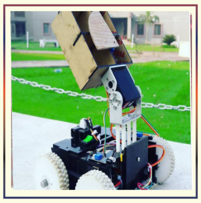

Wasim et al. LGURJCSIT 2023

ISSN: 2521-0122 (Online)
ISSN: 2519-7991 (Print)

LGU Research Journal of Computer Science & IT

doi: 10.54692/lgurjcsit.2023.073365

Vol (7): Issue (3), July – September 2023

A Comparative Study of Parallel and Distributed Big Data Programming Models: Methodologies, Challenges, and Future Directions

Muhammad Wasim[1], Fiza Gulzar Hussain[1*], Ayesha Nasir [1], M. Usman Ashraf [2]

[1]Department of Computer Science, University of Management and Technology, Lahore (Sialkot Campus), Pakistan.

[2]Department of Computer Science, GC Women University, Sialkot, Pakistan.

Email: 21001279011@skt.umt.edu.pk

ABSTRACT:

According to a survey conducted in 2021, users share about 4 petabytes of data on Facebook daily. The exponential increase in data (called big data) plays a vital role in machine learning, the Internet of Things (IoT), and business intelligence applications. Due to the rapid increase in big data, research in big data programming models gained much interest in the past decade. Today, many programming paradigms exist to handle big data, and selecting an appropriate model for a project is critical for its success. This study analyzes big data programming models such as MapReduce, Directed Acyclic Graph (DAG), Message Passing Interface (MPI), Bulk Synchronous Parallel (BSP), and SQL. We conduct a comparative study of distributed and parallel big data programming models and categorize these models into three classes: traditional data processing, graph-based processing, and query-based processing models. Furthermore, we evaluate these programming models based on their performance, data processing, storage, fault-tolerant, suitable language, and machine learning support. We highlight the benchmarks with their characteristics used for big data programming models. Finally, we discuss the models' challenges and suggest future directions for the research community.

KEYWORDS: Programming Models, Parallel computing; Distributed computing, Big data, Map Reduce, Directed Acyclic Graph, Message Passing Interface, Bulk synchronous Parallel, SQL-like

1. INTRODUCTION

In recent years, the emergence in the domain of IoT and social media platforms usage is becoming the source of generating a massive amount of digital data called big data. Daily, billions of users access social media platforms and share information regarding their activities and interests. Big data refers to the massive amount of data generated through messages, audio, and videos [1]. Big Data is a massive data set that might be unstructured, structured, or semi-structured. Different sources like sensors, cell phones, social media, and e-commerce websites generate big data. The concept of big data reflects the size of the extensive data. It is characterized by 3Vs (volume, velocity, variety) as shown in Figure 1. 1) Volume: alludes to the gigantic measure of information (Gigabytes, Terabytes, Petabytes) 2) Velocity: this alludes to the speed and frequency of the incoming data that needs to be processed and analyzed. 3) Variety: indicates data in different formats (e.g., XML, CSV, PDF, JSON) and types (e.g., text, sound, pictures, videos) [2][3].

Big Data is becoming dominant because of its usage in different fields like health care [4-6], agriculture [7-9], banking [10-12], media [13-15], entertainment [16-18], and telecom

[19-21] and researchers have proposed different models to handle different type of big data. Big data help organizations get better customer insights and design effective marketing campaigns. Machine Learning, Deep Learning, Cloud Computing, and IoT also rely on big data programming models [22-24]. Processing and handling large-scale data using traditional technologies like relational databases is impossible.

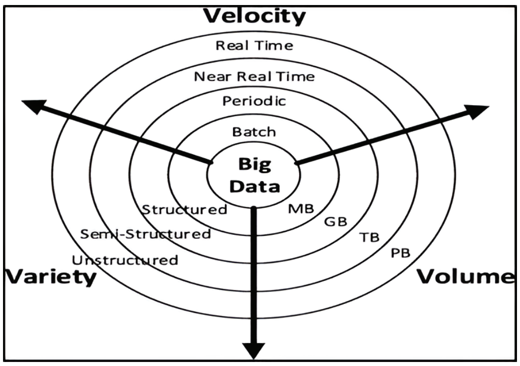

Figure 1: The Three Characteristics of Big Data

Therefore, researchers have proposed different programming models such as MapReduce, DAG, MPI, BSP, and SQL-like paradigms for handling big data. We categorize the programming models into three categories: traditional programming models (MapReduce, DAG, MPI), graph programming models (BSP, Pregel, Hama), and query programming models (SQL-Like), as shown in Figure 2. The overview of these programming models is presented in the following sections.

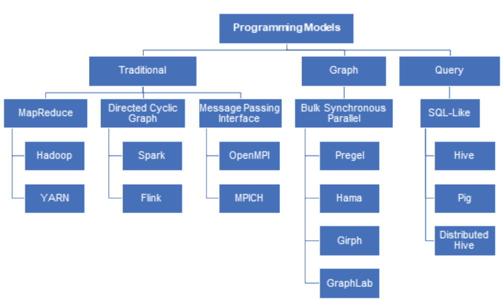

Figure 2: Traditional, Graph based and Query paradigm for Big Data Programming Models

1.1. MapReduce

The MapReduce programming model is used to develop large-scale big-data applications. This programming model uses two essential functions map and reduce. The map function splits the input data into different pieces or tasks and produces key-value pairs. Reduce function accepts these input pairs and combines these tasks.

The programming model also provides the facility of handling faults if any occur without disturbing the whole mechanism. If there is no response from the worker node for a specified time, this node is considered dead, and a master then assigns the same task to it to recover from faults. Due to disk processing of data in disk instead of memory, Hadoop performance is considered slow.

1.2. Directed Acyclic Graph

Directed Cyclic Graph (DAG) is an effective platform for modeling complex data analysis, such as blockchain and data mining applications. DAG is the combination of edges and vertices, and the vertices could be objects of any kind connected by edges [23].

1.3. Message Passing Interface

Message passing interface (MPI) provides process-to-process communication and exchange messages by connecting multiple
computers running parallel programs over distributed shared memory [26]. MPI aims to provide scalability, portability, and high performance. In MPI, the sender process sends information that is to be received by the receiving process [27]. Although the MPI offers high scalability and performance, it lacks support for fault tolerance [28].

1.4. Bulk Synchronous parallel

The Bulk Synchronous Parallel (BSP) model was introduced in the late 19s [39]. This model worked in three steps, i.e., super steps, barrier synchronization, and global computations. The local computations were performed in each super step, and the global communication step was used to take an update from each super step. Barrier synchronization was used to ensure all processing was done in super steps. This model performs efficiently on graph-based applications.

1.5. SQL-like

SQL-like programming models facilitate developers in writing big data applications in a distributed and parallel manner. These programming paradigms are generally considered the core part of big-data architecture. Moreover, the knowledge of these platforms helps developers to select suitable programming models according to the nature of the application. For example, some applications require large-scale data handling but not in real-time. On the other hand, some applications demand efficient machine learning (ML) platforms, and others require fault tolerance. Similarly, Different applications need efficient graph processing mechanisms. Moreover, a developer should consider a few limitations (fault-tolerant, real-time) in these programming models before selecting a model.

We also compare these models in Table 1 based on parameters, including data flow, computations, use case, and in-memory caching. We observe that only DAG is the in-memory big data model.

We also compare our survey paper with existing survey papers. D. Wu et al. published a survey on big data programming models in 2017. The study describes all big data programming models and their implementations [29]. They categorized the programming models into MapReduce, Functional, SQL-based, Actor, Statistical, Data flow, BSP, and high-level DSL. They explained the application of programming models and compared them based on Features, Abstraction, Semantics, and computation. The programming models in the survey were not compared based on their characteristics, parameters, qualities, and suitable applications.

