

An innovative fractal architecture model for implementing MapReduce in an open multiprocessing parallel environment

Muslim Mohsin Khudhair^{1,3}, Adil Al-Rammahi², Furkan Rabee³

¹Department of Computer Information System, College of Computer Science and Information Technology, University of Basrah, Basrah, Iraq

²Department of Mathematical Science, Faculty of Computer Science and Mathematics, University of Kufa, Najaf, Iraq

³Department of Computer Science, Faculty of Computer Science and Mathematics, University of Kufa, Najaf, Iraq

Article Info

Article history:

Received Nov 11, 2022

Revised Dec 25, 2022

Accepted Dec 28, 2022

Keywords:

Cloud computing

Cube model

Fractal

MapReduce

Open multiprocessing

Shared-memory

ABSTRACT

One of the infrastructure applications that cloud computing offers as a service is parallel data processing. MapReduce is a type of parallel processing used more and more by data-intensive applications in cloud computing environments. MapReduce is based on a strategy called "divide and conquer," which uses regular computers, also called "nodes," to do processing in parallel. This paper looks at how open multiprocessing (OpenMP), the best shared-memory parallel programming model for high-performance computing, can be used with the proposed fractal network model in the MapReduce application. A well-known model, the cube, is used to compare the fractal network model and its work. Where experiments demonstrated that the fractal model is preferable to the cube model. The fractal model achieved an average speedup of 2.7 and an efficiency rate of 67.7%. In contrast, the cube model could only reach an average speedup of 2.5 and an efficiency rate of 60.4%.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Muslim Mohsin Khudhair

Department of Computer Information System, College of Computer Science and Information Technology

University of Basrah

Basrah, Iraq

Email: muslim.khudhair@uobasrah.edu.iq

1. INTRODUCTION

Cloud computing is a new technology growing quickly in the information technology (IT) industry. It uses different ideas and technologies, such as virtualization, processing power, storage, sharing, distributed networks, and connectivity, to its advantage. Cloud-based services are now one of the best ways for users and businesses to get on-demand services and unlimited storage [1]. One of the most important things about a cloud-based platform is that it makes it easy to quickly process large data sets in cloud applications. To deploy big data applications on a cloud data centre network (DCN), the key challenges are volume, velocity, and variety. Volume is the amount of data, variety is the number of data types, and velocity is the speed at which data is processed [2], [3].

Big data is becoming an increasingly important way for a business to enhance its value proposition or the efficiency of its operations. Due to the sheer amount of data, big data relies heavily on parallel computing technology to finish processing data quickly. Many specialized programming models and runtime systems have been created to support big data [4]. The map/reduce approach is utilized by Hadoop and Spark [5]. GraphLab, Giraph, and GraphX use the Pregel model [6]. Storm can deal with flowing data [7]. Each system uses threads

and basic ways to establish a parallel and distributed computing environment. High-performance computing experts created programming models that make designing systems (like the ones above) easier and balance programming effort and performance.

Parallel computer architecture uses shared memory, distributed memory, and hybrid systems. Shared memory systems, such as open multiprocessing (OpenMP), can give all processors access to the same memory so processors don't have to communicate with each other. A distributed memory system, like message passing interface (MPI), is distinguished by its high-speed connection, which moves data between the system's nodes. Hybrids connect nodes with high-speed connections and share memory [8]. So, the main goal of this research is to find a way to use the OpenMP program to simulate MapReduce by using the fractal network models shown to meet the needs of processing big data.

The rest of this work is put together in the following way. In section 2, the main concepts behind MapReduce are explained. In section 3, the basic ideas of fractals are clarified. These ideas will be the basis for the suggested architectural framework. In section 4, OpenMP demonstrates the basic concepts and knowledge of the parallel computing environment. Section 5, the central part of this paper, describes the given fractal model in detail. Section 6 provides the details of the experiments and an analysis of the results. The research results are discussed in the last part of the paper, section 7.

2. MAPREDUCE FRAMEWORK

MapReduce is a parallel and distributed programming model created by Google. It is a powerful software that supports data-intensive applications because it can handle errors, is easy to use, and can grow as needed [9]. On the other side, Apache/Hadoop is an open-source framework licensed by Apache and the most well-known way to use MapReduce. It comprises two parts: Hadoop distributed file system (HDFS), which stores files, and MapReduce, which processes them. The HDFS is a scalable file system, has a high throughput, and can handle errors. Figure 1 shows the main tasks of MapReduce, which can be summed up in the following way [1], [10]:

- Mappers process the input data, which is read from the HDFS file system, and produce output pairs of "key/value."
- During the shuffling step, the outputs of the mappers are redirected to reducer nodes by the values of their respective "keys."
- The data that has been shuffled is processed by the reducers, providing the final outputs.

