Dheyab *et al. J Big Data* (2019) 6:112
https://doi.org/10.1186/s40537-019-0275-3

Journal of Big Data

METHODOLOGY Open Access

A novel approach for big data processing using message passing interface based on memory mapping

Saad Ahmed Dheyab[1,2*], Mohammed Najm Abdullah[3] and Buthainah Fahran Abed[4]

*Correspondence:
saad.theyab@uoitc.edu.iq
[1] College of Engineering, University of Information Technology and Communications, Baghdad, Iraq
Full list of author information is available at the end of the article

Abstract

The analysis and processing of big data are one of the most important challenges that researchers are working on to find the best approaches to handle it with high performance, low cost and high accuracy. In this paper, a novel approach for big data processing and management was proposed that differed from the existing ones; the proposed method employs not only the memory space to reads and handle big data, it also uses space of memory-mapped extended from memory storage. From a methodological viewpoint, the novelty of this paper is the segmentation stage of big data using memory mapping and broadcasting all segments to a number of processors using a parallel message passing interface. From an application viewpoint, the paper presents a high-performance approach based on a homogenous network which works parallelly to encrypt-decrypt big data using AES algorithm. This approach can be done on Windows Operating System using .NET libraries.

Keywords: Big data, Message passing, Memory mapping, High-performance computing, Parallel AES

Introduction

The field of big data analytics is fast becoming attractive to several companies such as banks, telecommunication and web-based companies (like Google, Twitter, and Facebook) who generate large volumes of data. Business users have nowadays explored how such data can be monetized. For instance, customer behavior data can be transformed into great monetary worth using several predictive marketing techniques. However, the implementation of platforms that are fast enough to load, store and execute ad-hoc analytical queries over big data remains a major challenge [1].

The definition of big data is commonly based on the '3Vs: Volume (huge amounts of data), Variety (different forms and emerging structures), and Velocity (fast generation, capturing, and consumption). Being that Hadoop can address the issues relating to volume and velocity in a virtually unlimited horizontal scaling, it has been considered a universal solution; however, the issue of variety can be addressed by storing data in schema-less databases [2]. Regarding performance, basic Hadoop technologies like HDFS, Pig, Hive, and MapReduce are scalable and cost-efficient; however, they are

associated with certain challenges, especially relating to their ability to support fast ad-hoc analysis and to process difficult queries [3].

The aim of this study is to use a novel approach which depends on memory management and big data architectures to tackle big data. The three stages of the proposed approach are depicted in Fig. 1. The proposed approach was validated by building a prototype and conducting a study on big data encryption and decryption using AES algorithm and parallel message passing interface (MPI).

This paper is organized in sections, "Related works" section reviewed previous works on big data analysis and their correlation with different platforms, while "Theoretical basis" section highlighted the theoretical basis of memory analysis and parallel processing on big data. In "The proposed approach MPIM" section, the proposed method was overviewed while the last section presented the experimental methods and the results of the experiments.

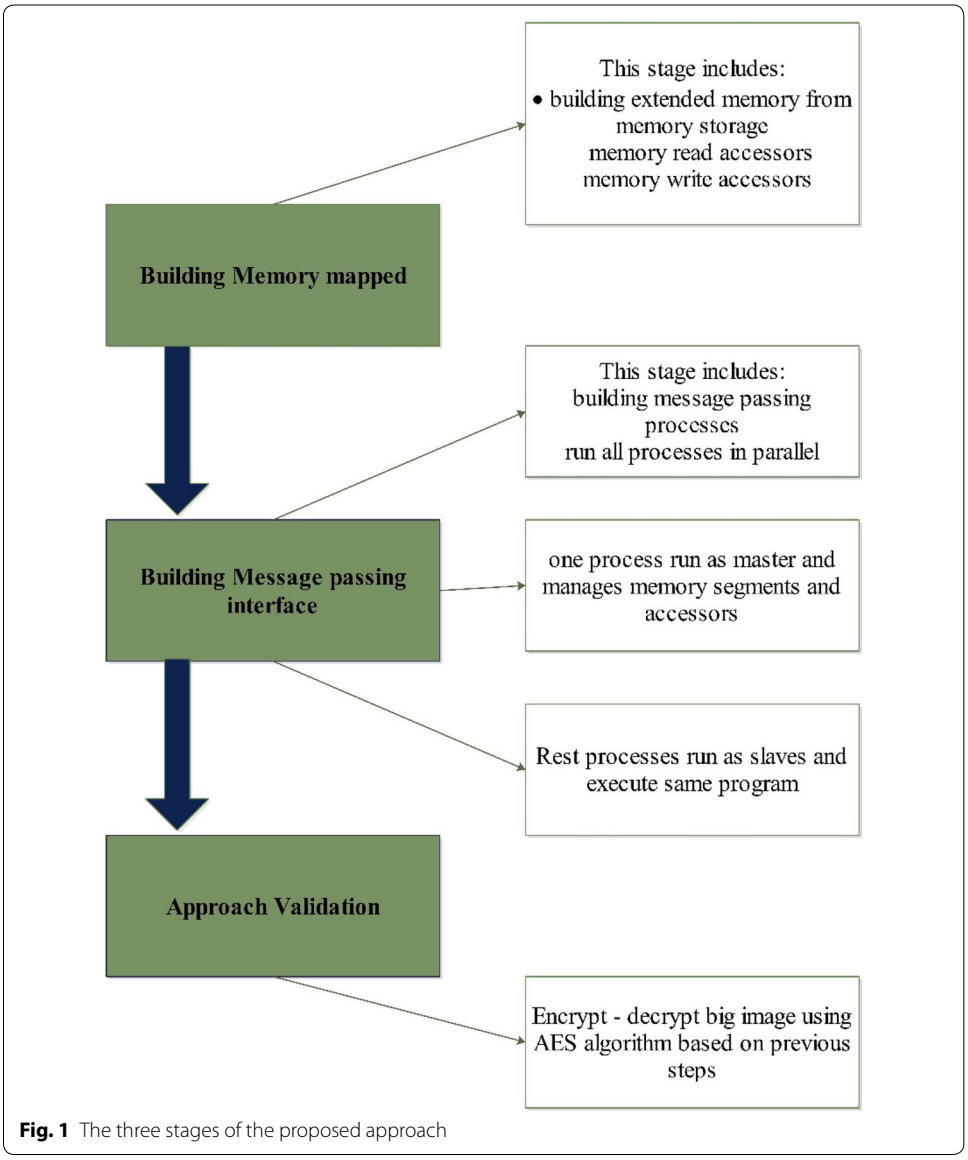

Fig. 1 The three stages of the proposed approach

Related works

Big data analysis has received several research interests, especially in relation to big events that deploy different techniques. This section provides a brief review of some of the previous works related to this study. A study on big data broadcasting issues from an algorithmic perspective has been reported by [4]. This study formalized big data broadcasting problem into the Lockstep tree (LSBT) model and equally considered designing such a single overlay tree and the models' maximum completion time. This is the first attempt at investigating the relationship between a single overlay tree with a fixed uplink rate and the maximum completion time in complex networks.

