Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

A secure operating system for data centers: A survey

Sikandar Ejaz [1,*], Muhammad Javed Iqbal [1], Hafsa Bibi [1], Shahbaz Pervez [2], Kawther A. Al-Dhlan [3], Seyed Ebrahim Hosseini [2]

[1]*Computer Science Department, University of Engineering and Technology, Taxila, Pakistan*
[2]*Department of Information Technology, Abacus Institute of Studies, Christchurch, New Zealand*
[3]*Computer Science and Information Department, College of Computer Science and Engineering, University of Hail, Hail, Saudi Arabia*

ARTICLE INFO

Article history:
Received 11 March 2019
Received in revised form
25 April 2020
Accepted 2 May 2020

Keywords:
Data centers
Operating system
Resource sharing
Security
Scalability
Application development
Cluster
Cloud computing

ABSTRACT

Data centers are now evolving source of computational hardware which have high potential to bring extraordinary computing capacity to use applications with resource sharing, fault tolerance, security, and scalability. To deliver the user with efficient computational power, with the support of data sharing, resource sharing and abstraction, an operating system-like software stack is needed for cloud computing hardware platforms. Existing distributed operating systems are not scalable to handle thousands of machines in clouds. As a result, current cloud computing environments are more complex at the user side. This paper surveys the existing data center functional platforms and discusses their worth and cost, to emphasis on development of a long-term mechanism with lasting impacts for present and future data center software infrastructure demands by considering all these factors which will help the organizations to select the best operating system for datacenter as per their particular needs and priorities.

1. Introduction

Commodity servers in data centers are now capable of providing dominant platforms not only for handling data-intensive and computational-intensive applications of enterprises, technical and objective workload but also for providing prevalent services such as e-commerce, social media networking, online communal interactions, gaming, computing resource provisioning and web search (Bahl et al., 2012). In these data centers, even one separate application can engage hundreds of servers, containing multiple software services. Concerning high stream performance and networking measurements of these data centers, the new requirements involve the development of a comprehensive protection model that would be able to program and manage the data center needs all alone. In other words, a complete operating system like-layer is required (Zaharia et al., 2011) to control data center applications and user diversity, and specialized tasks like flexible resources sharing and utilization, handling of substantial number of connections and churn, optimized solutions for minimizing constraints like delay and latency, and supervision of software programming.

Datacenter operating system can be taken as a software stack layer that can perform the same functionalities in a data center, as a typical OS performs in a single machine (Zaharia et al., 2011; Barroso et al., 2013; Patterson, 2008). In general, an operating system facilitates users with several packages and tools like programming languages and compilers, and an intermediate layer for managing virtualization of hardware resources. Typically an OS can carry out quite a lot of essential survival operations which can be: (i) support a machine for multitasking and then for interaction among user tasks by resource sharing, (ii) merger of standalone application programs through files and pipes to offer optimized solutions to problems i.e. data sharing among application programs, (iii) relief in creation of software applications through programming abstractions and (iv) extensive monitoring and debugging utilities for whole system. These crucial characteristics enable an operating system to encourage interdependent applications to be executed in a well-organized environment.

Like computers, data centers also require an OS-like layer to manage the growing diversity of users and applications. From application point of view,

* Corresponding Author.
Email Address: sikandar.ejaz@students.uettaxila.edu.pk (S. Ejaz)
https://doi.org/10.21833/ijaas.2020.08.007
Corresponding author's ORCID profile:
https://orcid.org/0000-0003-0232-2356

several cluster computing frameworks like Wang et al. (2014, 2013), Shvachko et al. (2010), Isard et al. (2007), Zaharia et al. (2010a), Dean and Ghemawat (2008) and Malewicz et al. (2010) are in existence. But they are using case-specific frameworks, no common interface is there for accessing data and resources for independent applications. This paper provides an overview of present frameworks along with their values and challenges and answers to the questions: How these frameworks offer some of the OS services? How can be a common interface for data centers? How an OS layer can be developed for large-scale data and computation extensive applications in single controller environments? By doing this, our work discusses the main challenges and requirements of data centers to raise spirits of researchers, so they look for the implementation of a software stack for interoperable cloud computing environments.

The remaining sections of the paper are structured as follows. Section II focuses on important data center requirements that an operating system must care about. Section III is the main part of this article, which constitute traits and silences of several cluster computing frameworks. A comparison of these frameworks has been made in section IV. Section V discusses the key features that have been obtained from the survey, followed by future directions. Finally, section VI concludes the paper.

2. Research problems and objectives

This section describes the problems and objectives of this research, which comprises of core concerns for which a datacenter OS is supposed to be responsible for.

2.1. Resource sharing

Diverse datacenter applications like web applications, long term services, storage systems, and batch programming are currently managed by only a set of nodes on which they execute. Although corresponding physical or virtual hosts do resource sharing among these applications at a coarse-grained level dynamic increase in a number of applications and their requirements need resource sharing at some sort of fine-grained level to entertain multiple computing frameworks. Besides, management of network sharing between multiple datacenter application traffic (Greenberg et al., 2008) and interdependent services, scheduling of resources with optimized response time, throughput and energy consumption, and virtualization of computing resources and then migration of virtual machines are also needed to be implemented efficiently (Jadeja and Modi, 2012; Zhang et al., 2010).

2.2. Data sharing

Parallel applications in a data center need intercommunication for data sharing to maintain the workflow of steps of single tasks written in multiple programs. For example, in MapReduce, Map and Reduce tasks need an exchange of data to complete an action. Client applications require abstractions and standard interfaces for data sharing. Existing distributed filesystems (Wang et al., 2014; Shvachko et al., 2010) are more interdependent abstractions because they may take more time in reading and writing datasets from file systems resulting in a lack of reliability. Also, these distributed filesystems do not perform well in the case of streaming data. Batch query-dependent live data analytics become a challenge and call for a more refined design approach.

2.3. Programming abstractions

Data centers involve more multifaceted hardware and their related issues such as failure detection, recovery, and miserable performance of nodes. During an application development phase, data centers require programming abstractions to cover the extra hardware complexities for making the program writing simpler. Several parallel cluster computing frameworks (Isard et al., 2007; Dean and Ghemawat, 2008; Malewicz et al., 2010) are problem-solving approaches, but they lack in a common abstraction to facilitate the system and software design. For flawless and time-saving system and productivity programming, a common abstraction in the form of data center OS is necessary to support fundamental primitives such as APIs for initiation and monitoring of tasks, similar communication patterns and coordination (Burrows, 2006) in different distributed joins and fault-tolerance in distributed data structures.

