EVALUATION OF HIGH-PERFORMANCE COMPUTING TECHNIQUES FOR BIG DATA APPLICATIONS

Waleed Al Shehri[*,1], Maher Khemakhem[2], Abdullah Basuhail[3], Fathy E. Eassa[4]
[1,2,3,4] Department of Computer Science, King Abdul-Aziz University, Jeddah, KSA.
[*,1] waleed.ab2@gmail.com

ABSTRACT: *The big data revolution has led to the emergence of many technologies for processing the enormous volume, high velocity and variety of data sources. The capabilities provided by high-performance computing (HPC) have had a substantial impact in supporting this revolution through the integration of heterogeneous hardware and crafting software and algorithms that exploit the parallelism of HPC. However, big data platforms, written in high-level programming languages, and their architectures differ from HPC languages and architectures. This misalignment has resulted in the underutilization of HPC resources and capabilities for big data problems. In this paper, we highlight and evaluate many of the techniques used in the HPC environment based on the most critical factors influencing performance and resource utilization: load balancing and data locality, job scheduling strategies, topology awareness, decomposition techniques and programming models. We recommend establishing a research roadmap that considers these influential factors in the field of resource and job management within HPC environments for big data computing.*

Keywords: High-Performance Computing; Big Data; Load-Balancing; Data-Locality; Resource Management; Parallel Programming models; Power Consumption;

1. INTRODUCTION

Parallel computing has become an essential part of computational technology. Multi-core processors enable the parallelization of daily computational tasks. Parallelization plays a vital role in research and scientific fields where large data sets need to be processed. Algorithms designed to parallelize tasks and computations are used to achieve high-performance results [1, 2].

The performance of such algorithms is highly dependent on the datasets being worked on. It has been found that input and output operations take a significant amount of time compared to the program that does the computation [3], [4]. This input-output includes the data transfer between memory and disk, implying that memory available to complete the computation plays a vital role. Generally, a program written to operate on large amounts of data, more than is able to be kept in active memory, requires data to be swapped from memory to disk and back. Disk access is much more costly in time than is accessing memory. In simple computer terms, it is advisable to reduce the input-output operations to attain maximum performance [5, 6].

Keeping this in mind, when a program is written, it is recommended that there be the least amount of data swapping among cache, memory and disk. In other words, the program should use cache memory as much as possible, before it gets loaded with fresh data from memory. Similarly, the program should avoid making use of data from the disk as much as possible. With consideration of big data applications, this limitation impacts performance significantly [7–9].

Recently, the volume of data produced in many different scientific fields has grown drastically in all big data characteristics. Big data can be characterized by different Vs. such as volume, velocity, variety, veracity, and value [10]. Many systems and applications in the real-world rely on collecting, storing, and analyzing such large-scale data, and this trend is supposed to grow rapidly [11].

Big data technologies, such as Hadoop, MapReduce, and Spark, have emerged to support parallelism in processing enormous datasets [12]. High-Performance Computing (HPC) is the process of aggregating the available computing power such that the maximum amount of work can be done in minimum possible time. HPC can be considered an attractive environment for supporting scientific workflows and big data computing due to its performance capabilities, which has led to a convergence of the HPC and big data fields [13].

Unfortunately, there is usually a performance issue when running big data applications on HPC clusters because such applications are written in high-level programming languages. Such languages may be lacking in terms of performance and may not encourage or support writing highly parallel programs in contrast to some parallel programming models like Message Passing Interface (MPI) [14]. Furthermore, these big data platforms are designed as a distributed architecture, which differs from the architecture of HPC clusters [15].

Alternately, the large volume of big data may hinder parallel programming models such as MPI, Open Multi-Processing (OpenMP) and accelerator models (CUDA, OpenACC, OpenCL) from supporting high levels of parallelism [16].

With a future road toward exascale computing, and with the revolution in the hardware industry for increasing the number of CPUs in each compute node, job scheduling techniques can be considered a cornerstone of any scalable computing infrastructure in terms of encouraging high levels of parallelism, controlling system processes and directly impacting overall performance and system efficiency. However, running big data applications on HPC clusters typically lacks efficient resource and job management techniques for reducing the performance gap between these two domains.

Section 2 of this survey investigates high-performance techniques of load balancing and data locality, job scheduling strategies, topology awareness, and decomposition techniques applied to big data applications to attain maximum performance. In Section 3, some of the relevant parallel programming models are introduced and compared with each other and with big data programming models. Some evaluation thoughts are presented in Section 4, followed by recommendations for future work.

2. TECHNIQUES USED IN HPC

HPC has been discussed in detail in many research papers. This survey paper studies various HPC techniques used for big data applications. Even though the topic itself is vast, this survey attempts to cover many aspects and techniques used in an HPC environment to support big data applications. Listed are some of these techniques, which are classified according to the most important factors affecting this research trend.

2.1 Load Balancing and Data-Locality

Work stealing is a type of load balancing technique. It is used for distributed task scheduling systems. (**Figure 1**). These scheduling systems have multiple schedules; they assist in making scheduling decisions. Through this method, tasks are assigned to idle schedulers and transferred from heavily loaded ones. This transfer can result in poor data locality in data-intensive applications because both tasks and their execution largely depend on large amounts of data processing; this can cause data-transferring overhead.

Work stealing can be enhanced via shared and dedicated queues [17]. The data size and location is used to organize

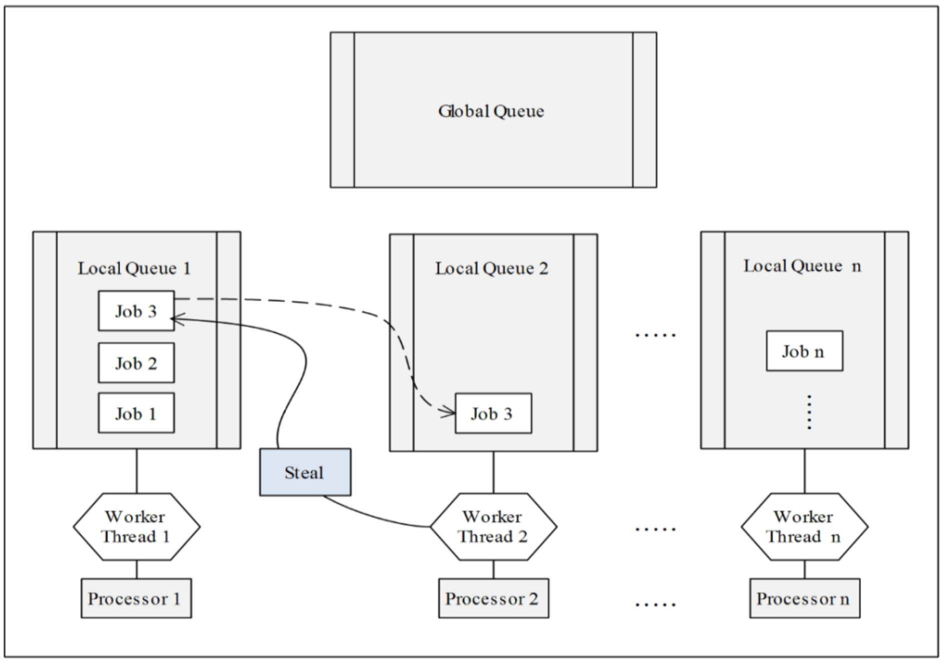

Figure 1. Work Stealing Scheduling Technique.

queues. In such a case, the MATRIX technique is used for distributing task scheduler for the purpose of computing multiple tasks. The metadata is scaled and organized, with the help of a distributed key-value store (DKVS). Other elements organized and scaled are task dependency and data-locality. It is noticeable that the data-aware work stealing technique gives a good performance.

Falkon [18] can be considered to be a centralized task scheduler providing support for Many-Task Computing (MTC) applications naïve hierarchical scheduling. Although Falkon provides better scale, it is prone to the issue of scaling in petascale systems, and its hierarchical implementation affects load balancing during uncertain task execution times. Moreover, for scheduling of data-intensive workloads, the data diffusion approach [19] was adopted by Falkon. During the data diffusion process, storage and compute resources are acquired dynamically, data are replicated with respect to demand, and then computations are scheduled close to data. Contrary to a distributed key-value store [17], Falkon suffers from poor scalability due to deploying a centralized index server for storing metadata.

Charm++ [20] is a machine independent parallel programming system, where load balancing occurs in either a static or centralized, hierarchical or fully dynamic or distributed style. The centralized approach has very poor scalability (i.e., up to 3,000 cores [20]), whereas the dynamic or distributed approach is based on the neighboring averaging schemes that ensure load balancing limits inside a local space and generates poor load balancing at extreme scales.

Mesos [21] is a resource sharing platform among many cluster computing frameworks for task scheduling. It lets frameworks attain data locality via reading data stored on each machine. It deploys the delay scheduling system and waits for a limited time to get data storing nodes. This approach generates substantial waiting time for any task to be scheduled on time particularly for larger datasets.

