair; see Sect. 8.3 for details. In these experiments, we use Rhode Island (RI) road network, USA, which consists of 53,658 vertices and 69,213 edges.

The results in Fig. 8.7 show that parallelization does not have a significant impact when executing a small number of queries. This is because the job is too small compared to the number of cores. It has an ineffective job distribution and takes a long time for I/O overhead among the cores which are distributed among up to 10 nodes (with 24 cores each). Three different queries are used in the figure: 10, 100, and 1000 queries. The horizontal axis shows results for varying number of cores: 23, 46, up to 230. Each Aziz node contains 24 cores. However, we keep one core for the operating system to perform its job. Thus, we utilize 23 cores for each node. The vertical axis gives the total runtime to compute the whole sets of queries.

A larger number of queries (10 K, 100 K, and 1 M) show a clear reduction and advantage in execution time while parallelizing the whole sets of queries as shown in Figs. 8.8 and 8.9. As usual the letter K denotes a thousand and M indicates a million.

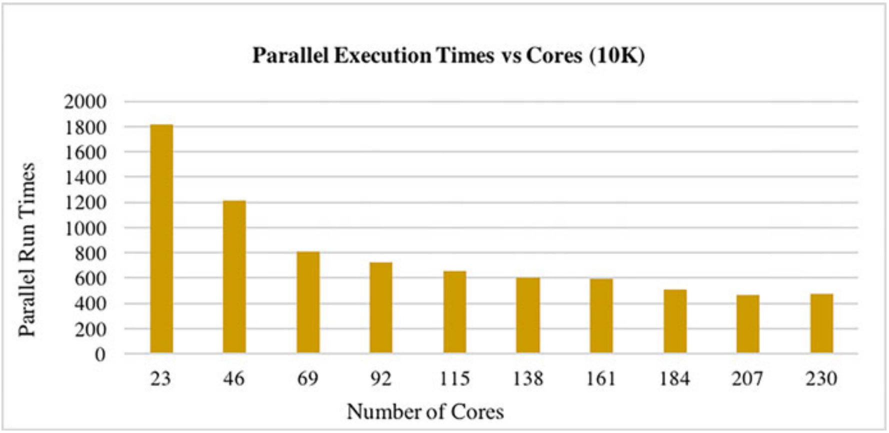

Fig. 8.8 Parallel execution time of varying number of nodes

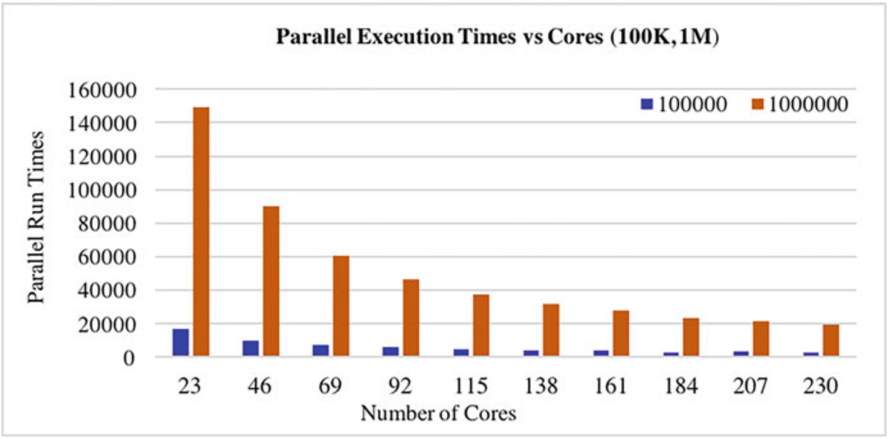

Fig. 8.9 Parallel execution time of varying number of nodes

8.4.3 Speedup

According to the experimental results in Sect. 8.4.2, we have calculated the achieved speedup. Figure 8.10 depicts that the achieved speedup is increasing with the increasing number of cores: 46 to 230. The figure depicts the speedups for six different query set sizes: 10, 100, 1 K, 10 K, 100 K, and 1 M. Note that the speedups for smaller computations get saturated for a smaller number of nodes compared to, for example, for larger query set of 1 M. The speedup is measured by using the following well-known formula.

$$Sp = \frac{T_s}{T_p}$$

Sp denotes the achieved speedup, while *Ts* denotes the execution time of the sequential computation, and *Tp* denotes the execution time of parallel computation.