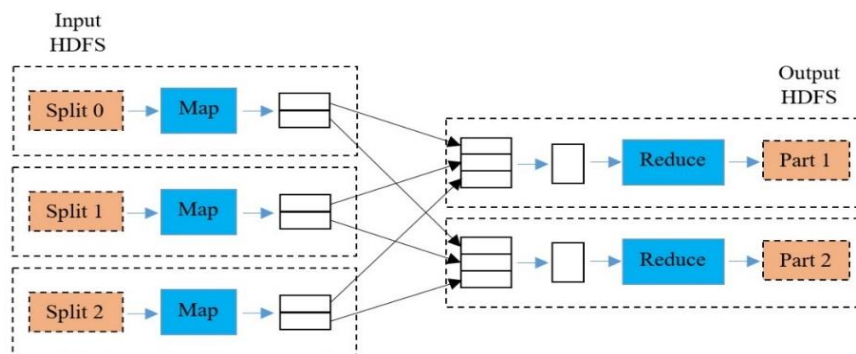


Figure 1. Main steps of the MapReduce framework

3. THE FUNDAMENTALS OF FRACTALS AND MEASUREMENTS

Fractals are patterns and structures based on geometry that may repeat themselves of any size, from the smallest to the largest. It is commonly known that fractals may describe structures and surfaces that cannot be defined by traditional Euclidean geometry [11], [12]. Fractals weren't studied widely until computer simulations improved and made it possible to make artificial and mathematical fractals and automatically make fractal dimensions' estimations easy [13].

Fractal dimension is a (usually non-integer) number that can be assigned to any natural, random, or fabricated fractal to calculate or quantify the complexity of the fractal concerning the space it lives in [13]–[15]. The following is the formula for computing the similarity dimension, often known as D_s , for self-similar fractals that consist of N copies, all of which are scaled by the same factor r :

$$D_S = \frac{\log(N)}{\log(1/r)} \tag{1}$$

3.1. Sierpinski triangle

The Sierpinski triangle is one famous fractal shape which builds via an iterative methodology [16], [17]. Level 0 consists of a single triangle, as seen in Figure 2, and is the first available level. If the triangle in the middle is removed, three additional triangles will be the same as in level 1. This results in the formation of four triangles, three of which are filled with the black color and one of which is blank. Continuing to remove triangles from the filled triangle, it will advance to the second level. On level two, nine triangles are black, and four blank triangles. In the same way, it follows to get to the third level [17], [18].

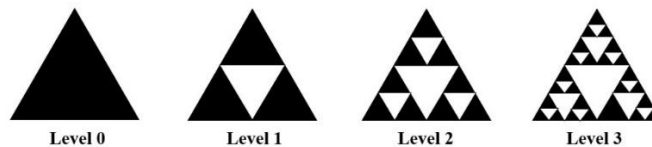


Figure 2. Four levels of Sierpinski triangle

4. OPEN MULTI-PROCESSING (OPENMP)

OpenMP is often used on symmetric multiprocessor platforms (SMPs), which have a CPU with many processing cores. It works best for fine-grained parallel computing. OpenMP-based computing is fast and has low latency [19], [20]. The compiler directives make it possible for the C/C++ and Fortran programming languages to use loop-based parallelization, tasking, work sharing, and synchronization. From a programming point of view, the OpenMP model is better because it doesn't take too much work to make sequential programs run simultaneously [21].

OpenMP's primary parallelization technique is known as multithreading. In this technique, a single master thread splits out several slave threads, as seen in Figure 3. The instructions in the serial environment are carried out one after the other by the master thread. In contrast, the instructions in the parallel environment are carried out simultaneously and independently by the slave threads [22], [23].

Because OpenMP uses a shared memory model, every thread can access the global memory by default. Slave threads can communicate by reading and writing to the global memory. When they update global or shared memory simultaneously, it can lead to a race condition that changes depending on how the threads are scheduled. When two or more threads access the same memory without the necessary synchronization, this is called a "data race condition." Additionally, OpenMP enables the parallelization of parallel areas, allowing parallel loops to be nested inside parallel loops. When this happens, the slave threads that were made spawn more threads to make the team [24], [25].