Quincy [22] is a highly dynamic and flexible framework for concurrently distributed job scheduling by means of fine-grain resources sharing. It uses both load balancing and data locality to find the best solution by mapping a graph structure, which requires a significant amount of time.

Dryad [23] is a broadly useful distributed execution system for coarse-grained information parallel applications. It underpins applications structured as work process directed acyclic graphs (DAGs) like the Hadoop scheduler [24], Dryad does centralized scheduling that maps undertakings to where the information is present, which is neither reasonable nor versatile.

CloudKon [25] architecture is just like the MATRIX architecture with the only difference being an emphasis on the cloud environment. It is dependent on various cloud services such as SQS [26] to perform circulated load balancing and deploy DynamoDB [27] as the DKVS system for metadata task keeping. Cloud services ensure easier and simpler development but show control and overall performance loss. Moreover, CloudKon also lacks support for data-aware scheduling.

SLAW [28] is an exceedingly versatile, locality-aware work-tasking scheduler framework supporting both work and help-first systems [29] adaptively at runtime on a for each assignment premise. In spite of the fact that it is intended to address adaptability work tasking issues, it accentuates the center and thread level too. That strategy of SLAW does not bolster the huge scale distributed frameworks by any means.

Another work is proposed in [30] which employed data-aware work stealing by deploying both dedicated and shared queues. Nevertheless, it relies upon the X10 worldwide address space programming model [31] to uncover the information territory data statically and to separate between the location adaptable and location sensitive tasks at the start. This task data-locality information stays unaltered subsequent to characterizing them. However, this approach is not adaptable for multiple data-intensive workloads.

Table 1. A comparison of some Work Stealing techniques.

Technique	Features	Limitations	Ref. no.
MATRIX	- It is used for distributing task scheduler for multiple tasks computing purpose. - Enhanced data locality via shared and dedicated queues. - The data size and location are used to organize queues, task dependency and data-locality. - The metadata is scaled and organized, with the help of a distributed key-value store.	- Not versatile for large-scale computing. - Not support HPC workloads.	[17]
Falkon	- A centralized task scheduler to support Many-task computing (MTC) applications via naive hierarchical scheduling.	- Difficult to scale to the petascale system. - The hierarchical implementation also affects the load balancing during uncertain task execution times.	[18]
Charm++	- An independent machine on a parallel programming system. - Entire load balancing occurs statically or dynamically or in distributed style.	- Poor scalability.	[20]
Mesos	- A resource-sharing platform of cluster computing for tasks scheduling. - It lets frameworks to attain data locality via data reading stored on each machine. - It deploys the delay scheduling system to get data storing nodes.	- Generates substantial waiting time for any task to be scheduled on time particularly for the larger set of data.	[21]
Quincy	- A highly dynamic and flexible framework for concurrent distributed jobs scheduling. - Uses fine-grain resources sharing system. - Uses both load balancing and data locality to find the best solution by mapping the graph structure.	- Requires a significant amount of time due to the mapping of graph structure. -	[22]
Dryad	- Distributed execution system for coarse-grained information parallel applications. - It supports running of applications structured as work process DAGs like the Hadoop scheduler.	- Centralized scheduling of mapping processes to where the information presents is neither reasonable nor versatile.	[23]
CloudKon	- Like the MATRIX architecture with emphasizing on the Cloud environment. - It is dependent on the various cloud services such to perform distributed load balancing.	- Lacks support for data-aware scheduling at an existing stage.	[25]

Generally, a computer-centric architecture is preferred for scientific applications that are based on MPI processes. The MPI processes run on various nodes using a file system, and they access the data needed to run the scientific applications. However, the high performance and efficiency of these MPI-based applications are threatened by the volatile growth of scientific data. The study [31] presents a solution to this problem: Data Locality MPI (DL-MPI). The DL-MPI proposes that the MPI-based applications obtain data information from computer nodes through a data locality API alongside a scheduling algorithm. The premise behind DL-MPI is that the scheduling algorithm will task each node based on their individual capacity thus increasing the efficiency and performance of the MPI. Similarly, data locality API will assess the amount of unprocessed local data and help the scheduling algorithm allocate processing to compute nodes. While the solution is novel and noteworthy, it requires sophistication as the algorithm does not scale effectively when the number of nodes is large. It would seem that the data movement overhead obstructs the scaling of the system in the baseline of the system.

In the literature, a multitude of data distribution methods for non-uniform memory access (NUMA) systems has been suggested. Huang et al. [32] proposition of extensions to OpenMP to allocate data to an undefined, immaterial notion of locations is one such suggestion. The method advocates block-wise distribution of data; while the effort required to program this distribution method is more than its counterparts, it allows for greater and more accurate control of data distribution.

Alternatively, the Minas framework [33] advocated for a data distribution API that uses an automated code transformation to find the best distribution for a given program based upon the unique profile of that program. Like, Huang et al. [32], this method also allows for precise control over the data distribution and expert programmers are needed for its execution.

Majo and Gross [34] proposes another method. Their method relies on the use of a fine-grained API to distribute data. The API makes use of execution profiling to obtain the data access patterns of loops that are simultaneously used for directing the distribution of code and data. Loop iterations are pivotal in this method, as data distribution is executed between the iteration of these loops which ensures that memory pages are accessed locally in every iteration.

Nikolopou-Los et al. [35] proposed yet another method, a runtime tracing technique that uses automatic page migration. The salient feature of this method is that unlike the above approaches, minimal programming effort is required. Moreover, the page migration is based on the continuous monitoring of hardware through performance counters. The need for programming is minimal because of the performance counters as well as the user-level framework approach in which page access runs in the background whereas hot pages are shifted closer to the node that will access it, thus increasing performance.

The data and task affinity techniques, as discussed in [35], rely heavily on the concept of OpenMP runtime systems NUMA optimizations. The co-scheduling and local distribution elements of the concept influence this work significantly and are implemented in the runtime systems developed within the paper. Broquedis *et al.* [36] have presented a similar methodology.

Data distribution and locality-aware scheduling on many-core processors have not been addressed adequately in the literature. Yoo et al. [37] give a point by quantitative point examination of territory aware planning for data parallel projects on many center processors. It has been concluded that work-stealing scheduling cannot capture locality because scheduling algorithms are ineffective in data-parallel programs. Therefore, a systemized locality-aware streaming and scheduling method is suggested that ensures probability maximization of exploring the joined memory impression of a task group in the least level store to guarantee accommodating footprint. The method, however, cannot perform effectively without prior order information and task groupings that are acquired via profiling read-write task sets and offline graph analysis.

Vikranth et al. [38] suggest confining stealing to groups of core centers in light of the topology of the processor. Tousimojarad and Vanderbauwhede [40] embraced a smart approach. They recommended diminishing the entrance latencies to data distributed by utilizing duplicates, rather than originals, whose access is local in nature to the thread access on the TILEPro64 processor. Alternatively, Zhou and Demsky [39] approached the problem from another angle and suggested that objects should be migrated to numerous processors to increase local access. They developed a NUMA- aware adaptive garbage collector for this purpose. However, programs written in C are more laborious to migrate and thus limit the efficiency of this approach.

Lu et al. [40] suggested the method proposed by Zhou and Demsky [39] of migrating data for multiple core processors. Lu et al. [40] claimed that rearranging affinity for-loops during compilation reduces access latency to data which has been distributed uniformly on the caches of multiple core processors. On the other hand, Marongiu and Benini [41] suggested partition arrays to rearrange data and increase access latency. Data is rearranged to reduce latency and then redistributed according to its profiling. Marongiu and Benini [41] were motivated to adopt this approach because they wanted to enable data distribution on MPSoCs framework without any memory management hardware support.

Another approach is by Li *et al.* [42, 43]. They deployed the compilation-time information system to monitor the data placement runtime system. Finally, R-NUCA [44], like Tousimojarad and Vanderbauwhede [45], uses an automated system to migrate shared memory pages to the cache memory.

The research study in [46] claims that energy waste from High-Performance Parallel-Systems can be reduced by leveraging the locality-awareness principle in order to implement optimized power-efficient techniques. It elaborates that two symbiotic techniques can be used to achieve power efficiency in high-performance parallel systems: (i) intra-locality power driven power optimization and (ii) inter-process locality-driven power optimizations. The intra-locality technique gives programmers and system designers the control and flexibility to assign processor frequency and manage sleep states within the sequence of the same process. On the other hand, the inter-process technique uses the concepts of co-placement and co-scheduling of jobs to a varied set of threads from a diverse set of processes which are executed simultaneously on an HPC cluster. Co-placement and co-scheduling group threads and processes based on the similarity of their affinity patterns and symbiosis allows for a reduction of as much as 25% in energy consumption. The efficiency of energy reduction depends heavily on the correct identification of CPU capacity and computer memory functions. These two techniques, when used in unison, proved to be more fruitful than independent attempts. Thus, it is prudent to analyze the results of these techniques in a systematic framework. Since, the breakthrough of measuring the framework itself is a great innovation, which contributes significantly to the growing numbers of researchers in HPC power optimization.

