electronics

Review

Data Locality in High Performance Computing, Big Data, and Converged Systems: An Analysis of the Cutting Edge and a Future System Architecture

Sardar Usman [1], Rashid Mehmood [2,*], Iyad Katib [3] and Aiiad Albeshri [3]

[1] Department of Computer Science, Grand Asian University, Sialkot 51060, Pakistan
[2] High-Performance Computing Center, King Abdulaziz University, Jeddah 21589, Saudi Arabia
[3] Department of Computer Science, FCIT, King Abdulaziz University, Jeddah 21589, Saudi Arabia
* Correspondence: rmehmood@kau.edu.sa

Abstract: Big data has revolutionized science and technology leading to the transformation of our societies. High-performance computing (HPC) provides the necessary computational power for big data analysis using artificial intelligence and methods. Traditionally, HPC and big data had focused on different problem domains and had grown into two different ecosystems. Efforts have been underway for the last few years on bringing the best of both paradigms into HPC and big converged architectures. Designing HPC and big data converged systems is a hard task requiring careful placement of data, analytics, and other computational tasks such that the desired performance is achieved with the least amount of resources. Energy efficiency has become the biggest hurdle in the realization of HPC, big data, and converged systems capable of delivering exascale and beyond performance. Data locality is a key parameter of HPDA system design as moving even a byte costs heavily both in time and energy with an increase in the size of the system. Performance in terms of time and energy are the most important factors for users, particularly energy, due to it being the major hurdle in high-performance system design and the increasing focus on green energy systems due to environmental sustainability. Data locality is a broad term that encapsulates different aspects including bringing computations to data, minimizing data movement by efficient exploitation of cache hierarchies, reducing intra- and inter-node communications, locality-aware process and thread mapping, and in situ and transit data analysis. This paper provides an extensive review of cutting-edge research on data locality in HPC, big data, and converged systems. We review the literature on data locality in HPC, big data, and converged environments and discuss challenges, opportunities, and future directions. Subsequently, using the knowledge gained from this extensive review, we propose a system architecture for future HPC and big data converged systems. To the best of our knowledge, there is no such review on data locality in converged HPC and big data systems.

Keywords: High-performance computing (HPC); big data; High-Performance Data Analytics (HPDS); convergence; data locality; Spark; Hadoop; design patterns; process mapping; in situ data analysis

Citation: Usman, S.; Mehmood, R.; Katib, I.; Albeshri, A. Data Locality in High Performance Computing, Big Data, and Converged Systems: An Analysis of the Cutting Edge and a Future System Architecture. *Electronics* **2023**, *12*, 53. https://doi.org/10.3390/electronics12010053

Academic Editor: Antonio Brogi

Received: 8 November 2022
Revised: 8 December 2022
Accepted: 19 December 2022
Published: 23 December 2022

1. Introduction

Data has grown exponentially during the last decade giving rise to the big data phenomenon [1,2]. Big data has revolutionized science and technology, leading to innovations in many sectors including urbanization [3], transport [4], energy [5], healthcare [6], education [7], economics [8], smart societies [9], computing infrastructure [10], and more; see, for example, [1,11], for details on big data technologies and applications. The paramount contribution of big data is the development of contemporary data-driven machine and deep learning and artificial intelligence (AI) technologies that have transformed our societies and infrastructure [12,13]. This has also given rise to the need for developing green AI approaches, a broad term that incorporates properties including energy efficiency, responsibility, fairness, etc. [14–17].

High-performance computing (HPC) provides the necessary computational power for big data analysis using machine learning and other artificial intelligence methods [18]. HPC has traditionally focused on compute-intensive simulations of natural phenomena, engineering design of static and dynamic objects (bridges, vehicles, etc.), and other scientific and engineering problems on high-end tightly coupled supercomputing systems. Big data technologies have grown to use relatively loosely coupled and inexpensive computer systems. HPC is good at compute-intensive tasks while big data systems have better performance for data-driven tasks. The last five or so years have seen developments of systems that integrate the advantages of both big data and HPC systems by converging the two approaches to system design [19–26].

Designing HPC and big data converged systems, also referred to as High-Performance Data Analytics (HPDA), is a hard task requiring careful placement of data, analytics, and other computational tasks such that the desired performance is achieved with the least amount of resources. Energy efficiency has become the biggest hurdle in the realization of HPC and HPDA systems capable of delivering exascale and beyond performance. In large-scale systems comprising millions of cores and thousands of nodes aiming to provide exascale performance is an arduous exercise. The classical Dennard scaling has stopped for more than a decade now as the scaling of a single processing core has ceased and performance scaling is achieved by mounting more cores on chips and exploiting explicit parallelism. The complexity of managing parallelism increases with increasing memory hierarchies from system and node levels to the processing unit level [27]. It becomes even more challenging in loosely coupled systems where nodes are typically geographically distributed with many uncertainties including network quality [10,28,29].

Locality, or how to improve data access and transfer, within the application, is one of the most significant challenges that will need to be addressed in the upcoming years. Addressing data locality issues in simple terms means narrowing the distance between data and processing for better performance. One problem relating to locality comes from the memory and the network: the affinity and location of processes within an application affect how quickly data is transferred between them. The cost of data movement has been under the scanner of researchers for years but now has gained momentum, as performance and energy consumption are heavily dependent on data locality. Researchers have realized that scalability cannot be addressed only by powerful infrastructure but is constrained by resource utilization and energy efficiency [27].

Locality refers to a phenomenon in which computations do not have a uniform or independent access to data but rather have clustered, dependent, co-related access [30]. The Principle of Locality or Locality of References can be categorized as temporal locality and spatial locality. Temporal locality aims to reuse data as much as possible once it has been brought in and spatial locality aims to use every data element that has been brought in [31]. Data Locality can be defined as bringing computations to the place (processor, cache, etc.) or nodes where the data to be processed actually resides. Data Movement is defined as the movement of data across cache hierarchies, inter- and intra-node data movement and, in the case of in situ data analysis, data movement mostly refers to data movement back and forth to persistent storage and retrieving data for post-data-analysis. In the rest of the paper, we refer to data locality as a means of bringing computation to data, minimizing data movement by efficient exploitation of cache hierarchies, reducing intra- and inter-node communications, locality-aware process and thread mapping, and in situ and in transit data analysis.

Data locality is a key parameter of HPDA system design as moving even a byte costs heavily both in time and energy with an increase in the size of the system. Performance in terms of time and energy are the most important factors for users, particularly energy, due to it being the major hurdle in high-performance system design and the increasing focus on green energy systems due to environmental sustainability.

This paper provides an extensive review of cutting-edge research on data locality in HPC, big data, and converged systems. To the best of our knowledge, there is no such review on data locality in converged HPC and big data systems.

In Section 2, we review earlier works on data locality surveys in HPC, big data, and converged systems and establish the case for this work. Section 3 reviews literature on data locality in HPC environments with topics covering applications perspective; programming languages, compiler, and libraries; cache optimization techniques; locality-aware scheduling and load-balancing; bulk synchronous processing (BSP); out-of-core computing; parallelism mapping; and in situ data analysis. Section 4 reviews works on data locality in HPC environments from perspectives including parallel programming models, data placement, scheduling, load balancing, and in-memory computations. Section 5 reviews literature on data locality in converged HPC and big data environments with topics covering MPI with map-reduce frameworks, map-reduce frameworks with high-performance interconnects, and map-reduce-like frameworks for in situ analysis. Section 6 discusses challenges, opportunities, and future directions covering programming paradigms, programming models and language support, programming abstractions, innovations in data layout strategies, locality-aware scheduling, software hardware co-design, and innovations in memory and storage technologies. In Section 7, we use the knowledge gained from this extensive review to propose a system architecture for future converged systems. Section 8 concludes the paper.

2. Related Works: Case for This Paper

This section reviews earlier works on data locality surveys in HPC (Section 2.1), big data (Section 2.2), and converged systems (Section 2.3) and establish the case for this work.

2.1. Big Data

Lores et al. [32] presented a survey of different techniques proposed to deal with data locality for high-performance and high throughput systems by categorizing different techniques into four major categories, i.e., application development, in-memory computing, task scheduling, and storage formats. A survey by Zhang et al. [33] focused on data processing and management strategies for in-memory computations. Dolev et al. [34] investigated different requirements and challenges in designing geographically distributed big data analysis frameworks and protocols by classifying and focusing on map-reduce-based systems, stream processing, SQL-style processing, geo-distributed frameworks, etc. Senthilkumar et al. [35] base their work on task scheduling in big data computations primarily focusing on scheduler classification, and algorithmic comparison with pros and cons, and also include various tools and frameworks for managing and enhancing the performance of map-reduce.

Idris et al. [36] presented a survey on context-aware scheduling in map-reduce frameworks. They classified scheduling techniques and algorithms and comparative analysis of these techniques. Mozakka et al. [37] presented a survey on adaptive job schedulers in map-reduce and discussed the benefits and drawbacks of different adaptive scheduling techniques. Nagina et al. [38] reviewed scheduling algorithms in big data. Sreedhar et al. [39] surveyed big data management and job scheduling. Akilandeswari et al. [40] presented a survey on task scheduling in cloud environments.