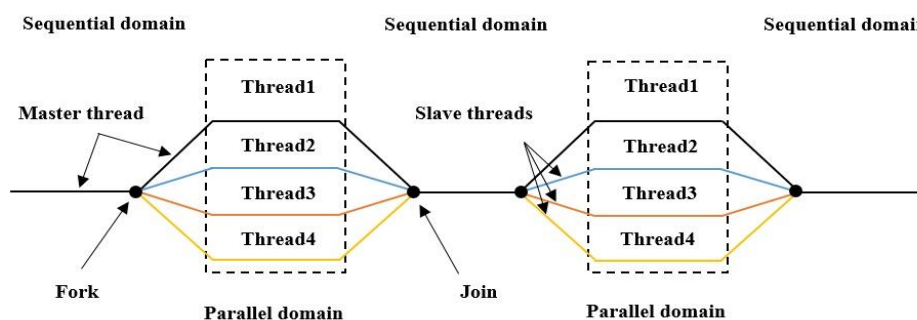


Figure 3. A fork-and-join approach of OpenMP parallelism

4.1. Scheduling methods

Scheduling work becomes the most important part of parallelization. OpenMP gives you several ways to set up a schedule, which are [26], [27]:

- The static schedule works best if all the iterations take the same time to compute.

- Through dynamic scheduling, a minimal amount of work is assigned to each thread, and after that work is completed, the thread receives more jobs. It, however, aids in a better load-balancing of work among the OpenMP threads when a loop's iterations do not have an evenly distributed demand.
- The guided schedule determines the chunk size based on the number of unallocated iterations. So, the first chunks of the threads are bigger. As the number of allocated iterations increases, the chunk size decreases.
- The auto schedule lets the compiler and the runtime decide how to schedule things. The behaviour of the automatic schedule will differ depending on the implementation-specific.
- The runtime schedule lets a program wait until runtime to decide which OpenMP schedule to use. Schedule and chunk-size options can be chosen at runtime.

4.2. Speedup and efficiency in parallel computing

Speedup and parallel efficiency measure how well a parallel algorithm works on a parallel architecture. The equations listed below are utilized in the process of determining them [25], [27]:

$$S_{Parallel} = \frac{T_{Sequential}}{T_{Parallel}} \quad (2)$$

$$E_{Parallel} = \frac{S_{Parallel}}{N_{Parallel}} \quad (3)$$

where S (*parallel*) is the parallel speedup, T (*sequential*) is the time it took for the sequential program to run, T (*parallel*) is the time it took for the parallel program to run, and E (*parallel*) is the efficiency of parallel processing, and N (*parallel*) is the number of processors used for parallel processing.

5. THE PROPOSED METHODOLOGIES

This section will simulate the fractal architecture model based on the recursive fractal Sierpinski triangle (Gasket) formula described in sub-section 3.1. These models system places the processing elements at the vertices (nodes) of the graph. The word count will be used here as an example of how efficiently the model's network work. Word count is a typical use of MapReduce, which reads text files and counts how many times each word appears. The number of lines in the text input file is counted right at the beginning of the test model. The file is then split into chunks, each with a certain number of text lines. This MapReduce will be implemented using the fractal architecture model within the OpenMP environment.

5.1. The fractal architecture model

The following is a comprehensive explanation of the methods involved in the design of the model by the following steps:

- Input text.
- Counting the number of lines of the entered text data file.
- Splitting the lines of the text file into equal chunks for distribution to processing nodes, as in Table 1.
- Distributing the chunks to the allocated first three nodes (1, 2, and 3) so that each node represents a processing unit that performs the mapping and shuffling process, as in Figure 4.
- The OpenMP environment's parallel processing works by having each node in the preceding phase designate a free thread to perform the necessary processing (mapping and shuffling).
- The outputs (value, keys) consider inputs to the rest nodes (4, 5, and 6) that perform the reducing process.
- Under the OpenMP environment parallel processing, each node in the previous step assigns a free thread to perform the required processing (reducing).
- Finally, the output completes the MapReduce operation to calculate the number of word repetitions for the input text file.