L. Belcastro et al. surveyed to compare big data programming models [23]. They divided the programming models into four categories: Level of abstraction, type of parallelism, infrastructure scale, and application classes. They compare these programming models based on data management and exchange, interoperability, and efficient parallel computations. It helped developers identify programming models according to their hardware needs. However, they did not categorize the programming models according to data processing techniques.

Similarly, L. Belcastro et al. conducted a detailed survey of programming models [30]. This survey explained the features of programming models along with the code snippets and real-world applications. They compared different programming models such as MapReduce, Spark, Flink, Pregel, and SQL based on programming features and diffusion and presented their advantages/disadvantages. Their study did not cover any benchmarks for evaluating the performance of these programming models. Our contribution in this study is described below:

- We explain different big data programming models and categorizes them into three categories (Traditional data processing, graph-based processing, and query-based processing) based on data processing.
- We present a detailed study of the evaluation of these models.
- We also discuss the different benchmarks vital for different model types.
- We identify the challenges developers face in the selection of big data programming models.
- We also identify and present the limitations in programming models to define new research directions for researchers in the field of big data programming
- We analyze the usage of big data programming models based on parameters such as performance, data processing, storage, fault tolerance, and machine learning support.

We conducted a detailed literature survey by studying the papers from 2015-2023. We studied a total of 84 research papers downloaded from Google Scholar. We found these research papers by searching with different keywords related to Big Data programming models like big data, parallel computing, distributed computing, programming models, Apache spark, Apache Hadoop, Map Reduce, and MPI. The rest of the paper is organized as follows: section 2 elaborates on classifying big data programming models into traditional, graph and query models. The big data benchmark datasets are describing in section 3. We discuss the crucial parameters, open problems and future directions in section 4. Finally, section 5. concludes our study with future directions.

2. LITERATURE REVIEW OF BIG DATA PROGRAMMING MODELS

We classify the programming models into three types: Traditional, Graph, and query, as shown in Figure 2. Different types of models under these paradigms are explained in this section.

2.1. Traditional Big Data Programming Models

Dean et al. discussed the first programming model, "map-reduce," for handling big data, and is proposed by Google [4]. Before this model, google faced the issue of parallelism, fault tolerance, and distribution of its computations. MapReduce programming model solved all these problems. This programming model was inspired by Lisp and other functional languages primitives "map" and "reduce." The MapReduce is simple but powerful enough to hold up different data-intensive applications [5]. MapReduce is used in different domains, including machine learning, social media, data mining, image processing, and information retrieval.

Table 1: Comparison Table of Big Data Programming Models

Model	Data flow	Computation	In-memory caching
Map Reduce	Map and Reduce phases	Batches	No
DAG	Directed Graph	Real- Time	Yes
MPI	Explicit Point-to-Point communication	Message passing between tasks	No
BSP	Synchronous Iterations	Iterative and parallel	No
SQL	Relation Algebra	Queries	No

Apache Hadoop platform is implementing the MapReduce model that came into existence in 2005 [4]. Yahoo first contributed and adapted 80% of the core of Hadoop [6]. Apache Hadoop handles large-scale data in a distributed manner and facilitates programmers by providing solutions like fault tolerance, load-balancing scalability, and cost.[7]. Hadoop uses the Hadoop Distributed File System (HDFS) for storing data. [8].

P. Natesn et al. proposed a two-stage MapReduce model using Apache Hadoop [77]. It was called MapReduce Multivariate linear regression model (MR-MLR). In the training phase, the mapper was used to correlate between regression variables. It reads the data from the HDFS file structure. The second phase was the prediction/classification of predictor values by reading test instances. This framework was evaluated on four UCI datasets of machine learning. The experimental results revealed that MR-MLP was scalable and efficient for big data applications.V. K. Vavilapalli et al. highlighted the shortcomings of the Hadoop MapReduce programming model and explained the new architecture of Hadoop On-Demand and Apache YARN [9]. The classical Hadoop MapReduce model was limited in scalability and strongly decoupled resource initializer with the programming model. Hadoop On-Demand (HoD) overcame these limitations. But resource allocation information was not adequately managed by HoD. Apache YARN managed resources. It consisted of three major components: Resource Manager, Application Manager, and Node Manager. The resource manager communicated with NM for resource availability and then issued container leases.

Apache Spark, which implements the DAG programming model, is used to process data in RAM instead of disk[10][85-90]. This feature of DAG, as a result, provides faster computation than Hadoop. In addition, Spark did not havSparktorage system, which is Big Data applications' primary and fundamental requirement. Spark uses other sources like HDFS Cloud storage and other NoSQL databases to overcome this limitation.

The authors in [78] proposed a word count application using big data. The application was implemented on Apache Spark 3.1.2 version with 8 GB with 2 cores and a single node. They used different data sizes for executing them on different numbers of cores. The experiment was performed by Running the word count application that analyses the speed and processing time. The results showed that the models take less processing time when increasing the number of cores.

The big data programming models can also be used for heavy computational time-consuming tasks like feature engineering. In [79], the authors extracted text features from the Wikipedia corpus to evaluate the RDD and Spark SQL APIS runtime of the Apache Spark programming model. The HDFS was used for storing and retrieving the corpus. More Apache yarn is used as a resource manager for managing hardware resources and batch jobs. The results showed that SparkSQL API performs better in running long batch jobs by decreasing the runtime from 67% to 80%.

P. Carbone et al. proposed the Apache flink programming model based on DAG [11]. The authors explained Flink's architecture and discussed how it was used for batch and stream processing. Apache Flink consisted of two APIs: batch processing Dataset API and stream processing dataStream API. The Flink process model had three components: Flink Client, Task Manager, and Job Manager. Flink client received the program code and made a dataflow graph which was passed to the Job manager. Job

Managers created checkpoints for fault tolerance. Actual processing executed in Task Manager.

Y. Benlachmi et al. compared big data programming models frameworks Hadoop and Spark. This paper evaluates the performances of these two frameworks [12]. These two implementations are compared regarding performance scalability, cost, security, and latency. By analyzing all the facts, the authors stated that apache spark is better at processing real-time stream data, but Apache Hadoop is better when large-scale data are in batch form. Another reason behind the fast performance of Spark is in-memory data processing. Hadoop is less costly than Spark due to the usage of local disks.

H. M. Makrani et al. presented an empirical analysis of the memory usage of Spark, Hadoop, and MPI [13]. It helped in understanding the overall impact of different memory parameters on the speed and performance of the big data frameworks. The memory parameters were capacity of memory, frequency of memory, and the number of channels. The results revealed that Spark and Hadoop don't require a large memory capacity, but MPI does.

M.Assefi et al. presented a real-world experiment on Apache Spark MLib [14]. Moreover, they also compared the performance of the Apache Spark MLIB platform with the Weka Hadoop version platform. They used different ML classifiers on four different datasets.

Another research focused on comparing the performance of the MPI model with MapReduce [15]. The authors made three randomly generated graphs with 1000 to 10,000 nodes. The results revealed that MPI performed better on iterative jobs for data-intensive iterative applications and when the dataset was moderate. On the other hand, when the dataset is large in scale and tasks don't require iterative jobs, MapReduce performs better than MPI.

A. Salzman et al. proposed novelties in the GFEM method [80]. They implemented a two-scale solver for local and global problems in linear elasticity problems using MPI. The authors developed a specific scheduling policy for local problems. And reference solution was proposed for the iterative process. The MPI model provided distributed memory access and used specific resolutions at the global level. The parallel workflow improved the scalability with a cost of less than 1.3%. I. Chebbi discussed thearchitecture of Hadoop and Spark in detail [16]. According to them, the platform of Hadoop and Spark is fault-tolerant by default. The platform of Hadoop recovers the lost data from other data nodes of the cluster through replication. On the other hand, sparks use its RDD data structure for recovering lost data. But if we consider the MPI programming model fault-tolerant feature, then according to [17]. MPI isn't fault tolerant by fault, and still, there isn't any mechanism proposed, yet that makes MPI fault tolerant.