2.4. Debugging and monitoring

Debugging and monitoring of massively distributed applications is also crucial due to the requirement of distributed debugging tools in data center environments. Error correction, pattern complexities and larger objects in parallel applications require a single tracking interface-like software stack so that old systems like Aviram et al. (2012), Fonseca et al. (2007) and Sakr et al. (2013) can be re-implemented to handle changing workloads.

3. Datacenter functional frameworks

From an application point of view, numerous well-known data center frameworks with their traits and silences toward the above mentioned requirements have been discussed in this section.

3.1. Mesos

Mesos permits fined-grained resource sharing through applications in data centers. Mesos uses an application control scheduling model called resource

offer, which is a distributed two-level mechanism. The main objective of Mesos is to provide a controlling layer that allows varied arrangements of cluster computing structures to efficiently allocate and utilize resources (Hindman et al., 2011).

Mesos defines an optimal interface that empowers the use of well-organized resource sharing across cluster structures in data centers, and then, allows the interface to forward control of task scheduling and implementation to that structures due to highly diverse and rapidly evolving assembly of clusters. Mesos is scalable and robust in achieving data locality, dealing with faults, evolving solutions independently and preserving the required change rate of the system at low frequency. In architectural terms, Mesos includes a slave process that is a dependent process and is managed and organized by a master process running on each cluster node, which is illustrated in Fig. 1. In operations, master employs fine-grained sharing of resources by using a technique, "resource offers" which shows a list of freely available resources on distributed slaves. Masters' organizational policy is responsible for offering a fair share of selected resources to each in working cluster groups. Mesos components include a master component which is combined with a scheduler to deliver resources and an originator process which is executed on multiple slave nodes to run the cluster's tasks. As Fig. 1 is showing the detail about the architecture.

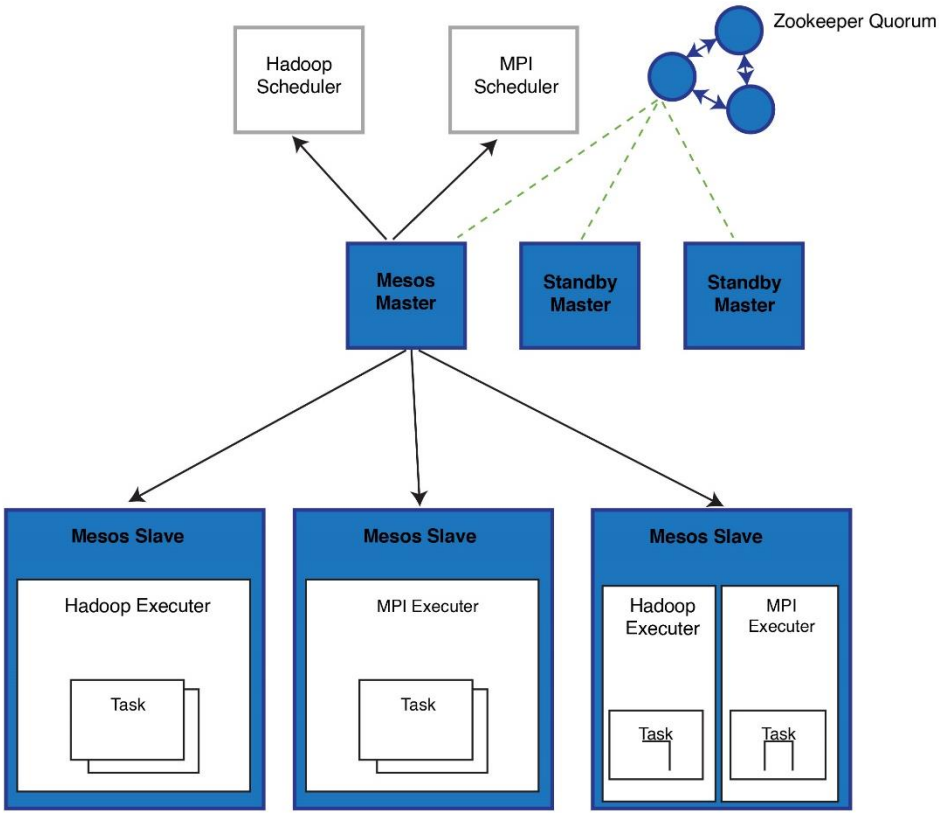

Fig. 1: Mesos architecture-Hadoop and MPI running (two frameworks)

It is noticed that Mesos works well with distributed scheduling but has limitations of fragmentation of resources in heterogeneous resources demand in uneven distribution of loads and tasks on small and large nodes, interdependent framework constraints and more complex scheduling of resources due to composite scheduling policies.

3.2. Dryad

Dryad is a general-purpose, parallel distributed programming system framework, meant for coarse-grained data-parallel applications. The Dryad application is aimed to form a data flow graph by associating computational vertices with interconnecting channels (Sakr et al., 2013). The main components and organization of the Dryad system are depicted in Fig. 2. Dryad implementation includes the execution of applications containing vertices of graphs on a set of available computers, cooperating by message passing through files, shared-memory FIFOs and TCP pipes. The vertices are typically written in sequential and consecutive programs without thread formation or locking in application program and then executed at the same time on multiple cores of a single processor inside a computer or multiple computers in a cluster efficiently schedule, allocate and utilize available resources (Isard et al., 2007) and the Fig. 2 is showing the detail regarding the system organization in dryad framework.

Dryad is intended to range from dominant multi-core solitary computers to trivial groups of clusters of computers, and then data centers containing an enormous number of machines. Dryad systems are good in recovering from node or communication failures and transportation of data among multiple

vertices, scheduling through resources, enhancing the degree of concurrency inside an independent computer, and delivering of data to its required place.