Table 1. Distributing the input text file chunks to processing nodes

Node 1	Node 2	Node 3
chunk 1	chunk 2	chunk 3
chunk 4	chunk 5	chunk 6
chunk 7	chunk 8	chunk 9
...
..	..	chunk n

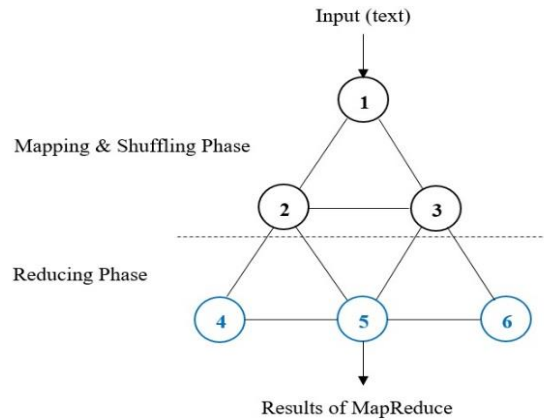


Figure 4. The fractal architecture model for implementing MapReduce

5.2. The comparison cube model

For evaluation and investigation, it utilizes a well-known network model called the "cube model" to compare and contrast the performance of the recently presented fractal model, as shown in Figure 5. This model evaluates using the same process methodology used in the model proposed. But the difference is that the file lines are split on the first four nodes (1, 2, 3, and 4), which do the Mapping and Shuffling process, as shown in Table 2. At the same time, the outputs of the previous nodes are used as inputs by the remaining nodes (5, 6, 7, and 8) to complete the Reducing process.

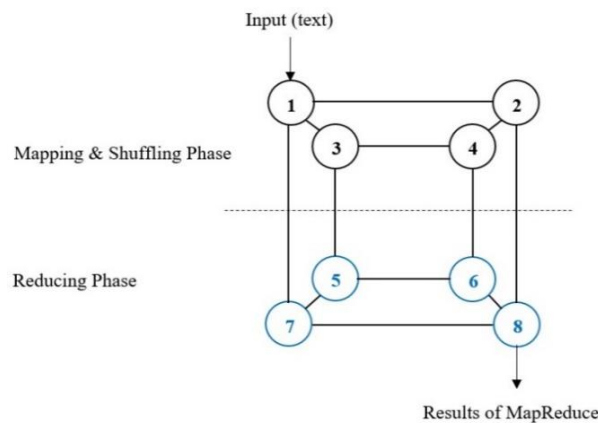


Figure 5. The cube architecture model for implementing MapReduce

Table 2. Distributing the input text file chunks to processing nodes for a cube model

Node 1	Node 2	Node 3	Node 4
chunk 1	chunk 2	chunk 3	chunk 4
chunk 5	chunk 6	chunk 7	chunk 8
chunk 9	chunk 10	chunk 11	chunk 12
...
..	chunk n

5.3. Algorithm of fractal architecture model

The algorithm of the fractal architecture model takes an input data file and returns an output file for repeating words (IPs). During the parallel processing process, six threads allocate. The first three threads use to run the Mapping and Shuffling processes, while the last three serve to run the reducing process. It notes that the nowait clause uses in a work-sharing loop construct, which means turning off the barrier and preventing the thread from waiting. Algorithm 1 describes the main basic steps implemented within the OpenMP environment.

Algorithm 1. MapReduce (fin, fo)

```

Input:
    fin: input file
    chunkNo: number of chunks
    nodeNo: node number
    threadNo: number of threads

Output:
    fo: output file

Begin
1:   set threadNo = 6
2:   call omp_set_num_threads(threadNo)
3:   #pragma omp parallel shared ()
4:   num = omp_get_thread_num ();
5:   #pragma omp single
6:   for (index = 1; index<=chunkNo; index++)
7:       nodeNo = index % 3           // Specify node number
8:       #pragma omp task
9:       if (num>=1 && num<=3)
10:          mapping (chunk_index, fo_map)
11:          shuffling (fo_map, fo_shuffle)
12:       end if
13:       if (num>=4 && num<=6)
14:          reducing (fo_shuffle, fo_reduce)
15:       end if
16:       #pragma omp taskwait
18:   aggregating the previous output reducing into the output file (fo)
19:   end for
20: end begin

```

6. RESULTS AND ANALYSIS OF EXPERIMENTS

This section analyzes and discusses the findings produced from serial and parallel implementations of the suggested fractal architectural model. In addition, compares these results with the cube architecture model. Experiments were performed on a quad-core HP Laptop (1.60 GHz CPU, 16 GB RAM). The C++ language is used to implement the experiment program. In a Windows environment, it compiled the programs using gcc 4.9.2 and OpenMP 5.0. The testing consisted of using text data sets (Data1, Data2, Data3, Data4, Data5, and Data6) that contained IPs addresses data of size lines (100000, 200000, 300000, 400000, 500000, and 600000) accordingly.