Furthermore, studies have been reported on big data quality management and metadata management layer for big data systems based on company needs [5]. The major focus of this study was on data quality management. To ensure the creation of higher quality insights on the company's product, the study filtered the company's tweets based on their quality.

Some schemes which mainly focus on the designing of some algorithms that can ensure the confidentiality of data have been proposed. However, such algorithms are often expensive and not efficiently applicable to big data [6].

A study on big data management as a traditional database management system, appliance, Hadoop, and Solid-State Disks (SSD) has been reported by [7]. The study reported that Hadoop offered the best service in terms of scalability, cost, and unstructured data.

A study by [8] presented three use cases for big data management and data integration which could be enhanced by employing semantic technologies and ontologies. The presented use cases are the integration of externally and internally sourced data, measurement of data quality, and improvement of business process integration.

A discussion on MPI design choices and a comprehensive optimization of the data pipelining and buffer management has been provided by Aji et al. [9]. The study investigated the efficiency of MPI-ACC application in the scientific field, mainly in the field of epidemiology and outlaid the lessons learned and the tradeoffs. The paper suggested that the optimization of the MPI-ACC's internal pipeline did not just improve the communication performance but equally provided avenues for the optimization of the application process, leading to a significant enhancement of the CPU-GPU and network use.

A globally shared memory abstraction to distributed-memory systems can be achieved using Partitioned Global Address Space (PGAS) models such as Global Arrays, Chapel, and OpenSHMEM. The extension of the PGAS models to incorporate GPUs as a part of the shared-memory abstraction has been explored in different studies [10, 11]. The PGAS models can utilize MPI on its own as an underlying runtime layer [12, 13] and can be considered complementary to MPI.

A new HPC-based data storage abstraction that combines memory and storage has been presented by [14]. This work extended the one-sided communication model concept by equipping MPI windows with parallel I/O functions such that programmers can maintain a unified familiar programming interface.

A new way of using MPI windows for concealing the heterogeneity of storage and memory sub-systems has been presented by [15]. This study provided a common programming platform for data movement across these layers. The proposed MPI storage windows were based on the OS memory-mapped file I/O mechanism to provide a

transparent and efficient integration of the storage support into both new and existing platforms without necessarily changing into the MPI standard. The system also allowed the definition of combined window allocations which merged storage and memory under a common virtual address space.

A set of ISA extensions and their related hardware support have been described by [16]. These extensions were shown to enhance the effectiveness of message passing executed on a shared memory multi-core. This approach merged software and hardware support to facilitate effective communication between processes executed in different spaces. The process of incorporating the extensions to 2 real-world MPI libraries to reduce energy utilization and improve communication performance transparently was also presented in the study. The major focus of the study was on MPI implementation.

The performance of an unstructured mesh Navier–Stokes solver was studied by Mavriplis [17] using MPI and OpenMP. The study also studied the same scheme for message transmission between threads using MPI. The outcome of the study reported a significant increase in speed when a two-level message passing scheme was used compared to the use of 140 processes on an SGI-Origin 2000.

A study by Cramer et al. [18] focused on the use of OpenMP on the Intel Xeon Phi chip. The outcome of the study showed the possibility of easily porting CPU-targeted code to the chip with decent speedup. Furthermore, a C++system was created by Dokulil et al. [19] for array-based operations. This system runs in hybrid on the host and Phi co-processor and depends on a parallel-for paradigm for work distribution.

The pseudo-polynomial dynamic programming algorithm which provides the optimal solution to the parallel identical processor scheduling problem in a bid to reduce the maximum time of job completion has been constructed by [20]. In this scheme, a random monotonic function that depends on the number of previously processed jobs was used to describe the processing time for each job.

Theoretical basis

Big data definition

Big data, unlike conventional data, refers to the huge volume of data sets that present in both structured, unstructured and semi-structured formats. Owing to the complex nature of big data, they require strong technologies and advanced frameworks to be analyzed. Thus, conventional static Business Intelligence tools may not be suitable for efficient big data applications anymore. Big data has been defined by most experts using the three main characteristics of big data as follows [21]:

1. Volume: Numerous devices and applications generate huge volumes of digital data continuously. McAfee et al. [22] reported an estimated 2.5 exabytes of data to be generated daily in the year 2012. This data volume doubles approximately every 40 months. The International Data Corporation estimated the total volume of digital data generated, replicated, and consumed in 2013 to be 4.4 Zettabytes (ZB) and this volume was expected to double every 2 years. The reported digital data volume by 2015 grew to 8 ZB. The IDC report forecasted that by the year 2020, the volume of generated digital data will reach 40 Zeta bytes [23].

2. Velocity: Being that data are rapidly generated, they should be rapidly processed to extract vital information. For example, Walmart is reported to generate > 2.5 PB of data per hour from its customers-related activities. Another company that generates huge data volume in a rapid manner is YouTube.
3. Variety: Big Data are generated in different formats from different sources. Such datasets are composed of both structured and unstructured, public or private, local or distant, complete or incomplete formats.

Some factors have added more Vs to provide a better definition of big data [24, 25]: such additions include: Vision (a purpose), Validation (a fulfilled purpose), Verification (data conformation to certain specifications), Value (possible to extract vital information for many sectors), Immutability (stored big data can be permanent if managed properly), and Complexity (cumbersome to analyze Big data due to the emerging data relationships).

Big data management

Data scientists are faced with many big data-related challenges and one of such challenges is related to the way to collect, integrate and store the huge volume of data generated from various sources using less hardware and software [26, 27]. Another problem is the issue of big data management. Reliable extraction of important information from a huge data volume requires efficient management of such dataset. Hence, the foundation of good big data analytics is a good data management. Big data management implies data cleaning for reliability, data aggregation from different sources, and data encoding for privacy and security. It also implies ensuring effectual big data storage and role-based access to multiple distributed endpoints. Therefore, big data management mainly aims at ensuring data reliability, easy accessibility, management, secured and properly stored [28].