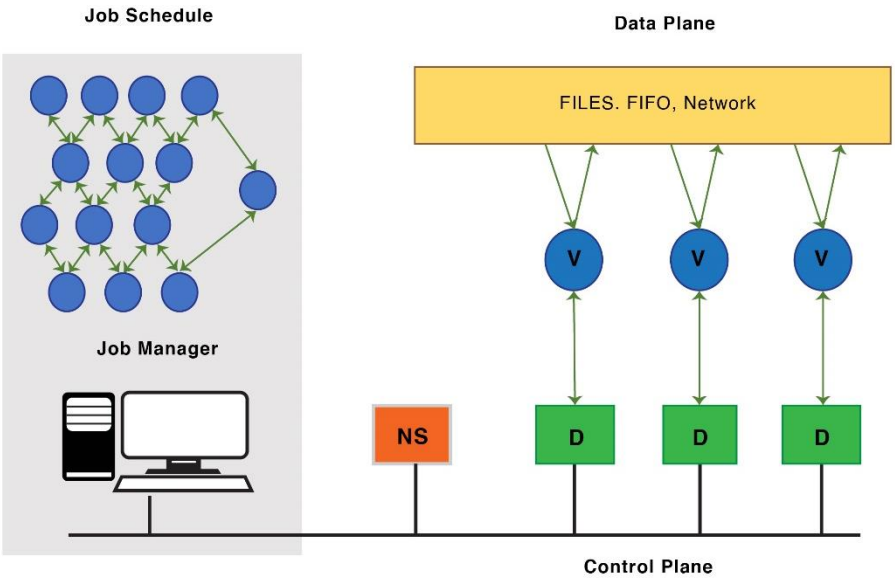

Fig. 2: System organization in dryad framework

The outstanding operations and results of Dryad are proved on numerous nontrivial, physical samples ranging from a single multi-core standalone computer to some clusters having thousands of machines, by demonstrating the use of optimum tools to improve parallelism and data distribution overhead adjustments. Scaling behavior of Dryad is demonstrated on trivial clusters, with complete performance and then to an enterprise database scheme for a hand-coded read-only query. On the other hand, network locality is exploited by enabling automatic execution of jobs having hundreds of thousands of vertices, on a larger cluster having the capability to process several TBs of input data in lesser time (Isard et al., 2007). In addition to this, Dryad supports developers to develop extensive distributed applications and do not demand them to use any concurrency procedures and data dependencies.

Comparison of Dryad with MapReduce system shows that this system is less complex in design, as it doesn't contain sequential steps of code like MapReduce, that is, a rigid sequence of the sort, map, and reduce steps.

3.3. Quincy

To handle the challenge of scheduling along with both fairness and locality constrictions, a new influential and reliable structure for scheduling of concurrent distributed jobs with fine-grained sharing of computing resources, named Quincy, is introduced in Isard et al. (2009). Quincy addresses the problem of concurrent scheduling of jobs in distributed compute and storage nodes and tries to settle the resource scheduling issues in several grid computing environments by extending the systems like MapReduce, Hadoop and Dryad where performance depends upon the availability of data near to their computations. Besides, Quincy is capable of mapping scheduling tasks to a data structure of graphs, where masses and edge dimensions encrypt the challenging loads of fairness, data locality, and starvation-freedom, and a regular solver calculates the optimal vacant schedule based on a defined global cost model. The Fig. 3 is demonstrating the process of cluster scheduling architecture.

Fig. 3 shows the cluster architecture of Quincy that is a queue-based scheduling model in which, once a worker task is forwarded to the scheduler, it is moved to the back-end of multiple queues, and when a resource is computed using scheduling algorithms, it is removed from the queue. When a new job is about to start, its root task is delivered to a random node among the nodes that are not performing root tasks at current time, and, any worker task that is presently in execution on that node, is tending to terminate and return into the scheduler queues as though it has just been forwarded. This approach applies the concepts of simple greedy fairness, fairness with preemption and flow based-scheduling with the queueing system, but has difficulties in handling sticky slots.

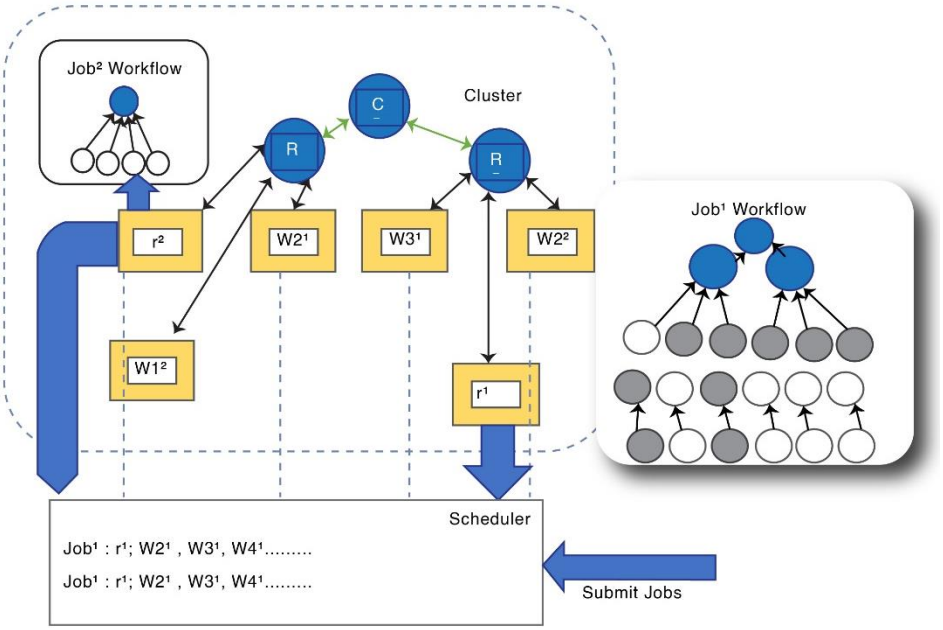

Fig. 3: The cluster scheduling architecture

The constraint regarding Quincy's existing design is: no explicit attempt is made to share the other network resources, and fairness is acquired totally based on a number of processers assigned to a task. Quincy's capability to decrease redundant network traffic enhances whole performance predictability and subsequently fairness, but on the other hand, when an application process requires inter-cluster communication, this constraint didn't demonstrate by this system. Quincy is flexible enough to limit the data transfers only in bottlenecks by taking account of network congestion, and monitor network traffic so that Quincy's cost would be adjusted. Also, one important property of Quincy is, it increases the throughput of clusters by a factor of three while reducing the traffic which is passing directly via the core switch of the tree-based network.

3.4. DryadLINQ

DryadLINQ is a scheme with an assemblage of language extensions which is introduced in Fetterly et al. (2009) which presents a novel programming paradigm for extensively distributed cluster computing. It is a comprehensive form of previous carrying out platforms, more specifically MapReduce, SQL, and Dryad in the following facets: By assuming an easy-to-read data representation of strongly coded .NET objects, and by assisting general-purpose essential and declarative activities on datasets inside a conventional best-in-class programming language. Fig. 4 is showing the LINQ-Expression implementation in DryadLINQ.in detail.

