A Survey On Applications Of Parallel And Distributed System

Md Mostafa , Tahmid Ashrafee Promit , Md Akibur Rahman Khan , Md Ibrahim Ratul , Fahmim Tanjila , Md Sabbir Hossain , Md Humaion Kabir Mehedi , and Annajiat Alim Rasel

Department of Computer Science and Engineering
Brac University
66 Mohakhali, Dhaka - 1212, Bangladesh
{*md.mostafa, tahmid.ashrafee.promit, md.akibur.rahman.khan, md.ibrahim.ratul, fahmim.tanjila, md.sabbir.hossain1, humaion.kabir.mehedi*}*@g.bracu.ac.bd, annajiat@gmail.com*

***Abstract*—This article offers several parallel and distributed system applications. Different distributed and parallel system architectures have been created and are widely utilized in daily life. The properties of distributed systems, as well as their difficulties and restrictions, are examined in this paper. Additionally, the need for parallel and distributed systems is supported. We thoroughly examine several distributed and parallel system types, highlighting their importance and capabilities. For a better understanding of the enormous disparities between these systems and conventional methods, evaluation results are also supplied.**

***Index Terms*—Distributed System, Parallel Systems**

I. Introduction

Without distributed and parallel systems, modern computing would not be conceivable. The functioning of wireless networks, cloud computing services, and the internet all depend on them. Any of these technologies would not exist if distributed systems did not.

The millions of nodes that make up the internet are connected by a wide variety of network hardware and protocols, and each one of them is equipped with a large amount of computational and storage power that, only a few years ago, could only be found in high-end servers. Additionally, the Internet's nodes are now more securely coupled to robust high-speed networks than ever before. An exponential increase in data volume. It is highly challenging to handle, store, and evaluate this data using a conventional method. We need distributed computing frameworks to process these enormous volumes of data more quickly.Because they may tap into the capabilities of other computing devices and processes,Features offered by distributed systems might be impossible or extremely tough to accomplish on a single system. Distributed systems offer scaling and improved productivity in certain ways that monolithic systems cannot.

Distributed environments are essential for processing and analyzing massive volumes of data in a wide range of applications, including aerodynamic research, weather forecasting, and scientific applications. One of the major problems for these kinds of computations as data volume grows is the requirement to offer effective, user-friendly solutions. The usage of Distributed File Systems is the greatest remedy for this problem.

In this modern era huge data is being sent globally. To process the data conveniently, a system must efficiently schedule the data. Three general approaches may be used to implement parallel scheduling algorithms: local scheduling, dynamic scheduling, and explicit co-scheduling. However, because there is no coordination between processors, the speed of these algorithms declines, and they only function for locally occurring events. They are also neither scalable nor stable. This distributed system initiates a new methodology which is known as Buffered Co-scheduling, it is a scheduling technique to share communication time among distributed and parallel systems. The new scheduling technique improves resource utilization. Additionally, it also permits the design of a global scheduling policy while keeping the implicit co-scheduling-provided overlap of processing and communication. Additionally, it is a highly quick and efficient strategy.

Huge amounts of data are transferred globally, much as the upper case. This data transmission generates a ton of load on cloud computing systems. Cloud platforms have not been able to consistently monitor inter-application traffic in a way that is independent of any specific application and without incurring considerable expenditure. Inter-application traffic is a major factor that directly represents how components interact. Black-box monitoring of a distributed system suitable for addressing these issues.This method uses a kernel-assisted event-driven approach is the technique to attaining maximum detail or little overhead specialized application instrumentation.This method also provides a distinct trade-off as compared to conventional monitoring techniques as a result of its focus on properly measuring the amount of data transported throughout processes.

In our modern era of information, high reliability is a crucial component of many systems. Due to difficulties sustaining dependability, the expansion of software and testing, among other issues, enough software trial frequently is impossible. Particularly, it could be hard to assess a distributed and parallel system in real life. The ability to tolerate failures in both hardware and software is a need for highly dependable systems. A revolution is brought about by the D-Cloud here. A fault-injection virtual machine and parallel software testing environments are offered by this distributed testing environment for dependable distributed and independent systems.

We are aware that deep learning and neural networks are the future. Applications for deep learning and machine learning are expanding rapidly. Efficiency in applying these models is a typical issue. Batching operations are a regular chore in these models. It requires parallel processing to complete this work effectively. The algorithm must execute concurrently. Thus, parallel processing is essential.