Big data analysis framework

The concept of big data generated an ever-increasing pressure for scalable data processing solutions, leading to the implementation of several data processing and management systems recently. A summary of the major features of big data analysis frameworks is presented in Table 1 [29]. The increasing data analysis requirements in virtually all application areas have necessitated need to design and build a new set of big data science tools which can seamlessly analyze huge data volumes, extract vital information, and establish important patterns and knowledge from such datasets [30].

Message passing interface (MPI)

Message passing (MP) is not a new concept; it has existed for long and has been the commonest programming tool for HPC applications with several nodes. The actor-based libraries and programming languages like Erlang and Akka/Scala [31, 32] were introduced to improve the popularity of the MP model beyond the HPC realm. The HPC was developed as a potential tool for building parallel applications which will exploit the growing number of available cores in the existing chip multiprocessors. However, shared memory multi-cores have come to stay as the ideal option for chip producers irrespective

Table 1 Major feature of big data analysis frameworks

Framework	Abstraction	Supported languages	Underlying engine	Availability
Hadoop	MapReduce	Java, Python	Hadoop	Open source
Spark	RDD	Scala, Python, Java, R	Spark, Yarn	Open source
H2O	Procedural + Library	REST, R, Python	H2O	Open source
SciDB	Declarative	SQL	SciDB	Open source
AzureML	Visual User Interface	REST	Microsoft Azure	Microsoft
R	Procedural	R	R	Open source
SparkR	Procedural	R	Spark	Open source
Mahout	Toolkit	Java, Scala	Hadoop, Spark	Open source
Spark MLib	Library	Scala, Python	Spark	Open source
Samsara	Declarative	Java, Scala	Spark, Flink, H2O	Open source
SystemML	Declarative	R, Python	Spark	Open source
Google ML	Visual User Interface	Python	Google Cloud Dataflow	Google
Amazon ML	Visual User Interface	N/A	Apache MXNet, TensorFlow, PyTorch	Amazon
BigML	Visual User Interface	Python	BigML Predict Server	BigML
Tensorflow	Visual User Interface	Python, Haskell, Java, Go, Julia, R	CUDA, TPU	Open source
KeystoneML	Procedural + Library	Scala	Spark	Open source

of the increasing popularity of MP. This is attributable to their wide adaptability, ranging from data centers to HPC settings, mobile devices, and the vast accumulated industrial experience on this domain. Traditionally, MPI is used in HPC systems with several numbers of distributed nodes where message propagation is relatively costly. As such, programmers always strive to limit communication and focus on exchanging large messages in small numbers. The trade-offs in shared memory multi-cores vary, and MP can be better deployed when compatible with the algorithm. Contrarily, few messages are embedded alongside the message meta-data in the same cache line, thus, keeping both bandwidth and latency to a minimum. Kernel extensions like KNEM [33] and LiMIC [34] can handle large messages and provide a single-copy message that passes across varying address spaces. For decades now, the MPI model has been deployed in building scalable applications for systems with several nodes. Meanwhile, the efficiency of this programming model alone cannot be maximized in systems where every node acts as a shared-memory multi-processor and executes several processes. Such a heterogeneous scenario has compelled programmers to resort to hybrid programming models such as MPI + OpenMP or combination of both MP and shared memory models in one program to ensure efficient utilization of the underlying hardware. Regrettable, programming with such hybrid models is a difficult task [35]. MPI architecture is depicted in Fig. 2.

Memory mapping

The process of associating the content of a file with an aspect of the virtual address space of a process is termed memory mapping. To maintain this association, the system generates a file mapping object, also called section object. The aspect of the virtual address space used to access the files' content by a process is known as the file view. Through file mapping, a process can utilize both random input/output (I/O) and sequential I/O and can also work effortlessly with a huge data file without the need of mapping the

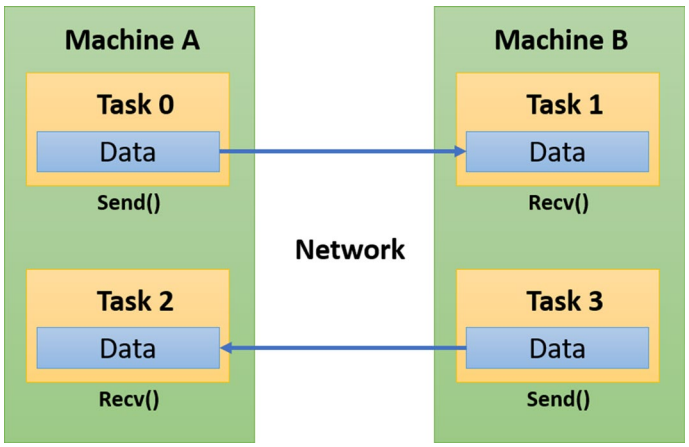

Fig. 2 MPI architecture

whole file first into memory. Multiple processes can also share data using memory-mapped files. Processes use pointers to read from and write to the file view as obtainable with dynamically allocated memory. Using file mapping enhances the efficiency due to the storage of the file on disk while the file view is in the memory. File view can also be manipulated by processes using the Virtual Protect function [36]. The relationship between a file mapping object, a file view, and a file on disk is depicted in Fig. 3. Despite the dramatic improvement in the performance of CPU performance, storage systems have not improved performance-wise compared to their capacity. Even though data can be quickly processed by the CPUs, there are still system problems due to the latency of the storage device. This wasted time is an important component of overhead for data-intensive workloads. The recent increase in the data-intensive nature of recent applications demands a substantial performance improvement. In-memory data processing has received great consideration due to its ability to process workloads more rapidly through the elimination of I/O overhead. However, such problems can be ultimately addressed using memory mapped file I/O [37].