8.4.4 Relative Speedup

To further elaborate the speedup saturation for increasing query set sizes and the number of cores, we now investigate relative speedup, the core-based speedup. The gained relative speedup is quite stable for large number of queries (1 M), and it is fluctuating for 100 K queries, as shown in Fig. 8.11. Whereas, for small queries less than 10 K, the relative speedup is decreasing. The following formula is used to calculate the relative speedup.

$$\text{Relative speedup} = \frac{Sp}{NC}$$

Sp and *NC* indicate the achieved speedup and the number of used cores, respectively.

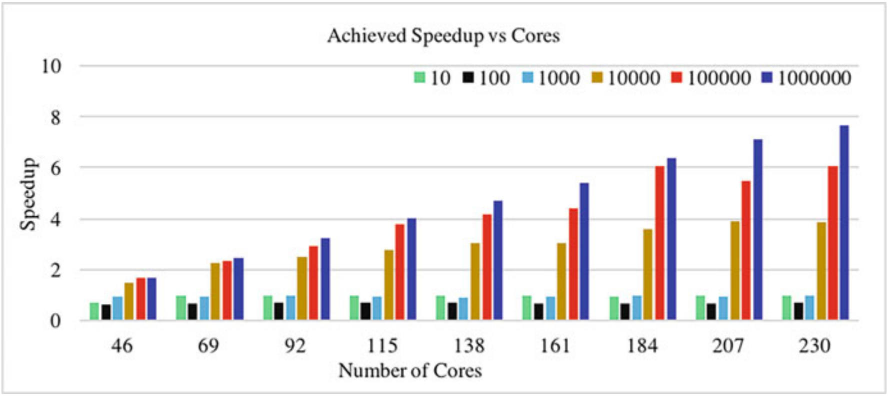

Fig. 8.10 Achieved speedup with different number of cores

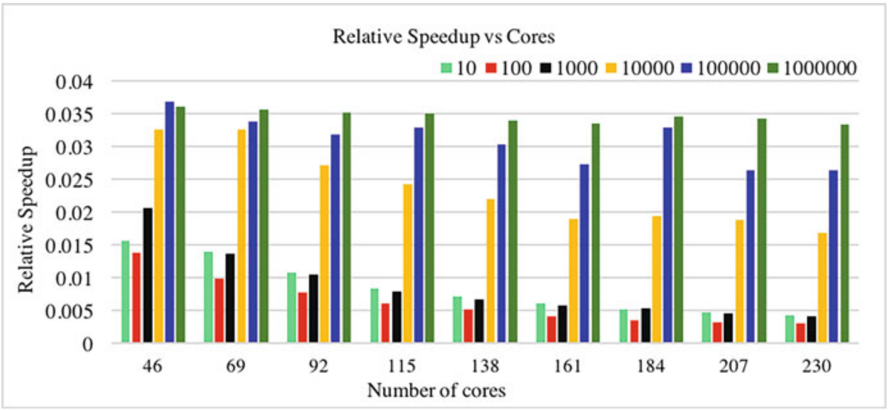

Fig. 8.11 Achieved relative speedup with different number of Aziz nodes

8.5 Conclusion

Smart applications and infrastructures are increasingly relying on graph computations to model real-life problems and process big data. The emergence of many graph-based software, programming languages, graph databases, and benchmarks, and their use in application domains provide the evidence for the increasing popularity of graph-based computing. In this chapter, we have our earlier work on single source shortest path computations of big data road network graphs using Apache Spark. In our earlier work [33], we had used the US road network data modelled as graphs and calculated shortest paths between two vertices over a varying number of up to 368 compute cores. The experiments were performed on the Aziz supercomputer (a former Top500 machine [34]). We had analyzed Spark's parallelization behavior by solving problems of varying graph sizes, i.e., various states of the USA with up to over 23 million vertices and 58 million edges. We call our system the Big Data Shortest Path Graph Computing (BDSPG) system.

In this chapter, we have focused on computing a set of large varying number of shortest path queries on a (source, destination) vertex pair. The number of queries used were 10, 100, 1 K, 10 K, 100 K, and 1 M, executed over up to 230 CPU cores. We achieved good performance, and as expected, the speedup is dependent on both the size of the data and the number of parallel nodes. In addition to the extended results, we have provided a detailed literature on shortest path graph computations. The system architecture for graph computing in Spark was explained with additional details using the architecture depiction and elaborated algorithms.