2.2. HPC

Hoefler et al. [41] presented an overview of topology mapping focusing on algorithmic strategies and mapping enforcement techniques, i.e., resource binding, rank reordering, etc. Unat et al. [27] presented a comprehensive survey of different trends in data locality abstractions available in the form of data structures, runtime systems, libraries, and languages and identified opportunities to combine different techniques to address data locality issues for future HPC systems. Singh et al. [42] presented a comprehensive survey on mapping methodologies by categorizing them as design time, runtime, on-the-fly, and

hybrid techniques and also provided upcoming trends, issues, and open challenges for many- and multicore systems.

2.3. HPC and Big Data Convergence

While the motivations and benefits for the convergence of HPC and big data have been noted in the literature, there are very few papers that have reviewed the literature on the convergence of big data and HPC. The most notable and earliest one (published in 2015) is by Reed and Dongara [18]. This is more of an agenda-setting article. Asaadi et al. [43] presented a data-supported comparative survey of HPC (MPI, PGAS, OpenMP) and big data (Spark, Hadoop, etc.) programming interfaces. They also conducted experiments on a series of benchmarks for comparison and discussed the potential benefits, issues, and challenges of convergence of HPC and big data. Jha et al. [44] presented a comparative analysis of HPC and big data paradigms by discussing their relevant features, functionalities, and common implementation characteristics. They also identified differences between HPC and big data software ecosystems and outlined some architectural similarities along with potential opportunities for the inevitable convergence of HPC and big data. [43,44] are both conference papers.

The authors of the technical report [45] reviewed convergence challenges and issues raised by the split between conventional HPDA and the explosive growth of data in recent years. They addressed and analyzed the application workflow level convergence challenges for widely distributed data resources, challenges imposed by converged infrastructure in edge environments (data flow between data centers and network edge and vice-versa), and opportunities for integrated centralized infrastructure. A survey of big data and HPC tracing from parallel programming models to clusters is provided in [46]. Golasowski et al. [47] discussed the convergence by reviewing four EU Horizon 2020 projects. Other discussion works on the convergence include [48–50], and a discussion article in the HPCwire magazine [2].

The works discussed above provide general discussions on HPC and big data convergence. To the best of our knowledge, there are no review papers on data locality in converged HPC and big data systems.

3. Data Locality in HPC Environments

HPC has been fundamental in developing many transformative applications [51–55]. These HPC applications require careful design of fundamental algorithms, e.g., the seven dwarfs [56–61]. Data locality has been coined as a key aspect among others and is regarded as one of the main challenges for exascale computing endorsed by many technical reports published in recent years [62–65]. The organization of memory into banks and NUMA regions, read-only memory, and multiple cache hierarchies make efficient data structure and optimization, a complex task. As we are heading towards exascale systems where the cost of data movement will be a dominant factor in terms of performance and energy efficiency. The complexities of managing data with different levels of memory hierarchies are further complicated by inter- and intra-node-level communication. Parallelism and communication are further constrained by heterogeneous core architecture. With the increase in platform heterogeneity, portability is a core issue that demands standardization of widely used approaches to automate performance tuning across different architectures [27].

The concurrency and input data size are on a rise and there is a need for efficient exploitation of heterogeneous architectures with an increasing number of cores to bridge a performance gap between application and target architecture. There is a need for optimization of data layout, data movement between processes/threads and data access patterns. Locality issues exist at different levels, from how application data is laid out to the increasing number of processing cores, complex memory hierarchies, inter-node communication, interconnects, and storage units [66]. The following section discusses the research related to data locality in the High-performance computing domain.

3.1. Application Perspectives

Data locality from an application's point of view demands the examining of a range of modeling methodologies, which needs exploration of a huge application space. The adaption of current HPC application code into exascale systems demands efficient utilization of heterogeneous resources, requiring innovations in application layout strategies, communication optimization, synchronization, resource availability, and data movement and access strategies.

3.2. Programming Languages, Compiler, and Libraries

The increasing level of hardware parallelism and deep hierarchies (core dies, chips, to node level) have posed a challenging task for efficient parallelism and data locality exploitation at all levels of hierarchy, which demands data locality support at different levels of the software ecosystem, i.e., programming language, compiler, libraries, etc. Majo et al. [67] proposed a parallel programming library for compose-able and portable data locality optimizations for NUMA systems. This library is based on Intel Threading Building Blocks (TBB) and allows the capturing of a programmer's insights for mapping tasks to available resources. Lezos et al. [68] presented a compiler-directed data locality optimization in MATLAB by developing a software tool, MemAssist, for efficient exploitation of targeted architecture and memory hierarchy. Regan-Kelley et al. [69] proposed a language and compiler support for optimizing parallelism, locality, and re-computation in image processing pipelines.

Current HPC models lack efficient exploitation of data locality due to the lack of language support for data locality policies e.g., array layouts, parallel scheduling, etc., an overexposure of target architecture, and a lack of support for user-defined policy abstractions. Chapel is a prominent parallel programming language developed by the joint efforts of Cray Inc., academia, and industry with a prime focus on the separation of parallelism and locality, and multi-resolution design with enhanced productivity features [70]. X10 [71] is based on a PGAS model and designed specifically for parallel computing that supports structured and unstructured parallelism, user-defined primitive struct types, and globally distributed arrays with co-location of execution and data. Huang et al. [72] addresses the data locality issues with explicit programming control of locality in the context of OpenMP and how it can be accomplished.

3.3. Cache Optimization Techniques

Locality optimizations by making efficient use of the cache, have been an active area of research for years. Locality can be categorized as temporal locality (reuse data once it has been brought into the memory) and spatial locality (use of every data element brought into the memory). The efficient exploitation of registers involves compiler, assembly-language, and programming-level optimizations. Cache line length, cache size, and cache replacement policy are some of the factors considered for the effective and efficient optimization of caches [33]. Gupta et al. [73] proposed a spatial locality-aware cache partitioning scheme by measuring spatial and temporal locality dynamically for optimal workload block size and capacity for effective cache sharing. Gonzalez et al. [74] proposed a dual data-cache organization for managing spatial and temporal locality by implementing a lazy cache policy that uses a locality prediction table to make necessary predictions based on recently used instructions. [75,76] related to on-chip caching for efficient cache utilization, while [77,78] based their approach on data-access frequency.

The following section explains the basic data access optimization techniques engineered to improve cache efficiency.

3.3.1. Data Access Optimization

Data access optimizations are generally code transformations whose prime motivation is to increase temporal locality by reordering iterations in a nested loop. These optimization techniques are also used to expose parallelism and help in vectorizing loop iterations.

Compilers used heuristics to decide the effectiveness of applying these transformations. If nested loops are not perfectly nested then loop skewing, unrolling, and peeling are used. Detailed information about these techniques can be found in [79,80].

Loop Skewing: When the carried dependencies prevent parallelizing, one can skew the loop-nest by modifying the aligning of iteration space coordinates and relabeling the statement instances in new coordinates [81].

Loop Peeling: Unfolding a few iterations of the loop to eliminate loop-carried dependencies is known as loop peeling.

Loop Interchange: When the order of a loop is not important then a loop-interchange transformation reverses the order of two adjacent loops in the loop-nest making sure all dependencies are preserved. Loop interchange is used to improve locality, enhance parallelism, register reuse, and vectorization [82].

Loop Fusion/Jamming: This technique involves the fusion of two adjacent loops having the same iteration space traversal into a single loop resulting in increased instruction level parallelism and data locality. The opposite of loop jamming is loop distribution, which divides a single loop into multiple loops with no carried dependencies. [82].

Loop Tiling: This loop transformation technique improves data locality by increasing the reuse of data in the cache by increasing the depth of the loop nest [83]. Selection of the best loop tile shape and size is a fundamental problem. Most of the current multicore processors have a shared last-level cache (LLC) and its space allocation depends on the co-execution of applications, which may cause interference in the shared cache [84]. Bao et al. [84] proposed a static compiler-based defensive tiling to choose the best tiling size for optimal performance. Some of the works related to loop tiling include [85,86] for the reduction of capacity misses, [87–89] for reducing communication, and [90–92] for auto/dynamic tuning of loop tiling.

3.3.2. Data Layout Optimizations

Data access optimization techniques may not be an optimal choice for data locality for computations with conflict misses, while data layout optimizations improve spatial locality by arranging data structure and variables in memory. Some of the most commonly used techniques for data locality optimization are inter- and intra-array padding, array merging, array transpose, etc. [82]. Parallelism and efficient exploitation of data locality have been considered separate objectives in the literature. Kennedy et al. [93] explored this trade-off between data locality and parallelization by proposing a memory model to determine the reuse of cache lines. The model uses loop optimization algorithms for efficient exploitation of data locality and in-memory optimizations.

3.3.3. Cache Bypassing

The depth of cache hierarchies and cache size have been increasing to meet the high processing demands, along with mounting more cores on the chip. The performance of an application with little data reuse is severely affected by cache use. Researchers over the years engineered different techniques to effectively bypass the cache to improve the performance of an application. The potential benefits of cache bypassing are obvious but bring many challenges including implementation overhead, memory and performance overhead, etc. The different cache techniques, their potential benefits, challenges, and taxonomy is explained in detail by Sparsh Mittal [94].