Advance Encryption Standard (AES) algorithm

The exchange of digital data over a network has exposed the multimedia data to various kinds of abuse such as Brute-Force attacks, unauthorized access, and network hacking. Therefore, the system must be safeguarded with an efficient media-aware security framework such as encryption methods that make use of standard symmetric encryption algorithms, which will be responsible for ensuring the security of the multimedia data. For the encryption of electronic data, one of the most prominent cryptographic algorithms is the Advanced Encryption Standard algorithm [38] which was established by Joan Daemen and Vincent Rijmen [39] as a symmetric encryption block cipher algorithm. Its implementation is mainly in applications where fast processing is required, such as in cellular phones, image-video encryption, and smart cards. The AES algorithm functions on a 4×4 array of bytes (128 bits) block size known as a state. This state undergoes series of encryption and decryption processes, made up of 4 procedures iterated over a given number of rounds based on the key length of the AES algorithm in

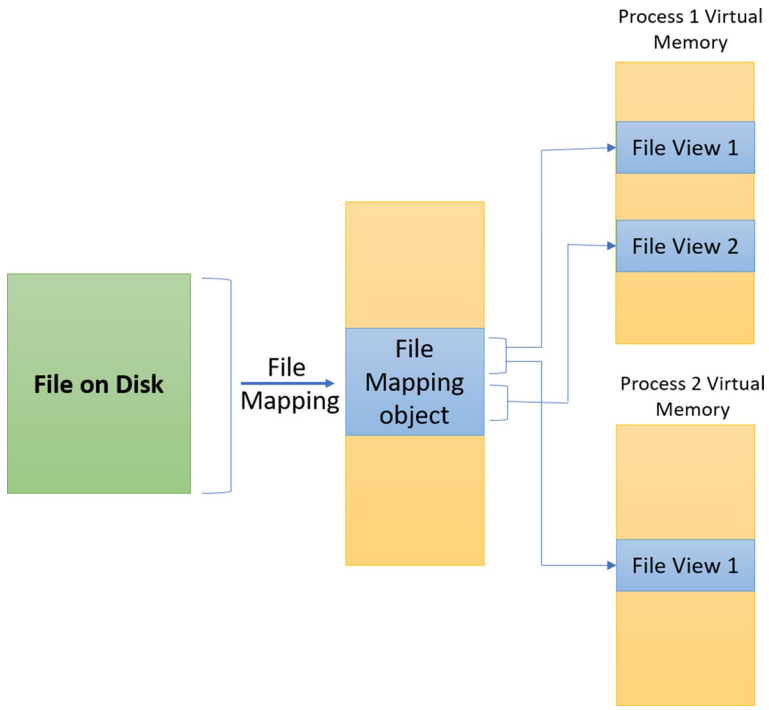

Fig. 3 The relationship between the file on disk, a file mapping object, and a file view

order to transform the data from an intelligible format to an unintelligible format. The number of the algorithmic rounds depends on the size of its key; 10 rounds for 128-bit keys, 14 rounds for 256-bit keys, and 12 rounds for 192-bit keys. The key length refers to the number of bits in the key used by a cryptographic framework. It expresses the highest level of security provided by a framework [40]. The encryption and decryption processes of the AES algorithm is depicted in Fig. 4. The encryption process is executed in 4 steps: SubByte, ShiftRow, Mixcolumn and AddRoundKey transformations on the state block array in addition to an initial round key. The round function repetition of 10, 12 or 14 rounds depends on the key length. The Mixcolumn step is not applied in the last round. The decryption procedure is carried out in the same four steps used in the encryption procedure [41].

The proposed approach MPIM

A novel approach has been proposed in this paper to process and manage big data. To discuss in detail, the approach was divided into three main stage-based methodologies, consisting of memory map construction, managing memory map using parallel Message Passing Interface, and data processing using an encryption algorithm. The paper suggests that this approach should be called MPIM, an abbreviation for Message Passing Interface and Memory Mapped.

First stage: constructing memory map

MPIM is based on parallel processing and management of big data, so, it was suggested to create the memory map because it allows access by more than one process and this is

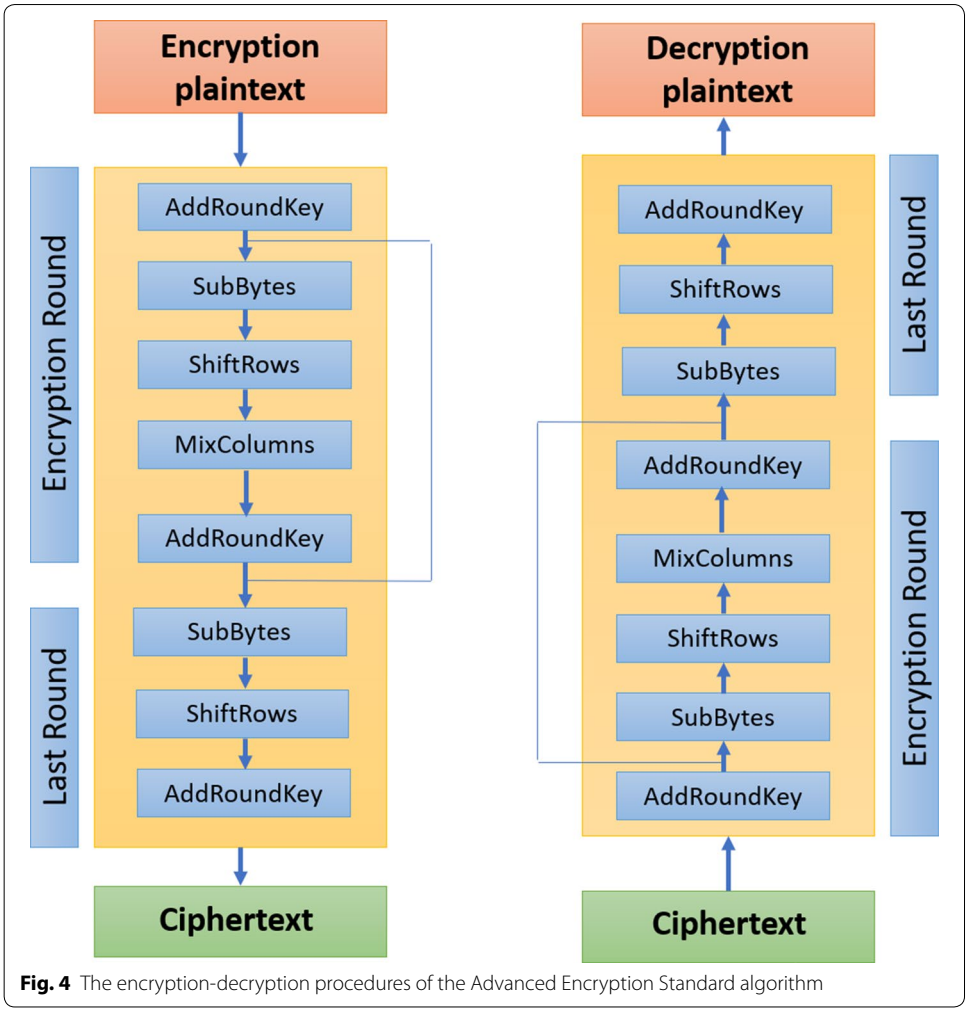

Fig. 4 The encryption-decryption procedures of the Advanced Encryption Standard algorithm