S.J. Kang et al. discussed the MPI and MapReduce parallel programming models [18]. The authors considered two problems first one is the all-pair-shortest path, and the second is computation intensive. MPI might be regarded as the framework when the data size is reasonable, and the task is computationally heavy. MapReduce may be a great framework when the vast data size and the jobs do not need iterative processing.

A. Mostafaeipour et al. analyzed the performance of Spark and Hadoop frameworks on the Machine Learning platform [10]. The model used the Higgs dataset with 11 million samples in the 28 features. The experiment was conducted using the KNN machine learning algorithm. The value of K used by the authors was 5 on the dataset for both platforms. The results indicated that for small datasets, the performance of the Spark increased by 4.5-5; for large datasets, the performance was 1.4-2 times higher than the Hadoop.

For evaluating the performance of MPI with Apache Spark, D. S Kumar et al. proposed a Twitter sentimental analysis on Twitter data[19]. The methodology was to read tweets line by line and then count positive and negative words. The dataset used for this experiment was 7GB, 500GB, 100GB, and 1TB. The results revealed that the execution time of MPI was 2 times greater than the execution time of Spark.

L.Xia et al. proposed a unified model named Blaze for handling high energy physics (HEP) big data [81]. It modified the Spark to add the message passing facility by OpenMPI. This model is used in data computer memory for efficient communication. HEP data was partitioned and used in parallel. The Spark computing engine was responsible for task allocation, and inter-task MPI was implemented. This model achieved 70% performance improvement as compared to the traditional Spark model.

X. Lu et al. experimented by combining the features of MPI and Hadoop to reduce delay [20]. The proposed idea worked with the MPI-D Library built on point-to-point primitives on MPI for supporting arbitrary operations of MapReduce. The methodology for adapting MPI was to use a communication platform for Hadoop, which was divided into two groups; first, by comparing Hadoop modules with MPI primitives to analyze the bandwidth and latency of these two platforms. Secondly, they implemented an MPI-D library that worked with key-value pairs.8 nodes were used to build the experiment with the MPI-D library. The results revealed that their proposed prototype reduces the execution time by 44%.

Another critical issue in combining HPC and Big data is the difference in their software stacks. The limitations interoperability between their programming models and languages is limited. To deal with this problem, the authors in [82] proposed a new model called IgnisHPC. This model was explicitly used for executing HPC and Big Data workloads. Moreover, IgnisHPC supports multiple language applications with Java Virtual Machine and non-Java Virtual Machine languages, as it relies on the MPI model. Hence, this framework takes advantage of network architectures and communication models. Moreover, the model executes MPI-based applications efficiently. The results showed that their model performed 1.1× to 3.9× faster than the traditional Spark.

M. M. Rathore et al. presented a Real-time and efficient stream data processing platform for analyzing big data [21]. The model worked with distributed and parallel environments of Hadoop with Apache Spark and GPU. The authors collected data from sources and then filtered it. After filtration, the data is transferred to the load balancing unit, where the controller and data nodes work together for parallel and distributed processing. The data nodes are attached with GPU, HDFS, and Apache Spark. Apache Spark uses its real-time processing feature and performs immediate action on data. The results reveal that the proposed system with GPU throughput processes 300-350 Mbps frames per second, whereas the CPU-based map-reduce framework has a throughput of 50 Mbps.

2.2. Graph Big Data Programming Models

Hadoop was mainly used for processing traditional data. It could also be used for processing graphs-based applications. The HADI algorithm for efficient MapReduce jobs in graphs was introduced in [22]. Another PEGASUS library was developed on top of Hadoop for graph mining tasks [23], but multiple map-reduce jobs involved can cause overhead and affect efficiency.

S. Sakr Proposed GraphLab project written in C++ [24]. It was used for graph processing Big Data with a high-level programming interface. It was used with both HDFS and POSIX file systems. It consisted of three main parts: a data graph, an update function, and a sync operation. Data graph used for user-modifiable program state and computational dependencies. Update function used to operate on data graph and transformed data in small overlapping contexts. It was used to represent user computations. Three operations, gather, apply, and scatter, were used in execution.

G. Malewicz et al. developed another separate framework for graph processing based on the BSP model named Pregel [25]. It was based on distributed computing. The architecture used a directed graph for input to Pregel computation. The vertex of this graph defined user-defined operation, and edges were associated with the source vertex. After graph initialization, a series of steps were performed in a sequence of super steps. After completion of tasks, all vertices vote to halt, and the process is terminated. An experiment was performed with a single-source shortest path on 300 multicore commodity PCs. 800 worker tasks were initiated, and it was observed that the running time of the graph took 10 minutes.

Z. Tian et al. proposed a BSP model for agent-based simulations [83]. The authors created a temporary artificial network for experimenting with simulation locally. They developed CloudCity, a distributed engine to improve the communication and locality in these simulations. The main area of concern was to improve the tolerance for distributed systems. To reduce the communication overhead, the author proposed a double buffering mechanism. They compared this framework with Giraph, GraphX, and Apache. The performance of this model was 100 times faster than Spark.

U. Kang Proposed a graph-based framework called GBASE on top of Hadoop [26]. It was deployed on the Yahoo Hadoop cluster. This framework comprised two components: The indexing stage and the query stage. The raw graph was given as input to the framework. The indexing stage then clustered it and divided it into blocks. Then these blocks were compressed and stored. GBASE was efficient in storage, indexing, and scalability.

S. Sakr proposed the Apache Giraph model based on the BSP model in 2012 [24]. It works in super steps. All graph processing programs were expressed as iteration sequences in super steps. It worked on Master-Slave architecture. The master node assigns partitions to a vertex which act as vertices. It used Zookeeper for synchronization.

R.S. Xin introduced the GraphX framework based on a resilient distributed graph system in Spark [27]. GraphX produced the resilient distributed graph (RDG) using RDDs. Two graph-based algorithms, Pregel and PowerGraph, were implemented using RDGs. GraphX interface provided the facility of graph construction along with graph transformations and queries.

P. Carbone provided support for graph processing using Gelly Flink [11]. Gelly is comprised of two datasets: the vertices and edges dataset. These dataset properties were used to generate a graph.

K. Siddique et al. proposed a new research direction in big data by introducing Apache Hama based on BSP [28]. The authors illustrated the architecture of Apache Hama in three major components: BSP master, Zookeeper, and Groom server. The BSP master was responsible for assigning tasks to Groom Server. Zookeepers acted as barrier synchronization. The BSP master supported the fault-tolerant property.

Siddique et al. worked on Apache Hama and discussed its architecture, advantages, and shortcomings[29]. They compared Apache Hama with other big data programming models, Apache Yarn, Apache Giraph, MapReduce, and Apache Spark. Apache Hama's core architecture was based on a BSP model. Apache Hama was useful for complex iterative applications and outperformed MapReduce in this domain. Apache Spark outperformed Hama in terms of usability. Hama outperformed MapReduce and Spark on top k joins on large datasets. Apache Giraph was not used for real-time processing, machine learning, and repartitioning. Hama used traditional graph partitioning techniques.

L.Y. Ho proposed another graph-based model named Kylin [30]. It was based on BSP but with three optimization techniques: vertex-weighted partitioning, pull messaging, and lazy vertex loading. This model outperformed Apache Hama up to five times due to efficient optimization techniques. Z. Wang proposed a new BC-BSP+ model based on BSP [31]. This model provided efficient and flexible configurations and graph partitioning techniques. This model used the disk buffer for managing data. BC-BSP+ provided simple APIs to users for implementing graph structures. The experiments were performed by running the PageRank algorithm. The results showed that BC-BSP+ outperformed the Hama and Giraph. The running time of BC-BSP+ was twice faster than Hama and six times faster than Giraph.