6.1. Speedup and parallel efficiency

The input data is divided into chunks for testing, depicted in Table 3. These chunks process in parallel using four threads. The amount of time needed to process data sets changes depending on the size of the chunk being processed, as shown in Figure 6. Consequently, it is considered a factor affecting how well a network works. It conducts the same test on the cube model illustrated in Table 4 and Figure 7 to compare and evaluate the performance efficiency.

From the Tables 3 and 4, it is clear that the fractal model is better than the cube model because it takes less time to implement for the same tested data. The other test between the fractal model and the cube model measures to assess a program's effectiveness by speedup and efficiency, as depicted in Table 5 and Table 6. It is important to remember that in (2) and (3) in sub-section 4.2 which used to calculate the findings in the Tables 5 and 6. Based on the results, the fractal model is better than the cube model. The findings demonstrated that the speedup achieved an average in the fractal model of 2.7, with an average efficiency of 67.666%. While the cube model achieved a speedup of 2.5 with an efficiency rate of 60.406%.

Table 3. The execution time of the fractal model for text datasets with varied chunk sizes in parallel

Chunks No.	6	10	20	30	40	50	60	70	80	90	100	110	120
Data1	3.527	2.563	1.790	1.482	1.492	1.539	1.462	1.509	1.590	1.627	1.750	1.818	1.852
Data2	14.032	8.304	4.844	3.713	3.525	3.088	3.031	2.962	3.055	3.048	3.139	3.233	3.456
Data3	36.864	21.362	11.011	7.187	6.193	5.640	5.375	5.088	5.069	5.053	5.031	5.059	5.155
Data4	65.320	39.512	17.447	11.969	9.675	9.086	8.107	7.459	7.500	7.176	7.001	6.977	7.050
Data5	98.637	59.332	30.687	20.935	14.201	13.710	12.651	11.003	10.713	10.828	9.600	9.414	10.154

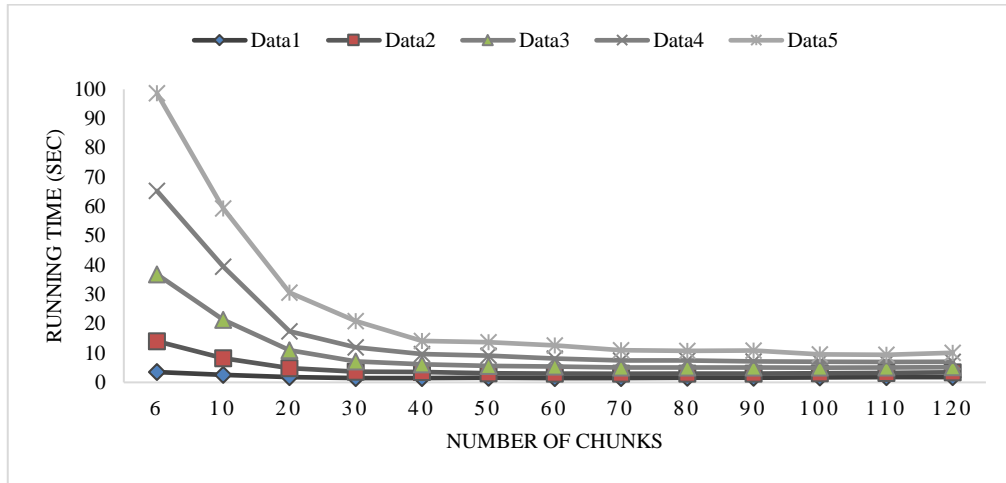


Figure 6. The execution time for several data sets with various chunk sizes running in parallel

Table 4. The execution time of the cube model for text datasets with varied chunk sizes parallel