II. Applications

A. Big Data Computation

Big data calls for extremely low-cost infrastructure and storage, as well as extremely quick processing of enormous amounts of data. Both cloud computing and cluster computing, which are subsets of distributed computing, are necessary for these. Therefore, batch processing and stream processing are necessary to speed up the processing of huge data. In Hadoop systems, MapReduce, Spark, and Flink are utilized for batch processing; Storm and Sanza are used for stream processing.

1) Big Data Distributed Computing Processing Frameworks: To test the effectiveness of Apache Spark and Hadoop MapReduce on various-sized datasets, an experiment [1] employing a single node Hadoop installation was carried out for this study. An i5 quad-core desktop machine with 8 GB of RAM, a 1 TB HDD, the Ubuntu 16.4 OS, and the Hadoop 2.7 software were utilized. NameNode, jobtracker, data node, and tasktracker are the components of a Hadoop single node installation that run on the same system and the outcomes are depicted below. For instance, the Linux operating system's word count function is not accessible, which means that utilizing the usual tools and approaches to analyze very large data sets will take much longer.

Datasets file size in MB (Approx)	Processing time (s)		
	Spark	MapReduce	Linux
2 MB	7.32	30.74	150.42
4 MB	7.91	31.52	170.33
40 MB	12.93	32.88	unreached
60 MB	14.59	39.12	unreached

Fig. 1. Comparison data on disk

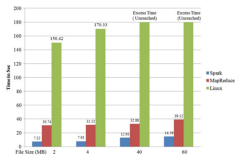

Fig. 2. Performance comparison

The conclusion and analysis highlight the significance of parallel and distributed computing within the Hadoop architecture. Depending on the situation, the state of the data to be processed, the requirements, the time limitations, and the user's interest in a particular category of results, the appropriate framework will change.

B. Distributed File System

The Internet currently has millions of nodes, each of which has crucial computing and storage hardware. The nodes are further connected by strong, quick networks. Additionally, the amount of data is now growing exponentially, necessitating quicker I/O. Consequently, the development of distributed file systems is required. It is possible to create a new set of file storage systems that collaborate with one another to support a range of end-user applications by properly combining edge and cloud computing systems.

1) Julunga A Distributed File Storage System: In this study [2],Julunga is an advanced distributed file storage solution for cloud computing environments. The metadata, file data blocks, and metadata are distributed across the file storage system. Exabyte-sized files with multiple the same file or directory is updated, read, and written to by many users concurrently can be easily handled by Julunga. A name is made up of a string of one or more alphabetic letters. JuFS offers advanced security that guarantees both the non-repudiation of transactions done in the file system name-space and the privacy, correctness, and consistency of the data recorded.

Multiple concurrent users can be supported by specifying lock levels at the byte level. A brand-new load balancing and consensus algorithm mechanism is also part of the distributed file system. Unlike other consensus algorithms that don't use broadcast techniques, it doesn't require a leader election.

2) Bigtable For Structured Data: A distributed storage system called Bigtable is capable of managing petabytes of data over tens of thousands of shared machines. A few Google projects that use Bigtable to store data include Google Finance, Google Earth, and web crawling.

Implementation :The Bigtable system consists of 3 main parts: one master server, multiple tablet servers, and a library that is connected into each client. Each tablet server oversees a certain number of tablets; typically, Each tablet server has between ten and a thousand devices. For reads and writes, clients connect directly to tablet servers.

Tablet Location : Even though tablet locations are maintained in memory and no longer require GFS trips, even if the client library is instructed to prefetch tablet locations, nevertheless lower this cost in the typical scenario.

Tablet Assignment: If the split notification is absent, the master notices the new tablet when it requests that the split tablet be loaded from a tablet server. Bigtable's ability to recoup the resources used by deleted data while simultaneously ensuring that it is immediately eliminated from the system through major compactions is essential for services that store sensitive data.By sending a tablet load request to the tablet server, the master can assign a tablet that is available but not yet assigned and has enough space for it. The redo points in this metadata provide links to any commit logs that might have tablet data as well as a list of the SSTables that make up a tablet. Then, a frozen memtable-derived SSTable is written to GFS. In order to help the master distribute its tablets more quickly, whenever a tablet server crashes, it tries to unlock its lock.