3.4. Locality-Aware Scheduling and Load-Balancing

Applications need to exploit parallelism at multiple scales at the fine granularity and across a variety of irregular program and data structures and program inputs. Parallel algorithms demand extra space to enable the temporal decoupling necessary for achieving parallelism, as compared to sequential algorithms, which attempt to minimize space usage. As applications are becoming more and more data-intensive and task execution involves the processing of a huge volume of data, optimal load balancing, and locality-aware

scheduling are critical issues to be resolved at the highest priority for exascale systems. Locality optimization (horizontal—between processing elements and vertical—between levels of hierarchy)) has a direct impact on energy consumption for exascale systems. Scheduling millions of tasks per second within latency constraints is one of the major challenges and current centralized scheduling systems (Falkon [95], SLURM [96], SGE [97], and Condor [98]) are deprived of handling fast scheduling. This issue is addressed by Ousterhout et al. [99] by presenting a stateless distributed scheduler called sparrow, which supports data-aware scheduling to help the collocating of task and input data.

Load balancing can be achieved by work stealing but the random migration of tasks results in poor data locality. Load balancing is a challenging task in a fully distributed environment as the scheduler has information about its own state. Load balancing techniques have been extensively studied over the years and can broadly be classified into static and dynamic techniques. These techniques achieve optimal load balancing in a centralized or distributed manner. Although work stealing is a widely used efficient load balancing technique e.g., OpenMP [100], Cilk [101], X10 [71], random work stealing results in poor scalability on large-scale systems [102]. Falt et al. [103] proposed a locality-aware task scheduling for parallel data stream systems.

Muddukrishna et al. [104] proposed a locality-aware task scheduling and runtime system-assisted data distribution algorithm for OpenMP tasks on NUMA systems and multicore processors. Ding et al. [105] proposed a cache hierarchy-aware loop–iterations–to–core mapping strategy by exploiting data reuse and minimizing data dependencies, which results in improved data locality. Lifflander et al. [106] proposed locality-aware optimization at different phases of fork/join programs with optimal load balance based on Cilk and also provides programmatic support for work-stealing schedules which helps in user guidance on data locality.

Xue et al. [107] proposed a hybrid locality-aware dynamic load balancing and locality-aware loop distribution strategy to multiprocessors and enhanced performance is reported compared to other static/dynamic scheduling techniques. Isard et al. [108] proposed a multipurpose execution engine called Dryad for coarse-grain data-parallel applications for efficient fault resilience, data transportation, and data-aware task scheduling. Maglalang et al. [109] proposed a locality-aware dynamic task graph scheduler with optimal locality and load balance and minimum overhead.

Yoo et al. [110] proposed a locality-aware task scheduling for unstructured parallelism by developing a locality analysis framework. The offline scheduler takes workload profuse information as input and makes scheduling decisions that are optimized with underlying cache hierarchies. Paidel et al. [111] focused on the selection of tasks that are most favorable to migrate across nodes in a distributed environment which is further supported by application-level data locality. Choi et al. [112] proposed locality-aware resource management and workflow scheduling by balancing resource utilization and achieving data locality based on network bandwidth in an HPC cloud environment.

Work scheduling and stealing are the two most commonly used scheduling paradigms for scheduling multithreaded computations to workers in typical task-based parallel systems. Guo Yi [113] in his PhD work proposed a locality-aware work-stealing framework for the efficient exploitation of data locality (affinity) and an adaptive work-stealing scheduling algorithm.

Hindman et al. [114] proposed Mesos, a thin resource-sharing layer with the prime objective being to engineer a scalable and efficient system for sharing resources between heterogeneous frameworks by presenting an abstraction called a resource offer. The resources offered to the framework are decided by Mesos based on organizational policy, while which policies to accept, and tasks to run on them are decided by the framework. Mesos uses delay scheduling and data locality is achieved by taking turns reading data stored on each node. Isard et al. [115] proposed a fair scheduling of concurrent jobs with fine-grain resource sharing for distributed computing clusters called Quincy, which achieved better fairness and improved data locality.

3.5. Bulk Synchronous Processing (BSP)

The BSP model, which was presented by Valiant [116] and modified by McColl, is a bridging model to design parallel algorithms and mainly consists of processing components, equipped with local memory, a network for communication, and synchronization between components. Communication is facilitated by one-sided put-and-get calls rather than two-way send-and-receive operations. A barrier ensures that all one-sided communication is completed. The oversubscription of processing elements and problem decomposition are exploited by the BSP model for automatically distributed memory management. Logical processes are randomly assigned to processors for optimal load balancing and communication [117]. Google used BSP for graph analytics with Pregel [118] and map-reduce, while some open-source projects (Apache Hama [119] and Giraph [120]) also extended the use of BSP by employing high-performance parallel programming models on top of Hadoop. There are different programming languages and interfaces based on the BSP model including BSPLib [121], BSPonMPI [122], Bulk Synchronous Parallel ML (BSML), and Multicore-BSP [123,124].

3.6. Out-of-Core Computing

Out-of-core algorithms (e.g., solvers for large systems of linear equations, as in nuclear physics) are used when the data to be processed are too large to fit in memory and data need to be fetched from storage devices e.g., hard and tape drives or memory attached via a network. As these auxiliary storage devices are slow and acceptable performance is achieved by data reuse in memory and how data are laid out in these storage devices for I/O on larger blocks [125].

The use of virtual memory, which enables programmers to access much larger than available memory without a need to know the actual location of data in the memory hierarchy. Different techniques proposed over the years to efficiently exploit data movement across different memory hierarchies i.e., caching, swapping and demand paging, etc. There are applications where virtual memory systems do not meet the programmer's expectations and do not provide enough virtual memory space. Out-of-core algorithms are used when primary and virtual memory is not enough to hold application data [126]. The principle of locality plays an important role in the performance of an application. Locality in terms of an out-of-core algorithm means that data must be laid out in a storage device to allow blocks of data to exchange between the memory of storage devices and also the reuse of data in memory. In a traditional disk-based approach, the processor remains idle until the data is loaded into memory and the next read is not initiated until the computation is finished. The author of [127] addresses this issue by proposing a two-process approach where disk I/O and computation are performed concurrently. One approach to overcome the limitations of speed at which data can be accessed from storage devices is the use of shared and distributed memories across the cluster, which demands the use of high-speed interconnects (InfiniBand) for the dataset to be loaded before the start of the algorithm in large aggregated distributed/shared memory. The use of high-speed interconnects improves performance but at the expense of a tangible cost of initial setup, maintenance, and energy consumption over time. The trend of the use of non-volatile memory NVM, e.g., low-power flash-based Solid-State Drives (SSDs) to speed up the I/O has increased over the years. These SSDs are used along with traditional storage devices on I/O nodes in a cluster environment, which results in enhanced performance with faster data access/load from these SSDs to I/O nodes [128].

Jung et al. [128] addresses the issues and potential benefits of co-locating the non-volatile memory and compute nodes, by presenting a compute local NVM architecture. They identify the drawbacks of modern file systems and proposed a novel Unified File System (UFS). In addition, there are numerous efforts in the literature focused on the usage of SSDs that include the use of SSDs as caches [129], FlashTier [130], and Mercury [131]. Salue et al. [132] presented an out-of-core task-based middleware for data-intensive com-

puting and [125,133,134] are related to out-of-core algorithms for solving dense and linear equation systems.

3.7. Parallelism Mapping

The increase in system concurrency introduced a massive challenge for applications and system software to deal with large-scale parallelism. The widely deployed message passing model MPI may not be a suitable choice to deal with the demands of extreme scale parallelism for exascale systems. Finding a single anomalous process among millions of running processes and threads is not an easy task [135]. The issues related to parallelism are diverse and are very much program-dependent. Parallelism mapping is very much platform/hardware-dependent and optimal mapping decisions depend on many factors like scalability (how much potential parallelism should be exploited), the number of processors in use, scheduling policies, the relative cost of communication and computation, etc. There have been various efforts to address this issue. In multi-cluster environments, the bandwidth among nodes inside a single cluster is normally much higher than the bandwidth between two clusters; similarly, within node cores sharing, the cache can communicate much faster. Entities exchanging or sharing lots of data could be placed on hardware processing units physically close to each other. By doing so, the communication costs are reduced, thus decreasing the application's overall execution time and, as a consequence, its energy consumption [66]. Along with communication, application performance is also dependent on load imbalance, communication patterns, and memory usage.

Mapping threads/processes to cores is dependent on many factors like operating system, implementation (different implementations of MPI e.g., OpenMPI, MPICH, IntelMPI), and runtime system, i.e., Message Passing Interface (MPI), Partitioned Global Address Space (PGAS). The work related to the efficient mapping of processes to reduce the communication cost is based on finding the communication topology (communication pattern/graph of the application e.g., number and size of messages between processes, etc.) and network topology graph (e.g., the latency and bandwidth between different processor cores, inter- and intra-node communication cost, etc.) and then the appropriate selection of optimal cores where the processes should be mapped. One can achieve the task of the optimal mapping of processes to cores by running an application with monitoring tools to understand the communication pattern. Trace libraries can provide communication details, e.g., MPI Trace [136]. There is a lack of standardization for thread/process mapping at start-up but this can be implemented at the MPI-execution level. Unfortunately, current parallel programming paradigms seem unable to address the data locality issue to improve parallel-application scalability. There is a need for some evolutionary and revolutionary changes in parallel programing models to address these problems.

There is a considerable amount of work in the literature related to process placement for MPI applications based on correlating the communication and network topology by algorithmic means using graph theory and implementation based on a scheduler, compiler, or exploiting the MPI runtime environment.