DryadLINQ scheme is like a sequence-based program, a collection of LINQ terminologies practicing secure and random renovations on datasets and can be generated and debugged by means of typical .NET development tools. Moreover, the DryadLINQ scheme can automatically and visibly interpret the data-parallel sections of an application program through a distributed execution strategy which is then delivered to the Dryad application execution environment.

From the implementation and evaluation point of view, DryadLINQ is successfully tested on a wide-ranging set of applications retrieved from areas like large-scale log mining, web-graph analysis, and machine learning. Steps for run-to-completion of a program in DryadLINQ is displayed in Fig. 4. Brilliant absolute performance of DryadLINQ is achieved on a general-purpose class of 1012B of data which is processed in a total time of 319sec on a 240-computer with 960-disk cluster. A drawback of DryadLINQ is its irreverent behavior towards many distributed applications, due to this, lack of simplification is there in policy choices in both Dryad and LINQ.

3.5. Delay scheduling

Zaharia et al. (2010a) highlighted the need to sharing clusters among users in fair scheduling and data locality aspects. They reported this problem through an experiment of the development of a fair scheduler for a Hadoop cluster of 600 nodes at Facebook. To represent the competition between fairness and locality, a naive algorithm named delay scheduling has been offered in Zaharia et al. (2010a), which is, when the job that is waiting to be scheduled in next turn according to fairness is not able to initiate a local task, it would wait for a short amount of time, leasing other jobs to initiate their remaining tasks.

Now, the pseudocode for delay scheduling algorithm (Zaharia et al., 2010a) for the scenario where we permit to skip a certain job for T times is given below:

Algorithm 1 Fair Sharing with Simple Delay Scheduling
Initialize i.skipcount to 0 for all jobs j.
When a heartbeat is received from node n:

if *n has a free slot* ***then***
Sort jobs in increasing order of number of running tasks
for *j in jobs* ***do***
if *j has unlaunched task t with data on n* ***then***
Launch t on n
Set j.skipcount=0
else if *j has unlaunched task t* ***then***
if *j.skipcount≥T* ***then***
Launch t on n
else
Set j.skipcount=j.skipcount + 1
end if
end if
end for

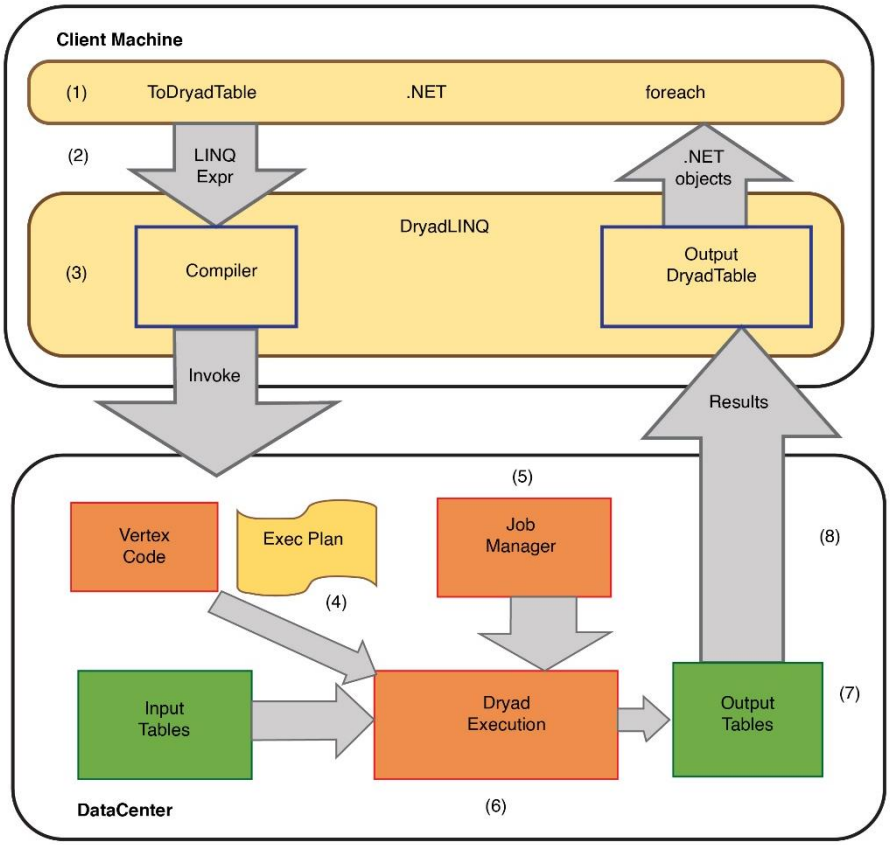

Fig. 4: LINQ-expression implementation in DryadLINQ

It is found that scheduled delay can accomplish approximately ideal data locality even in diverse workloads and can improve throughput by a factor of two while maintaining fairness. Besides this, the straightforwardness of delay scheduling makes it appropriate even under extensive variations in scheduling policies away from fair sharing. To implement fair sharing, two compromises between consumption and fairness are there to be considered: (i) either to halt running tasks or allow them to complete successfully and terminate when new tasks are ready to execute, (ii) how data locality can be attained in separate ways. In view of these two concerns, delay scheduling can achieve both locality and fairness by providing space for tasks to complete.

Experiments indicate that two crucial characteristics of the cluster computing environment including the brief time of tasks as compared to jobs and running of tasks on multiple locations support the delay scheduling to show remarkable performance. Delay scheduling gives its best performance in situations where above mentioned two properties hold, for instance, in Hadoop environments at Yahoo! and Facebook that supports multiple job executions in each node. Delay scheduling have limitations of nodes having larger job fractions with few slots available and sticky slots problem like Quincy (Isard et al., 2009) systems.

3.6. IX OS

A data plane operating system named as IX is presented in Belay et al. (2014), which can give acceptable I/O performance while maintaining the primary benefit of strong protection provided by prevailing kernels. Strong protection of kernels can be preserved by a data plane operating system IX for high I/O throughput and performance. Fig. 5 demonstrates the architectural components of IX. IX is aimed to split the scheduling and administrative responsibilities of the kernel from network function in a data plane by using hardware virtualization. The data plane structural design has been completely built upon a zero-copy inherited API which provides the best performance for delays and bandwidth together to data plane traffic by offering hardware threads and buffers in network elements. Also, IX restricts the groups of data traffic to completely process by reducing coherent traffic and multi-core management. IX gives better performance in data plane by reducing end-to-end-latency and improving

throughput (Comminiello et al., 2016). Fig. 5 is showing the detail of control and data plan-separation and protection and Fig. 6 is demonstrating inclosing of protocol processing and application execution process.