Chunks No.	6	10	20	30	40	50	60	70	80	90	100	110	120
Data1	3.753	2.596	1.833	1.552	1.470	1.818	1.879	1.933	1.900	1.659	1.850	1.912	1.983
Data2	14.246	9.046	5.040	3.953	3.340	3.138	3.696	2.987	3.663	3.458	3.304	3.289	3.497
Data3	37.157	23.418	11.916	8.184	6.898	6.608	5.764	5.252	5.140	5.181	5.204	5.489	5.715
Data4	67.819	41.225	18.999	13.758	10.201	10.623	8.467	8.009	7.624	7.295	7.680	7.835	7.353
Data5	108.998	63.750	33.365	23.785	17.939	16.292	13.759	11.720	11.418	11.331	11.199	9.729	10.173

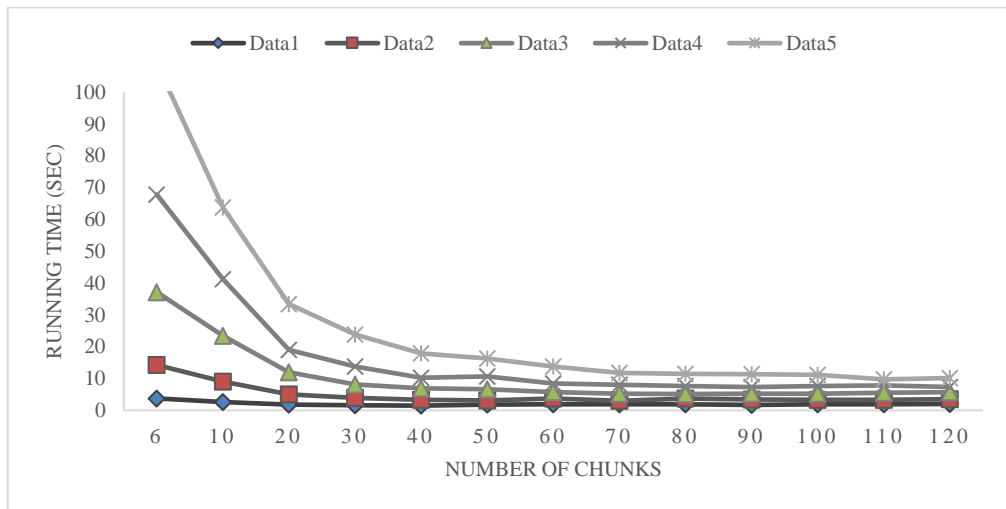


Figure 7. The execution time of the cube model for various chunk sizes running in parallel

Table 5. Efficiency and speedup of the fractal model for diverse data sets in parallel mode

Data sets	Serial (Sec.)	Parallel (4 Thread) (Sec.)	Speedup	Efficiency %
Data 1	12.934	3.415	3.787	94.675
Data 2	46.284	11.989	3.861	96.525
Data 3	101.531	30.307	3.350	83.750
Data 4	177.794	59.750	2.975	74.375
Data 5	277.537	97.414	2.849	71.225
Data 6	341.785	151.017	2.263	56.575
Average	159.644	58.982	2.707	67.666

Table 6. Efficiency and speedup of the cube model for diverse data sets in parallel mode

Data sets	Serial (Sec.)	Parallel (4 Thread) (Sec.)	Speedup	Efficiency %
Data 1	11.785	5.030	2.343	58.575
Data 2	46.085	18.214	2.530	63.250
Data 3	100.824	40.046	2.518	62.942
Data 4	176.886	70.095	2.524	63.088
Data 5	275.917	102.441	2.693	67.336
Data 6	339.579	157.790	2.152	53.800
Average	158.513	65.603	2.416	60.406

7. CONCLUSION

The OpenMP parallel computing model analyses the parallel performance of the given fractal model to determine the parallel speedup and efficiency. In the cases where this model utilized the MapReduce methodology to accomplish the word count computation, the findings demonstrated that the fractal model's average execution time in parallel is 10% faster than the time required by the cube model. Also, comparing the fractal model's average efficiency with the cube model reveals that the fractal model attained a higher success level by approximately 7%. Accordingly, considering the results obtained, the fractal model is preferable to the cubic model. Because the cubic model has more links than the fractal model, communication complexity and time are higher.