2.2 Job Scheduling Strategies

Resource and energy allocation techniques are needed for Quality of Service (QoS) and to slash system operational costs, which is needed in a large-scale parallel computing environment. The reason why the cost of energy consumption needs to be reduced is due to its importance in both the user and the owner's budget. Resource allocation strategies are very complicated when the matter of energy efficiency is a focus, and this can infringe on both the response and queue time.

In the same large-scale and parallel computing system, another factor impacting storage and computational resource access is the Resource Provider (RP). It uses the finite resource at hand, for some users. It also plays a part in maintaining QoS levels. Job scheduling technique should be chosen carefully in order to tackle the resource management dilemma. Scheduling techniques are used in performance optimization. Resource management scheduling techniques are a focus in research such as [47]–[53]. These researches solve the resource allocation problem under different QoS constraints. For example, there is a metric-aware scheduling policy suggested by Wei *et al*. [47] where the scheduler balances the different competing scheduling objectives. These objectives include efficiency, job response time and system utilization. Research conducted by Khan et al. deploys a self-adaptive and a weighted sum method as mentioned in [48] and [49]. Moreover, game theoretical methodologies [50] and goal programming approach [51]. These are used for optimization of system performance for grid resource allocation, and this is done under different QoS constraints. Eleven static heuristics were researched by Braun et al. [52]; he wanted to map independent tasks on a heterogeneous distributed computing system. There were common assumptions undertaken by the author, who evaluated and executed task mapping policies.

2.2.1 Resource Management

There are applications that have resource requirements that need HPC schedulers, and these schedules are specifically designed for the applications. Monolithic simulation is an example of an HPC application. It has a requirement for a massive set of cores and uses tightly coupled MPI-base parallelism. These MPI applications are high latency sensitive. The resource demands in these applications do not vary. The number of cores, memory and wall clock time needed do not change. Gang scheduling can be used here. Scheduling is primarily carried out on application-level tasks, which are also called job-level tasks. These tasks include the implementation of individual processing on compute nodes and are not shown to the resource manager. Both short-running and long-running batch-oriented tasks all come under the heterogeneous scheduling workload. This then challenges the traditional centralized and monolithic cluster scheduling systems. In order to eliminate the flexibility limitation and to provide support to the dynamic application that has heterogeneous tasks, Pilot-Jobs are used. HPC does not focus on data locality. The intensive data workloads can be broken down into loosely coupled parallel tasks. The utilization of resources, along with the improvement in their fairness, can be done at the task, rather than job, level [54].

YARN [55] and Mesos [21] are multi-level schedulers. The advantage of these two schedules is that they are can support data-intensive workloads that have lightly coupled tasks. They also make dynamic resource allocation and usage possible with the help of application-level scheduling. The interactive workloads low latency and scalability bottleneck requirements are addressed through the decentralized schedulers. There are two application-level schedulers for Spark, called Omega, which is from Google [56], and Sparrow [54]. These are decentralized schedulers.

Hadoop deploys a centralized task scheduling approach that is based on the Job Tracker, which is, in fact, a resource manager. Hadoop presents the Job Tracker with an advanced scalability bottleneck displayed in the MapReduce framework, which controls flexibility. Initially, this issue was not identified at all, and Hadoop was in use in the top HPC clusters. Hadoop used five things: Hadoop on Demand [57], SAGA Hadoop [58], My Hadoop [59], Amazon's Elastic Map Reduce [60] and Microsoft's HDInsight [61]. These approaches can be limited by not having sufficient data locality, which means that moving data to the Hadoop filesystem (HDFS) becomes necessary and this data move has to happen before computation. However, there were usage modes for Hadoop clusters that started to emerge. There was an emerging demand for an increase in different sizes and varieties of frameworks and applications and their requirements. This meant that supporting batch, interactive data processing, and streaming became very important.

Resource usage and performance was not easy to predict when a higher-level framework, like HBase or Spark, was deployed with Hadoop daemons. YARN [55] was the core of Hadoop 2 and was developed to address this problem while supporting the Hadoop environment based on large, heterogeneous workloads. Mesos [21] is a multi-level scheduler developed for Hadoop. The ecosystem that evolved on top of HDFS and YARN was large. The runtime system integrates an application-level scheduler that synchronized with YARN including HBase, Spark, and MapReduce. The general requirements are then embedded with the advanced-level shared runtime systems structures, like using DAGs, which support longer, interactive or multi-stage applications. Llama [62] provides a longer application controller for YARN-based applications. It is designed specifically for the Imapa SL engine. An example of a DAG processing engine is TEZ [63], which is designed to support the Hive SQL engine.

Short-running and data parallel tasks can be found in typical data-intensive workloads. If a job-level scheduler is used instead of a task-level scheduler, like YARN, then the overall cluster utilization can be made better, as the resources can be shifted between the applications. The scheduler can do things like removing resources from applications by waiting until task completion. In order to make resource usage as dynamic as possible, tighter integration of an application is required by YARN. The application should register as an Application Master process, which has a subscription to callbacks. The container is the unit of scheduling, and they are required from the Resource Manager. As compared to HPC schedulers, the requested number of resources is not asked for by the Resource Manager. This means that the application should use the resources elastically when they are allocated by YARN and YARN can even request container de-allocation as well. This means that the application will have to make a note of the resources that are available at that time.

2.2.2 A Convergence of High-Performance and Big Data Computing

HPC and Hadoop were designed to support multiple workload types and requirements. In the Hadoop case, the combination of various levels can be seen as the top of the line in parallel computing in HPC against modest data storage and recovery. Hadoop clusters have deployed more compute-demanding workloads. Data parallel tasks and workflows are also implemented on the Hadoop infrastructure. As Mesos and YARN were introduced, the ecosystem of Hadoop developed the ability to support heterogeneous workloads. There were tools like Pilot-Jobs, which supported loosely coupled and data-intensive workloads on HPC infrastructure. However, the Pilot-Jobs tools and other tools supported a large number of computing tasks for specific domains, and they could not reach the scalability and diversity of the ecosystem made for Hadoop.

MapReduce, iterative MapReduce, diagram analytics, machine learning and other advanced level applications for data storage, data processing and data analytics were supported by the Apache Big Data Stack (ABDS) ecosystem. This ecosystem is created from extensible components like the HDFS and YARN. In contrast to HPC schedulers, YARN is made to help heterogeneous workloads that deploy staggered and data-aware scheduling. To achieve this goal, a higher level of joining between the application or the structure and the framework level scheduler is a necessity. This is higher as compared to the HPC schedulers. YARN applications do not request static resources before they run, but instead request resources in an efficient way. When applications optimize resource usage, the cluster's usage becomes better as well. Even though YARN is not part of the HPC resource fabric, there have been suggestions to integrate Hadoop/YARN and HPC because of the many advantages that YARN brings. Three things need to be considered. Firstly, assurance of local resource management level system integration. Secondly, integration with multiple external resources such as parallel and shared file systems, which are HPC storage resources. Thirdly, the deployment of advanced network features like remote direct memory access (RDMA) and several other efficient abstractions that include joint operations. In resource management integration, integration can be achieved with the help of system level resource management system. The Hadoop-level scheduler needs to be adopted on top of the system-level scheduler. Condor and SLURM are resource managers that give support to Hadoop. Moreover, there are third-party systems like SAGA-Hadoop [58], JUMMP [64] and MyHadoop [59] as well. However, the main problem with the approach is that data locality is missing, and the system-level scheduler is not aware of it. Further, when the HDFS is used, data is copied from HDFS, after which it is processed. This also represents a substantial overhead. Hadoop is used in a single user mode here, and there is no optimal way to use the resource cluster.

In Storage Integration, a pluggable filesystem abstraction is given by Hadoop. This filesystem interoperates with any POSIX compliant filesystem. Parallel file systems can be used with Hadoop, but there are cases in which the Hadoop layer will not be aware of data locality maintained on the parallel file system. This includes the Intel-supported Hadoop, which is on top of Lustre [65], IBM on GPFS [66]. With the help of the shared file system, the MapReduce shuffling phase is carried out and, therefore, is an optimization concern [67]. These filesystems and their scalability, as compared to HDFS, is constrained. This is because much of the data is processed locally requiring data movement across the network. HDFS does not rely on fast interconnects.

In Network Integration, Ethernet environments are a fit for Hadoop as they use Java sockets for communication. RDMA, which is a high-performance feature, is not used here. In order to solve this issue, it is proposed that RDMA be used in supporting MapReduce, HDFS, HBase and a wide range of other components for 10 Gigabit Infiniband networks.

Direct Acyclic Graphs are scientific workflows, and even though they are useful in understanding the dependence of relationships, they do not give information about temporary data files, input or output. This type of information is essential because it helps eliminate performance issues. The paper in [68] shows a multi-workflow store-aware scheduler that is found in a cluster environment, known as Critical Path File Location (CPFL). In this approach, disk access time is imperative when contrasted with the entire network system, as it is an augmentation to the traditional list scheduling policies. The primary point here is the discovery that the best area for data documents in a progressive stockpiling framework and the algorithm that left it was tried in an HPC cluster with impressive execution performance.