R. Chen et al. worked on graph processing frameworks and proposed a new model for graph processing named Cyclops and CyclopsMT [32]. Cyclops was based on Master-Slave architecture. The working model was based on Pregel and Hama's core. In Cyclops, the master used to send replicas to other necessary nodes. Cyclop's performance compared to Hama was 2.06X using the Metis partition algorithm. T. Li et al. proposed a GraphZ framework for graph processing based on BSP [33]. It consisted of three components: master node, server node, and storage node. This model used the ZHT server. It was tested by implementing the PageRank algorithm on a different number of machines. It was considered best for load balancing and data locality compared to Hama.

G. Dai et al. proposed a new framework for graph processing named FGPG [34]. This framework consisted of processing kernels, block RAMs, and FGPG chips. On-chip cache mechanism for data locality in the graph was implemented. The experiment was performed by implementing Breadth-First Search (BFS) in Twitter data. The proposed framework did not achieve state-of-the-art performance on FGPG.

S. Aridhi et al. proposed a framework BLADYG for dynamic graph processing [35]. BLADYG was used to collect online graph data using HDFS, Database, or Amazon S3. Data can be stream-based or complete with one graph. Graph Partitioning techniques were also applied. R. Dathathri et al. proposed Gluon to improve communication optimization in existing frameworks for distributed graph analytics [36].

M. Twenty et al. proposed GraphOpt to improve the performance of existing frameworks Giraph and GraphX [37]. Moreover, it was used for three optimization algorithms. Experiments were performed on different benchmarks and showed that performance was increased up to 47.8% using random search and 5.7% on average.W. Fan et al. [38] proposed a GraphScope framework for parallel and distributed graph processing. It consisted of data flow runtime for distributed execution of graph processing. The architecture also included the graph library to perform standard graph computations. This framework can be implemented in cyber security monitoring, fraud detection, and link prediction. It was 34.7 times faster on iterative graph queries as compared to PowerGraph. W. Daluwatta et al. proposed a CGraph framework for graph processing [39]. It was based on graph repartitioning techniques to reduce the overhead. It improved performance up to 3.9 times compared to another graph-based Chaos framework.

2.3. Query Big Data Programming Models

S.Arora et al. emphasized the problem of MapReduce Hadoop that Java developers were required to perform any task on this model [40]. The authors explained two new implementations of the big data programming model SQL. Yahoo proposed Apache Pig to resolve the issue of a few available Java developers. They introduced a new language named Pig Latin, similar to SQL. Pig Latin was found to replace a hundred lines of Java code into four lines of Apache Hive proposed by Facebook and used this model on top of Hadoop for ease of use. It used an SQL-like query language called HQL. Apache Hive architecture comprised three main components: Hive client, driver, and Hadoop. The limitation of Apache Hive was latency issues for hive queries and was not suited for low-level updates.

V. Garg focused on the problem of using Apache Hive for big data that increases the execution time of tasks [41]. The author proposed a multiple query optimization (MQO) component to reduce the execution time. A new architecture of Apache Hive named distributed Hive was proposed. The user submitted Hive Queries in this architecture through a web interface or command line interface. Incoming queries were suspected and made common global queries. These global queries were passed to the Driver component that passed the query to the compiler. The compiler generated a logical plan that DAG uses for defining map-reduce tasks. An experiment was performed to evaluate the performance of distributed Hive by varying data sizes and several queries. It was observed that the execution time of queries with MQO was 50% reduced compared to conventional Hive architecture. In [84], the Apache hive model, MongoDB, and Microsoft SQL server are analyzed to construct the data warehouse for online learning platforms. The corpus construction and descriptive analytics process were evaluated with the assistance of the above-defined technologies. The Apache hive was used for different contexts in handling big data design principles in constructing data warehouses. Also, it was implemented on an Azure virtual machine with the same region and hardware configuration. Their evaluations showed that the Apache Hive platform requires less maintenance and performs faster in contrast to MongoDB and Microsoft SQL. This is because the scalability mechanism of Hive's used commodity hardware and the simplified mechanism of this programming model favors this decision.

K. Bansal et al. worked on Apache Pig and Apache Hive and experimented with massive datasets to analyze their performance [90-96]. A dataset was installed on Hadoop, and different queries were performed to extract data using Apache Pig and Apache Hive. The authors explained the architecture of Apache Pig and Apache Hive. Apache Pig was based on the Pig Latin language, which provided a high-level program of Java MapReduce jobs. Apache Hive was based on an SQL-like language called HiveQL. A medical dataset of 125,087 records from the United States was used to experiment. The authors observed that on increasing dataset size, Hive was slow in execution as compared to Pig. Regarding Storage, Hive was more efficient for data extraction than Pig. For ease of use, Pig was considered difficult to use because some knowledge of Java was required. On the other hand, Hive was easy to use because of the SQL-like structure. In terms of cost, both Apache Hive and Apache Pig were cost-effective.

3. BIG DATA BENCHMARKS

Benchmarks are used to compare the performance of big data programming models. A series of experiments and tests are performed to evaluate the programming models. The Benchmark process comprises five steps:
planning step, generating data, generating data, developing tests, execution of tests, and evaluation and analysis of results. Some critical extensive data benchmarks are presented in Table 3 with their characteristics and description.

Table 2: Summary of Big Data Programming Models, the frameworks based on those models with their respective pros and cons

Ref	Programming Model	Framework	Methodology	Pros	Cons
Traditional Big Data Programming Models					
[6]	MapReduce	Hadoop: MapReduce implementation	The model, along with its file system HDFS takes advantage of "map" and "reduce" functions for solving big data problems in a distributed and parallel manner.	Flexibility Scalability Fault-Tolerant	Low-level programming, Not for iterative tasks
[11]	Directed Acyclic Graph (DAG)	Apache Flink	The model processes data in a stream, batch, and iterative way with an in-memory computational mode.	Real-time Processing	Memory Management
[43]	Message Passing Interface (MPI)	Open MPI	Process to process communication for parallel processing of data.	Fast processing of Large-scale data, then Hadoop and Spark.	Not fault Tolerant
Graph Big Data Programming Models					
[25]	Bulk synchronous parallel (BSP)	Pregel	Using vertex-centric approach	Efficient graph processing	Slow speed
[28]	Bulk synchronous parallel (BSP)	Apache Hama	BSP based three components: BSP master, Zookeeper, and Groom server	Improved performance over Pregel	Unnecessary communication in graph partition strategy
Query Big Data Programming Models					
[41]	SQL	Distributed Hive	Add Multiple query optimization components in Hive	50% improved performance over traditional Hive	Less speed
[40]	SQL	Apache Pig, Apache Hive	Abstraction over Hadoop using SQL-based language in Hive and Pig Latin in Pig	Easy to use and implement	No storage system

Table 3: Big Data Benchmarks for different significant data programming paradigms with their characteristics

Ref	Name	Description	Characteristics
Traditional Big Data Programming Models			
[44]	YCSB	No SQL databases	Used for comparing two non-relational databases (Hbase & Cassandra).
[45]	Grid Mix	Suitable for Hadoop Clusters	Suitable when multiple users perform the same jobs.
[46]	TPC	Online Transaction Processing Workload	Workloads are implemented using different Arithmetic operators. De-facto standard for evaluating DBMS.
[47]	Big Bench	The industry benchmark for big data analytics for Hadoop	It comprises 30 queries and four key steps: system setup, generating, loading, and executing data.