REFERENCES

- [1] S. Saleti and R. B. V. Subramanyam, "A MapReduce solution for incremental mining of sequential patterns from big data," *Expert Systems with Applications*, vol. 133, pp. 109–125, 2019, doi: 10.1016/j.eswa.2019.05.013.
- [2] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *Journal of Network and Computer Applications*, vol. 143, no. January, pp. 1–33, 2019, doi: 10.1016/j.jnca.2019.06.006.
- [3] D. Malik and P. K. Goel, "A brief about big data," *Technology and Challenges*, vol. 22, no. 1, pp. 1–5, 2020, doi: 10.9790/0661-2201020105.
- [4] N. Y. Devi, "A parallel direct-vertical map reduce programming model for an effective frequent pattern mining in a dispersed environment," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 24, pp. 1–17, 2021, doi: 10.1002/cpe.6470.
- [5] Y. Benlachi and M. L. Hasnaoui, "Big data and spark: Comparison with hadoop," *Proceedings of the World Conference on Smart Trends in Systems, Security and Sustainability, WS4 2020*, pp. 811–817, 2020, doi: 10.1109/WorldS450073.2020.9210353.
- [6] C. Ai, B. Chen, L. Liu, J. Dong, L. He, and X. Qiu, "Design and implementation of information dissemination simulation algorithm in large-scale complex network based on spark," *Proceedings - 2018 IEEE 3rd International Conference on Data Science in Cyberspace, DSC 2018*, pp. 457–464, 2018, doi: 10.1109/DSC.2018.00074.
- [7] F. Jenhani, M. S. Gouider, and L. B. Said, "Streaming social media data analysis for events extraction and warehousing using hadoop and storm: Drug abuse case study," *Procedia Computer Science*, vol. 159, pp. 1459–1467, 2019, doi: 10.1016/j.procs.2019.09.316.
- [8] W. Kwedlo and P. J. Czochanski, "A hybrid MPI/OpenMP parallelization of K-means algorithms accelerated using the triangle inequality," *IEEE Access*, vol. 7, no. c, pp. 42280–42297, 2019, doi: 10.1109/ACCESS.2019.2907885.
- [9] H. Wang, H. Shen, C. Reiss, A. Jain, and Y. Zhang, "Improved intermediate data management for mapreduce frameworks," *Proceedings - 2020 IEEE 34th International Parallel and Distributed Processing Symposium, IPDPS 2020*, pp. 536–545, 2020, doi: 10.1109/IPDPS47924.2020.00062.
- [10] M. Khader and G. Al-Naymat, "Density-based algorithms for big data clustering using MapReduce framework," *ACM Computing Surveys*, vol. 53, no. 5, 2020, doi: 10.1145/3403951.
- [11] S. Ri, "Fractal functions on the Sierpinski Gasket," *Chaos, Solitons and Fractals*, vol. 138, p. 110142, 2020, doi: 10.1016/j.chaos.2020.110142.
- [12] J. Rodríguez-Cuadrado and J. S. Martín, "Sierpinski-Takagi combination for a uniform and optimal point-surface load transmission," *Applied Mathematical Modelling*, vol. 105, pp. 307–320, 2022, doi: 10.1016/j.apm.2021.12.040.
- [13] F. Jahanmiri and D. C. Parker, "An overview of fractal geometry applied to urban planning," *Land*, vol. 11, no. 4, 2022, doi: 10.3390/land11040475.
- [14] X. Yang, W. Zhou, P. Zhao, and S. Yuan, "Confined electrons in effective plane fractals," *Physical Review B*, vol. 102, no. 24, pp. 1–10, 2020, doi: 10.1103/PhysRevB.102.245425.
- [15] S. Anarova, F. Nuraliev, and O. Narzullov, "Construction of the equation of fractals structure based on the rvachev r-functions theories," *Journal of Physics: Conference Series*, vol. 1260, no. 7, 2019, doi: 10.1088/1742-6596/1260/7/072001.
- [16] E. M. Anitas, "Structural properties of molecular sierpiński triangle fractals," *Nanomaterials*, vol. 10, no. 5, pp. 1–12, 2020, doi: 10.3390/nano10050925.
- [17] M. Saltan, "Intrinsic metrics on Sierpinski-like triangles and their geometric properties," *Symmetry*, vol. 10, no. 6, 2018, doi: 10.3390/sym10060204.
- [18] A. Ali, H. Rafique, T. Arshad, M. A. Alqami, S. H. Chauhdary, and A. K. Bashir, "A fractal-based authentication technique using sierpinski triangles in smart devices," *Sensors (Switzerland)*, vol. 19, no. 3, 2019, doi: 10.3390/s19030678.
- [19] A. Afzal, C. A. Saleel, K. Prashantha, S. Bhattacharyya, and M. Sadhikh, "Parallel finite volume method-based fluid flow computations using OpenMP and CUDA applying different schemes," *Journal of Thermal Analysis and Calorimetry*, vol. 145, no. 4, pp. 1891–1909, 2021, doi: 10.1007/s10973-021-10637-1.
- [20] D. A. Rockenbach, J. Löff, G. Araujo, D. Griebler, and L. G. Fernandes, "High-level stream and data parallelism in C++ for GPUs," *ACM International Conference Proceeding Series*, pp. 41–49, 2022, doi: 10.1145/3561320.3561327.