Other businesses can increase the amount of per-user data in their own columns thanks to Bigtable's design. Users have access to their search history, which enables them to examine earlier searches and actions and receive customized search results based on their past Google usage patterns. From April 2005, bigtable clusters have been used in actual production. Google invested approximately seven person-years in the design and development of its products. Bigtable was being used by more than sixty projects as of August 2006. As users' resource requirements alter over time, the clusters' capacity may be increased.

C. Distributed Co-Scheduling

1) Buffered Co-Scheduling System: [3] A scheduling technique for sharing communication time among parallel and distributed systems - focuses mainly on communication buffering and strobing. By scheduling the communicating process, it more systematically makes use of global information. Strobing enables the consistent, global sharing of information, separating communication from synchronization. The experiment made use of two 32 processing node networks, each representing a different architectural approach. The first one consisted of a 5-dimensional cube topology and network adapters and routing that resemble Myrinet.

The second network is built on a 32-port, 100 Mb/s Intel Express switch, a well-liked technology because of its good price-performance ratio. This allows the design of a global scheduling policy while keeping the implicit co-scheduling-provided overlap of processing and communication. In conclusion, this study implemented a new approach to multitasking in distributed and parallel systems.

D. Distributed Monitoring System

1) Black Box Monitoring: A distributed system's black-box monitoring that can be applied in a number of ways. Using a kernel-assisted event-driven method, rather than customized application instrumentation, is the key to getting great detail and cheap overhead. [4] applies a prototype implementation to multi-platform micro-service deployments to demonstrate how to implement a prototype implementation [4] mentioned a monitoring agent that creates a system representation in the form of a volume-weighted graph of data transferred by intercepting important Linux system calls using eBPF and carefully aggregating data from the operating system kernel [4] use well-known applications for micro-services made by Google and WeaveWorks to illustrate it.

This paper [4] states a working prototype method, which uses a black box to monitor resource usage and cross-process interactions. In this prototype, the graph representation is stored and queried using Neo4j. for queuing, together with Apache Kafka.

Agent and Data Processor are its two key parts. While the Agent filters the data using kernel space, the Data Processor turns the data into a graph representation. It includes a full eBPF application, a proc reader, and tools for kernel routine probing.Operators can run a graph query for an analysis. While monitoring network traffic directly without any in-kernel aggregation, [3] first demonstrates eBPF instrumentation when performing, producing a large number of occurrences to test the suggested strategy. Iperf is a device for calculating the highest bandwidth that an IP network can provide. They created a localhost iperf server.

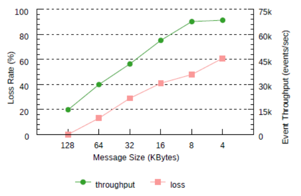

Fig. 3. Loss rate with increasing throughput

The message size for Iperf was set to 128 KB, which resulted in a loss rate of 0 in the eBPF scalability test. The findings undeniably demonstrate that CPU utilization in user mode has a significant impact on iperf performance, amounting to roughly 68 percent.This paper [4] conducted a validity test of their strategy by utilizing two complex, contrasting microservice-oriented applications: Hipster Shop and Sock Shop. It is possible to draw the conclusion that our black-box strategy is compatible with large, complicated, and heterogeneous architectures, numerous platforms, and even gets across virtualized network layers. They [4] also mentioned two strategies Q1(Scan Opt) and Q2(Shuffle Opt) one through automatic placement and another through manual optimization. Another overhead monitoring method that compares runtimes was put into practice. The observation result confirms that a commodity server would be able to manage events from a deployment with hundreds of servers carrying out a similar workload as well as carry out concurrent queries on the obtained data. So to conclude, the mentioned method provides a distinct trade-off as compared to conventional monitoring techniques because It concentrates on correctly calculating the amount of data.

E. Distributed Testing System

1) D-Cloud: D-Cloud was built on top of the Eucalyptus cloud computing platform. It uses fault injection to test distributed systems' software, which uses cloud technology. System tests may be executed via fault injection without changing the software. While utilizing a virtual machine, a flaw in the virtual machine's operating system has no effect on the host OS. However, while utilizing a virtual machine, a flaw in the virtual machine's operating system has no effect on the host OS, which is running on a real system.