Future work will look into improving algorithms for sequential shortest path algorithm and its parallelization including data locality. There is a need for further performance analysis of our proposed system. We wish to apply the BDSPG system to the smart city case studies developed in [5, 6, 55].

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-651-611-38. The experiments reported in this chapter 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 Recogn. **35**, 639–650 (2002)
3. Ding, Y., Yan, S., Zhang, Y., 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 '16. pp. 1–7. ACM Press, New York, New York, USA (2016)
5. 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. **37**, 75–104 (2017)
6. Mehmood, R., Graham, G.: Big Data Logistics: A health-care Transport Capacity Sharing Model. In: Procedia Computer Science. pp. 1107–1114 (2015)
7. Mehmood, R., Bhaduri, B., Katib, I., Chlamtac, I. (eds.): Smart Societies, Infrastructure, Technologies and Applications, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (LNICST), Volume 224. Springer International Publishing, Cham (2018)
8. El-Gorashi, T.E.H., Pranggono, B., Mehmood, R., Elmirghani, J.M.H.: A data mirroring technique for SANs in a metro WDM sectioned ring. In: ONDM 2008 - 12th Conference on Optical Network Design and Modelling (2008)
9. Ayres, G., Mehmood, R., Mitchell, K., Race, N.J.P.: Localization to enhance security and services in Wi-Fi networks under privacy constraints. In: Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, Volume 16. pp. 175–188. Springer (2009)
10. El-Gorashi, T.E.H., Pranggono, B., Mehmood, R., Elmirghani, J.M.H.: A mirroring strategy for SANs in a metro WDM sectioned ring architecture under different traffic scenarios. J. Opt. Commun. **29**, 89–97 (2008)
11. Mehmood, R., Pranggono, B., El-Gorashi, T., Elmirghani, J.: Performance evaluation of a metro WDM slotted ring network with san extension. In: Proceedings of the 7th IASTED International Conferences on Wireless and Optical Communications, WOC 2007. pp. 231–236 (2007)
12. Mehmood, R., Alturki, R., Faisal, M.: A Scalable Provisioning and Routing Scheme for Multimedia QoS over Ad Hoc Networks. (2009)
13. Mehmood, R., Alturki, R.: Video QoS analysis over wi-fi networks. Adv. Video Commun. over Wirel. Networks. 439–480 (2013)
14. Alturki, R., Mehmood, R.: Cross-Layer Multimedia QoS Provisioning over Ad Hoc Networks. Using Cross-Layer Tech. Commun. Syst. Tech. Appl. IGI Glob. Hershey, PA. 460–499 (2012)
15. Hendrickson, B., Kolda, T.G.: Graph partitioning models for parallel computing. Parallel Comput. **26**, 1519–1534 (2000)
16. Mehmood, R., Crowcroft, J.: Parallel iterative solution method for large sparse linear equation systems. Technical Report Number UCAM-CL-TR-650, Computer Laboratory, University of Cambridge, Cambridge, UK (2005)