[48]	Cloudsuite	For testing the applications running on cloud platforms, Hadoop and GraphLab.	Used for scale-out applications.
[49]	Hi-bench	A shell script set published by Apache License.	Four categories are classified into thirteen workloads. Perform operations on real-world applications and synthetic micro-benchmarks.
Graph Big Data Programming Models			
[48]	Cloudsuite	For testing the applications running on cloud platforms, Hadoop and GraphLab.	Used for scale-out applications.
[50]	Graphalytics	Used for graph processing models	Distributed processing framework Support RDF databases.
Query Big Data Programming Models			
[51]	Pig Mix	Query evaluation of pig based system	17 queries perform different operations.
[52]	Big Data Bench	In the real world, synthetic and big data workloads.	Perform three basic operations: relational queries, microbenchmark and essential data store operation. Generate six seeds model.

4. DISCUSSION

In this section, we discuss the different challenges of big data programming models. We evaluate the programming models present in literature review according to parameters like in-memory data, batch processing, stream processing, efficient resource management, and iterative tasks. We also present a comparative table (Table 4) of the big data programming models based on performance, data processing, storage, fault-tolerant, suitable language, and machine learning support. Based on our extensive study, we analyze the challenges regarding big data programming models and provide solutions to these challenges.

The most common and widely used MapReduce programming model has HDFS storage which is suitable for handling large data sets. This model mainly uses batch processing to manage the data effectively. It is able to handle the data if any fault occurs and is highly resilient. On the other hand, DAG is proposed to be an effective solution for real-time applications. It manages data in streams. MPI is not fault-tolerant, so it is important to note that this model is less suitable and might not be ideal for applications that demand high availability and robustness in case of node failure and system breakdown.

Graph-based applications like social networks, network optimization, and maps require iterative computations. BSP-based models like Apache Hama, Pregel, GraphLab, and Apache Giraph are most suitable for graph-based data processing applications.

The application developer must explicitly design and implement fault tolerance features, such as recovery through barriers, in the BSP model because fault tolerance is a great concern in graph processing applications. This manual fault tolerance method in the iterative BSP model causes data inconsistency and increases complexity. Alternative programming models like Apache Spark's GraphX offers automatic fault tolerance, which can be more advantageous than the manual implementation of this mechanism. We show these features in Figure3.

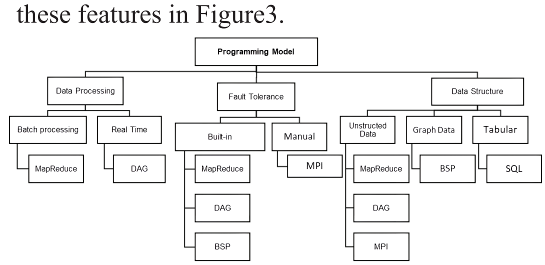

Figure 3: Characteristics based categorization of Big Data Programming Models

4.1. Open Problems: Challenges and Future Directions

4.1.1. Data Management

Data Management is a challenging task in programming models for handling big data. Hadoop uses disk management, which creates problems when processing data and causes delays. Therefore, in-memory data management was introduced in Apache Spark [119-120]. Although in-memory data management overcomes the problem of inefficient data retrieval, it also has a limitation in that data size must be small enough to load in memory or memory size must be large enough to store all data.

4.1.2. Processing

There are different ways to process data in big data programming models. Some programming models like Hadoop process data in batches, whereas Spark can process data streams.

4.1.3. Lack of Professional Expertise

Developers also face the challenge of a lack of professional knowledge and expertise in different languages to handle big data programming frameworks [97-112]. Developers with expertise in query languages find it easy to deal SQL based models like Apache Pig and Apache Hive. Similarly, developers with poor knowledge of Java face problems in writing map-reduce programs for Apache Hadoop.

4.1.4. Resource Management

Resource management is one of the crucial challenges in big data programming models. Managing the resources efficiently when working in a distributed environment is essential. Apache Hadoop and other versions of Hadoop, like Common Hadoop, were poor in resource utilization.

4.1.5. Graph Management

There was a problem with managing graph big data. Hadoop is not suitable for graph processing. Initially, Hadoop was used for graph processing. However, it was limited to up to two iterations and increased overhead. A developer must perform repeated map and reduce functions to perform iterative tasks in MapReduce [112-114].

This study aimed to find the most suitable programming model for developers and the research community. Different parameters and their associated best programming models [114-118] are presented in Table 5. We propose the following future directions:

i. Real-time processing of data should be implemented for big data applications

ii. For ease of programmers, different programming languages APIs should be introduced.

iii. Resources should be distributed in different clusters for the efficient development of big data applications

iv. For handling visual modality, different mechanisms for reducing overhead should be introduced

v. Apache Spark is a promising solution if the data is small that fits in memory.

vi. Apache Flink can be considered if the application requires batch, stream, or iterative processing.

vii. If the application needs to process graph-based data, BSP-based models like Apache Hama, Pregel, GraphLab, and Apache Giraph can be used.

viii. If the application requires handling big data in the backend and using query-based information in front, Apache Hive and Apache Pig are a clear winners.

ix. MapReduce is easy to use for Java developers from the language perspective

x. Apache Pig and Apache Hive can be the best choice for SQL developers

Table 4: Comparison of big data programming models based on different parameters

Ref	Programming Model	Framework	Performance	Processing	Storage	Fault-Tolerant	API	Language	ML
Traditional Big Data Programming Models									
[6]	MapReduce	Apache Hadoop	Fast for large data sizes.	Disk Batch-processing	HDFS	Yes	No	Java	Yes

[9]	Hadoop	Apache YARN	Improved performance by separating resources	Containers	HDFS	Yes	No	Java	Yes
[12] [10]	DAG	Apache spark	Fast for processing real-time short data	In-memory stream processing	Cloud, Amazon s3, HDFS	Yes	No	Scala	Yes
[11]	DAG	Apache flink	Improved performance over stream data	Stream, batch and iterative	Memory-based	Yes	No	Java and Scala	Yes
[9]	Message Passing Interface	Open MPI	Fast for iterative tasks	In-memory stream processing	NFS and HFS	No	No	C++	Yes
Graph Big Data Programming Models									
[25]	BSP	Pre-gel	Improved performance by data on the same machine	BSP Supersteps	Distributed and local	Yes	C++ API	Java	No
Query-based Big Data Programming Models									
[40]	SQL	Apache pig	Good on all types of data	Pig Scripts	Database	Yes	No	Pig Latin	No
[40]	SQL	Apache hive	Data Partition	Query Based	HDFS	Yes	No	Hive QL	No

Table 5: Application Requirement Vs. the most suitable big data programming model

Parameters	**Programming Model**
In-Memory Data	Apache Spark & Open MPI
Batch Processing	Apache Hadoop
Stream Processing	Apache Flink
Efficient Resource Management	Apache YARN
Iterative Tasks	Bulk Synchronous Parallel
SQL	Apache Hive

5. CONCLUSION

In this paper, we performed a comprehensive survey of parallel and distributed big data programming models along with benchmarks for different types of classified under three broader categories: Traditional big data programming models (MapReduce, message passing interface, directed acyclic graph), graph-based big data programming models (Bulk synchronous parallel, Pregel, Hama), and SQL-Like (Apache hive, Apache pig, and distributed Hive). We provided a detailed overview of the frameworks of these programming models. We identified the parameters for big data programming models which can be used to assess the suitability of a model for a particular application. These parameters include fault tolerance, scalability, language, storage, and data processing. Furthermore, we overviewed the evaluation of these programming models. The application needs and the most suitable programming model was presented. We recommend the Apache Spark with in-memory storage for real-time data applications. Developers with basic SQL expertise should use models like Apache Hive and Apache pig. We also strongly suggest the implementation of APIs in other languages for the

MapReduce model. We suggest using Apache YARN for efficient resource utilization. MapReduce model. We suggest using Apache YARN for efficient resource utilization. In the future, we also plan to perform experiments on the benchmarks to evaluate and compare the performance of these programming models.

REFERENCES

[1] J. E. Grable and A. C. Lyons, "An Introduction to Big Data," *J. Financ. Serv. Prof.,* vol. 72, no. 5, 2018.