D-Cloud is capable of testing distributed systems using a range of guest OSes. The virtualization program QEMU [5] and the open-source implementation Eucalyptus are both used by D-Cloud. Being an open source software it's very convenient to add a fault injection function to it. After that if a testing environment is reproducible, the tester can identify flaws that only appear under certain circumstances and resolve them. The D-Cloud software test environment system has the ability to automatically build test environments, run tests, and insert defects into hardware components in a virtual machine utilizing Eucalyptus and QEMU.

When a tester provides a test scenario together with the setup files and scripts, the flow is started. While snapshots are taken in line with the test scenario, system testing and fault injection are being performed.

F. Distributed Object Model

1) Distributed Object Model For Java: This distributed object model system is especially designed to function in the Java environment in order to support distributed objects in Java. In order to satisfy the semantics of object invocation, [6] constructed object model systems that need RMI. The proposed system for the Java programming language assumes that the Java Virtual Machine is a homogenous environment, allowing the system to follow the Java object model whenever feasible.

A remote object is one whose functions can be accessed from another address space, possibly on a separate system. A remote object is a sort of object that lacks access to a local object's methods. The act of calling a method (of a remote interface) on a distant object is known as remote method invocation or RMI. Remote interfaces, not their implementation classes, are what clients of remote objects program to. An additional failure mode that can happen during any remote method invocation must be handled by clients. The failure semantics of local items are fundamentally distinct from the failure modes of accessing remote objects. A remote object's remote interface must be defined before it can be implemented. This proposed interface has no methods and is entirely abstract. There are two ways to implement a remote interface. First one is remote implementation reuse and the second one is local implementation reuse.

There are three fundamental layers that make up the proposed RMI system. Those are: stub, RMI reference layer and the last one is transport. The stub layer also connects the application layer to the remainder of the RMI system.The lower level transport interface is controlled by the RMI reference layer. This layer is also in charge of carrying out a special invocation protocol that is independent of the consumer stubs and service skeletons. Lastly, It is possible for there to be more than one transport implementation because a transport specifies what a concrete representation of an endpoint is.

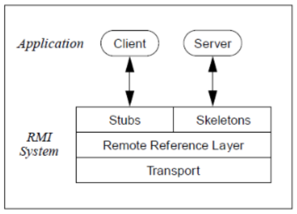

Fig. 4. System Architecture

The methods employed in [6] are intended to reduce the level of complexity felt by both the clients utilizing remote objects and the servers implementing them. [6] also makes it possible for Java objects to be invoked remotely in a seamless manner across different virtual machines. The end product offers a straightforward model that integrates nicely with the Java framework and is usable on a range of devices.

G. Distributed Transactions and Notifications

Every day, billions of updates to Google's indexing engine, which is performed on thousands of machines, store and analyze tens of petabytes of data. Small updates cannot be handled individually by MapReduce or other batch processing systems since they depend on the creation of big batches to be effective. These tasks are beyond what the present infrastructure is able to handle.

1) Distributed Transactions and Notifications for Large-Scale Incremental Processing: In this study [7], researchers developed Percolator, which was used to build the Google online search index and progressively process updates to a big data collection. The changes allowed for the processing of the same volume of documents each day while minimizing the average age of information obtained by Search queries by percent.

Imagine a system that is attempting to create a web index. A sequence of MapReduce operations can be used to define this bulk processing task. Every web page must be crawled in order to process each input and keep an index with a set of invariants. Moreover, how should a web index be updated following a thorough web crawl? Run the Map Reduces command over the whole repository, including the new and old pages. There are links in between new updated pages and the rest of the internet, so running it over just those pages is insufficient. A data processing system that is geared for incremental processing would be excellent. One would be able to efficiently update a sizable document repository as each new document was crawled. The indexing system might use transactions to ensure invariants while updating individual documents and storing the repository in a DBMS.