3.7.1. Message Passing Interface Support for Process Mapping

The HPC application community has begun experimenting with the manual placement of individual processes in a parallel job, commonly referred to as "process placement" or "process affinity". MPI implementation provides different mapping patterns like by-node (a.k.a., scatter, cyclic) and by-slot (a.k.a., bunch, pack, block). The different MPI implementations also provide numerous mpirun command line options and bind with different runtime configuration parameters to enhance the process mapping and binding. Assigning more than one process to a single processor is considered oversubscribing in most HPC environments and is generally discouraged as MPI/HPC applications are CPU-intensive; sharing multiple processes on a single processor causes starvation and performance degradation [137].

Given a parallel application, it is essential to efficiently map the MPI processes to the processors on the nodes. The communication pattern needs to be understood by running the application with profiling tools. Trace libraries can provide the communication details, e.g., MPI Trace, and process mapping can be done manually. Communication-assisted communication analysis can also be performed to map MPI processes to processors. The ultimate goal is to reduce communication by mapping processes with frequent communication on a single node. There could be a number of taxonomies to classify the mapping methodologies, like target architecture-based, optimization criteria-based, workload-based (Static or dynamic), etc. Static mapping methodologies are best suited for static workload scenarios where a predefined set of applications with known computation and communication behavior and a static platform are considered. As optimization is performed at design-time, the methodologies can use more thorough system information to make decisions for both homogeneous and heterogeneous architectures [42].

3.7.2. Algorithmic Approaches for Process Mapping

The speed of communication among cores in a multicore processor chip (intra-chip) varies with core selection since some cores in a processor chip share certain levels of cache and others do not. Consequently, intra-chip inter-process communication can be faster if the processes are running in cores with shared caches. The situation gets even worse when among cores on distinct processor chips in a cluster. Rodrigues et al. [138] used a graph mapping technique for the mapping process to cores by considering intra-chip, intra-node, and inter-node communication costs to improve the performance of applications with a stable communication pattern. The approach was tested by comparing the execution times of a real-world weather forecast model using default mapping and the proposed solution and obtained an improvement of up to 9.16%.

Rashti et al. [139] merged the node physical topology with network architecture and used graph embedding tools with an MPI library to override the trivial implementation of the topology functions and effectively reorder the initial process mapping. Hestness et al. [140] presented a detailed analysis of memory system behavior and effects for applications mapped to both CPU and GPU cores. Understanding the memory system behavior is very important as multiple cores are integrated on the same die that shares numerous resources. This paper presents a detailed comparison of memory access behavior for parallel applications executing on each core type in a tightly controlled heterogeneous CPU–GPU processor simulation.

The communication topology of an application and its underlying architecture affect the performance of point-to-point communication and MPI provides different primitives to gather such information such as MPI Cart create and MPI Graph create. An application communication graph can be created by calculating the communication cost between processes using message count and message volume. HU Chen et al. [141] proposed a profile-guided approach to finding the optimized mapping automatically to minimize the cost of point-to-point communications for arbitrary message-passing applications called MPIPP (MPI process placement toolset). This tool acquires the communication profile of the MPI application and the network topology of the target clusters. They also proposed an algorithm for optimized mapping and enhanced performance is reported by comparing their solution with existing graph portioning algorithms.

Zhang et al. [142] proposed an approach for optimized process placement to handle collective communication by transforming them into a series of point-to-point communication operations. They decomposed a collective communication into point-to-point and then generated the communication pattern of the whole application. They used a graph-partitioning algorithm for optimized process mapping. Pilla et al. [143] proposed a topology-aware load balancing algorithm for multicore systems by modeling distances and communication among hardware components in terms of latency and bandwidth by exploiting the properties of the current parallel systems, i.e., network interconnection, multi-levels of cache, etc. The introduction of multicore processors introduces numerous

challenges including competition between the various physical cores for shared resources. Having more cores in a single node causes multiple requests for the network interface, resulting in performance degradation [144]. This demands the distribution of parallel processes in available computing nodes such that requests arriving at each network interface be decreased. The queuing time of messages at interface queues will be decreased as well, resulting in enhanced performance. Zarrinchain et al. [144] addressed this issue by proposing a solution for mapping parallel processes to multicore clusters to reduce network interface contention by determining the length of the messages among processes and an appropriate value for the threshold (number of processes in each compute node) using the number of adjacent processes of each process and the number of available free cores in the computing nodes.

Guillaume et al. [145] proposed an efficient process mapping of an MPI application to better take advantage of a multicore environment without any modification of MPI implementation and improved performance is reported solely on the basis of relevant process placement. They extract an embedding of the application's graph from the target machine's graph and used scotch software to solve NP graph problems. Scotch applies graph theory, with a divide-and-conquer approach, to scientific computing problems such as graph and mesh partitioning, static mapping, and process ordering. The information is then used to create a mapping between MPI process ranks and each node's core numbers. Finally, an application-specific command line is generated.

Other work related to mapping includes architecture-specific mapping [146–148] for Blue Gene systems, [139,149,150] targeting multicore networks, [151] targets hybrid MPI/OpenMP mapping, [152] proposes a mapping library, and [153,154] advance programming standards that support virtual topology mapping.

3.7.3. Machine Learning-Based Parallelism Mapping

Programming with target architecture in mind and mapping parallelism to processors/cores to avoid/minimize communication are two alternative ways of optimizing application performance. Selecting the correct mapping scheme has a significant impact on performance and these mapping schemes are very much architecture-dependent. So, there is always a need for an automatic and portable solution for assigning tasks to a target architecture to achieve scalable parallelism.

Castro et al. [155] proposed a machine learning-based approach to do efficient thread mapping in transactional memory (TM) applications. Software TM libraries usually implement different mechanisms to detect and solve conflicts. As a consequence, it becomes much more complex to determine a suitable thread-mapping strategy for an application since it can behave differently according to conflict detection and resolution mechanisms. Grewe et al. [156] proposed a portable partitioning scheme for OpenCL programs on heterogeneous CPU and GPU architectures by extracting code features statically and using ML algorithms for predicting the best task partitioning. Tournavitis et al. [157] proposed profile-driven parallel detection and ML-based mapping to overcome the limitations of static analysis and traditional parallelizing compilers by using profiling data to extract control and data dependencies and then used an ML-based trained predictor to select the best scheduling policy offered by OpenMP i.e., CYCLIC, GUIDED, STATIC, and DYNAMIC.

Wang et al. [158] proposed a compiler-based automatic and portable approach for selecting the best thread scheduling policy based on an ML model learned offline, to map parallel programs to multicores. ML-based predictors use profiling information to characterize the code, data, and runtime features of a given program. The feature extractor needs several profiling runs for a program to extract these features. They used an Artificial Neural Network ANN and Support Vector Machine SVM to build two different learning models to predict program scalability and classify scheduling policies. The models were trained using a set of training data that consisted of pre-parallelized programs with selected features and desired mapping decisions. Long et al. [159] used an ML-based approach for cost-aware parallel workload allocation by using static program features. More specifically

they used ML to determine the thread number allocated for a parallel java loop on the run time and don't tackle portability. Adaptive multi-versioning for OpenMP parallelization via machine learning integrated in the compiler, to achieve parallelism. Pinel et al. [160] proposed an ML-based automatic parallelization of a scheduling heuristic, by defining a generic parallel pattern-matching engine that learns the algorithm to parallelize.

Emani et al. [161] proposed a predictive modeling approach that dynamically considers a number of thread selection policies and chooses the one it believes will perform best at every parallel loop and called this approach a mixture of experts. Their approach predicts the optimal number of threads for a program and the run-time environment. Emani et al. [162] proposed an efficient parallel mapping based on online change detection by combining an offline model with online adaption to find the optimal number of threads for an OpenMP program. Luk et al. [163] proposed a fully automatic mapping technique to map computations to processing elements on heterogeneous multiprocessors. They measured the parallelization speedups on matrix multiplication with a heterogeneous machine and implemented it in an experimental system called Qilin and report a performance close to manual mapping with an adaptability feature for different problem sizes and hardware configurations.

Dominguez et al. [164] extended their previous work by using Servet to map applications on multicore systems and analyze the performance of different parallel programming models i.e., message-passing, shared memory, and Portioned Global Address Space (PGAS). The featured extracted by Servet can be used to optimize the performance by choosing a more appropriate mapping policy without source code modification.

3.8. In Situ Data Analysis

The increasing data size, limited storage and bandwidth, efficient use of compute resources, difficulties in examining output data, and storing and retrieving output data for post data analysis are considered impractical for an exascale environment. The term data movement for in situ data analysis is mostly used to describe data movement back and forth to persistent storage and retrieving data for post data analysis. In situ analysis translates to saving in execution times, power, and storage cost, and avoiding either completely, or to a very large extent, massive data movement of simulations output to persistent storage.

In situ data analysis is performed on the data in the transition phase before they are written back into the parallel file system. Tiwari et al. [165] exploit the compute power in SSDs for in situ data analysis and called this approach Active Flash. This approach provides energy-efficient data analysis as computation near storage devices reduces the data movement cost; in addition, SSDs are equipped with low-power, multicore ARM-based controllers. In situ analysis has become one of the core aspects of data interpretation in large-scale scientific simulations. However, for data that already reside in backend storage systems, efficient data analysis is still a core issue.

Zheng et al. [166] proposed an in situ middleware system to facilitate the underlying scheduling tasks e.g., cycle stealing. They created an agile run-time and called it GoldRush, fine-grained scheduling to steal idle resources by ensuring minimal interruption to the simulations and in situ data analysis. The system makes use of the idle wasted resources of compute nodes to be efficiently used for in situ analysis. The experiment results showed enhanced performance, low data-movement cost, efficient resource utilization, and minimum interference with simulation. Sewell et al. [167] proposed a framework that uses both in situ and co-scheduling approaches for large-scale output data. They compare the performance by analyzing different setups to perform data analysis, i.e., in situ, co-scheduling, and a combination of both.