17. 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, The Netherlands (2004)
18. Mehmood, R.: Disk-based Techniques for Efficient Solution of Large Markov Chains, PhD Thesis, School of Computer Science, University of Birmingham, (2004)
19. Mehmood, R., Parker, D., Kwiatkowska, M.: An efficient BDD-based implementation of Gauss-Seidel for CTMC analysis. Technical report CSR-03-13, School of Computer Science, University of Birmingham, Birmingham, UK (2013)
20. Eleliemy, A., Fayze, M., Mehmood, R., Katib, I., Aljohani, N.: Loadbalancing on Parallel Heterogeneous Architectures: Spin-image Algorithm on CPU and MIC. In: EUROSIM 2016, The 9th Eurosim Congress on Modelling and Simulation. p. 6. Oulu, Finland (2016)
21. Schlingensiepen, J., Mehmood, R., Nemtanu, F.C., Niculescu, M.: Increasing sustainability of road transport in European cities and metropolitan areas by facilitating autonomic road transport systems (ARTS). In: Wellnitz, J., Subic, A., Trufin, R. (eds.) Sustainable Automotive Technologies 2013 Proceedings of the 5th International Conference ICSAT 2013, pp. 201–210. Springer International Publishing, Ingolstadt, Germany (2014)
22. Junghanns, M., Petermann, A., Neumann, M., Rahm, E.: Management and analysis of big graph data: current systems and open challenges. In: handbook of big data technologies. Pp. 457–505. Springer international publishing, Champions (2017)
23. Altowaijri, S., Mehmood, R., Williams, J.: A quantitative model of grid systems performance in healthcare organisations. In: ISMS 2010 - UKSim/AMSS 1st International Conference on Intelligent Systems, Modelling and Simulation. pp. 431–436 (2010)
24. Tawalbeh, L.A., Bakhader, W., Mehmood, R., Song, H.: Cloudlet-based mobile cloud computing for healthcare applications. In: 2016 IEEE Global Communications Conference, GLOBECOM 2016 - Proceedings (2016)
25. Muhammed, T., Mehmood, R., Albeshri, A., Katib, I.: UbeHealth: a personalized ubiquitous cloud and edge-enabled networked healthcare system for smart cities. IEEE Access. **6**, 32258–32285 (2018)
26. 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)
27. Mehmood, R., Faisal, M.A., Altowaijri, S.: Future networked healthcare systems: a review and case study. In: Boucadair, M., Jacquenet, C. (eds.) Handbook of Research on Redesigning the Future of Internet Architectures, pp. 531–558. IGI Global, Hershey, PA (2015)
28. 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)
29. Xin, R.S., Gonzalez, J.E., Franklin, M.J.: GraphX: A Resilient Distributed Graph System on Spark
30. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: GraphX: Graph Processing in a Distributed Dataflow Framework
31. Apache Spark GraphX, https://spark.apache.org/graphx/
32. Apache Spark, https://spark.apache.org/
33. Arfat, Y., Mehmood, R., Albeshri, A.: Parallel shortest path graph computations of United States road network data on apache spark. In: Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, Volume 224. pp. 323–336. Springer, Cham (2018)
34. Aziz Supercomputer, Top500, https://www.top500.org/site/50585
35. Büscher, M., Coulton, P., Efstratiou, C., Gellersen, H., Hemment, D., Mehmood, R., Sangiorgi, D.: Intelligent mobility systems: Some socio-technical challenges and opportunities. In: Communications Infrastructure. Systems and Applications in Europe, Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST 16. pp. 140–152 (2009)

36. Ayres, G., Mehmood, R.: On discovering road traffic information using virtual reality simulations. In: 11th International Conference on Computer Modelling and Simulation, UKSim 2009. pp. 411–416 (2009)
37. Mehmood, R.: Towards understanding intercity traffic interdependencies. In: 14th World Congress on Intelligent Transport Systems, ITS 2007. pp. 1793–1799. ITS America, Beijing (2007)
38. Ayres, G., Mehmood, R.: LocPriS: A security and privacy preserving location based services development framework. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), LNAI, Volume 6279, Part 4. pp. 566–575. Springer (2010)
39. Elmirghani, J.M.H., Badic, B., Li, Y., Liu, R., Mehmood, R., Wang, C., Xing, W., Garcia Zuazola, I.J., Jones, S.: IRIS: An inteligent radio-fibre telematics system. In: Proceedings of the 13th ITS World Congress, London, 8–12 October (2006)
40. 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)
41. Suma, S., Mehmood, R., Albeshri, A.: Automatic event detection in smart cities using big data analytics. In: International Conference on Smart Cities, Infrastructure, Technologies and Applications (SCITA 2017): Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, Volume 224. pp. 111–122. Springer, Cham (2018)
42. Alomari, E., Mehmood, R.: Analysis of tweets in Arabic language for detection of road traffic conditions. In: Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, Volume 224. pp. 98–110. Springer, Cham (2018)
43. Mehmood, R., Nekovee, M.: Vehicular Ad hoc and grid networks: Discussion, design and evaluation. In: 14th World Congress on Intelligent Transport Systems, ITS 2007. pp. 1555–1562. ITS America, Beijing (2007)
44. Gillani, S., Shahzad, F., Qayyum, A., Mehmood, R.: A survey on security in vehicular ad hoc networks. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 59–74 (2013)
45. Alvi, A., Greaves, D., Mehmood, R.: Intra-vehicular verification and control: A two-pronged approach. In: 7th IEEE International Symposium on Communication Systems, Networks and Digital Signal Processing, CSNDSP 2010. pp. 401–405 (2010)
46. Schlingensiepen, J., Nemtanu, F., Mehmood, R., McCluskey, L.: Autonomic Transport Management Systems—Enabler for Smart Cities, Personalized Medicine, Participation and Industry Grid/Industry 4.0. In: Intelligent Transportation Systems – Problems and Perspectives, Volume 32 of the series Studies in Systems, Decision and Control. pp. 3–35. Springer International Publishing (2016)
47. Schlingensiepen, J., Mehmood, R., Nemtanu, F.C.: Framework for an autonomic transport system in smart cities. Cybern. Inf. Technol. **15**, 50–62 (2015)
48. Alam, F., Mehmood, R., Katib, I.: D2TFRS: An object recognition method for autonomous vehicles based on RGB and spatial values of pixels. In: Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, Volume 224. pp. 155–168. Springer, Cham (2018)
49. Alazawi, Z., Altowaijri, S., Mehmood, R., Abdljabar, M.B.: Intelligent disaster management system based on cloud-enabled vehicular networks. In: 2011 11th International Conference on ITS Telecommunications, ITST 2011. pp. 361–368. IEEE (2011)
50. Alazawi, Z., Abdljabar, M.B., Altowaijri, S., Vegni, A.M., Mehmood, R.: ICDMS: An intelligent cloud based disaster management system for vehicular networks. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), LNCS, Volume 7266. pp. 40–56. Springer, Vilnius, Lithuania (2012)
51. Alazawi, Z., Alani, O., Abdljabar, M.B., Altowaijri, S., Mehmood, R.: A smart disaster management system for future cities. In: Proceedings of the 2014 ACM international workshop on Wireless and mobile technologies for smart cities - WiMobCity ‘14. pp. 1–10. ACM Press, New York, New York, USA (2014)