A prologue/epilogue mechanism is used as a foundation for a Resource and Job Management System (RJMS) [69]. It promotes communication between HPC and Big Data systems, as it reduces the disturbance on HPC workloads. This is carried out by leveraging the resilience of Big Data frameworks.

There is advanced performance hardware in HPC, including Parallel File Systems (PFS), and fast interconnect systems which can be advantageous [70]. Some take HPC traditional technologies, totally opposite to what is mentioned before, and build new Big Data tools. MPI is one way to do it. They use such tools because of their resilience, and they do not compensate for the lack of performance [71]. Using traditional Big Data tools is not a waste when used on HPC infrastructure if their resilience and dynamism are leveraged. There are many techniques to use Big Data when it comes to the resource management level. Big Data and HPC workloads can run side by side.

The straightforward approach is to design two clusters, which will be committed to each workload type. However, the exchange between the groups will be a bottleneck. There is likewise no heap when the two clusters are adjusted to account for the partition of concern. A less complicated approach here is giving the client a chance to deal with Big Data programming inside HPC all alone. This can be connected to straightforward work processes that don't convey much information. However, it has its drawbacks. Right off the bat, regardless of whether the contents are useful for the client, the client may think that it is hard to maintain awareness of the system.

Furthermore, the Big Data programming stacks must be sent, arranged, begun and shutdown for each activity completed. The overhead should be noted if there should be a need for one-off applications or frameworks. Finally, if the information is too vast and can't fit on the client's resources, an outsider stockpiling would be required for moving information and this can affect execution time [72]. Another thing to do here is to run HPC jobs, using a Big Data stack. However, most HPC applications are not compatible with Big Data RJMS, and a communication layer is required. HPC applications have to be rewritten to make use of the Big Data systems. In an integrated approach, new abstractions are needed to make it one system; this converts Big Data RJMS resource requirements into HPC RJMS allocations [73]. However, these approaches apply to some technologies. They evolve with them but have limitations as well. Thus, the solution is limited to implemented adaptors, and their cost can be high. A more suitable approach here is to summarise HPC on Hadoop or vice versa, by using a Pilot base abstraction [74]. The two systems can be joined using software, as suggested by the authors.

2.3 Topology Awareness

The computing platforms expanding intricacy fueled the requirement for programmers and developers to comprehend hardware organization and to adjust their applications accordingly. As a component of the general enhancement process, there is a substantial requirement for an ability to visualize a model of the hardware platform. hwloc is the most famous programming tool for uncovering a static perspective of the topology of CPUs and memory.

The paper in [75] shows how hwloc reaches out to these computing assets by joining I/O devices topology and offering approaches to deal with different hubs. I/O territory data has been added to the hwloc tree speaking to the equipment and additionally to enable applications to recognize the resources they utilize, put undertakings close to them or to adjust their area. Later on, to control the remote host's topologies with pressure for useful adaptability, an API was introduced.

The features that this paper is highlighting are mentioned in the hwloc v1.9 (released in Spring 2014). On-going work is currently concentrating on enhancing topology discovery on developing ARM designs for elite figuring and also programmed conflict management between constant sources of data.

Deploying the advanced hardware resource to a particular application is not something new and has been discovered in earlier researches. It can be grasped clearly in a grid environment context [76–78], as the choice of the best combination of resources (clusters) to use is an essential factor. The advantage of using such an approach is to reduce the WAN communication impact in grids but does not account for real topology details, such as the effects of NUMA or cache hierarchy.

More recently, a specific type of application has been targeted by some work, such as MapReduce- based applications. Like Topology-Aware Resource Adaptation (TARA) [79] that uses the application description for resources allocation purposes. However, the work is designed for a particular set of applications only and not meant to address other hardware details.

The network topology backs parallel applications mapping jobs to physical hardware. It can lead to significant improvements in performance [80]. The scheduler can consider the network topology characteristics [81] to favor selecting a set of nodes being connected to the same network under the same switch or even placed near each other to avoid long-distance communication. Most open-source and proprietary RJMS takes this kind of feature into account by using the underlying physical topology characteristics. However, they eventually failed to consider the importance of application behavior during resource allocation. An approach named HTCondor, formerly Condor [82], is proposed to take advantage of the matchmaking method, which allows matching the requirements of applications with the available hardware resources. However, this matchmaking method does not consider the application behavior, and HTCondor applies to both clusters and connected workstations. Slurm [83] provides a feature to minimize the number of network switches used in the allocation, thereby reducing communication costs during the execution of the application. Obviously, switches located deep in the topology tree are supposed to have less communication cost than those at the top layer.

Similarly, PBS Pro [84] and LSF [85] exploit the same concept of topology-aware placement. Os Fujitsu [86] employs a similar technique, but only for its proprietary interconnect called Tofu. According to our knowledge, Slurm [83] is still the only one that provides a best-fit topology-aware selection system while the others only suggest the first-fit algorithm system.

Part of RJMS offers alternatives for task placement that can provide an appropriate arrangement for different application forms. Torque [87] recommends NUMA based job task placement system. Be that as it may, in this current work, just the system topology is considered and the hubs intuitive design is left unresolved when execution results are conventional from misusing the memory chain of command.

Multiple binding strategies are accessible and supported with the Open MPI approaches. In every one of these solutions, the user first needs to recover the compositional points of interest before delivering their job or task. Moreover, the placement choices offered to put the weight on the client to decide on their approach and the application correspondence plot is not considered.

2.4 Decomposition Techniques

The way toward software application parallelization can be exceptionally dreary, and error inclined, specifically the data decomposition task [88], [89].

While choosing a parallelization methodology, data decomposition and the consequent conveyance of data or task processing running on available cores is a fundamental point to consider [90], [91].

Several researchers have conducted different studies to a better understanding of parallel program needs and the state-of-practice of parallelization [92–94]. Such studies have played a role in understanding the project organization, software usage, debugging, testing and tuning. On the other hand, data decomposition is an area of limited attention. This surprise leads us to the fact that data decomposition is a significant challenge in a parallel programming environment [89].

Up until this point, there have been insufficient empirical researches in the HPC field that has concentrated on these issues. The assignment of performing data disintegration and correspondence while parallelizing applications has been examined in empirical research [95]. The use of techniques to help with this task and the current state of practice was examined. To support the undertaking of data deterioration and correspondence, while parallelizing applications, an arrangement of crucial necessities is inferred for tools of this system.

Table 2. Key Features of OpenACC, OpenMP, OpenCL, and CUDA

Features	*OpenMP*	*OpenACC*	*OpenCL*	*CUDA*
Supported parallelism	- Data and task parallelism.	- Data and task parallelism.	- Data and task parallelism.	- Data and task parallelism.
Supported parallelization system	- Host and device.	- Device only.	- Host and device.	- Device only.
Synchronization	- Barrier - Reduction - Join	- Reduction - Join	- Barrier - Reduction	- Barrier
Implemented code	- Programa (compiler directives) for C/C++ and Fortran. - Compiler-time.	- Programa (compiler directives) for C/C++ and Fortran. - Compiler-time.	- C/C++ extension. - Run-time.	- C/C++ extension. - Run-time.
Architecture level	- High level.	- High level.	- Low level.	- Low level.
Supported GPUs manufacturers	- Numerous core accelerated devices.	- Numerous core accelerated devices.	- Numerous core accelerated devices.	- NVIDIA GPUs.

The idea of decomposing multi-dimensional arrays in a multi-view data model [96] is common to that of distributed arrays supported by specific languages and frameworks. High-Performance Fortran (HPF) offers compiler directives that define array element distributions [97]. The easiest way to decompose an array is to designate keywords, such as BLOCK or CYCLIC, to each dimension. Partitioned Global Address Space (PGAS) provides a locality-aware shared address space among multiple processes and is supported by languages developed for high-performance computing, such as Co-array Fortran, Unified Parallel C [98], UPC++ [99], X10 [100], and Chapel [101]. PGAS languages also provide simple notations for describing data decomposition. However, HPF and this PGAS do not have a concept of View as in [96]. As a result, users have to manually write code for mapping tasks to processes by considering data location when the task itself should be run in parallel. Hierarchically Tiled Array (HTA) is a distributed array that enhances locality and parallelism for improving performance on multicore systems [102]. An HTA is composed of hierarchical tiles that contain sub-tiles or values, and a unit of tiles performs data assignment to processes and data processing. Although it may be possible to achieve the same effect of data distribution and processing by using HTA, users have to carefully consider the hierarchy and indices of tiles when writing such codes. Habanero-Java provides Array-View, in which a one-dimensional array can be viewed as any higher-dimensional array, to improve productivity [103].