3.8.1. In Situ Compression

Scientific data are mostly regarded as effectively incompressible due to their inherently random nature and decompression also imposes extra overhead. Sriram et al. [168]

addresses this problem of compression by exploiting temporal patterns in scientific data to compress data with minimal overhead on simulation runtime.

Zou et al. [169] worked on data compression for the removal of redundant data reduction, by using general compression techniques and proposed a use-specific method that allows users to remove redundant or non-critical data by using simple data queries. This method allows users to optimize output data and explicitly identify the data that need to be retained. General-purpose lossy compression techniques do not provide this level of flexibility.

3.8.2. Use of Indexing for In Situ Data Analysis

As the computation power increases at a brisk speed compared to storing data to disks and reading data back from these disks, J Kim et al. [170] used indexing and in situ processing to address these challenges. Indexing is a powerful and effective way of addressing data access issues, while the implementation of an indexing technology is embedded in DBMS, which lacks the ability to manage most scientific datasets. They used in situ data processing to create indexing in parallel, thus reducing the resources utilized, to store data back and forth from disks. The usage of indexes improved the data access time and in situ data processing reduced the index creation time.

According to Sriram et al. [171], current state-of-the-art indexes require computation and memory-intensive processing, thus making indexing impractical for in situ processing. They propose DIRAQ, a parallel in situ, query-driven visualization, and analysis during simulation time that transforms the simulation output to a query-accessible form. This technique has a minimum overhead on simulation runtime and speeds up query-response time.

Yu Su et al. [172] proposed in situ Bitmap generation and performing data analysis based on these Bitmaps. Bitmap indices can be used as a summary structure for offline analysis tasks. Their work basically focused on in situ analysis of selected bitmaps, thus reducing the amount of simulation output data to be stored on the disk and reducing the memory requirements for in situ analysis. HPC systems with in situ data analysis analyze temporary datasets as they are generated. Permanent datasets are stored in the backend persistent storage systems; their efficient analysis is still a challenging task.

3.8.3. In Situ Visualization

As scientific simulations produce a huge volume of raw data, saving a vast amount of raw data for offline analysis is a complex task and not a suitable method for current petascale and future exascale systems. Karimabadi et al. [173] addresses this I/O issue through in situ visualization strategies. The main idea is to extract important features from raw data parallel with simulation and thus reducing the amount of raw data stored on the disk. Their work focused on the overhead associated with computation needs for in situ visualizations in parallel with the simulation run.

Yu et al. [174] investigated in situ visualization for turbulent-combustion simulations and explored in situ data processing and visualization strategies in an extremely parallel environment. Their results showed that in situ visualization enhanced performance and can be used for accelerating high-performance supercomputing and scientific discovery. Zou et al. [175] proposed an online data query system (FlexQuery) using inline performance monitoring and minimized data movement with low latency data-query execution. They demonstrated the dynamic deployment of queries by the proposed query system for low-latency remote data visualization.

Woodring et al. [176] addresses the issue of reducing memory footprints by sharing data between visualization libraries and simulations by using a zero-copy data structure. They optimized the traditional way of coupling different mesh-based codes for in situ data analysis where data needs to be explicitly copied from one implementation to another with the necessary translation. This results in redundant data, which ultimately increases memory footprints. They proposed an alternative way of sharing data between simulations through optimized dynamic on-demand data translation, with reduced memory

footprints and memory per core. Nouanesengsy et al. [177] proposed a generalized analysis framework for automatically evaluating the relative importance of data in order to reduce data products, thus ensuring enough resources to process reduce datasets. The proposed framework prioritizes large-scale data by using user-defined prioritization measurements.

3.8.4. In Situ Feature Selection

Landge et al. [178] proposed in situ feature extraction techniques that allow state-of-the-art feature-based analysis to be performed in situ with minimal overhead on simulation runtime.

Zhang et al. [179] proposed a framework for in situ feature-based object tracking on distributed scientific datasets with decentralized online clustering DOC and a cluster tracking algorithm. They run in parallel with the simulation process and obtain data from on-chip-shared memory directly from the simulation. Their results showed that the proposed framework can be efficiently utilized for in situ data analysis of large-scale simulations.

4. Data Locality in Big Data Environments

Big data has transformed society with many innovative applications [180–189]. Different solutions emerged over the years to deal with big data issues and were successfully implemented. However, these solutions do not satisfy the ever-growing demands of big data. The issues related to big data are immense and cover a variety of challenges that needs careful consideration. These challenges include data representation, redundancy reduction, data compression, data life cycle management, analytical mechanism, data confidentiality, energy management, expandability and scalability, high dimensionality, computational complexity, real and distributed computation, non-structured processing, etc. The key advantage that big data technologies brought over traditional HPC is data locality. Hadoop brings computation to data and Spark further enhances it through in-memory computation.

Big data analysis is done at various levels, i.e., when server memory is huge, in-memory analysis can be used by keeping the hot data in memory for the sake of efficiency. This memory-level technology is ideal for real-time analysis, e.g., MangoDB [1]. Business intelligence (BI) has different tools and procedures for analyzing data when it exceeds the memory capacity. Map/reduce is used most widely for massive data analysis, which is beyond BI capacity and mostly falls into the offline analysis category [190]. Chasing a correct solution depends on the size of the data, the urgency of results, prediction about the need for more processing power as the size of data increases, fault tolerance for applications in case of hardware failure, data rate, scalability, etc. Data locality has been recognized as one of the major issues to be resolved for exascale systems and considerable efforts have been done to move computation closer to data. Map-Reduce [191] is the most widely used data processing model in data-centric computation environments. Google MapReduce, based on Google File System (GFS) [192], achieves locality by relying on data-aware task scheduling and block replication. Hadoop [193], based on HDFS (Hadoop Distributed file System) [194], followed the same approach. Different solutions emerged as the processing requirements of applications change i.e., HBase [195] for random-access data, Apache Giraph [120] for graph processing, Spark streaming, and TEZ [196] and Twister [197] for iterative streaming. Data locality in data-parallel systems by bringing flexibility to scheduling algorithms has a major edge over traditional HPC systems. In the following section, we analyze the research efforts and current trends by focusing specifically on enforcing data locality in big-data environments.

4.1. Parallel Programming Models

We can broadly classify parallel programming models in big-data environments as batch processing and iterative. The following section gives a brief overview of both.

4.1.1. Batch Processing

Map-reduce is probably the most widely used data-centric programming model where a piece of work is divided among several parallel map/reduce tasks. The original implementation by Google relies on the Google File System GFS [192], whereas the open implementation of map-reduce (Hadoop [193]) relies on the Hadoop Distributed File System HDFS, and data locality is achieved by data-aware scheduling and block replication. The input files are split up in input format, which selects the file, defines the input splits, breaks the file into tasks, and provides the place for the record reader objects. The input format defines the list of tasks that makes up the mapping phase. The task then assigns to the node of the system based on where the input file chunks are physically resident [198]. The input split describes the unit of work that comprises a single map task in a map-reduce program. The record reader loads that data and converts them into (key value) KV pairs that can be read by the mapper. The mapper performs the first phase of the map-reduce program given the key and the value, and the mappers export key and value pairs and sends these values to the reducers. The process of moving map outputs to the reducers is known as shuffling [199].

Due to the lack of support for processing multiple heterogeneous datasets, map-reduce-merge was proposed by Yang et al. [200] to facilitate the use of map-reduce in relational operations like join.

4.1.2. Iterative

Traditional map-reduce lacks support for iterative tasks. The mapper needs to read the data and, after each iteration, results need to be written back to the disk for the subsequent iteration. Disk I/O is a bottleneck here and for each iteration, a new mapper and reducer need to be initialized. [201]. Different wrappers or extensions have been developed to overcome the shortcomings of Hadoop for improved performance, e.g., programming model extensions.

Map-iterative-reduce [202] is an iterative framework that has emerged from map-reduce (which lacks support from reduce-intensive workloads) to support reduce-intensive applications. One of the implementations of iterative-map-reduce is Twister. Ekanayake et al. [197] presented a programming model and architecture of Twister and compared its performance with other programming models like Hadoop and DryadLINQ and reported efficient iterative map-reduce computation performance. Bu et al. [203] also targeted the lack of support for map-reduce and Dryad for iterative programs by proposing Haloop for efficient iterative data processing. They further enhanced the performance by presenting loop-aware scheduling and efficiently exploiting various cache mechanisms.

Spark [204] provides a data flow execution engine that supports cyclic data flows, with support for various languages, e.g., Scala, Python, and Java. Sparks abstracts data with Resilient Distributed Datasets (RDD), which is a spark representation of a set of data, distributed across multiple machines and allows fault-tolerant in-memory computations on large clusters.

4.1.3. Language Support

The map-reduce programming model is often referred to as a low-level model by analysts, who are used to SQL-like or declarative languages. It also requires advance programming skills and an in-depth understanding of system architecture for developing efficient map-reduce applications. [205]

Hive provides the ability to analyze large amounts of data stored in HDFS. Hive was designed to appeal to a community comfortable with SQL and uses an SQL-like language known as HiveQL. It supports map and reduce transform scripts in the language of the user's choice, which can be embedded within the SQL, and is widely used in Facebook. Hive is a framework for performing analytical queries, while its dominant use is to query flat files. Currently, Hive can be used to query data stored in Hbase. The worker nodes in Hive keep small tables in a distributed memory setup for quick data access.