52. Alazawi, Z., Alani, O., Abdljabar, M.B., Mehmood, R.: An intelligent disaster management system based evacuation strategies. In: 2014 9th International Symposium on Communication Systems, Networks and Digital Signal Processing, CSNDSP 2014. pp. 673–678 (2014)
53. Alazawi, Z., Alani, O., Abdljabar, M.B., Mehmood, R.: Transportation evacuation strategies based on VANET disaster management system. Procedia Econ. Financ. **18**, 352–360 (2014)
54. Aqib, M., Mehmood, R., Albeshri, A., Alzahrani, A.: Disaster management in smart cities by forecasting traffic plan using deep learning and GPUs. In: Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, Volume 224. pp. 139–154 (2018)
55. Mehmood, R., Lu, J.A.: Computational Markovian analysis of large systems. J. Manuf. Technol. Manag. **22**, 804–817 (2011)
56. Arfat, Y., Aqib, M., Mehmood, R., Albeshri, A., Katib, I., Albogami, N., Alzahrani, A.: Enabling Smarter Societies through Mobile Big Data Fogs and Clouds. In: Procedia Computer Science (2017), 109, 1128
57. Quddus, M., Washington, S.: Shortest path and vehicle trajectory aided map-matching for low frequency GPS data. Transp. Res. Part C Emerg. Technol. **55**, 328–339 (2015)
58. Szucs, G.: Decision support for route search and optimum finding in transport networks under uncertainty. J. Appl. Res. Technol. **13**, 125–134 (2015)
59. Feng, L., Lv, Z., Guo, G., Song, H.: Pheromone based alternative route planning. Digit. Commun. Networks. **2**, 151–158 (2016)
60. Zeng, W., Church, R.L.: Finding shortest paths on real road networks: the case for a *. Int. J. Geogr. Inf. Sci. **8816**, (2017)
61. Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: A System for Large-Scale Graph Processing. Proc. 2010 ACM SIGMOD Int. Conf. Manag. data. 135–145 (2010)
62. Yan, J., Tan, G., Mo, Z., Sun, N.: Graphine: programming graph-parallel computation of large natural graphs for multicore clusters. IEEE Trans. Parallel Distrib. Syst. **27**, 1647–1659 (2016)
63. Selim, H., Zhan, J.: Towards shortest path identification on large networks. J. Big Data. **3**, (2016)
64. Zhou, X., Chang, P., Chen, G.: An Efficient Graph Processing System. Asia-Pacific Web Conf. LNCS. 401–412 (2014)
65. Cao, Z., Guo, H., Zhang, J., Niyato, D., Fastenrath, U.: Finding the shortest path in stochastic vehicle routing: a cardinality minimization approach. IEEE Trans. Intell. Transp. Syst. **17**, 1688–1702 (2016)
66. Hou U, L., Zhao, H.J., Yiu, M.L., Li, Y., Gong, Z.: Towards online shortest path computation. IEEE Trans. Knowl. Data Eng. 26, 1012–1025 (2014)
67. Strehler, M., Merting, S., Schwan, C.: Energy-efficient shortest routes for electric and hybrid vehicles. Transp. Res. Part B Methodol. **103**, 111–135 (2017)
68. Hong, I., Murray, A.T., Rey, S.: Obstacle-avoiding shortest path derivation in a multicore computing environment. Comput. Environ. Urban. Syst. **55**, 1–10 (2016)
69. Mozes, S., Nussbaum, Y., Weimann, O.: Faster shortest paths in dense distance graphs, with applications. Theor. Comput. Sci. **1**, 1–25 (2014)
70. Abraham, I., Goldberg, A. V, Werneck, R.F.: A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. Springer-Verlag Berlin Heidelb. 2011. 230–241 (2011)
71. Sanders, P., Schultes, D.: Highway hierarchies hasten exact shortest path queries. Algorithms–Esa 2005. 568–579 (2005)
72. Peng, S., Sankaranarayanan, J., Samet, H.: SPDO: High-throughput road distance computations on Spark using Distance Oracles. 2016 IEEE 32nd Int. Conf. Data Eng. ICDE 2016. 1239–1250 (2016)
73. Zhu, A.D., Ma, H., Xiao, X., Luo, S., Tang, Y., Zhou, S.: Shortest Path and Distance Queries on Road Networks: Towards Bridging Theory and Practice. 857–868 (2013)
74. Zheng, C.Y., Wang, J.: All-Pairs Shortest Paths in Spark