needed in the processing of big data. The creation of memory map is supported by some programming languages like c#; the two types (either to be memory map based on the file in the hard disk (secondary memory) or memory creation in an empty space of the hard disk) depends on the nature of the required processing. After the creation of the memory map, the view must be created and should be one memory each or more. The creation of view is very important, specifically with the files larger than 2 GB, and this has been discussed later in this paper. The presence of more view has the importance of cutting big data files into several small files which are easy to process. The view consists of pointers that have a beginning and an end and is determined in advance by the file size. Figure 5 shows the algorithm for the creation of the memory mapped file.

Stack overflow is the most important problem faced by programmers in dealing with big data. It is an undesirable situation because it causes the interruption of the work of any proposed system; so, the usage of memory map will address this problem. The capacity of the memory stack in the operating system of 64 bit is equal to 1 Mega, so, the total data associated with these addresses must not exceed 2 Giga. Stack overflow problem makes handling big data within the traditional operating systems almost impossible, therefore, the first step in MPIM is to address this problem. All views can read data from

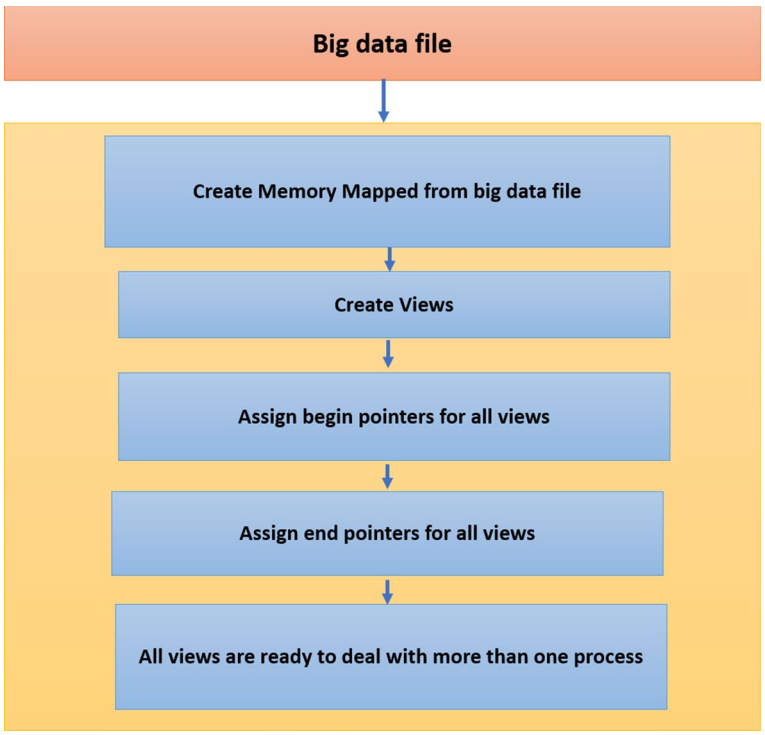

Fig. 5 Diagram of the creation of memory mapped file

the hard disk in a manner of memory map with the size of the data exceeding 2 Giga. Finally, after reading the data through the views, the process moves to the other stages of MPIM and completion of data processing operations. The writer views should create the process of writing on the original file or on an empty space in the hard disk to store the processed file.

Second stage: building message passing interface

MPIM suggests the use of MPI to achieve parallel processing on big data distributed to a number of views in the previous step. The master computer, which includes the main process, manages all views and reads each part of the big file and sends it to the specific slave process which will be received and processed using step 3 of MPIM. Figure 6 illustrates the proposed architecture for MPI in managing and manipulating big data. Although MPI has is costly for data transmission, yet, it is the best solution to address big data that exceeds the size of the memory of the master computer. It is necessary to introduce a set of computers to work in parallel to handle big data in order to achieve high performance and supercomputing. MPIM suggests that MPI work in the master–slave method; there is a master-process in master computers whose task is only to read the big file fragments from views and to manage the process of sending and receiving to the slave-process which is located on a group of computers connected to each other within a computer network. The required processing is the same in all the CPUs and works concurrently. The processing is according to

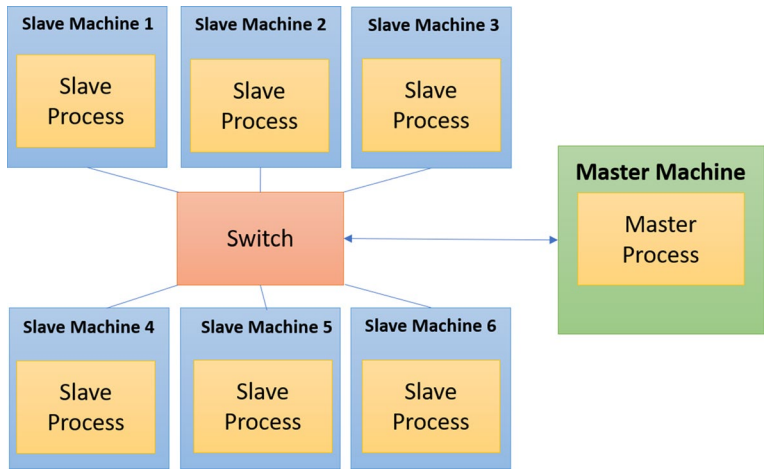

Fig. 6 The proposed architecture for MPI

big data analysis requirements. Therefore, data encryption was chosen as a validation method for MPIM; this will be explained in Step 3 of MPIM. Table 1 illustrates the different frameworks used to handle big data; each framework has its own method of performing parallel processing, all of which rely on almost the same programming languages (Python, Java, Scala, R). So, MPIM suggested using C# programming language in implementing all stages of MPIM. The first step was implemented by supporting the .NET libraries but the second step (MPI) is a major challenge due to the lack of a library supported in the .NET for use in building MPI. Therefore, a special MPI has been built which can send and receive big data through the .NET sockets Library. A specific port is allocated to send and receive data for each salve machine. The proposed MPI receives the data in the form of a byte matrix and sends it in the same form.