Olston et al. [206] developed a programming language (Pig-Latin) that gives a programming abstraction of Java map-reduce (low-level procedural style) to make it somewhat similar to the high-level declarative style of SQL for RDBMs.

4.1.4. Locality-Aware Partitioning

The map-reduce-based join operation is not optimized when dealing with skewed data and proposed solutions often result in a huge volume of data in the shuffle phase affecting the performance of the map-reduce-based join. Lin et al. [207] addresses this problem by proposing SALA (A Skew-Avoiding and Locality-aware) algorithm and locality-aware partitioning to ensure data locality, even data distribution to reducers, without any modification of the map-reduce framework. Similarly, Ibrahim et al. [208] proposed LEEN for locality-aware and fairness-aware key partitioning by incorporating an asynchronous map and reduce scheme. Their results show enhanced data locality with minimum intermediate shuffle data. Rhine et al. [209] proposed a locality-aware scheduling algorithm and input data split, by partitioning data belonging to a node, in a single split.

4.2. Data Placement

4.2.1. Locality-Aware Data Placement

Hadoop relies on HDFS to store data with high availability to multiple nodes but data is placed randomly to achieve load balance without taking the characteristics of the data into consideration. By default, Hadoop lacks the ability to collocate data on the same set of nodes. Eltabakh et al. [210] addresses this issue by proposing CoHadoop, an extension of Hadoop, giving control to applications to manage locality-aware storage by modifying the HDFS data replacement policy. Minimizing off-switch communication increased the performance of map-reduce in Hadoop. Yu et al. [211] argues that the grouping of blocks of data in fewer racks enhanced performance by reducing off-chip communication and proposed a methodology to group data and scheduling mechanisms by exploring the trade-off between off-chip communication and parallelism.

Schedulers often consider map tasks for locality and ignore reduce tasks while fetching intermediate data, which results in performance degradation. Tan et al. [212] proposed a stochastic optimization framework for improving reduce task data locality for sequential map-reduce jobs. Wang et al. [213] addresses the issues related to the random distribution of data placement in traditional Hadoop by proposing a data-grouping-aware (DRAW) data placement scheme. DRAW optimizes group sizes and optimizes parallelization per group by re-organizing data layouts.

4.2.2. Locality-Aware Data Placement in a Heterogeneous Environment

The Hadoop implementation by default assumes that cluster nodes are homogeneous and data locality is not taken into consideration to map tasks. Xie et al. [214] addressed this issue by proposing a data placement strategy in heterogeneous environments (Hadoop Clusters) for optimal load balance. The data placement algorithm must consider the node heterogeneity (processing capabilities) to partition input and intermediate data. Arasanal et al. [215] proposed load-balanced data placement enhancements and an input data distribution algorithm in Hadoop based on the processing capabilities of the nodes. A similar approach was proposed by Wei Lee et al. [216], who proposed a dynamic data placement algorithm to balance workload based on the computing capabilities of each node in a heterogeneous environment. The proposed strategy dynamically adjusts workload and reduces data transfer time.

Ubarhande et al. [217] analyzed various scheduling techniques from a data locality prospective and proposed a data placement methodology based on computation ratio for Hadoop data nodes and enhanced performance is reported by executing standard map-reduce applications, i.e., Grep and word count. The input to the task must be present on a node where the task is supposed to be executed, and otherwise needs the transferring

of input data, which ultimately increased execution time. Sujitha et al. [218] proposed a methodology to address the issues of heterogeneity and data locality in Hadoop.

4.3. Scheduling and Load Balancing

4.3.1. Locality-Aware Scheduling and Load Balancing

Locality-aware scheduling is one of the prominent features of MapReduce, which allows schedulers to bring compute to data rather than vice versa. The cross-rack traffic must be reduced for optimal performance. Guo et al. [219] proposed an algorithm to improve data locality by exploiting available resources and considering all tasks together rather than a task-by-task approach, as in traditional Hadoop. They also proposed another algorithm by integrating fairness and locality by allowing users to define the trade-off for desired performance. Chen et al. [220] proposed a partition algorithm CLP (Cluster Locality Partition) for optimal load balance and performance. Locality partitioning achieved data locality by assigning data clusters to appropriate nodes.

Network traffic is a major bottleneck in data-intensive applications and can be reduced by locality-aware scheduling. Chen et al. [221] addresses this issue and proposed the LaSA locality-aware scheduling algorithm for data-aware resource assignment. Wang et al. [102] proposed a data-aware work-stealing technique implemented at the node/scheduler level to achieve an enhanced load balance and efficient exploitation of data locality by proposing a fully distributed task scheduling system for Many Task Computing (MTC) systems. Park et al. [222] proposed a runtime reconfiguration scheme (Dynamic Resource Reconfiguration DRR) that schedules a task to nodes where data resides and also dynamically increases or decreases the computing capability of each node for optimal data-aware scheduling. Zaharia et al. [223] addresses the conflict between data locality and fairness in scheduling by proposing a delay-scheduling algorithm, which increases throughput with optimal data locality and guaranteed fairness. Hadoop has been optimized to reduce the amount of network traffic, i.e., delay scheduling achieves nearly optimal data locality for a variety of workloads, which ultimately results in a low volume of network traffic [223]. Intermediate data shuffling in Hadoop still generates a huge volume of network traffic.

4.3.2. Locality-Aware Scheduling and Load Balancing in a Heterogeneous Environment

The scheduling of map-reduce tasks is further complicated in heterogeneous environments and was addressed by Zhang et al. [224] by proposing a locality-aware scheduling algorithm. Hsu et al. [225] proposed locality and load-aware virtual machine mapping techniques to improve map-reduce performance in heterogeneous environments, by portioning data before the mapping phase and using virtual machine mapping in the reduce phase. Xue et al. [226] proposed the dynamic scheduling algorithm BOLAS (Bipartite-Graph Oriented Locality-aware Scheduling) by modeling the scheduling problem as a bipartite-graph matching problem using the Kuhn–Munkeres algorithm and achieved improved data locality for both homogeneous and heterogeneous environments. Sadasivam et al. [227] proposed the Hybrid Particle Swarm Optimization-Genetic Algorithm (HPSO-GA) for the efficient execution of tasks and utilization of resources achieved by capacity-aware load distribution in heterogeneous environments. Zhang et al. [228] addressed the issues related to the performance of map-reduce in heterogeneous environments by proposing a methodology to separate the map shuffle and reduce stages for optimized task allocation and controlled dynamic execution.

4.3.3. Adaptive Scheduling

Guo et al. [229] proposed a data-distribution-aware task scheduling methodology by overcoming the uneven data-distribution strategy of default scheduling techniques. The network overhead is reduced and high system efficiency is reported by mapping tasks to nodes with a high probability of data availability and the task's scheduling priority. Hammoud et al. [230] proposed a locality-aware reduced task scheduler called LARTS, which collocates data and reduce tasks. LARTS achieves high data locality and reduce

scheduling delay/skew and exploits resources efficiently. Ahmad et al. [231] proposed communication-aware load balancing and the scheduling of map computation, predictive load balancing for reduce computation to optimize the performance of map-reduce in heterogeneous environments. Similarly, Kumar et al. [232] proposed a context-aware scheduling technique (CASH); Zhao et al. [233] proposed a job-scheduling algorithm on map-reduce; Hammoud et al. [234] presented a locality- and skew-aware scheduling algorithm (CoGRS); and Ibrahim et al. [235] proposed replica-aware scheduling for map-reduce. A detailed survey on adaptive scheduling in map-reduce is presented by Mozakka et al. [37].

4.3.4. Delay Scheduling

Delay scheduling involves scanning jobs more than once before some threshold is reached, after which the job is scheduled. Delay scheduling may also delay high-priority jobs. Sethi et al. [236] proposed a mechanism to force the scheduler to launch high-priority jobs to be executed locally or on some nodes based on the availability of data. Yang et al. [237] proposed a scheduling algorithm by segregating map and reduce as separate stages of the scheduling problem and addressing the issues of map-reduce stage deadline, execution time, and data locality. Bezerra et al. [238] proposed data locality-aware job scheduling with the prime motive of running tasks that handle the same blocks of data on the same node where the blocks reside.

4.4. In-Memory Computations

Memory storage capacity and bandwidth are increasing at a brisk speed and the time is not far off when memory will replace hard drives. This transformation will ultimately set trends for building in-memory systems where a major portion of the data fits in memory with an obvious performance gain. The in-memory capacity of machines in map-reduce clusters is often underutilized and can be effectively utilized by in-memory prefetching input data to improve data locality. Sun et al. [239] proposed a prefetching service-based task scheduler HPSO (High-performance Scheduling Optimizer) to predict the optimal node for future tasks and prefetching needed data.