[2] M. Assefi et al., "A Study of Big Data Analytics using Apache Spark with Python and Scala," in *International Journal of Parallel, Emergent and Distributed Systems,* vol. 34, no. 6, pp. 632–652, 2017.

[3] K. Vassakis, E. Petrakis, et al., "Big data analytics: applications, prospects and challenges," in *Mobile big data, Springer,* pp. 3–20, 2018.

[4] E. Baro, S. Degoul, et al., "Toward a literature-driven definition of big data in healthcare," *Biomed Res. Int.,* vol. 2015, 2015.

[5] D. V. Dimitrov, "Medical internet of things and big data in healthcare," *Healthc. Inform. Res.,* vol. 22, no. 3, pp. 156–163, 2016.

[6] S. Shilo, H. Rossman, et al., "Axes of a revolution: challenges and promises of big data in healthcare," *Nat. Med.*, vol. 26, no. 1, pp. 29–38, 2020.

[7] A. Kamilaris, A. Kartakoullis, et al., "A review on the practice of big data analysis in agriculture," *Comput. Electron. Agric.*, vol. 143, pp. 23–37, 2017.

[8] K. Bronson and I. Knezevic, "Big Data in food and agriculture," *Big Data \& Soc.*, vol. 3, no. 1, pp. 2053951716648174, 2016.

[9] I. Carbonell, "The ethics of big data in big agriculture," *Internet Policy Rev.,* vol. 5, no. 1, 2016.

[10] N. Sun, J. G. Morris, et al., "iCARE: A framework for big data-based banking customer analytics," *IBM J. Res. Dev.,* vol. 58, no. 5/6, pp. 1–4, 2014.

[11] U. Srivastava and S. Gopalkrishnan, "Impact of big data analytics on banking sector: Learning for Indian banks," *Procedia Comput. Sci.,* vol. 50, pp. 643–652, 2015.

[12] H. Hassani, X. Huang, et al., "Digitalisation and big data mining in banking," *Big Data Cogn. Comput.,* vol. 2, no. 3, pp. 18, 2018.

[13] M. L. Stone, "Big data for media," 2014.

[14] D. V Shah, J. N. Cappella, et al., "Big data, digital media, and computational social science: Possibilities and perils," *Ann. Am. Acad. Pol. Soc. Sci.,* vol. 659, no. 1, pp. 6–13, 2015.

[15] A. Oboler, K. Welsh, et al., "The danger of big data: Social media as computational social science," *First Monday,* 2012.

[16] M. D. Smith and R. Telang, "Streaming, sharing, stealing: Big data and the future of entertainment*," Mit Press,* 2016.

[17] H. Lippell, "Big data in the media and entertainment sectors," *in New Horizons for a Data-Driven Economy, Springer, Cham,* pp. 245–259, 2016.

[18] H. W. Kim and M. Lee, "Big data and entertainment content: Case studies and prospects," *J. Internet Comput. Serv.*, vol. 17, no. 2, pp. 109–118, 2016.

[19] H. Zahid, T. Mahmood, et al., "Big data analytics in telecommunications: literature review and architecture recommendations," *IEEE/CAA J. Autom. Sin.,* vol. 7, no. 1, pp. 18–38, 2019.

[20] H. A. A. Hafez, "Mining Big Data in telecommunications industry: challenges, techniques, and revenue opportunity," *Int. J. Comput. Electr. Autom. Control Inf. Eng,* vol. 10, no. 1, pp. 183–190, 2016.

[21] M. Z. Kastouni and A. A. Lahcen, "Big data analytics in telecommunications: Governance, architecture and use cases," *J. King Saud Univ. Inf. Sci.,* 2020.

[22] J. Chin, V. Callaghan, et al., "Understanding and personalising smart city

services using machine learning, The Internet-of-Things and Big Data," in *2017 IEEE 26th international symposium on industrial electronics (ISIE)*, pp. 2050–2055, 2017.

[23] L. Belcastro, F. Marozzo, et al., "Programming models and systems for big data analysis," *Int. J. Parallel, Emergent Distrib. Syst.,* vol. 34, no. 6, pp. 632–652, 2019.

[24] D. G. Páez, F. Aparicio, et al., "Big data and IoT for chronic patients monitoring," in *International Conference on Ubiquitous Computing and Ambient Intelligence,* pp. 416–423, 2014.

[25] Y. Benlachmi and M. L. Hasnaoui, "Big data and spark: Comparison with hadoop," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4),* pp. 811–817, 2020.

[26] N. Sultana, M. Rüfenacht, et al., "Understanding the use of message passing interface in exascale proxy applications," *Concurr. Comput. Pract. Exp.,* vol. 33, no. 14, p. e5901, 2021.

[27] F. Nielsen, "Introduction to MPI: the message passing interface," *in Introduction to HPC with MPI for Data Science, Springer,* pp. 21–62, 2016.

[28] J. M. Abuin, N. Lopes, et al., "Big data in metagenomics: Apache spark vs MPI," *PLoS One,* vol. 15, no. 10, pp. e0239741, 2020.

[29] D. Wu, S. Sakr, et al., "Big data programming models," in *Handbook of Big Data Technologies, Springer,* pp. 31–63, 2017.

[30] L. Belcastro, R. Cantini, et al., "Programming big data analysis: principles and solutions," *J. Big Data,* vol. 9, no. 1, pp. 1–50, 2022.

[31] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," 2004.

[32] K. Shvachko, H. Kuang, et al., "The hadoop distributed file system," in 2*010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pp. 1–10, 2010.

[33] B. Jia, T. W. Wlodarczyk, et al. "Performance considerations of data acquisition in hadoop system," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science,* pp. 545–549, 2010.

[34] J. Nandimath, E. Banerjee, et al., "Big data analysis using Apache Hadoop," in *2013 IEEE 14th International Conference on Information Reuse \& Integration (IRI)*, pp. 700–703, 2013.

[35] V. K. Vavilapalli, A.C. Murthy, et al., "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing,* pp. 1–16, 2013.

[36] A. Mostafaeipour, A. J. Rafsanjani and et al., "Investigating the performance of Hadoop and Spark platforms on machine learning algorithms," *J. Supercomput.,* vol. 77, no. 2, pp. 1273–1300, 2021.

[37] P. Carbone, A. Katsifodimos, et al., "Apache flink: Stream and batch processing in a single engine," *Bull. IEEE Comput. Soc. Tech. Comm. Data Eng.,* vol. 36, no. 4, 2015.

[38] H. M. Makrani, S. Rafatirad, et al., "Main-memory requirements of big data applications on commodity server platform," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID),* pp. 653–660, 2018.

[39] M. Assefi, E. Behravesh, et al., "Big data machine learning using apache spark MLlib," in *2017 IEEE international conference on big data (big data),* pp. 3492–3498, 2017.

[40] S. J. Kang, S. Y. Lee, et al., "Performance comparison of OpenMP, MPI, and MapReduce in practical problems," *Adv. Multimed.,* vol. 2015, 2015.

[41] I. Chebbi, W. Boulila, et al., "A comparison of big remote sensing data processing with Hadoop MapReduce and Spark," in *2018 4th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, pp. 1–4, 2018.

[42] Y. Guo, W. Bland, et al., "Fault tolerant MapReduce-MPI for HPC clusters," in

Proceedings of the *International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2015.

[43] A. C. Sodan, "Message-passing and shared-data programming models-wish vs. reality," in 1*9th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, pp. 131–139, 2005.

[44] D. S. Kumar and M. A. Rahman, "Performance Evaluation of Apache Spark Vs MPI: A Practical Case Study on Twitter Sentiment Analysis," *J. Comput. Sci.*, vol. 13, no. 12, pp. 781–794, 2017.

[45] X. Lu, B. Wang, et al., "Can MPI benefit Hadoop and MapReduce applications?," in *2011 40th International Conference on Parallel Processing Workshops,* pp. 371–379, 2011.