This [7] paper's portion examines how MapReduce might be expanded to handle documents individually rather than performing global scans of the repository using Percolator. Percolator offers ACID-compliant transactions because high throughput requires the simultaneous transformation of the repository by numerous threads running on numerous machines. Percolator is designed specifically for processing massive volumes of data incrementally. Bigtable or MapReduce are superior at handling calculations whose output cannot be divided into little updates. Traditional DBMSs like Spark or NetBeans are capable of handling simpler computations. The new Google Percolator tool processes pages as they are crawled and added to the live online search index. The need to operate at huge sizes and the absence of a necessity for extremely low latency had an impact on the architecture. The distributed storage platform Bigtable serves as the foundation for Percolator. The web search index for Google was created by Percolator. With a tolerable increase in resource use, the system met the objectives established for lowering the delay of indexing a single document. The findings point to an exciting area for distributed storage system development in the future.

H. Batching Operation In Neural Network

1) Static Automatic Batching in TensorFlow : It can be difficult to efficiently utilize dynamic neural networks, despite their increasing popularity. Manual batch version creation can be difficult and error-prone, and run time overheads are an issue for such models when batching operations on the fly. To address this, a parallel-for loop improved by static loop vectorization [8], to TensorFlow is included. Users may define estimation using pfor by using layered loops and conditional structures. Benchmarks show speedups of one to a pair of orders for a variety of workloads. Implementations: Using SPMD parallel for loop [8] a stacked tensor structure was generated. Graph Auto-Vectorization:By utilizing a GraphDef to represent the core of the feedback path and a scalar Tensor to indicate the number of iterations, the vectorization approach [8] creates a collection of GraphDef components that provide functionality equivalent to running pfor with the given parameters. Higher order gradients along with Jacobians Researchers [8] have been able to determine the eigenfunctions of linear operators using stochastic optimization, study RNN fixed points, and examine the eigenvalues of Jacobians in machine translation decoders. Benchmarks : Given that sequence lengths are chosen at random, pfor's sequential loop repeats increasingly work on smaller batches, lowering the effective batch size and accelerating processing. In order to achieve hardware speed as the batch size increases, pfor may recast the computation for this setup. The trials were run using an NVIDIA Maxwell Titan X GPU, 64GB of RAM, and a 6 core Intel [Xeon E5-1650 3.60GHz CPU] [8]. Since the kernel execution is iterated through the output items in a stream-based execution scenario on the GPU, the execution gets serialized. Due to the experimental nature of TensorFlow, models that employ batch standards have been left out of these benchmarks. Pfor requires much larger batch sizes to perform better here than in other implementations since the margins are narrower. For both small and big batch sizes, For continues to perform better than the other three.

TensorFlow's sequential loop can be accelerated by up to two orders of magnitude, while DyNet with runtime batching can accelerate by an order of magnitude, according to GPU benchmarks. A parallel-for loop with SPMD semantics has been introduced to Tensorflow.

III. Discussions And Limitations of Existing Works

We have discussed many distributed system types in this paper, along with their benefits and applications. Below are the thoughts and advice that we came to.

A. Big Data Computation

Frameworks for Big Data Distributed Computing

- The best framework will differ based on the conditions, the readiness of the data to be processed, the requirements, the time restrictions, and the user's interest in a particular category of results.
- There are trade-offs when working on broadly focused tasks versus implementing as a whole, and the same logic holds true when choosing to prioritize cutting-edge and creative achievements over time-tested alternatives.

B. Distributed File System

1) Julunga:

- Due to space constraints, the theoretical findings and experiments conducted in the real world are deleted

2) BigTable:

- Large-scale distributed systems are vulnerable to a variety of failures, in addition to the usual network partitions and fail-stop problems that are inherent in many distributed protocols.
- Delaying the addition of new features is crucial until it is obvious how the new features will be used.
- The value of thorough system-level monitoring
- A simple design's value.

C. Batching Operation In Neural Network

1) Static Automatic Batching in TensorFlow:

- By converting at the tensor level of abstraction, one can take advantage of the mathematical features of the operations rather than inferring them from highly nested loop nests.
- TensorFlow operations, which in turn employ highly optimized kernels, are used to describe the vectorization process' output.
- TensorFlow kernel-level vectorizing necessitates navigating a considerably more extensive and challenging instruction set.
- This is a problem for vectorization, both in terms of accuracy and optimization tailored to the various meanings.

D. Distributed Co-Scheduling System

1) Black-box monitoring of a distributed system:

- The working sets of different occupations are affected differently by the memory hierarchy, however this is not taken into account in the preliminary experimental results.
- Buffered co-scheduling requires more main memory in order to avoid memory swapping.
- Unable to employ buffered co-scheduling, assess the throughput and reaction time of parallel jobs.
- Could not evaluate its performance in comparison to commercial space-sharing solutions or explicit co-scheduling.