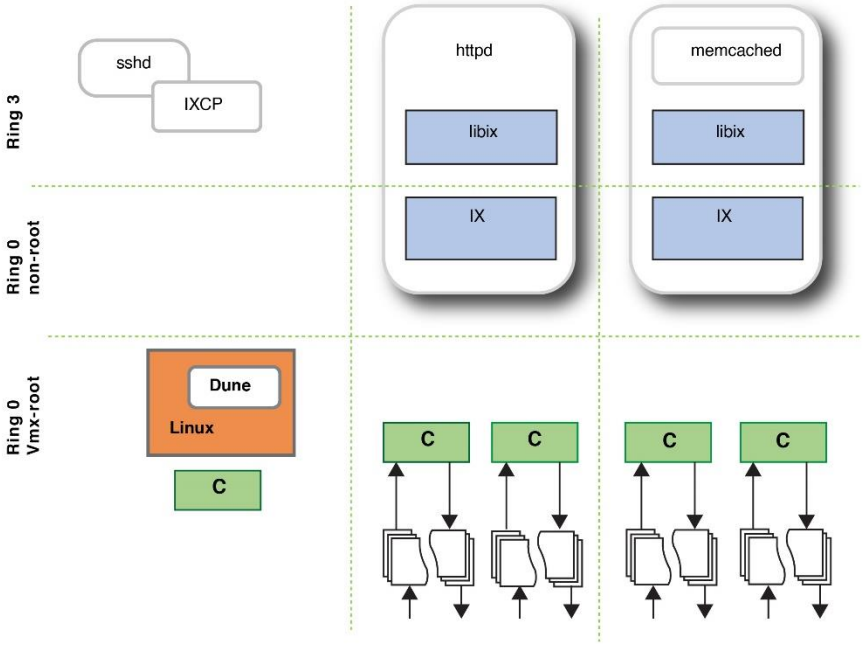

Fig. 5: Control and data plane-separation and protection

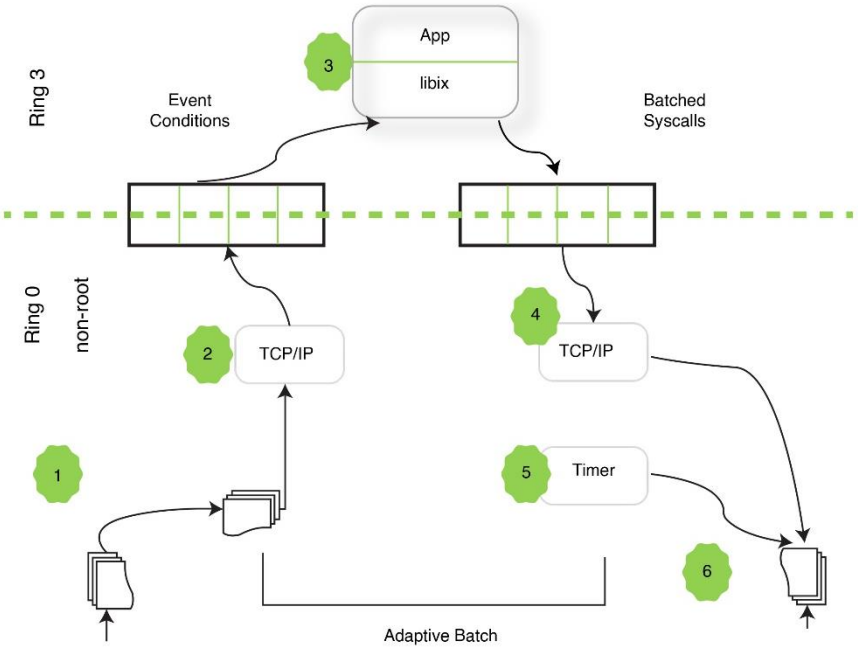

Fig. 6: Inclosing of protocol processing and application execution

IX enhances the data plane framework to service untrusted, general-purpose applications and satisfy all requirements, which includes, portioning and defense of control and data plane, completion and termination of with adaptive batching execution, native, zero-copy API along with flow control, flow consistent, synchronization-free processing, and finally, implemented resource allocation strategies. The flow of execution of a thread in the data plane of IX is represented in Fig. 6. The progress of well-organized allocation strategies involves consideration of problematic compromises between data plane energy consumption, resource sharing and utilization between interrelated applications and their returns.

The set-up of the IX implementation model is proficient enough for the management of events in a fast and non-blocking style. In its execution, operations with extended execution delays are likely to be passed on to contextual threads as compared to execute within the background of elastic threads.

3.7. Plan9 IX OS

Distributed Cloud/IX operating system is a derivative of plan9 which was developed for management and organization of the ARM-based

server environments. The main characteristics of this operating system involve naming and availability of resources are managed by hierarchical file systems, there is a standard protocol for retrieving remote and local resources and multiple services are interconnected together in boundless hierarchical file namespaces (Leokhin and Panfilov, 2015).

The fundamental values of plan 9 OS are defined to execute the disseminated processing schemes for distributed and parallel programming and computational prototypes using supercomputers, multi-server environments, and distributed embedded systems. The key advantage of this operating system is its simplicity for inter-node communication. This is done by deploying file namespaces for the creation and termination of applications irrespective of their basic system hardware. By this, they would execute at anyplace on any computer in the system, on any framework. A standard protocol called 9P protocol has been built in plan 9 that is responsible for assigning available resources to tasks. Plan 9 enables easy program development by refining the modularity of information by representing data sets in plain files.

3.8. MapReduce

MapReduce (Sakr et al., 2013) is a programming paradigm and a supplementary application that is very responsive to a vast variety of real-world functions for creating, managing and processing large datasets. A Map and a Reduce function in which MapReduce jobs are carried out in key/value pair input and output functions are used for indication of user requests and required computations and then an automated underlying operative system distributes the large extensive computations to the number of compute nodes in the cluster. This system is also responsible for dealing with node failures and their recovery, intra-cluster communications during the execution of tasks and effectual use of computing and storage resources (Dean and Ghemawat, 2008). An overview of MapReduce's operational flow is here in Fig. 7. MapReduce fully provides the data locality, fault tolerance, backups, fine-grained sharing of resources, and efficient bandwidth utilization.