Both Hadoop and Spark [240] are big-data frameworks that perform the same tasks, are not mutually exclusive, and are able to work together. Spark is reported to work 100 times faster than Hadoop in some situations and doesn't have its own distributed storage system [241]. Apache Spark is the most widely used distributed in-memory computing framework that handles in-memory operations by copying data from distributed file systems into faster logical RAM [242]. Map-reduce writes all data back to the distributed storage system after each iteration to ensure full recovery, whereas Spark arranges data in resilient distributed datasets that are capable of full recovery in case of failure. The efficiency of Spark is questioned for applications where the dataset is loaded and evicted at runtime. Shen Li et al. [243] addresses this problem by proposing Stark for optimizing in-memory processing on dynamic dataset collection by avoiding replication and shuffling. Engle et al. [244] proposed Shark (Hive on Spark) for deep data analysis based on Resilient Distributed Datasets (RDDs) [204] to achieve scalability, performance, resilience, and the efficient execution of iterative algorithms with intra-query temporal locality for in-memory computation on large clusters. Sentos-Neto et al. [245] exploited storage affinity by data reuse and used a replication strategy for efficient scheduling without any run-time information. Reynold et al. [246] proposed GraphX by combining data parallel and graph parallel systems by efficiently distributing graphs, and exploiting resilience and in-memory computation.

4.4.1. Registers and Cache-Centric Optimizations

The effective and efficient usage of registers is usually targeted at the compiler or assembly-language level. The use of in-memory databases with traditional iterative-style queries often results in poor data locality [33]. Efficient utilization of cache hierarchies

of different levels is of paramount importance for obvious performance gains. The work related to cache optimization includes compression [247], coloring [248], re-organizing data layouts by organizing records in columns lay out [249,250], or using decomposition storage model [251] and cache conscious indexes [252,253]. A detailed survey is provided by Zhang et al. [33].

4.4.2. Non-Uniform Memory Access (NUMA)

Each processor in the NUMA architecture has faster access to its own local memory and relatively slow and higher-latency access to remote memory. In the context of NUMA, much of the work in the literature is related to data partitioning [254–256] and data shuffling [257]. A detailed survey is provided by Zhang et al. [33].

4.4.3. NVRAM

The use of non-volatile memory is an emerging trend in High-performance computing environments to provide memory with high speed and capacity. The NVRAM technology provides better performance compared to traditional hard drives/flash drives and comparable performance to DRAM, e.g., phase change memory technology [258], memristive devices [259], and STT-MRAM [260]. A detailed survey is provided by Zhang et al. [33].

4.4.4. In-Memory Data Processing Systems

In-memory data processing is much faster compared to other data-centric computational models, e.g., Hadoop, and is often used for the analysis of huge volumes of data within a time constraint. Spark [240], Mammoth [261], and Piccolo [262] are data analytics systems, whereas S4 [263] and Map-reduce online [264] are real-time data processing systems.

4.4.5. In-Memory Data Storage Systems

In-memory data storage systems include relational, NoSQL databases, and cache-based systems (cache between the application server and database). [265–268] describe examples of in-memory relational databases, [269–271] describe in-memory NoSQL data bases, and [272–274] describe in-memory cache-based systems. A detailed survey is provided by Zhang et al. [33].

5. Data Locality in Converged HPC and Big Data Environments

Now that we have reviewed the literature on data locality in HPC and big data environments, in this section, we review the literature on converged systems. We first capture in Table 1 a summary of the efforts in big data and HPC that were reviewed in Sections 3 and 4. Specifically, the table gives a brief overview of research efforts related to data locality at different levels of the software ecosystem and also highlights some of the convergence challenges.

Table 1. Data Locality-aware research efforts in HPC & Big data Environment and Convergence Challenges.

	HPC	Big Data	Convergence Challenges
Parallel Programming Models	Majo et al. [67], Lezos et al. [68], Regan-Kelley et al. [69], X10 [71], Huang et al. [72], BSP [116], Pregel [118], including BSPLib [121], BSPonMPI [122], Bulk Synchronous Parallel ML (BSML), Multicore-BSP [123,124].	Google File System GFS [192], Yang et al. [200], Map-iterative-reduce [202], Ekanayake et al. [197], Bu et al. [203], Spark [204], Olston et al. [206], SRM [275], iRODS [276], MapReduce-MPI [277], Pilot-MapReduce [278], Lustre [279], GPFS [192], PVS [280]	• Scalability • Programming Abstraction • Exploiting dynamic parallelism • Data locality through abstraction layer • Datacentric abstraction. • Heterogeneity

Table 1. *Cont.*

	HPC	Big Data	Convergence Challenges
Scheduling and Load Balancing	Ousterhout et al. [99], Falt et al. [103], Muddukrishna et al. [104], Ding et al. [105], Lifflander et al. [106], Xue et al. [107], Isard et al. [108], Maglalang et al. [109], Yoo et al. [110], Paidel et al. [111], Guo Yi [113], Hindman et al. [114], Isard et al. [115]	Guo et al. [170], Chen et al. [220], Chen et al. [221], Wang et al. [102], Park et al. [222], Zaharia et al. [223], Zhang et al. [224], Hsu et al. [225], Xue et al. [226], Sadasivam et al. [227], Zhang et al. [228], Guo et al. [229], Hammoud et al. [230], Ahmad et al. [231], Kumar et al. [232], Zhao et al. [233], Hammoud et al. [234], Ibrahim et al. [235], Mozakka et al. [37], Sethi et al. [236], Yang et al. [237], Bezerra et al. [238]	• Poor scalability • Heterogeneity • Locality-aware scheduling algorithms • Portability • Complexity
Parallelism Mapping	Jeannot et al. [281], Rashti et al. [139], Hestness et al. [140], HU Chen et al. [141], Zhang et al. [142], Zarrinchain et al. [144], Guillaume et al. [145], Blue Gene systems [146–148], multicore networks [139,149,150], hybrid MPI/OpenMP mapping [151], mapping library [152], Grewe et al. [156], Tournavitis et al. [157], Wang et al. [158]	Map-Reduce [191], Hadoop [193], Map-iterative-reduce [202], Spark [204], Engle et al. [244], Olston et al. [206].	• Manual Mapping is time-consuming and error-prone • Portability • High Complexity (Compiler based techniques) • Expensive compilation Overhead • Lack of Intelligence
In situ Data Analysis	Tiwari et al. [165], Zheng et al. [166], Sewell et al. [167], Sriram et al. [168], Zou et al. [169], Kim et al. [170], Sriram et al. [171], Yu Su et al. [172], Karimabadi et al. [173], Yu et al. [174], Zou et al. [175], Woodring et al. [176], Nouanesengsy et al. [177], Landge et al. [178], Zhang et al. [179]	Wang et al. [282], Xu et al. [283], [165], Wang et al. [282], Xu et al. [283], Spark on demand [284].	• Data Size • Energy Efficiency • Resource utilization • Limited storage and bandwidth • Data Movement cost • Efficient Data Analysis • Compression/decompression overhead • Indexing (compute and memory intensive) • I/O issues • In situ visualization
Locality-aware Partitioning	Zhang et al. [33], NUMA data shuffling [257], data partitioning [254–256], NVRAM Memristive devices [259] STT-MRAM [260],	Lin et al. [207], Ibrahim et al. [208], Rhine et al. [209]	• Lack of Intelligence • Complexity • Load balancing • Data Dependencies
Data Placement		Eltabakh et al. [161], Yu et al. [162], Tan et al. [163], Wang et al. [164], Xie et al. [214], Arasanal et al. [215], Wei Lee [216], Ubarhande et al. [217], Sujitha et al. [218]	• Locality-aware storage • Communication

Table 1. *Cont.*

	HPC	Big Data	Convergence Challenges
In-Memory Computation	Sun et al. [239], Shen Li et al. [243], Engle et al. [244], Sentos-Neto et al. [245], Reynold et al. [246], In Memory Data Processing Systems Spark [240], Mammoth [261], Piccolo [262], S4 [263], Map-reduce online [264].		• Emerging new non-volatile memory technologies • Network/storage aggregation • Efficient utilization of cache • Optimized utilization of storage management
Cache-centric Optimization	Compression [247], coloring [248], decomposition storage model [251], re-organizing data layouts [252,253], Gupta et al. [73], Gonzalez et al. [74], on-chip caching [75,76], data access frequency [77,78], Kennedy et al. [93], Sparsh Mittal [94].		• Efficient exploitation of cache • Cache-aware partitioning • Smart, dynamic, and predictive optimizations.

In typical HPC environments, both compute and storage servers are separated and the cost of moving these large datasets is very high. High-end computation machines and storage clusters running parallel file systems are connected via a high-speed network. Data-intensive applications in this setup demand high data movement across the network, which is a major bottleneck. In contrast to the big-data paradigm, data management in HPC environments lacks higher-level abstraction [44]. Solutions have emerged over the years to deal with massive amounts of data in data-intensive applications, e.g., SRM [275], iRODS [276], MapReduce-MPI [277], Pilot-MapReduce [278], etc.

HPC applications use parallel programming paradigms such as MPI to exploit parallelism, rely on low-latency networks for message passing, and use parallel file systems, for example, Lustre [279], GPFS [192], PVS [280], etc. Data-intensive computing makes use of distributed file systems, which include the Google file system GFS [192], the HDFS Hadoop distributed file system [194], etc. HPC applications use a data-intensive distributed file system through an interface, for example, libHDFS [285]. Although these file systems are tailored for different targeted applications and computing environments, they have somewhat identical abstract-level designs [286]. Data consistency is not a priority for data-intensive file systems and is usually compromised for better performance by introducing a client-side cache to improve bandwidth. Parallel file systems support concurrency, while cache coherency is maintained in data-intensive file systems through data-locking techniques. The client and server process is collocated for enhanced I/O performance, while data locality is not a prime design choice for parallel file systems [44].