3 PROGRAMMING MODELS

3.1 Parallel Programming Models

Recently, parallel computing systems have been making greater use of heterogeneous tools at the node level. The types of nodes in use may include universally useful CPUs and performance accelerators, e.g., GPUs or Intel Xeon Phis, that give superior qualities in terms of energy utilization. However, it is not easy to exploit the performance available in heterogeneous systems, and this may be a tedious and challenging process. There are a wide range of parallel programming models (such as OpenCL, OpenMP, CUDA) and selecting the right one for the target context is not so easy and straightforward.

above summarises features of parallel programming models as considered in the study in [104], i.e., OpenMP [105], OpenACC [106], OpenCL [107], and CUDA [108]. OpenACC and OpenMP are mainly implemented as C, C++, and FORTRAN compiler directives, which significantly hide the detailed information of the computing architecture from the programmer. Interestingly, CUDA and OpenCL are executed as programming/software libraries for only C and C++ and open the developer to lower architectural details. As far as the boost from parallelism is concerned, the majority of the models considered widely support task parallelism and data parallelism. While OpenCL and OpenMP give parallelization to both multi-center CPUs and numerous core accelerated devices, CUDA and OpenACC support for parallelization are implied only for accelerators such as NVIDIA GPUs [109].

Concerning distributed computing, MPI based parallel executions are, for the most part, flexible to a broad number of processors but may be less efficient when used on specialist hardware. Precisely when various processor centers are established their parallel efficiency is significantly lower [110]. Of course, OpenMP is a conventional memory parallel programming system, which grants thread level parallelism. Joining OpenMP and MPI parallelization for hybrid program assembly can achieve two levels of parallelism. This approach decreases the overhead of MPI while minimizing OpenMP overhead due to thread creation and deletion. MPI/OpenMP have been shown to have better execution in comparison with MPI frameworks, for a couple of use cases [111, 112]. The OpenACC Application Program Interface gives compiler orders, library calls and condition factors that empower programmers to make the parallel code for various systems such as General Purpose Graphics Processing Units (GPGPUs) [113]. Uniting OpenACC and MPI gives the limit of running a parallel code in a collecting mode with more than one GPU, distributed among the cluster nodes, increasing the total number of cores joined.

3.2 Parallel and Big Data Programming Models: Performance Perspective

It has been widely noted that MPI-based HPC frameworks outperform Spark or Hadoop-based big data frameworks by order of magnitude or more for a variety of different application domains, e.g., support k-nearest neighbors and vector machines [71], k-means [114], graph analytics [115, 116], and large-scale matrix factorizations [117]. Recent performance analysis of Spark showed that compute load was the primary bottleneck in some Spark applications, specifically serialization and deserialization time [118]. Other work has tried to bridge the HPC-big data gap by using MPI-based communication primitives to improve performance. For example, Lu et al. [119] indicate how replacing map-reduce communicators in Hadoop (Jetty) with an MPI derivative (DataMPI) can lead to better performance; the limitation of this approach is that it is not a drop-in replacement for Hadoop and existing modules need to be re-coded to use DataMPI. It has been shown that it is possible to extend simple MPI-based applications to be elastic in the number of nodes [120] through periodic data redistribution among required MPI ranks. Efforts to add fault tolerance to MPI have been ongoing since at least 2000 when Fagg and Dongarra proposed FTMPI [121]. Although the MPI standard has still not integrated any fault tolerance mechanism, proposed solutions continue to be put forth, e.g., Fenix [122]. However, there is a tremendous productivity gap between the APIs of Fenix and Spark. Several machine learning libraries have support for interoperating with Spark, such as H2O.ai [123] and deeplearning4j [124], and include their communication primitives. However, these are Java-based approaches and do not provide for direct integration of existing native-code MPI-based libraries. Spark With Accelerated Tasks (SWAT) [125] creates OpenCL code from JVM code at runtime to improve Spark performance. As opposed to our work, SWAT is limited to single-node optimizations; it does not have access to the communication improvements available through MPI.

4 EVALUATION

The techniques discussed in this survey are widely used by high-performance computing for big data applications. Each of these techniques has its own advantages and disadvantages when compared to each other. While a technique is good for a given dataset, it may not be suitable for another dataset or application. In one case, the technique may give very positive results with high performance, but it may fail in another case.

Recently, big data has emerged as a universal concept, and its management has attracted the attention of the research community. In any big data computing system, there are two important factors: the data-centric programming model, and the underlying infrastructure, which is an integration of storage and computation resources in each node [126]. Processing such large and complex datasets require collaborative High-Performance Computing (HPC). One of the prime challenges in HPC is how to allocate resources effectively, especially since each paradigm has a different software stack [127] as shown in **Figure 2**. This raises this question: are the existing techniques of HPC scheduling and resource allocation effective enough to support big data computing?

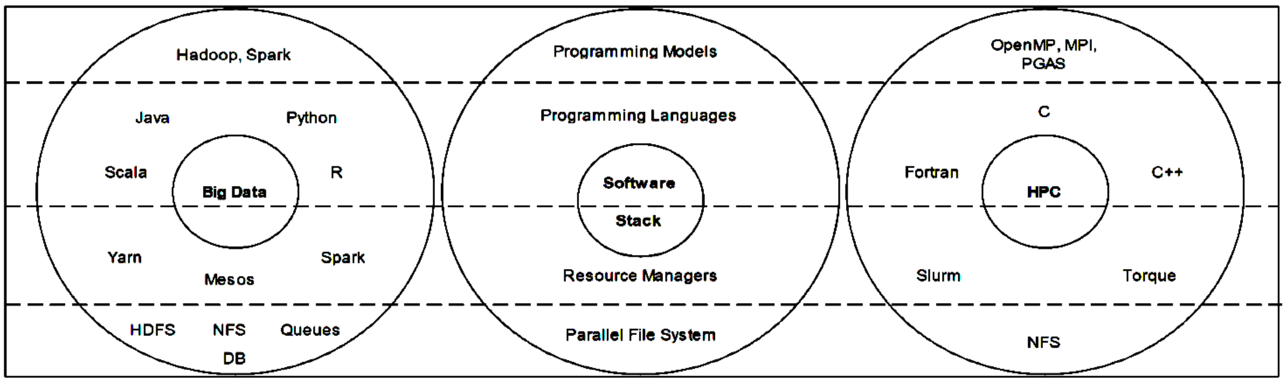

Figure 2. HPC and Big Data Software Stacks.

4.1 Optimum performance

Performance is an important metric used to evaluate software systems. Optimality of system performance varies from one system to another based on system requirements, software packages used, and the underlying hardware infrastructure supporting the computation. All the techniques presented in Sections 2 and 3 were designed to enhance performance explicitly or implicitly based on relevant influencing factors.

Many big data platforms, such as MapReduce and Spark, have the ability to support data locality by sending processes near the location of the data to be processed. However, such platforms are written in high-level programming languages, which do not provide the same level of support for parallelism as dedicated parallel programming models. This consequently impacts performance when running on HPC clusters.

Additionally, the enormous volume of big data may hinder parallel programming models from maximizing parallelism. In this sense, it is necessary to use effective domain decomposition to tackle this concern. Granularity of decomposition can play a vital role with options ranging from fine-grain to coarse-grain. Choice of granularity can affect performance with fine-grain granularity encouraging high parallelism by decomposing the data into small chunks and evenly distributing them among processors, which facilitates load balancing. It would seem that performance could be enhanced by employing a large number of processors, but this leads to communication and synchronization overhead, and inevitably consumes power. This kind of parallelism is appropriate for fast communication architectures like shared memory. In contrast, coarse-grained granularity splits the data into large tasks and places them in processors. This approach employs a lesser number of processors that results in some processors working on some tasks with other processors being idle, resulting in load imbalance. However, this approach has low communication and synchronization overhead, which is suitable for MPI based architectures. Medium-grained granularity supports parallelism as a compromise between both fine-grained and coarse-grained parallelism. Accordingly, it can be deduced that to achieve optimal performance, all of these factors should be employed collectively based on the given application and dataset.

There is, ordinarily, a positive correlation between performance and energy consumption: increasing performance increases the amount of power consumed. The problem of sacrificing power consumption as a result of increased performance can be solved by using power saving strategies such as race to halt [128]. This strategy aims to employ maximum system speed to create long intervals of idle time during which power consumption can be minimized.

Parallel programming can be achieved by implementing parallel programming models, whether for CPUs and GPU accelerators. By comparing these programming models, via the features in Table 3, it can be summarised that OpenMP, OpenAcc and OpenCL re-enforce portability. Moreover, it can be claimed that Cuda and OpenCL make a preference for control of the low-level system. It can be argued that OpenAcc can have the same performance, which makes it a more productive model [109, 128]. Trends are now targeting exascale computing and there is a need to employ hybrid models (dual or tri) to fulfill the requirements of the exascale future.

Integrating parallel and big data programming models such as MPI with Spark [14] can enhance the performance and get the best from both worlds. However, this will work for specific scenarios and cannot be considered a general purpose solution that can be applied as an independent big data platform.

4.2 Resource utilization

Exploiting HPC resources is highly dependent on providing metadata of the capabilities and availabilities of these resources. Although this paper has already mentioned static techniques to explore hardware topology, like hwloc [75], such techniques have to be dynamic since resource monitoring and dynamic updates can play a very critical role for supporting data locality, load balancing and decomposition from a scheduling perspective. The availability of resources and system architecture can determine the decomposition paradigm and granularity options that support efficient resource and job management system.