Advantages of MapReduce include its simple and easy usability, flexibility, independence of storage, fault tolerance, high scalability, simple language, flexible data flow, and I/O optimization. However, ensuring increased efficiency, scalability, energy consumption and fault tolerance is a major challenge (Lee et al., 2012). The Fig. 7 is showing MapReduce Execution Overview.

3.9. Spark

Spark framework enhances the functionality of MapReduce variants while preserving the fault tolerance and scalability for data-intensive applications by discouraging iterative data flow model on commodity clusters (Zaharia et al., 2010b). Fig. 8 illustrates the spark architecture. Several machine learning algorithms have been used in collaborative data exploration tools for processing of working datasets over multiple parallel distributed tasks. Spark presented a novel abstraction named Resilient Distributed Datasets (RDDs) which consists of distributed and restorable read-only sets of objects for processing of iterative working datasets even in node failures. The use of RDDs enhances the performance by a factor of 10 with a 39 GB dataset response time in interactive machine learning jobs as contrasted with Hadoop (Zaharia et al., 2010b; 2012).

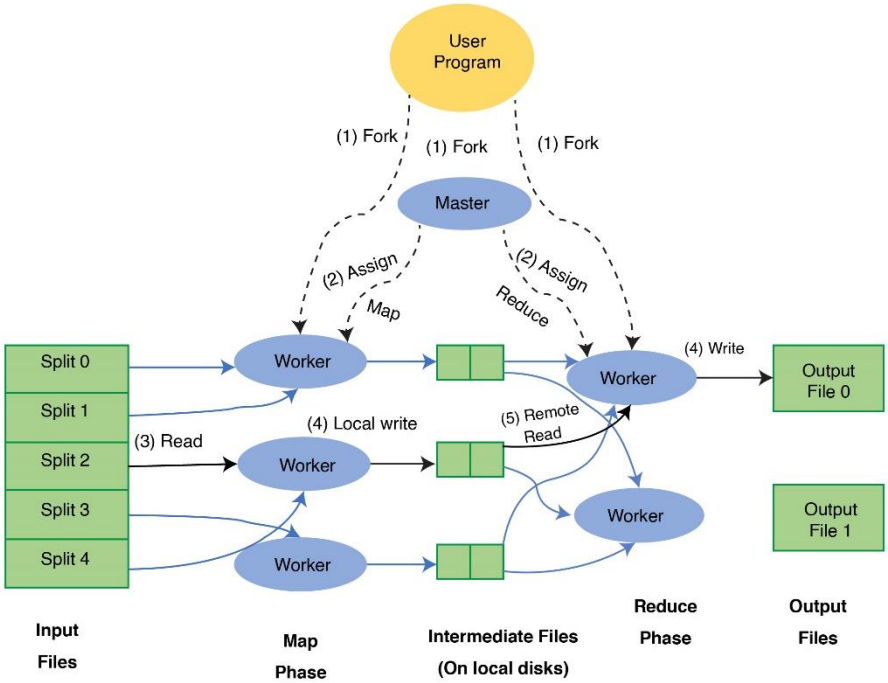

Fig. 7: MapReduce execution overview

RDDs in spark can be constructed by using file from a distributed file system like Hadoop Distributed File System (HDFS), partitioning and parallel scheduling of a driver program to multiple nodes, transforming of a present RDD having a specific type of datasets into another RDD with different datasets, and altering the persistence of current RDDs by cache or save actions. Spark uses shared variables such as broadcast variables and accumulators in closures to implement tasks of the map, reduce and filter. Broadcast variables make certain the distribution of large read-only sections of data just the once in multiple workers' operations without pushing it with all closures, while accumulators help in making the system fault-tolerant due to their add-only property. Accumulators employ associative operations to provide parallel sums with a type of data with zero value and an add operation. Fig. 8 is illustrating the detail of Spark architecture.

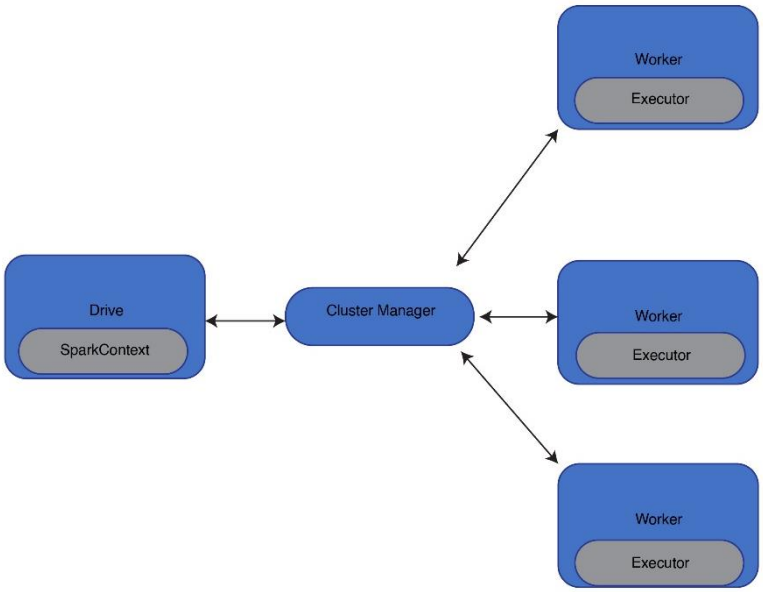

Fig. 8: Spark architecture (Mullender et al., 1990)

3.10. Amoeba

A distributed operating system Amoeba is intended for high-performance communications between clients and servers side nodes using the well-known RPC model in Tanenbaum et al. (1990). The architecture of the Amoeba system is presented in Fig. 9, contains hardware components like workstations, several servers, a processor pool, and gateways to transparently connect Amoeba systems over wide-area networks. Amoeba file server is so fast and its boundaries are only associated with communication bandwidth.

In an operational point of view, Amoeba exploits the remote procedure calls to operate on objects with RPC protocols. One remarkable advantage of this system is its robust security. To prove the performance aspects of Amoeba, it has been compared with SUN RPC. It is found that Amoeba runs a small RPC 9 times faster, and realized over 3 times the bandwidth for large RPCs. Amoeba also gives the best services in high load scenarios, by supporting its fair share of the vacant bandwidth resources. Fig. 9 is showing the detail of four components of Amoeba architecture.