E. Distributed Testing System

1) D-Cloud:

- Making decisions about how to allocate resources and where to deploy employees purely on the basis of information collected without the application's awareness and while treating it like a black box.

- To take advantage of the application's features both automatically and manually by utilizing all of their potential.
- Only could implement and test the system in python.

F. Distributed Transactions and Notifications

1) Distributed Transactions and Notifications for Large-Scale Incremental Processing:

- Percolator lacks a global deadlock detection and a central place for transaction management.
- The challenge for the team is to provide functionality that Bigtable does not have, such as multi row transactions and the observer architecture. Bigtable rows and columns are used to arrange the data, with extra columns holding the metadata.
- Percolator must effectively identify dirty cells with observers that need to be tracked in order to provide alerts, then apply those findings to the data table.
- The researcher selected a design that, in comparison to conventional database systems, has a considerable 30-fold overhead and scales linearly across many orders of magnitude on commodity equipment.

G. Distributed Object Model

1) Distributed Object Model For Java:

- The RMI system must handle remote object garbage collection and support additions like server replication and the activation of persistent objects to handle invocations.
- In order for programmers to handle this failure properly, their challenge was to expose the extra exception RemoteException in all calls to remote methods.

Conclusion

In the age of the fourth industrial revolution, huge data is generated every second. The data must be managed in a way that is scalable, capable of delivering capabilities that would be difficult or impossible to create on a single system, and allows for better data processing. Our aim was to give a brief about different applications of distributed systems which are able to facilitate data handling effectively. We evaluated all of these models by examining the dependability of different system components that impact on the overall system's dependability. We came to the conclusion that existing models have identified key issues and generated a number of models to increase system reliability.

References

[1] G. S. Bhathal and A. Singh, "Big data computing with distributed computing frameworks," in *Innovations in Electronics and Communication Engineering*, pp. 467–477, Springer, 2019.

[2] H. Dewan and R. C. Hansdah, "Julunga: A new large-scale distributed read-write file storage system for cloud computing environments," in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pp. 942–951, IEEE, 2018.

[3] F. Petrini and W.-c. Feng, "Buffered coscheduling: A new methodology for multitasking parallel jobs on distributed systems," in *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000*, pp. 439–444, IEEE, 2000.

[4] F. Neves, R. Vilaça, and J. Pereira, "Detailed black-box monitoring of distributed systems," *ACM SIGAPP Applied Computing Review*, vol. 21, no. 1, pp. 24–36, 2021.

[5] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, and M. Sato, "D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 631–636, IEEE, 2010.

[6] A. Wollrath, R. Riggs, and J. Waldo, "A distributed object model for the javaˆ tˆ m system," *Computing Systems*, vol. 9, pp. 265–290, 1996.

[7] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications," in *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[8] A. Agarwal, "Static automatic batching in tensorflow," in *International Conference on Machine Learning*, pp. 92–101, PMLR, 2019.

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[10] T. Lu, S. Hoyer, Q. Wang, L. Hu, and Y.-F. Chen, "Distributed data processing for large-scale simulations on cloud," in *2021 IEEE International Joint EMC/SI/PI and EMC Europe Symposium*, pp. 53–58, IEEE, 2021.

[11] U. A. Kashif, Z. A. Memon, A. R. Balouch, and J. A. Chandio, "Distributed trust protocol for iaas cloud computing," in *2015 12th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pp. 275–279, IEEE, 2015.

[12] A. Wollrath, R. Riggs, and J. Waldo, "A distributed object model for the java system," *Comput. Syst.*, 1996.

[13] https://www.researchgate.net/publication/330931223_Big_Data_Computing_with_Distributed_Computing_Frameworks. Accessed: 2022-8-29.

[14] G. S. A. Singh, "Comparison data-on-disk different size datasets and time is taken by frameworks [photograph," in *Big Data Computing with Distributed Computing Frameworks*, 2019.

[15] F. Neves, R. Vilaça, and J. Pereira, "Detailed black-box monitoring of distributed systems," *ACM SIGAPP Appl. Comput. Rev.*, vol. 21, pp. 24–36, Mar. 2021.