In this context, [129] investigated the efficiency of HPC resource allocation using Google cluster datasets and various data mining tools to determine the correlational coefficient between resource allocation, resource usage and priority. Initial analysis focused on the correlation between resource allocation and resource use. The results show that a high proportion of the resources allocated by the system for a job did not get used by that job. Furthermore, it was identified, using clustering, classification and prediction techniques, that there is a very loose correlation between the priority of the jobs with the allocation and use of resources. This research finding emphasizes the need to improve current HPC scheduling and resource allocation techniques in order to enhance the performance of big data applications and to efficiently accommodate its challenges.

As far as we know, and based on the review of prior work, there is a necessity to establish a research roadmap that considers all the aforementioned factors in the context of resource and job management within HPC environments for big data applications. This trend stems from the performance gap between big data platforms and high-performance computing due to different software stacks and architecture designs for each domain. This disparity leads to the fact that these platforms do not fully exploit the performance available in HPC clusters. As a result, improving the performance of big data applications and optimizing HPC resource utilization without sacrificing the power consumption of HPC is still a key challenge.

5 CONCLUSION

In this survey paper, various HPC techniques based on the most important factors have been studied and compared in the domain of big data and scientific workflow. All of the aforementioned factors can play a critical role in determining the efficiency of any HPC-based system used to support big data applications. Each technique has its own benefit and drawback. We have studied the techniques of load balancing and data locality, job scheduling strategies, topology aware, and data decomposition. Furthermore, different features of parallel programming models have been compared, highlighting a performance gap in comparison with some big data programming models. While each method is suitable for a given scenario and datasets, it cannot be generalized for all kinds of parallelization. Accordingly, it is desirable to customize each method for the given application, datasets, and the underlying architecture.

We have discussed how resource allocation is a critical part in the case of high-performance computing as well as big data. In both cases, a separate resource manager is generally needed to manage the resources and to ensure that the resources are loaded optimally to get the best performance out of the clustered environment.

In future work, we will consider all of the previous factors for employing a hybrid parallel programming model to build a high-level scheduling technique to enhance the performance of big data applications and exploit the availability of HPC resources without sacrificing the power consumption of HPC.

REFERENCES

[1] Y. Georgiou and M. Hautreux, “Evaluating Scalability and Efficiency of the Resource and Job Management System on Large HPC Clusters,” Springer, Berlin, Heidelberg, 2013, pp. 134–156.

[2] E. Jeannot, E. Meneses, G. Mercier, F. Tessier, and G. Zheng, “Communication and topology-aware load balancing in Charm++ with TreeMatch,” in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, 2013, pp. 1–8.

[3] H. Luu *et al.*, “A Multiplatform Study of I/O Behavior on Petascale Supercomputers,” *Proc. 24th*

Int. Symp. High-Performance Parallel Distrib. Comput. - HPDC '15, pp. 33–44, 2015.

[4] Z. Zhou *et al.*, "I/O-aware batch scheduling for petascale computing systems," *Proc. - IEEE Int. Conf. Clust. Comput. ICCC*, vol. 2015–Octob, pp. 254–263, 2015.

[5] N. Capit *et al.*, "A batch scheduler with high level components," in *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, 2005, p. 776–783 Vol. 2.

[6] E. Jeannot and G. Mercier, "Near-Optimal Placement of MPI Processes on Hierarchical NUMA Architectures," 2010, pp. 199–210.

[7] Z. Tan, L. Du, D. Feng, and W. Zhou, "EML: An I/O scheduling algorithm in large-scale-application environments," *Futur. Gener. Comput. Syst.*, vol. 78, pp. 1091–1100, 2018.

[8] R. Ross, R. T.-P. of the 4th annual L. showcase, and undefined 2000, "PVFS: A parallel file system for Linux clusters," *usenix.org*.

[9] X. Zhang, S. Jiang, and K. Davis, "Making resonance a common case: A high-performance implementation of collective I/O on parallel file systems," in *2009 IEEE International Symposium on Parallel & Distributed Processing*, 2009, pp. 1–12.

[10] S. Bergamaschi, C. Cavazzoni, A. Curioni, and G. Fox, "New Opportunities in High-Performance Data Analytics (HPDA) and High-Performance Computing (HPC)," *ieeexplore.ieee.org*.

[11] K. M. Tolle, D. S. W. Tansley, and A. J. G. Hey, "The Fourth Paradigm: Data-intensive scientific discovery," *Proc. IEEE*, vol. 99, no. 8, pp. 1334–1337, 2011.

[12] S. Fiore, M. Bakhouya, and W. W. Smari, "On the road to exascale: Advances in High-Performance Computing and Simulations—An overview and editorial," *Futur. Gener. Comput. Syst.*, vol. 82, pp. 450–458, 2018.

[13] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Commun. ACM*, vol. 58, no. 7, pp. 56–68, 2015.

[14] M. Anderson *et al.*, "Bridging the gap between HPC and big data frameworks," *Proc. VLDB Endow.*, vol. 10, no. 8, pp. 901–912, 2017.

[15] P. Xuan, J. Denton, P. K. Srimani, R. Ge, and F. Luo, "Big data analytics on traditional HPC infrastructure using two-level storage," *Proc. 2015 Int. Work. Data-Intensive Scalable Comput. Syst. - DISCS '15*, pp. 1–8, 2015.

[16] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mob. Networks Appl.*, vol. 19, no. 2, pp. 171–209, 2014.

[17] K. Wang, X. Zhou, T. Li, D. Zhao, M. Lang, and I. Raicu, "Optimizing load balancing and data-locality with data-aware scheduling," *Proc. - 2014 IEEE Int. Conf. Big Data, IEEE Big Data 2014*, pp. 119–128, 2015.

[18] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing - SC '07*, 2007, p. 1.

[19] I. Raicu, Y. Zhao, I. T. Foster, and A. Szalay, "Accelerating large-scale data exploration through data diffusion," in *Proceedings of the 2008 international workshop on Data-aware distributed computing - DADC '08*, 2008, pp. 9–18.

[20] G. Zheng, E. Meneses, A. Bhatele, and L. V. Kale, "Hierarchical Load Balancing for Charm++ Applications on Large Supercomputers," in *2010 39th International Conference on Parallel Processing Workshops*, 2010, pp. 436–444.

[21] B. Hindman, A. Konwinski, A. Platform, F.-G. Resource, and M. Zaharia, "Mesos: A platform for fine-grained resource sharing in the data center," *Proc. ...*, p. 32, 2011.

[22] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles - SOSP '09*, 2009, p. 261.

[23] M. Isard *et al.*, "Dryad," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 - EuroSys '07*, 2007, vol. 41, no. 3, p. 59.

[24] BIALECKI and A., "Hadoop : a framework for running applications on large clusters built of commodity hardware," *http://lucene.apache.org/hadoop*, 2005.

[25] I. Sadooghi *et al.*, "Achieving Efficient Distributed Scheduling with Message Queues in the Cloud for Many-Task Computing and High-Performance Computing," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2014, pp. 404–413.

[26] "What is Amazon Simple Queue Service? - Amazon Simple Queue Service." [Online]. Available: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html. [Accessed: 12-Aug-2018].

[27] G. DeCandia *et al.*, "Dynamo," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, p. 205, Oct. 2007.

[28] Y. Guo *et al.*, "SLAW," in *Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming - PPoPP '10*, 2010, vol. 45, no. 5, p. 341.

[29] Yi Guo, R. Barik, R. Raman, and V. Sarkar, "Work-first and help-first scheduling policies for async-finish task parallelism," in *2009 IEEE International Symposium on Parallel & Distributed Processing*, 2009, pp. 1–12.

[30] J. Paudel, O. Tardieu, and J. N. Amaral, "On the Merits of Distributed Work-Stealing on Selective Locality-Aware Tasks," in *2013 42nd International Conference on Parallel Processing*, 2013, pp. 100–109.

[31] J. Yin, A. Foran, and J. Wang, "DL-MPI: Enabling data locality computation for MPI-based data-intensive applications," *Proc. - 2013 IEEE Int. Conf. Big Data, Big Data 2013*, pp. 506–511, 2013.

[32] L. Huang, H. Jin, L. Yi, and B. Chapman, "Enabling locality-aware computations in OpenMP," *Sci. Program.*, vol. 18, no. 3–4, pp. 169–181, 2010.

[33] C. Pousa Ribeiro, M. Castro, J. F. Méhaut, and A. Carissimi, "Improving memory affinity of geophysics applications on NUMA platforms using minas," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6449 LNCS, pp. 279–292, 2011.

[34] Z. Majo and T. R. Gross, "Matching memory access

patterns and data placement for NUMA systems," *10th Int. Symp. Code Gener. Optim. CGO 2012*, pp. 230–241, 2012.

[35] D. S. Nikolopoulos, T. S. Papatheodorou, C. D. Polychronopoulos, J. Labarta, and E. Ayguadé, "Is Data Distribution Necessary in OpenMP?," *ACM/IEEE Int. Conf. High Perform. Comput. Networking, Storage Anal.*, 2000.