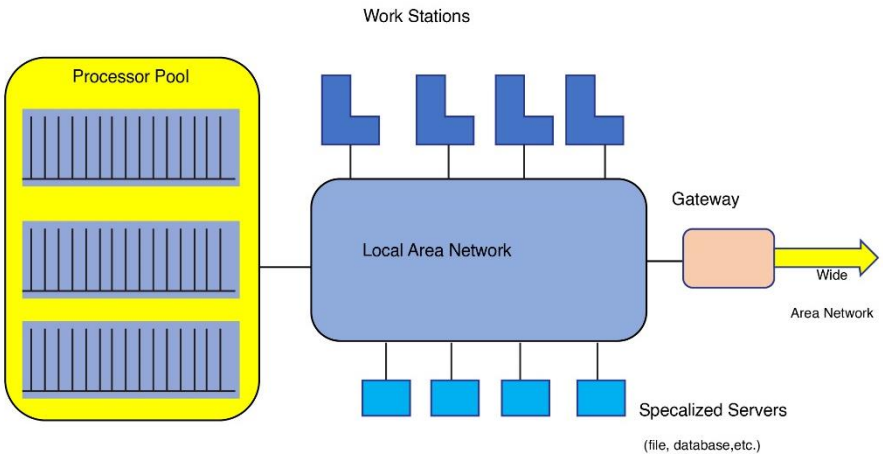

Fig. 9: Four components of amoeba architecture

The bullet file server is the file service of Amoeba. Bullet ensures the high availability of resources through duplication. Also, the Amoeba file server is four times faster for writing large files and two times faster reading large files than SUN RPC (Van Renesse et al., 1989). Limitation of this system is its delay in some read operations.

4. Comparison of various data center framework based on benchmark parameters

This work has presented a comprehensive comparison and analysis of various state of art operating systems for data centers and cloud setups. The comparison and analysis were carried out using benchmark parameters which are based on resource sharing, data sharing, data/network locality, programming abstraction, debugging and monitoring, fault tolerance and file system. Table 1 illustrates the comparison between distinctive frameworks proposed by renowned researchers based on the mentioned benchmarks parameters. The discussed frameworks can play a significant role in cloud and data center environments. Previous operating systems developed for distributed systems are not suitable for cloud and data center environments. The analysis shows that resource sharing is very critical for part of every framework. The support for other parameters is less likely. Most of the frameworks are lacking fault tolerance and file systems. Most of the cloud computing environments have high complexity compared to others. Moreover, the proper employment of these frameworks can eliminate the need for the stacking of different software in small or large environments. The next section highlights various future work directions to overcome mentioned and related limitations.

5. Key findings and future discussion

In an effort of designing a proper secure operating system for distributed environments in data centers, different approaches have been made. We discussed some of those platforms and their precise comparison has been made in Table 1. Findings of Table 1 show that some platforms give some of the characteristics of an operating system but lack in providing all the services that an operating system gives in a machine. Mesos is a notable effort in this sense because it is originally intended for resource sharing in data centers. Others features like abstraction, monitoring and debugging, file systems and memory management, which are the typical characteristics of an operating system are there, but to some extent.

Table 1: Comparison between distinctive characteristics of above-mentioned frameworks based on different benchmarks parameters

Ref.	Frameworks	Resource Sharing	Data Sharing	Data/Network Locality	Programming Abstraction	Debugging and Monitoring	Fault-tolerance	File System
1.	Mesos (Hindman et al., 2011)	✓	✓	✓	✓	✓	✓	-
2.	Dryad (Isard et al., 2007; Sakr et al., 2013)	✓	✓	✓	-	✓	✓	✓
3.	Quincy (Isard et al., 2009)	✓	✓	✓	✓	-	-	-
4.	DryadLINQ (Fetterly et al., 2009)	✓	-	-	✓	✓	-	-
5.	Delay Scheduling (Zaharia et al., 2010a)	✓	-	✓	✓	-	-	-
6.	IX Operating System (Belay et al., 2014; Comminiello et al., 2016)	✓	-	-	✓	✓	-	✓
7.	Plan9 IX OS (Leokhin and Panfilov, 2015)	✓	✓	-	✓	✓	-	✓
8.	MapReduce (Sakr et al., 2013; Dean and Ghemawat, 2008; Lee et al., 2012)	✓	✓	-	✓	✓	✓	✓
9.	Spark (Zaharia et al. 2010b; 2012; Mullender et al., 1990)	✓	✓	-	-	-	✓	✓
10.	Amoeba (Tanenbaum et al., 1990; Van Renesse et al., 1989)	-	-	-	✓	✓	-	✓

Dryad is a less complex better approach in parallel programming having similarity with Hadoop and MapReduce. It performs well in distributed systems but does not work in data centers where a centralized controller is responsible for the management of the whole network. Similarly, DryadLINQ, Quincy and Delay schedules are intended to focus on resource sharing and programming abstractions. IX operating system and plan9 based IX OS also contributes to giving some services of operating system like file servers, resource sharing and data abstractions but have an absence of operations in multi-cluster to a data center like environments. Spark preserves the output of tasks in fault-prone systems and efficiently share data in MapReduce (Sakr et al., 2013; Dean and Ghemawat, 2008), Hadoop (Shvachko et al., 2010), Pregel (Malewicz et al., 2010) and SQL

environments. Amoeba gives a file system for distributed environments but its operational delays are its limitations. Even though these frameworks are managing huge workloads but cloud computing services still need operating system-like software stack for set-ups.

6. Conclusion

This work presented a comprehensive comparison and analysis of various state of art frameworks for cloud and data centers. This work has an impactful contribution to cloud computing and the data center domain. As the employment of data centers grows to keep pace with future application and user demands, need for an operating system-like software stack increases to facilitate the computing environments with traditional kernel operating system services like resource and data sharing, abstractions, file systems, access to debugging tools, fault tolerance and service elasticity in distributed computations. Some considerable efforts have been done and different frameworks have been developed at the ad-hoc level, but it becomes a challenge in single controller settings. Although these platforms are dealing with increased diversity of clusters and workloads interoperable applications development, storage systems, data processing frameworks and services still need operating system-like software stack for inter-data center and intra-data center operations. In the future, this work can be extended to deal with various other functionalities and the detailed working of these frameworks.

Compliance with ethical standards

Conflict of interest

The authors declare that they have no conflict of interest.