Third stage: approach validation

At this stage, the data is processed in a parallel way based on the previous two steps. The data processing may include data encryption, data mining, data analysis, image processing, deep learning or machine learning. The processed data is an image with 2 Giga volume. AES encryption algorithm was chosen to encrypt the image because of its complexity and the time it takes to process. Therefore, data processing challenge is how to process big data with a complex encryption algorithm; this is what MPIM requires to prove its success from its failure, taking into consideration the high-performance computing. One of the most important contributions in this paper after MPIM is to encrypt the big data in real time; as the data is received by the slave machine in the form of a byte matrix and enter into the encryption algorithm, it is sent to the master computer after encryption to collect all the encrypted data from different slave machines to aggregate and write them again. Figure 7 shows the details of MPIM in all steps.

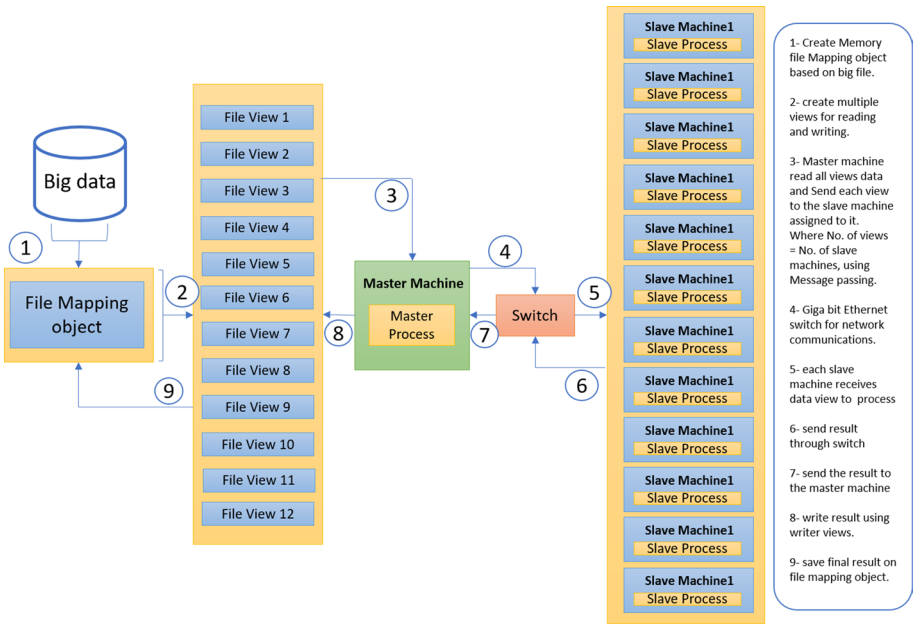

Fig. 7 MPIM approach

Methods and experience

Mathematical model

The mathematical models for MPIM were proved mathematically as below. Let S be the size of a big file, R as the data bit rate = 1 Gbps, P = {p1, p2, p3, ...} as a number of (message passing) processes that run in parallel, and let's consider T as the execution time of data processing. The idea of the mathematical model was simply based on the transfer of big data to assume its size equal 2 Giga in the Ethernet network at a transmission speed of 1 Gigabit per second, equivalent to 128 Megabyte per second. The transmission line was divided into the number of message passing processes that will run in parallel. So, the mathematical model was represented in Eq. 1. This equation contains a multiplication operation (of 2) because the transfer process will be back and forth with the same transmission line and the same data size from the master computer and other computers within the process (send-receive). In the case of non-homogeneous computers, the Max execution time will be taken within the network, but if they are homogeneous, the execution time will be uniform in all computers.

$$E = (S/R/P) * 2 + MAX(Ti) \quad (1)$$

where E is the performance evaluation of the execution time of the overall system. The performance of MPIM was examined using Eq. 1, and the execution times were identical to Eq. 1 (as discussed in "Results" section).

Implementation

Systems for big data analytics must have enough capacity, memory, and bandwidth to perform real-time parallel task processing. Hence, there is a need to implement the

proposed system over a cluster of 2 servers. Each of the selected servers in its implementation has 2 Intel Core i7-7700HQ CPU 2.8 GHz processors running on 64-bit Window 10 Ultimate. The servers were also equipped with 16 DDR3 RAM and 2 TB hard disk; Gigabit switch was used to achieve the highest level of data transmission within the network at a rate of 1 Gigabit per second.

Data gathering

The dataset was used in this paper is a satellite image of Nineveh province in Iraq. The size is 2 Giga and available on the link mentioned at the end of the paper and illustrated in Fig. 8. This image was cut into several parts according to the implementation of various attempts to examine the MPIM.

Data processing

The data processing was done by cutting the image and reading each piece through a view as mentioned in the first step of MPIM. The file cuts were equal in sections based on the size of the file; the process of cutting was divided between 2 and 12 by the size of the file on the figures, as shown in Table 2. MPIM was examined through all the resulting sizes. At first, the file was divided into two parts and the approach was implemented by using two message passing processes and measuring the result. Then, the execution was done by dividing the file into three parts and executing them. Then measuring them down to dividing file to 12 parts executed and measured. The goal of the execution in such a way is to know the performance of MPIM and the effect of increasing the number of processes that work in parallel on the performance.