[36] N. Furmento, B. Goglin, and R. Namyst, "Dynamic Task and Data Placement over NUMA Architectures : an OpenMP Runtime Perspective e Wacrenier To cite this version : Dynamic Task and Data Placement over NUMA Architectures : an OpenMP Runtime Perspective," 2009.

[37] R. M. Yoo, C. J. Hughes, C. Kim, Y.-K. Chen, and C. Kozyrakis, "Locality-aware Task Management for Unstructured Parallelism: A Quantitative Limit Study," *Proc. 25th ACM Symp. Parallelism algorithms Archit.*, pp. 315–325, 2013.

[38] B. Vikranth, R. Wankar, and C. Raghavendra Rao, "Topology aware task stealing for on-chip NUMA multi-core processors," *Procedia Comput. Sci.*, vol. 18, pp. 379–388, 2013.

[39] J. Zhou, B. Demsky, J. Zhou, and B. Demsky, "Memory management for many-core processors with software configurable locality policies," in *Proceedings of the 2012 international symposium on Memory Management - ISMM '12,* 2012, vol. 47, no. 11, p. 3.

[40] Q. Lu *et al.*, "Data layout transformation for enhancing data locality on NUCA chip multiprocessors," *Parallel Archit. Compil. Tech. - Conf. Proceedings, PACT*, pp. 348–357, 2009.

[41] A. Marongiu and L. Benini, "An OpenMP Compiler for Efficient Use of Distributed Scratchpad Memory in MPSoCs," *IEEE Trans. Comput.*, vol. 61, no. 2, pp. 222–236, Feb. 2012.

[42] Y. Li, A. Abousamra, R. Melhem, and A. K. Jones, "Compiler-assisted Data Distribution for Chip Multiprocessors," *Parallel Archit. Compil. Tech.*, vol. 23, no. 11, pp. 501–512, 2010.

[43] Y. Li, R. G. Melhem, and A. K. Jones, "Practically Private: Enabling High Performance CMPs through Compiler-Assisted Data Classification," *21st Int'l Conf. Parallel Archit. Compil. Tech.*, pp. 231–240, 2012.

[44] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: Near-Optimal Block Placement and Replication in Distributed Caches," *Proc. Int. Symp. Comput. Archit.*, no. June, pp. 184–195, 2009.

[45] A. Tousimojarad and W. Vanderbauwhede, "A Parallel Task-based Approach to Linear Algebra," 2014.

[46] B. A. I. Relations, "for High-Performance Parallel Systems," 2016.

[47] W. Tang, D. Ren, Z. Lan, and N. Desai, "Adaptive Metric-Aware Job Scheduling for Production Supercomputers," in *2012 41st International Conference on Parallel Processing Workshops*, 2012, pp. 107–115.

[48] S. Khan, "A Self-adaptive Weighted Sum Technique for the Joint Optimization of Performance and Power Consumption in Data Centers.," *Isca Pdccs*, pp. 13–18, 2009.

[49] S. Khan, C. A.-I. J. of Electrical, undefined Computer, undefined and, and undefined 2009, "A weighted sum technique for the joint optimization of performance and power consumption in data centers," *Citeseer*.

[50] S. U. Khan and I. Ahmad, "Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, 2006, p. 10 pp.

[51] S. U. Khan, "A goal programming approach for the joint optimization of energy consumption and response time in computational grids," in *2009 IEEE 28th International Performance Computing and Communications Conference*, 2009, pp. 410–417.

[52] T. D. Braun *et al.*, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, Jun. 2001.

[53] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo, "A comparative study of online scheduling algorithms for networks of workstations," *Cluster Comput.*, vol. 3, no. 2, pp. 95–112, 2000.

[54] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Scalable scheduling for sub-second parallel jobs*2013*",.

[55] V. K. Vavilapalli *et al.*, "Apache Hadoop YARN," in *Proceedings of the 4th annual Symposium on Cloud Computing - SOCC '13*, 2013, pp. 1–16.

[56] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega," in *Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys '13*, 2013, p. 351.

[57] Apache Hadoop Project, "Hadoop on demand," 2008. [Online]. Available: https://svn.apache.org/repos/asf/hadoop/common/tags/release-0.17.1/docs/hod.html.

[58] SAGA, "SAGA-Hadoop," 2014. [Online]. Available: https://github.com/drelu/saga-hadoop.

[59] MyHadoop, "myhadoop," 2013. [Online]. Available: https://portal.futuregrid.org/tutorials/%0Arunning-hadoop-batch-job-using-myhadoop.

[60] Amazon Web Services, "Elastic Map Reduce Service," 2013. [Online]. Available: https://aws.amazon.com/emr/.

[61] Microsoft Azure, "HDInsight Service," 2013. [Online]. Available: https://azure.microsoft.com/en-us/services/hdinsight/.

[62] "Llama," 2013. [Online]. Available: http://cloudera.github.io/llama.

[63] "Apache TEZ," 2014. [Online]. Available: https://hortonworks.com/apache/tez/.

[64] W. C. Moody, L. B. Ngo, E. Duffy, and A. Apon, "JUMMP: Job Uninterrupted Maneuverable MapReduce Platform," in *2013 IEEE International Conference on Cluster Computing (CLUSTER),* 2013, pp. 1–8.

[65] O. Kulkarni, "Hadoop mapreduce over lustre–high performance data division," 2013.

[66] P. Zikopoulos et al., *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. 2011.

[67] R. H. Castain, O. Kulkarni, and X. Zhenyu, “MapReduce and Running Hadoop in a High Performance Computing Environment Lustre : Agenda,” *Lustre User Gr. 2013 China Japan*, 2013.

[68] C. Acevedo, P. Hernández, A. Espinosa, and V. Méndez, "A Critical Path File Location (CPFL) algorithm for data-aware multi-workflow scheduling on HPC clusters," *Futur. Gener. Comput. Syst.*, vol. 74, pp. 51–62, 2017.

[69] M. Mercier, D. Glesser, Y. Georgiou, and O. Richard, “Big Data and HPC collocation : Using HPC idle resources for Big Data Analytics,” pp. 347–352, 2017.

[70] M. Wasi-ur-Rahman, X. Lu, N. S. Islam, R. Rajachandrasekar, and D. K. Panda, “High-Performance Design of YARN MapReduce on Modern HPC Clusters with Lustre and RDMA,” in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 291–300.

[71] J. L. Reyes-Ortiz, L. Oneto, and D. Anguita, “Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf,” *Procedia Comput. Sci.*, vol. 53, pp. 121–130, Jan. 2015.

[72] S. Krishnan and D. Ph, “myHadoop 0 . 2a : Hadoop-on-demand on Traditional HPC Resources,” *Lustre*, pp. 0–3, 2011.

[73] M. V. Neves, T. Ferreto, and C. De Rose, “Scheduling MapReduce Jobs in HPC Clusters,” Springer, Berlin, Heidelberg, 2012, pp. 179–190.

[74] A. Luckow, I. Paraskevakos, G. Chantzialexiou, and S. Jha, “Hadoop on HPC: Integrating Hadoop and Pilot-Based Dynamic Resource Management,” in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 1607–1616.

[75] B. Goglin, “Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc),” *Proc. 2014 Int. Conf. High Perform. Comput. Simulation, HPCS 2014*, pp. 74–81, 2014.

[76] Chuang Liu, Lingyun Yang, I. Foster, and D. Angulo, “Design and evaluation of a resource selection framework for Grid applications,” in *Proceedings 11th IEEE International Symposium on High Performance Distributed Computing*, pp. 63–72.

[77] O. Sonmez, H. Mohamed, and D. Epema, “Communication-Aware Job Placement Policies for the KOALA Grid Scheduler,” in *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science’06)*, 2006, pp. 79–79.

[78] A. Sahai and S. F. Wu, *Utility computing : 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2004, Davis, CA, USA, November 15-17, 2004 : proceedings*. Springer, 2004.

[79] G. Lee, N. Tolia, P. Ranganathan, and R. H. Katz, “Topology-aware resource allocation for data-intensive workloads,” in *Proceedings of the first ACM asia-pacific workshop on Workshop on systems - APSys ’10*, 2010, p. 1.

[80] A. Bhatelé, E. Bohm, and L. V. Kalé, “Topology aware task mapping techniques,” *ACM SIGPLAN Not.*, vol. 44, no. 4, p. 301, Feb. 2009.

[81] J. Navaridas, J. Miguel-Alonso, F. J. Ridruejo, and W. Denzel, “Reducing complexity in tree-like computer interconnection networks,” *Parallel Comput.*, vol. 36, no. 2–3, pp. 71–85, Feb. 2010.

[82] R. Raman, M. Livny, and M. Solomon, “Matchmaking: distributed resource management for high throughput computing,” in *Proceedings. The Seventh International Symposium on High Performance Distributed Computing (Cat. No.98TB100244)*, pp. 140–146.

[83] A. B. Yoo, M. A. Jette, and M. Grondona, “SLURM: Simple Linux Utility for Resource Management,” Springer, Berlin, Heidelberg, 2003, pp. 44–60.