References

Aviram A, Weng SC, Hu S, and Ford B (2012). Efficient system-enforced deterministic parallelism. Communications of the ACM, 55(5): 111-119. https://doi.org/10.1145/2160718.2160742

Bahl P, Han RY, Li LE, and Satyanarayanan M (2012). Advancing the state of mobile cloud computing. In the 3rd ACM Workshop on Mobile Cloud Computing and Services, ACM, Low Wood Bay, Lake District, UK: 21-28. https://doi.org/10.1145/2307849.2307856

Barroso LA, Clidaras J, and Hölzle U (2013). The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synthesis Lectures on Computer Architecture, 8(3): 1-154. https://doi.org/10.2200/S00516ED2V01Y201306CAC024

Belay A, Prekas G, Klimovic A, Grossman S, Kozyrakis C, and Bugnion E (2014). {IX}: A protected dataplane operating system for high throughput and low latency. In the 11th {USENIX} Symposium on Operating Systems Design and Implementation, USENIX Association, Broomfield, USA: 49-65.

Burrows M (2006). The Chubby lock service for loosely-coupled distributed systems. In the 7th Symposium on Operating Systems Design and Implementation, USENIX Association, Seattle, USA: 335-350.

Comminiello D, Michele S, Simone S, Raffaele P, and Aurelio U (2016). Smart innovation, systems and technologies. Springer Science and Business Media, Berlin, Germany.

Dean J and Ghemawat S (2008). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1): 107-113. https://doi.org/10.1145/1327452.1327492

Fetterly YYMID, Budiu M, Erlingsson Ú, and Currey PKGJ (2009). DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In the 8th USENIX Symposium on Operating Systems Design and Implementation, USENIX Association.

Fonseca R, Porter G, Katz RH, Shenker S, and Stoica I (2007). X-trace: A pervasive network tracing framework. In the 4th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, Cambridge, USA: 20-20.

Greenberg A, Hamilton J, Maltz DA, and Patel P (2008). The cost of a cloud: research problems in data center networks. ACM SIGCOMM Computer Communication Review, 39(1): 68-73. https://doi.org/10.1145/1496091.1496103

Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz RH, and Stoica I (2011). Mesos: A platform for fine-grained resource sharing in the data center. In the 8th USENIX conference on Networked systems design and implementation: 295-308.

Isard M, Budiu M, Yu Y, Birrell A, and Fetterly D (2007). Dryad: Distributed data-parallel programs from sequential building blocks. In the ACM SIGOPS Operating Systems Review, ACM, Lisbon, Portugal, 41(3): 59-72. https://doi.org/10.1145/1272998.1273005

Isard M, Prabhakaran V, Currey J, Wieder U, Talwar K, and Goldberg A (2009). Quincy: Fair scheduling for distributed computing clusters. In the ACM SIGOPS 22nd Symposium on Operating Systems Principles, ACM, Big Sky, USA: 261-276. https://doi.org/10.1145/1629575.1629601

Jadeja Y and Modi K (2012). Cloud computing-concepts, architecture and challenges. In the International Conference on Computing, Electronics and Electrical Technologies, IEEE, Kumaracoil, India: 877-880. https://doi.org/10.1109/ICCEET.2012.6203873

Lee KH, Lee YJ, Choi H, Chung YD, and Moon B (2012). Parallel data processing with MapReduce: A survey. ACM SIGMOD Record, 40(4): 11-20. https://doi.org/10.1145/2094114.2094118

Leokhin Y and Panfilov P (2015). A study of cloud/IX operating system for the ARM-based data center server platform. Procedia Engineering, 100: 1696-1705. https://doi.org/10.1016/j.proeng.2015.01.545

Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, and Czajkowski G (2010). Pregel: A system for large-scale graph processing. In the 2010 ACM SIGMOD International Conference on Management of Data, ACM, Indianapolis, USA: 135-146. https://doi.org/10.1145/1807167.1807184

Mullender SJ, Van Rossum G, Tanenbaum AS, Van Renesse R, and Van Staveren H (1990). Amoeba: A distributed operating system for the 1990s. Computer, 23(5): 44-53. https://doi.org/10.1109/2.53354

Patterson DA (2008). The data center is the computer. Communications of the ACM, 51(1): 105-105. https://doi.org/10.1145/1327452.1327491

Sakr S, Liu A, and Fayoumi AG (2013). The family of mapreduce and large-scale data processing systems. ACM Computing Surveys, 46: 1. https://doi.org/10.1145/2522968.2522979

Shvachko K, Kuang H, Radia S, and Chansler R (2010). The hadoop distributed file system. In the Conference of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), IEEE, Washington, USA: 1-10. https://doi.org/10.1109/MSST.2010.5496972

Tanenbaum AS, Van Renesse R, Van Staveren H, Sharp GJ, and Mullender SJ (1990). Experiences with the amoeba distributed operating system. Communications of the ACM, 33(12): 46-63. https://doi.org/10.1145/96267.96281

Van Renesse R, Van Staveren H, and Tanenbaum AS (1989). The performance of the Amoeba distributed operating system. Software: Practice and Experience, 19(3): 223-234. https://doi.org/10.1002/spe.4380190303

Wang L, Tao J, Ranjan R, Marten H, Streit A, Chen J, and Chen D (2013). G-Hadoop: MapReduce across distributed data centers for data-intensive computing. Future Generation Computer Systems, 29(3): 739-750. https://doi.org/10.1016/j.future.2012.09.001

Wang M, Li B, Zhao Y, and Pu G (2014). Formalizing google file system. In the IEEE 20th Pacific Rim International Symposium on Dependable Computing, IEEE, Singapore, Singapore: 190-191. https://doi.org/10.1109/PRDC.2014.32

Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, and Stoica I (2010a). Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In the 5th European Conference on Computer Systems, ACM, Paris, France: 265-278. https://doi.org/10.1145/1755913.1755940

Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, and Stoica I (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In the 9th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, San Jose, USA.

Zaharia M, Chowdhury M, Franklin MJ, Shenker S, and Stoica I (2010b). Spark: Cluster computing with working sets. HotCloud, 10(10-10): 95. Available online at: https://bit.ly/2zSQieG

Zaharia M, Hindman B, Konwinski A, Ghodsi A, Joseph AD, Katz RH, and Stoica I (2011). The datacenter needs an operating system. In HotCloud. Available online at: https://bit.ly/3eD0UNz

Zhang Q, Cheng L, and Boutaba R (2010). Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1): 7-18. https://doi.org/10.1007/s13174-010-0007-6