Results and discussion

To validate the MPIM approach, firstly, its performance was evaluated and compared with other big data frameworks such as Hadoop, Spark, and flink. Finally, the feasibility of the proposed approach was evaluated. In a comparative paper [42], a set of programs

Fig. 8 Satellite image of Nineveh province in Iraq

Table 2 The size of the segment based on the number of processes

No. of processes	Segment size
1	0x80000000
2	0x40000000
3	0x2aa99999
4	0x20000000
5	0x19999999
6	0x1554cccc
7	0x12480000
8	0x10000000
9	0xe380000
10	0xcccccccc
11	0xba19999
12	0xaa99999

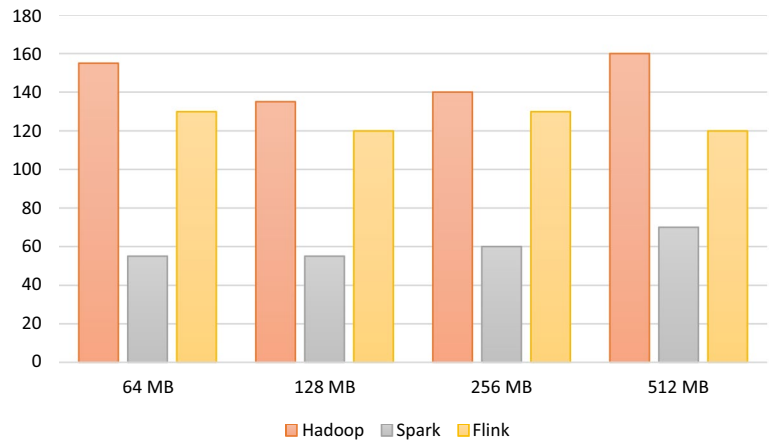

Fig. 9 Execution time (s) for word count program using multiple frameworks

have been implemented in different ways and in more than one framework. The program was chosen as the easiest processing method (word count program) applied to data close to the size of data used in this paper. The results were shown in Fig. 9. The results obtained from the application of the MPIM approach was illustrated in Fig. 10. The results corresponded to Eq. 1, considering that the execution time (T) of the encryption program was 10 s.

Although the encryption program that processed the data in MPIM was complex and the data size is four times the data used in the comparative paper, it should be noted that the results of the MPIM were much stronger as it achieved the fastest execution time of 12.6 s with a data volume of 2 GB. The execution time achieved with Spark in the comparative paper using a data volume of 512 MB was approximately 65 s while the data volume. This proved that MPIM succeeded in addressing big data problems through the proposed parallel processing method.

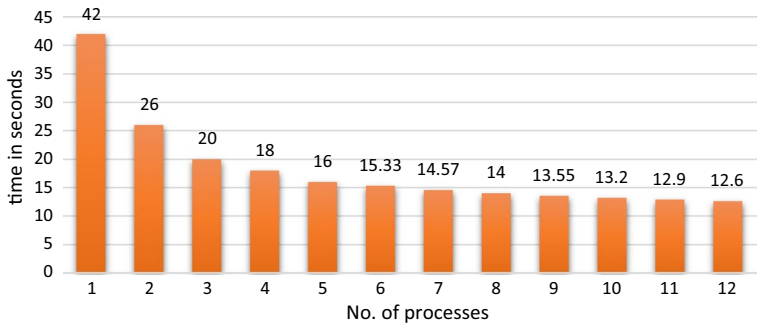

Fig. 10 Execution time (s) for AES encryption program using MPIM

Conclusion

The use of parallel message passing based on memory mapping is one of the best ways to handle big data by exploiting computer resources over a strong communication network. However, there is a challenge that increasing the number of processors working within the network should be calculated based on artificial intelligence. This may be a future research work. The large load on the processors may interrupt the work of the processor or may cause delays in work, thus, reduce system performance. This should be avoided when dealing with large data. Even with all the message passing features, the problem of time loss in transmission over the network exists and is considered one of the most important message passing problems. Therefore, other techniques should be used to reduce this cost to get stronger results commensurate with the size of the challenges of big data.

Abbreviation

MPIM: message passing interface based on memory mapped.

Acknowledgements

A particular acknowledgment for University of Information Technology and Communications and ICCI, Informatics Institute for Postgraduate Studies For their great support in accomplishing this research.

Authors' contributions

SAD performed the primary literature review, experiments, proposed the method and also drafted the manuscript. BFA supervised the programming of the application, wrote a part of the manuscript. MNA worked with SAD to develop the application. SAD, BFA, and MNA provided reviews on the manuscript. All authors read and approved the final manuscript.

Funding

None.

Availability of data and materials

The data sets are available in (https://drive.google.com/file/d/1SMeTuEeRZJgJ-1kYNTl_cDG3orPR__Rr/view).

Ethics approval and consent to participate

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Author details

[1] College of Engineering, University of Information Technology and Communications, Baghdad, Iraq. [2] Informatics Institute for Postgraduate Studies, ICCI, Baghdad, Iraq. [3] Computer Engineering Dept, University of Technology, Baghdad, Iraq. [4] Smart Cities Dept, University of Information Technology and Communications, Baghdad, Iraq.