[84] PBSWorks, “PBS Professional.” [Online]. Available: https://www.pbsworks.com/PBSProduct.aspx?n=PBS-Professional&c=Overview-and-Capabilities.

[85] C. Smith, B. Mcmillan, and I. Lumb, “Topology Aware Scheduling in the LSF Distributed Resource Manager,” *Proc. Cray User Gr. Meet.*, 2001.

[86] T. Mikamo, “Technical Computing Suite Job Management Software,” 2012.

[87] Adaptive computing, “Torque resource manager.” [Online]. Available: http://docs.adaptivecomputing.com/torque/6-0-0/Content/topics/torque/2-jobs/monitoringJobs.htm. [Accessed: 14-Aug-2018].

[88] H. Vandierendonck and T. Mens, “Averting the Next Software Crisis,” *Computer (Long. Beach. Calif).*, vol. 44, no. 4, pp. 88–90, Apr. 2011.

[89] A. Meade, J. Buckley, and J. J. Collins, “Challenges of evolving sequential to parallel code,” in *Proceedings of the 12th international workshop and the 7th annual ERCIM workshop on Principles on software evolution and software evolution - IWPSE-EVOL ’11*, 2011, p. 1.

[90] B. L. Massingill, T. G. Mattson, and B. A. Sanders, “Reengineering for Parallelism: an entry point into PLPP for legacy applications,” *Concurr. Comput. Pract. Exp.*, vol. 19, no. 4, pp. 503–529, Mar. 2007.

[91] V. Pankratius, C. Schaefer, A. Jannesari, and W. F. Tichy, “Software engineering for multicore systems,” in *Proceedings of the 1st international workshop on Multicore software engineering - IWMSE ’08*, 2008, p. 53.

[92] R. Eccles and D. A. Stacey, “Understanding the Parallel Programmer,” in *20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS’06)*, 2006, pp. 12–12.

[93] V. R. Basili *et al.*, “Understanding the High-Performance-Computing Community: A Software Engineer’s Perspective,” *IEEE Softw.*, vol. 25, no. 4, pp. 29–36, Jul. 2008.

[94] L. Hochstein and V. R. Basili, “The ASC-Alliance Projects: A Case Study of Large-Scale Parallel Scientific Code Development,” *Computer (Long. Beach. Calif).*, vol. 41, no. 3, pp. 50–58, Mar. 2008.

[95] A. Meade, D. K. Deeptimahanti, J. Buckley, and J. J. Collins, “An empirical study of data decomposition for software parallelization,” *J. Syst. Softw.*, vol. 125, pp. 401–416, Mar. 2017.

[96] S. Takizawa, M. Matsuda, N. Maruyama, and Y. Nakamura, “A Scalable Multi-Granular Data Model for Data Parallel Workflows,” *Proc. Int. Conf. High Perform. Comput. Asia-Pacific Reg. - HPC Asia 2018*, pp. 251–260, 2018.

[97] D. B. Loveman, "High Performance Fortran," *IEEE Parallel Distrib. Technol. Syst. Appl.*, vol. 1, no. 1, pp. 25–42, Feb. 1993.

[98] C. Coarfa *et al.*, “An evaluation of global address space languages,” in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming - PPoPP ’05*, 2005, p. 36.

[99] Y. Zheng, A. Kamil, M. B. Driscoll, H. Shan, and K. Yelick, “UPC++: A PGAS Extension for C++,” in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, pp. 1105–1114.

[100] P. Charles *et al.*, “X10,” in *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming systems languages and applications - OOPSLA '05*, 2005, vol. 40, no. 10, p. 519.

[101] B. L. Chamberlain, S. J. Deitz, D. Iten, and S.-E. Choi, “User-defined distributions and layouts in chapel: philosophy and framework,” *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*. USENIX Association, pp. 12–12, 2010.

[102] G. Bikshandi *et al.*, “Programming for parallelism and locality with hierarchically tiled arrays,” in *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming - PPoPP ’06*, 2006, p. 48.

[103] V. Cavé, J. Zhao, J. Shirako, and V. Sarkar, “Habanero-Java,” in *Proceedings of the 9th International Conference on Principles and Practice of Programming in Java - PPPJ ’11*, 2011, p. 51.

[104] S. Memeti, L. Li, S. Pllana, J. Kolodziej, and C. Kessler, “Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: programming productivity, performance, and energy consumption,” pp. 1–14, 2017.

[105] OpenMP, “OpenMP 4.0 Specifications.” [Online]. Available: https://www.openmp.org/specifications/. [Accessed: 14-Aug-2018].

[106] S. Wienke, P. Springer, C. Terboven, and D. an Mey, “OpenACC — First Experiences with Real-World Applications,” Springer, Berlin, Heidelberg, 2012, pp. 859–870.

[107] J. E. Stone, D. Gohara, and G. Shi, “OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems,” *Comput. Sci. Eng.*, vol. 12, no. 3, pp. 66–73, May 2010.

[108] NVIDIA, “CUDA C Programming Guide.”

[109] and M. W. Y. Yan, B. M. Chapman, “A Comparison of Heterogeneous and Manycore Programming Models,” 2015. [Online]. Available: https://www.hpcwire.com/2015/03/02/a-comparison-of-heterogeneous-and-manycore-programming-models/. [Accessed: 14-Aug-2018].

[110] D. Li, X. Ji, Z. Zhou, and Q. Wang, “A Hybrid MPI/OpenMP Model Based on DDM for Large-Scale Partial Differential Equations,” in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012, pp. 1839–1843.

[111] H. Jin, D. Jespersen, P. Mehrotra, L. Huang, and B. Chapman, “High performance computing using MPI and OpenMP on multi-core parallel systems,” *Parallel Comput.*, vol. 37, no. 9, pp. 562–575, Sep. 2011.

[112] P. D. Mininni, D. Rosenberg, R. Reddy, and A. Pouquet, “A hybrid MPI–OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence,” *Parallel Comput.*, vol. 37, no. 6–7, pp. 316–326, Jun. 2011.

[113] D. Kirk and W. Hwu, *Programming massively parallel processors : a hands-on approach*. Morgan Kaufmann Publishers, 2010.

[114] S. Jha, J. Qiu, A. Luckow, P. Mantha, and G. C. Fox, “A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures,” in *2014 IEEE International Congress on Big Data*, 2014, pp. 645–652.

[115] N. Satish *et al.*, “Navigating the maze of graph analytics frameworks using massive graph datasets,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD ’14*, 2014, pp. 979–990.

[116] G. M. Slota, S. Rajamanickam, and K. Madduri, “A Case Study of Complex Graph Analysis in Distributed Memory: Implementation and Optimization,” in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 293–302.

[117] A. Gittens *et al.*, “Matrix factorizations at scale: A comparison of scientific data analytics in spark and C+MPI using three case studies,” in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 204–213.

[118] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun, “Making Sense of Performance in Data Analytics Frameworks,” *12th USENIX Symp. Networked Syst. Des. Implementation (NSDI 2015)*, pp. 293–307, 2015.

[119] X. Lu, F. Liang, B. Wang, L. Zha, and Z. Xu, “DataMPI: Extending MPI to Hadoop-Like Big Data Computing,” in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, pp. 829–838.

[120] A. Raveendran, T. Bicer, and G. Agrawal, “A Framework for Elastic Execution of Existing MPI Programs,” in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, 2011, pp. 940–947.

[121] G. E. Fagg and J. J. Dongarra, “FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World,” Springer, Berlin, Heidelberg, 2000, pp. 346–353.

[122] M. Gamell, D. S. Katz, H. Kolla, J. Chen, S. Klasky, and M. Parashar, “Exploring Automatic, Online Failure Recovery for Scientific Applications at Extreme Scales,” in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 895–906.

[123] “H2O.ai. Sparkling Water.” [Online]. Available: https://github.com/h2oai/sparkling-water. [Accessed: 14-Aug-2018].

[124] “deeplearning4j. Deep Learning for Java. Open-Source, Distributed, Deep Learning Library for the JVM.” [Online]. Available: https://deeplearning4j.org/. [Accessed: 14-Aug-2018].

[125] M. Grossman and V. Sarkar, “SWAT,” in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing - HPDC ’16*, 2016, pp. 81–92.

[126] S. Caíno-Lores, A. Lapin, J. Carretero, and P. Kropf, “Applying big data paradigms to a large scale scientific workflow: Lessons learned and future directions,” *Futur. Gener. Comput. Syst.*, pp. 1–13, 2018.

[127] H. R. Asaadi, D. Khaldi, and B. Chapman, “A comparative survey of the HPC and big data paradigms: Analysis and experiments,” *Proc. - IEEE Int. Conf. Clust. Comput. ICCC*, pp. 423–432, 2016.

[128] W. Wang, “Performance, power, and energy tuning using hardware and software techniques for modern parallel architectures,” 2016.

[129] B. R. Ray, M. Chowdhury, and U. Atif, “Future Network Systems and Security,” vol. 759, pp. 97–112, 2017.