75. Djidjev, H., Chapuis, G., Andonov, R., Thulasidasan, S., Lavenier, D.: All-pairs shortest path algorithms for planar graph for GPU-accelerated clusters. J. Parallel Distrib. Comput. **85**, 91–103 (2015)
76. Aridhi, S., Lacomme, P., Ren, L., Vincent, B.: A MapReduce-based approach for shortest path problem in large-scale networks. Eng. Appl. Artif. Intell. **41**, 151–165 (2015)
77. Faro, A., Giordano, D.: Algorithms to find shortest and alternative paths in free flow and congested traffic regimes. Transp. Res. Part C Emerg. Technol. **73**, 24–28 (2016)
78. Kajdanowicz, T., Kazienko, P., Indyk, W.: Parallel processing of large graphs. Futur. Gener. Comput. Syst. **32**, 324–337 (2014)
79. Liu, X., Zhou, Y., Guan, X., Sun, X.: A feasible graph partition framework for random walks implemented by parallel computing in big graph. Chinese Control Conf. CCC. 2015–Septe, 4986–4991 (2015)
80. 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)
81. 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)
82. 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)
83. 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)
84. Aridhi, S., Mephu Nguifo, E.: Big graph mining: frameworks and techniques. Big Data Res. **6**, 1–10 (2016)
85. 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)
86. 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)
87. Mohan, A., G, R.: A Review on Large Scale Graph Processing Using Big Data Based Parallel Programming Models. Int. J. Intell. Syst. Appl. 9, 49–57 (2017)
88. Miller, J.A., Ramaswamy, L., Kochut, K.J., Fard, A.: Research Directions for Big Data Graph Analytics. Proc. - 2015 IEEE Int. Congr. Big Data, BigData Congr. 2015. 785–794 (2015)
89. Chakaravarthy, V.T., Checconi, F., Petrini, F., Sabharwal, Y.: Scalable single source shortest path algorithms for massively parallel systems. Proc. Int. Parallel Distrib. Process. Symp. IPDPS. **28**, 889–901 (2014)
90. 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. Proc. 2014 IEEE BigData Conf. 942–951 (2014)
91. Zhang, M., Shen, F., Zhang, H., Xie, N., Yang, W.: Fast Graph Similarity Search via Locality Sensitive Hashing. Adv. Multimed. Inf. Process. PCM 2015. 9315, 447–455 (2015)
92. Pollard, S., Norris, B.: A Comparison of Parallel Graph Processing Benchmarks. (2017)
93. GraphX | Apache Spark
94. DIMACS Implementation Challenge, http://www.dis.uniroma1.it/challenge9/download.shtml
95. Gephi - The Open Graph Viz Platform, https://gephi.org/