[46] M. M. Rathore, H. Son, et al., "Real-time big data stream processing using GPU with spark over hadoop ecosystem," I*nt. J. Parallel Program.,* vol. 46, no. 3, pp. 630–646, 2018.

[47] U. Kang, C. Tsourakakis, et al., "Hadi: Fast diameter estimation and mining in massive graphs with hadoop," *ACM Trasactions Knowl. Discov. from Data,* vol. 5, no. 2, pp. 8, 2008.

[48] U. Kang, C. E. Tsourakakis, et al., "Pegasus: A peta-scale graph mining system implementation and observations," in *2009 Ninth IEEE International Conference on Data Mining*, pp. 229–238, 2009.

[49] S. Sakr, "Processing large-scale graph data: A guide to current technology," *IBM Dev.,* vol. 15, 2013.

[50] G. Malewicz, M. H. Austernet, al., "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 135–146, 2010.

[51] U. Kang, H. Tong, et al., "Gbase: a scalable and general graph management system," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1091–1099, 2011.

[52] R. S. Xin, J. E. Gonzalez, et al., "Graphx: A resilient distributed graph system on spark," in *First international workshop on graph data management experiences and systems*, pp. 1–6, 2013.

[53] K. Siddique, Z. Akhtar, et al., "Investigating Apache Hama: a bulk synchronous parallel computing framework," *J. Supercomput.*, vol. 73, no. 9, pp. 4190–4205, 2017.

[54] K. Siddique, Z. Akhtar, et al., "Apache Hama: An emerging bulk synchronous parallel computing framework for big data applications," *IEEE Access,* vol. 4, pp. 8879–8887, 2016.

[55] L.Y. Ho, T.H. Li, et al., "Kylin: An efficient and scalable graph data processing system," in *2013 IEEE International Conference on Big Data,* pp. 193–198, 2013.

[56] Z. Wang, Y. Bao, et al., "A BSP-based parallel iterative processing system with multiple partition strategies for big graphs," in *2013 IEEE International Congress on Big Data,* pp. 173–180, 2013.

[57] R. Chen, X. Ding, et al., "Computation and communication efficient graph processing with distributed immutable view," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing,* pp. 215–226, 2014.

[58] T. Li, C. Ma, et al., "Graph/z: A key-value store based scalable graph processing system," in *2015 IEEE International Conference on Cluster Computing,* pp. 516–517, 2015.

[59] G. Dai, Y. Chi, et al., "FPGP: Graph processing framework on FPGA a case study of breadth-first search," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays,* pp. 105–110, 2016.

[60] S. Aridhi, A. Montresor, et al., "BLADYG: A graph processing framework for large dynamic graphs," *Big data Res.,* vol. 9, pp. 9–17, 2017.

[61] R. Dathathri, G. Gill, et al., "Gluon: A communication-optimizing substrate for

distributed heterogeneous graph analytics," in *Proceedings of the 39th ACM SIGPLAN conference on programming language design and implementation,* pp. 752–768, 2018.

[62] M. Twaty, A. Ghrab, et al., "GraphOpt: a Framework for Automatic Parameters Tuning of Graph Processing Frameworks," in *2019 IEEE International Conference on Big Data (Big Data),* pp. 3744–3753, 2019.

[63] W. Fan, et al., "GraphScope: a unified engine for big graph processing," *Proc. VLDB Endow.,* vol. 14, no. 12, pp. 2879–2892, 2021.

[64] W. Daluwatta, R. D. Silva, et al., "CGraph: Graph Based Extensible Predictive Domain Threat Intelligence Platform," *arXiv Prepr. arXiv2202.07883,* 2022.

[65] S. Arora, A. Verma, et al., "An Overview of Apache Pig and Apache Hive," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.,* pp. 432–436, 2019.

[66] V. Garg, "Optimization of multiple queries for big data with apache Hadoop/Hive," in *2015 International Conference on Computational Intelligence and Communication Networks (CICN),* pp. 938–941, 2015.

[67] K. Bansal, P. Chawla, et al., "Analyzing performance of Apache Pig and Apache hive with hadoop," in *Engineering Vibration, Communication and Information Processing, Springer,* pp. 41–51, 2019.

[68] B. F. Cooper, A. Silberstein, et al., "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing,* pp. 143–154, 2010.

[69] T. Ivanov, T. Rabl, et al., "Big data benchmark compendium," in *Technology Conference on Performance Evaluation and Benchmarking,* pp. 135–155, 2015.

[70] R. Nambiar, M. Poess, et al., "TPC benchmark roadmap 2012," in T*echnology Conference on Performance Evaluation and Benchmarking*, pp. 1–20, 2012.

[71] A. Ghazal, T. Rabl, et al., "Bigbench: Towards an industry standard benchmark for big data analytics," in *Proceedings of the 2013 ACM SIGMOD international conference on management of data,* pp. 1197–1208, 2013.

[72] R. Han, X. Lu, et al., "On big data benchmarking," in *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware,* pp. 3–18, 2014.

[73] S. Huang, J. Huang, et al., "Hibench: A representative and comprehensive hadoop benchmark suite," in *Proc. ICDE Workshops,* pp. 41–51, 2010.

[74] M. Capotua, T. Hegeman, et al., "Graphalytics: A big data benchmark for graph-processing platforms," in *Proceedings of the GRADES'15,* pp. 1–6, 2015.

[75] K. Ouaknine, M. Carey, et al., "The PigMix benchmark on Pig, MapReduce, and HPCC systems," in *2015 IEEE International Congress on Big Data,* pp. 643–648, 2015.

[76] L. Wang, J. Zhan, et al., "Bigdatabench: A big data benchmark suite from internet services," in *2014 IEEE 20th international symposium on high performance computer architecture (HPCA),* pp. 488–499, 2014.

[77] P. Natesan, V. E. Sathishkumar, et al., ''A distributed framework for predictive analytics using big data and MapReduce Parallel Programming,'' *Mathematical Problems in Engineering,* pp. 1–10. doi:10.1155/2023/6048891, 2023.

[78] K. Subha and N. Bharathi, "Apache Spark based analysis on word count application in Big Data," *2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM),* vol. 2, pp. 491–495, 2022.

[79] A. Esmaeilzadeh, M. Heidari, et al., "Efficient large scale nlp feature engineering with apache spark," *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC),* pp. 274–280, 2022.

[80] A. Salzman and N. Moës, "A two-scale solver for linear elasticity problems in the context of parallel message passing," *Comput. Methods Appl. Mech. Eng.,* vol. 407, p. 115914, 2023.

[81] L. Xia, W. Sun, et al., “Blaze: A High performance Big Data Computing System for High Energy Physics,” *Journal of Physics: Conference Series,* vol. 2438, no. 1, pp. 12012, 2023.

[82] C. Piñeiro and J. C. Pichel, “A unified framework to improve the interoperability between HPC and Big Data languages and programming models,” *Futur. Gener. Comput. Syst.*, vol. 134, pp. 123–139, 2022.

[83] Z. Tian, P. Lindner, et al., “Generalizing Bulk-Synchronous Parallel Processing for Data Science: From Data to Threads and Agent-Based Simulations,” Proc. ACM Manag. Data, vol. 1, no. 2, pp. 1–28, 2023.

[84] A. Rudniy, “Data Warehouse Design for Big Data in Academia,” *Comput. Mater. \& Contin.,* vol. 71, no. 1, 2022.

[85] Bukhsh, Madiha, et al., “An Interpretation of Long Short-Term Memory Recurrent Neural Network for Approximating Roots of Polynomials,” IEEE Access 10, pp. 28194-28205, 2022.