- [21] J. Klinkenberg, P. Samfass, M. Bader, C. Terboven, and M. S. Müller, "CHAMELEON: reactive load balancing for hybrid MPI+OpenMP task-parallel applications," *Journal of Parallel and Distributed Computing*, vol. 138, pp. 55–64, 2020, doi: 10.1016/j.jpdc.2019.12.005.
- [22] J. Zhao, M. Zhang, and H. Yang, "Code refactoring from openmp to mapreduce model for big data processing," *Proceedings - 2019 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Internet of People and Smart City Innovation, SmartWorld/UIC/ATC/SCALCOM/IOP/SCI 2019*, pp. 930–935, 2019, doi: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00186.
- [23] X. Peng *et al.*, "Parallel computing of three-dimensional discontinuous deformation analysis based on OpenMP," *Computers and Geotechnics*, vol. 106, no. November 2018, pp. 304–313, 2019, doi: 10.1016/j.compgeo.2018.11.016.
- [24] J. Zhao and M. Zhang, "Refactoring OpenMP code based on MapReduce Model," *2018 IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCLOUD/SocialCom/SustainCom)*, pp. 1040–1041, 2019, doi: 10.1109/BDCLOUD.2018.00153.
- [25] P. S. Pacheco and M. Malensek, "An introduction to parallel programming," *An Introduction to Parallel Programming*, pp. 1–468, 2021, doi: 10.1016/B978-0-12-804605-0.00002-6.
- [26] P. Yu, X. Peng, G. Chen, L. Guo, and Y. Zhang, "OpenMP-based parallel two-dimensional discontinuous deformation analysis for large-scale simulation," *International Journal of Geomechanics*, vol. 20, no. 7, pp. 1–14, 2020, doi: 10.1061/(asce)gm.1943-5622.0001705.
- [27] M. Aldinucci *et al.*, "Practical parallelization of scientific applications with OpenMP, OpenACC and MPI," *Journal of Parallel and Distributed Computing*, vol. 157, pp. 13–29, 2021, doi: 10.1016/j.jpdc.2021.05.017.

BIOGRAPHIES OF AUTHORS






Muslim Mohsin Khudhair    is a staff member in the Department of Computer Information Systems, College of Computer Science and Information Technology, University of Basrah, Basrah, Iraq. Currently a Ph.D. student in the Computer Science Department, Faculty of Computer Science and Mathematics, at the University of Kufa. He received the B.Sc. and M.Sc. degrees in computer science from the College of Science, University of Basrah, Basrah, Iraq. His areas of interest include wireless sensor networks, artificial intelligence, image processing, and applied mathematics. He can be contacted at email: muslim.khudhair@uobasrah.edu.iq and mos1970@yahoo.com.



Adil Al-Rammahi    he has Ph.D. from 2005 in fractal geometry and was awarded the title of Professor of Mathematics in 2014. Included in the official business are the Head of the Mathematics Department and the Assistant Dean for Administrative and Scientific Affairs as well. He has several publications in Scopus journals and research contributions to conferences in Los Angeles, London, Paris, and Geneva. He has books on programming and functional analysis. He can be contacted at email: adilm.hasan@uokufa.edu.iq.



Furkan Rabee    is a staff member in the Computer Science Department, Faculty of Computer Science and Mathematics, at the University of Kufa. He got BSc and MSc in Computer Engineering from AL-Nahrian University in 2000 and 2008. He obtained PhD in Computer Science and IT from the School of Computer Science and Engineering, UESTC, Chengdu, China 2015. The research interests include real-time scheduling algorithms, real-time locking protocols, operating systems, parallel processing, distributed system, computer network, IoT, mobile computing, cloud computing, and smart cities. He can be contacted at email: furkan.rabee@uokufa.edu.iq.