Received: 21 March 2019 Accepted: 3 December 2019

Published online: 16 December 2019

References

1. Golov N, Rönnbäck L. Big data normalization for massively parallel processing databases. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 9382. Springer Verlag; 2015, p. 154–63.
2. Golov N, Rönnbäck L. Big data normalization for massively parallel processing databases. Comput Stand Interfaces. 2017;54(Part 2):86–93.
3. Kalavri V, Vlassov V. MapReduce: limitations, optimizations and open issues. In: 2013 proceedings of the 12th IEEE international conference on trust security and privacy in computing and communications (TrustCom); 2013, p. 1031–8. ⟨http://dx.doi.org/10.1109/TrustCom.2013.126⟩.
4. Wu C, Ku C, Ho J, Chen M. A novel pipeline approach for efficient big data broadcasting. In: IEEE transactions on knowledge and data engineering, vol 28, no. 1, p. 17–28, 1 Jan 2016.
5. Pääkkönen Pekka, Jokitulppo Juha. Quality management architecture for social media data. J Big Data. 2017. https://doi.org/10.1186/s40537-017-0066-7.
6. Zhang X, Tao Du H, Quan Chen J, Lin Y, Jie Zeng L. Ensure data security in cloud storage. In: 2011 International conference on network computing and information security (NCIS), vol 1, May 2011, p. 284–7.
7. Ji C, Li Y, Qiu W, Awada U, Li K. Big data processing in cloud computing environments. In: International symposium on pervasive systems, algorithms and networks, IEEE, p. 1087–4089/12, 2012.
8. Eine Bastian, Jurisch Matthias, Quint Werner. Ontology-based big data management. Systems. 2017;5:1–14. https://doi.org/10.3390/systems5030045.
9. Aji AM, et al. MPI-ACC: accelerator-aware MPI for scientific applications. In: IEEE transactions on parallel and distributed systems, vol 27, no. 5, p. 1401–14, 1 May 2016.
10. Potluri S, Bureddy D, Wang H, Subramoni H, Panda D. Extending OpenSHMEM for GPU computing. In: Proc. IEEE int. parallel distrib. process. symp.; 2013, p. 1001–12.
11. Sidelnik A, Chamberlain BL, Garzaran MJ, Padua D. Using the high productivity language chapel to target GPGPU architectures. Champaign: UIUC Dept. Comput. Sci; 2011.
12. Hammond J, Ghosh S, Chapman B. Implementing OpenSHMEM using MPI-3 one-sided communication. In: Proc. 1st workshop OpenSHMEM related technologies. Experiences, Implementations, Tools; 2014, p. 44–58.
13. Dinan J, Balaji P, Hammond JR, Krishnamoorthy S, Tipparaju V. Supporting the global arrays PGAS model using MPI one-sided communication. In: Proc. IEEE int. parallel distrib. process. symp.; 2012, p. 739–50.
14. Jones T, Brim MJ, Vallee G, Mayer B, Welch A, Li T, Lang M, Ionkov L, Otstott D, Gavrilovska A, et al. UNITY: unified memory and file space. In: Proceedings of the seventh international workshop on runtime and operating systems for supercomputers ROSS 2017, ACM; 2017, p. 6.
15. Rivas-Gomez S, Gioiosa R, Bo Peng I, Kestor G, Narasimhamurthy S, Laure E, Markidis S. MPI windows on storage for HPC applications. Parallel Comput. 2018;77:38–56. https://doi.org/10.1016/j.parco.2018.05.007 **(ISSN 0167-8191)**.
16. Titos-Gil R, Palomar O, Unsal O, Cristal A. Architectural support for efficient message passing on shared memory multi-cores. J Parallel Distrib Comput. 2016;95:92–106. https://doi.org/10.1016/j.jpdc.2016.02.005 **(ISSN 0743-7315)**.
17. Mavriplis DJ. Parallel performance investigations of an unstructured mesh Navier-Stokes solver. Int J High Perform Comput Appl. 2002;16(4):395–407.
18. Cramer T, Schmidl D, Klemm M, an Mey D, Openmp programming on intel r xeon phi tm coprocessors: an early performance comparison; 2012.
19. Dokulil J, Bajrovic E, Benkner S, Pllana S, Sandrieser M, Bachmayer B. High-level support for hybrid parallel execution of C++ applications targeting intel xeon phiTM coprocessors. In: Procedia Comput. Sci., 2013 international conference on computational science, vol 18; 2013, p. 2508–11. http://dx.doi.org/10.1016/j.procs.2013.05.430.
20. Rudek R. Scheduling on parallel processors with varying processing times. Comput Oper Res. 2017;81:90–101. https://doi.org/10.1016/j.cor.2016.12.007 **(ISSN 0305-0548)**.
21. Furht B, Villanustre F. Introduction to big data. Big Data Technologies and Applications. Cham: Springer International Publishing; 2016. p. 3–11.
22. Rajaraman V. Big data analytics. Resonance. 2016;21:695–716.
23. Kune R, Konugurthi PK, Agarwal A, Chillarige RR, Buyya R. The anatomy of big data computing. Softw Pract Exp. 2016;46:79–105.
24. Emani CK, Cullot N, Nicolle C. Understandable big data: a survey. Comput Sci Rev. 2015;17:70–81.
25. Gandomi A, Haider M. Beyond the hype: big data concepts, methods, and analytics. Int J Inf Manag. 2015;35:137–44.
26. Chen M, Mao S, Zhang Y, Leung VC. Big data: related technologies, challenges and future prospects. Berlin: Springer; 2014.
27. Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E. Deep learning applications and challenges in big data analytics. J Big Data. 2015;2:1.
28. Oussous A, Benjelloun FZ, Lahcen AA, Belfkih S. Big data technologies: a survey. J King Saud Univ Comp Inf Sci. 2018;30(4):431–48. https://doi.org/10.1016/j.jksuci.2017.06.001 **(ISSN 1319-1578)**.
29. Elshawi R, Sakr S, Talia D, Trunfio P. Big data systems meet machine learning challenges: towards big data science as a service. Big Data Res. 2018;14:1–11. https://doi.org/10.1016/j.bdr.2018.04.004 **(ISSN 2214-5796)**.
30. Talia D, Trunfio P, Marozzo F. Data analysis in the cloud. Amsterdam: Elsevier; 2016.
31. Armstrong J. The development of Erlang. In: ACM SIGPLAN notices, vol 32, ACM, 1997, p. 196–203.
32. Haller P. On the integration of the actor model in mainstream technologies: the scala perspective. In: Proceedings of the 2nd workshop on programming systems, languages and applications based on actors, agents, and decentralized control abstractions; 2012, p. 1–6.
33. Goglin B, Moreaud S. KNEM: a generic and scalable kernel-assisted intranode MPI communication framework. J Parallel Distrib Comput. 2013;73:176–88.
34. Jin H-W, Sur S, Chai L, Panda D. LiMIC: support for high-performance MPI intra-node communication on Linux cluster. In: International conference on parallel processing; 2005, p. 184–91.
35. Rabenseifner R, Hager G, Jost G. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In: International conference on parallel, distributed and network-based processing; 2009, p. 427–36.

36. Choi J, Kim J, Han H. Efficient memory mapped file I/O for in-memory file systems; 2017.
37. Ousterhout J, Gopalan A, Gupta A, Kejriwal A, Lee C, Montazeri B, Ongaro D, Park SJ, Qin H, Rosenblum M, Rumble S, Stutsman R, Yang S. The RAMCloud storage system. ACM Trans Comput Syst. 2015;33(3):7.
38. Hameed M, Ibrahim MM, Manap NA. Review on improvement of advanced encryption standard (AES) algorithm based on time execution, differential cryptanalysis and level of security. J Telecommun Electron Comput Eng. 2018;10:139–45.
39. R. Concepts. White paper AES encryption and related concepts, p. 0–4.
40. Dhatrika SC, Puvvula D, Gopal SV. A novel approach for AES algorithm in image encryption. Int J Adv Res Sci Eng. 2015;8354(4):85–92.
41. Abood OG, Guirguis SK. A survey on cryptography algorithms. Int J Sci Res Publ. 2018;8:495–516. https://doi.org/10.29322/IJSRP8.7.2018.p7978.
42. Veiga J, Expósito RR, Pardo XC, Taboada GL, Tourifio J (2016) Performance evaluation of big data frameworks for large-scale data analytics. p. 424–31. https://doi.org/10.1109/bigdata.2016.7840633.

Publisher's Note