[86] H. Tufail, M. U. Ashraf, et al., “The Effect of Fake Reviews on e-Commerce During and After Covid-19 Pandemic: SKL-Based Fake Reviews Detection,” IEEE Access 10 , pp. 25555-25564, 2022.

[87] M. Mumtaz, N. Ahmad, et al., “Modeling Iteration’s Perspectives in Software Engineering,” *IEEE Access 10, pp. 19333-19347,* 2022.

[88] M. Asif, J. K. K. Asamoah, et al., “A Novel Image Encryption Technique Based on Cyclic Codes over Galois Field,” *Computational Intelligence and Neuroscience 2022 ,* 2022.

[89] S. Mehak, M. U. Ashraf, et al., “Automated Grading of Breast Cancer Histopathology Images Using Multilayered Autoencoder.” *CMC-COMPUTERS MATERIALS & CONTINUA* 71.2 : 3407-3423, 2022.

[90] M. R. Naqvi, M. W. Iqbal, et al., “Ontology Driven Testing Strategies for IoT Applications,” *CMC-Computers, Materials & Continua.,* 70(3), pp. 5855-69, (1 Jan. 2022).

[91] S. Tariq, N. Ahmad, et al., “Measuring the Impact of Scope Changes on Project Plan Using EVM,” vol. 8, 2020.

[92] M. Asif, S. Mairaj, et al., “A Novel Image Encryption Technique Based on Mobius Transformation,” C*omputational Intelligence and Neuroscience,* (17 Dec. 2021).

[93] M. U. Ashraf, “A Survey on Data Security in Cloud Computing Using Blockchain: Challenges, Existing-State-Of-The-Art Methods, And Future Directions,” *Lahore Garrison University Research Journal of Computer Science and Information Technology,*Vol. *5,* no. 3, pp. 15-30, 2021.

[94] M. U. Ashraf, M. Rehman, et al., “A Survey on Emotion Detection from Text in Social Media Platforms,” *Lahore Garrison University Research Journal of Computer Science and Information Technology*; 5(2), pp. 48-61, (21 Jun. 2021).

[95] K. Shinan, K. Alsubhi, et al., “Machine learning-based botnet detection in software-defined network: a systematic review,” *Symmetry,* 13.5 , pp. 866, 2021.

[96] A. Hannan, F. Hussain, et al., “A decentralized hybrid computing consumer authentication framework for a reliable drone delivery as a service,” *Plos one,* 16.4 , e0250737, 2021.

[97] S. Fayyaz, M. K. Sattar, et al., “Solution of combined economic emission dispatch problem using improved and chaotic population-based polar bear optimization algorithm,” *IEEE Access,* 9, pp. 56152-56167, 2021.

[98] I. Hirra, M. Ahmad, et al., “Breast cancer classification from histopathological images using patch-based deep learning modeling,” *IEEE*, 9, pp. 24273-87, (Access. 2 Feb. 2021).

[99] M. U. Ashraf, F. A. Eassa, et al., “AAP4All: An Adaptive Auto Parallelization of Serial Code for HPC Systems,” *INTELLIGENT AUTOMATION AND SOFT COMPUTING,* 30(2), pp. 615-39, (1 Jan. 2021).

[100] T. Hafeez, S. M. U. Saeed, et al., "EEG in game user analysis: A framework for expertise classification during gameplay," *Plos one*, 16(6), e0246913, (18 Jun. 2021) .

[101] N. Siddiqui, F. Yousaf, et al., "A highly nonlinear substitution-box (S-box) design using action of modular group on a projective line over a finite field," *Plos one*,15(11), e0241890, (12 Nov. 2020).

[102] M. U. Ashraf, A. Hannan, et al., "Detection and tracking contagion using IoT-edge technologies: Confronting COVID-19 pandemic," 2020 international conference on electrical, communication, and computer engineering (ICECCE), IEEE, 2020.

[103] K. Alsubhi, Z. Imtiaz, et al., "MEACC: an energy-efficient framework for smart devices using cloud computing systems," *Frontiers of Information Technology & Electronic Engineering,* 21.6, pp. 917-930, 2020.

[104] S. Riaz, M. U. Ashraf, et al., "A Comparative Study of Big Data Tools and Deployment PIatforms," *In 2020 International Conference on Engineering and Emerging Technologies (ICEET), (pp. 1-6), IEEE,* (22 Feb. 2020).

[105] M. U. Ashraf , F. A. Eassa, et al., "Empirical investigation: performance and power-consumption based dual-level model for exascale computing systems," *IET Software*, 14(4), pp. 319-27, (27 Jul. 2020).

[106] M. U. Ashraf, K. M. Jamb, et al., "IDP: A Privacy Provisioning Framework for TIP Attributes in Trusted Third Party-based Location-based Services Systems," *International Journal of Advanced Computer Science and Applications (IJACSA),* 11.7, pp. 604-617, 2020.

[107] A. Manzoor, W. Ahmad, et al., "Inferring Emotion Tags from Object Images Using Convolutional Neural Network," *Applied Sciences,* 10.15, pp. 5333, 2020.

[108] K. Alsubhi, F Alsolami, et al., "A Tool for Translating sequential source code to parallel code written in C++ and OpenACC," *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), IEEE,* 2019.

[109] M. U. Ashraf, M. Naeem, et al., "H2E: A Privacy Provisioning Framework for Collaborative Filtering Recommender System," *International Journal of Modern Education and Computer Science,* 11(9),pp. 1, (1 Sep. 2019).

[110] M. U. Ashraf, I. Ilyas, et al., "A Roadmap: Towards Security Challenges, Prevention Mechanisms for Fog Computing," In *2019 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), (pp. 1-9), IEEE,* (24 Jul. 2019).

[111] M. U. Ashraf, R. Qayyum, et al., "State-of-the-art Challenges: Privacy Provisioning in TPP Location Based Services Systems," *International Journal of Advanced Research in Computer Science (IJARCS),* 10(2), pp. 68-75, (20 Apr. 2019).

[112] M. U. Ashraf, A. Arshad, et al., "Improving Performance In Hpc System Under Power Consumptions Limitations," *International Journal of Advanced Research in Computer Science,* 10(2), (Mar. 2019).

[113] Javed, Rushba, et al. "Prediction and monitoring ents using weblogs for improved disaster recovery in cloud." *Int. J. Inf. Technol. Comput. Sci.(IJITCS)* 11.4 : 9-17, 2019.

[114] Ali, Muhammad, et al. "Prediction of Churning Behavior of Customers in Telecom Sector Using Supervised Learning Techniques." *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE). IEEE*, 2018.

[115] M. U. Ashraf, F. A. Eassa and et al. Performance and power efficient massive parallel computational model for HPC heterogeneous exascale systems. *IEEE Access.* 2018 Apr 9;6:23095-107.

[116] M. U. Ashraf, F. A. Eassa and et al. Toward exascale computing systems: An energy efficient massive parallel computational model. *International Journal of Advanced Computer Science and Applications.* 9(2). (Jan. 2018).

[117] M. U. Ashraf, S. Arif and et al. Provisioning quality of service for multimedia applications in cloud computing. *Int. J. Inf. Technol. Comput. Sci.(IJITCS).*10(5):40-7. *2018.*

[118] M. U. Ashraf, F. A. Eassa and et al. Efficient Execution of Smart City's Assets Through a Massive Parallel Computational Model. InInternational Conference on Smart Cities, Infrastructure, Technologies and Applications, (pp. 44-51). Springer, Cham. (27 Nov. 2017).

[119] Alrahhal, M. Shady, et al. "AES-route server model for location based services in road networks." *International Journal Of Advanced Computer Science And Applications* 8.8 : 361-368. 2017.

[120] M. U. Ashraf, F. A. Eassa and et al. High performance 2-D Laplace equation solver through massive hybrid parallelism. In *8th International Conference on Information Technology (ICIT), (pp. 594-598). IEEE.* (17 May. 2017).