As discussed before, data locality is considered a major concern for optimized data movement and the co-location of computing and data to reduce communication between process/threads/compute nodes to achieve energy-efficient exascale computing. There is a huge body of work related to data locality, as has been presented in the previous sections for both the big data and HPC domains. The following sections discuss the research efforts, which can be considered as baby steps, toward data locality-aware convergence of HPC and big data.

5.1. MPI with Map-Reduce Frameworks

MPI is a de facto standard and is widely used in High-performance computing environments for effective and efficient communication. Researchers have successfully experimented with the idea of using MPI for data-intensive computing. Hoefler et al. [287] proposed a scalable implementation of map-reduce functionality using MPI and numerous possible extensions in MPI to support map-reduce. Hadoop map-reduce provides fault

tolerance through redundant storage and reallocation of work, whereas MPI-based implementations are deprived of that. The MPI-map-reduce integration can provide efficient implementation with optimized data movement by controlling where data reside for each map and reduce phase, which can be achieved by user-defined hash functions [277].

The parallelization of many task applications has been tried with different workflow systems, e.g., MPI, ad-hoc Hadoop [193], CloudBlast [288], Spark [240], and HTCondor [289]. Zhang et al. [290] used Apache Spark to parallelize many task applications by using Kira (an astronomy image processing toolkit) and compared its performance with equivalent parallel implementation using an HPC toolset and improved performance is reported. Lu et al. [291] identified the challenges and potential benefits of reducing the communication overhead by using MPI with map-reduce and also highlighted possible MPI extensions for optimized integrated MPI-map-reduce programming paradigms. DataMPI exploits the overlapping of the map, shuffle, and merge phases of the map-reduce framework and increases data locality during the reduce phase. This approach provides the best performance and average energy efficiency [292]. Mohamed et al. [293] proposed the overlapping of the map and reduce phases by running them concurrently, and MPI is used as a message-passing communication medium between the two to exchange partial intermediate data.

5.2. Map-Reduce Frameworks with High-Performance Interconnects

Data-intensive applications have been extensively used in HPC infrastructure with multicore systems using the map-reduce programming model [294]. Hadoop relies on legacy TCP/IP protocols for the transferring of intermediate data, which makes Hadoop incapable of utilizing the benefits of RDMA. So, it finds it difficult to use high-performance interconnects in an optimal way, and, so, different HPC-oriented map-reduce solutions have been proposed that addresses the problem of leveraging high-performance interconnects [295], i.e., RDMA–Hadoop, DataMPI [296], etc. Hadoop has its own limitations of disk and network bandwidth, and network bandwidth is increased with the use of InfiniBand. The TCP/IP protocol is used as a communication protocol in Hadoop through Java sockets [292]. Different solutions emerged to address this problem for efficient use of map-reduce with high-performance interconnects.

Yandang et al. [297] presented a comparative analysis of InfiniBand and 10GigaBit and the performance of both is evaluated on Hadoop. Performance is considerably improved when the intermediate data size is small, while, with a large intermediate data size, performance degradation is reported. Disk bottleneck and scalability also improved with the use of Hadoop with high-performance interconnects. Yu et al. [298] proposed an acceleration framework to optimize Hadoop for fast data movement and a network-levitated merge algorithm. The reduce task gets the intermediate data from the map output and stores it locally in the memory, which leads to multiple disk access and I/O operations. The proposed algorithm overcomes this by fetching only a header of the segment instead of the whole segment.

Dhabaleshwar. K Panda [299] emphasizes the effectiveness of using InfiniBand in terms of cost for large-scale clusters compared to its counterpart, the standard Ethernet. Most of the HPC-based map-reduce solutions (RDMA, DataMPI [296], and HMOR) are affected by the degree of change in the default Hadoop framework to exploit the benefits of high-speed interconnects, but Mellanox UDA [300,301] and IP over InfiniBand (IPoIB) [302] require minimum-to-no changes to the Hadoop configuration. Hadoop is linearly scalable and with the increasing size of clusters, organizations started using InfiniBand and solid-state drives (SSDs). InfiniBand along with RDMA delivered almost four times the bandwidth of a 10GigaBit Ethernet port [299].

According to project Aloja [303], there are numerous Hadoop performance tuneable parameters like Hardware, RAM capacity, storage type, HDFS block size, number of mappers and reducers, network speed, etc. According to their findings, adding InfiniBand does not improve the performance but using it with solid state drives (SSDs) delivered 3.5X better performance compared to Gigabit Ethernet. Islam et al. [304] identifies different challenges of pipelined replication schemes and proposes an alternative parallel replication scheme and compared the performance of the latter with existing pipelined replication in HDFS over Ethernet, IPoIB, 10 GigE, and RDMA and showed performance enhancement with a parallel model for large data sizes and high-performance interconnects.

Lu et al. [295] highlighted the potential benefits of integrating Spark and the RDMA framework and proposed an RDMA-based solution to accelerate data shuffling in Spark by using high-performance interconnects. Similarly, Wasi-ur-Rehman et al. [305] proposed Hadoop map-reduce over InfiniBand using RDMA, Islam et al. [306] presented HDFS with RDMA over InfiniBand and Lu et al. [307] proposed a Hadoop RDMA-based Hadoop RPC over InfiniBand.

5.3. Map-Reduce-like Framework for In Situ Analysis

Scientific applications are often run on High-performance computing clusters, followed by offline data analysis tasks on smaller clusters. The expense of CPU hours on High-End Computing (HEC) machines is one of the main reasons for this offline cluster analysis. So, the compute-intensive simulations are run on the HEC machine and data analysis tasks are performed on smaller clusters after the completion of the simulations. This approach has several disadvantages in terms of performance, energy consumption, and redundant I/O, which ultimately results in an increase in data traffic between compute and storage subsystems [165]. Scientific datasets are stored in backend storage servers in HPC environments and these datasets can be analyzed by the YARN map-reduce program on compute nodes. As both compute and storage servers are separated in HPC environments, the cost of moving these large datasets is very high. Wang et al. [282] proposed a map-reduce-like framework for in situ data analysis that requires minimal modification to the simulation code. Compared to traditional map-reduce, their system performs the analysis task by fetching data directly from memory in each node and keeps memory utilization low by avoiding key-value pair output. They evaluated the system by using different scientific simulations on both multicore and many-core clusters with minimum overhead. Although many HPC systems have exploited in situ data analysis, there is still a need for the efficient analysis of data stored in the backend storage system. Xu et al. [283] proposed a virtualized Analytics Shipping (VAS) framework with fast network and disk I/O for efficient shipping of map-reduce programs to Lustre storage servers. Spark on-demand allows users to use Apache Spark for in situ data analysis of big data on HPC resources [284]. With this setup, there is no longer a need to move petabytes of data for advance data analytics. Table 2 summarizes the research efforts related to the convergence of HPC and big data along with the challenges and future directions.

Table 2. Data Locality-aware HPC and Big Data Convergence Efforts.

Convergence	Convergence Efforts	Challenges/Future Directions
MPI with Map-Reduce	Hoefler et al. [287], MPI, ad-hoc Hadoop [193], CloudBlast [288], HTCondor [289], Zhang et al. [290], Lu et al. [291], DataMPI [292], Mohamed et al. [293], Pilot-Jobs [308], Pregel [118], Apache Hama [119] and Giraph [120], SRM [275], iRODS [276], MapReduce-MPI [277], Pilot-MapReduce [278].	• Programming Abstraction • Minimizing Complexity (Degree of change of default MapReduce). • Improving Parallel Replication Scheme • Adaptability • Innovation in Data placement and data access strategies • Improving data layout strategies

Table 2. *Cont.*

Convergence	Convergence Efforts	Challenges/Future Directions
Map-Reduce with High-Performance Interconnects	DataMPI [296], [240], Yandang et al. [297], Yu et al. [298], Dhabaleshwar. K Panda [299], Mellanox UDA [300,301], IP over InfiniBand (IPoIB) [302], Aloja [303], Islam et al. [304], Lu et al. [295], Wasi-ur-Rehman et al. [305], Islam et al. [306], Lu et al. [307]	• Scalability • Complexity • High Bandwidth and Low-latency interconnects • Efficient Data transfer • Energy Efficiency • Compatibility • S/w H/W co-design
In Situ Analysis	Wang et al. [282], Xu et al. [283], Spark on demand [284].	• Data Volume • Data Management • Energy Efficiency • Complexity • Cognitive computing and storage

6. Challenges, Opportunities, and Future Directions

HPC and big data are different paradigms (compute-centric vs. datacentric) but also have different software ecosystems. The convergence of both these paradigms demands collaborative efforts at different levels of their ecosystems. Hadoop is relatively new but has matured over the years and has started to support different heterogeneous workloads [44], especially with the introduction of YARN [309] and Mesos [114]. Pilot-Jobs [308] and other tools emerged for data-intensive jobs in HPC environments but lack the scalability of Hadoop [44]. Table 3 summarizes the differences between big data (Hadoop) and HPC ecosystems.

Table 3. HPC vs. Hadoop Ecosystems.

	Big Data	HPC
Programming Model	Java Applications, SparQL	Fortran, C, C++
High-level Programming	Pig, Hive, Drill	Domain-specific Language
Parallel run time	Map-reduce	MPI, Open MP, OpenCL
Data Management	HBase, MySQL	iRODS
Scheduling (Resource Management)	YARN	SLRUM (Simple LINUX utility for resource management)
File system	HDFS, SPARK (Local storage)	LUSTRE (Remote storage)
Storage	Local shared-nothing architecture	Remote shared parallel storage
Hardware for Storage	HDDS	SSD
Interconnect	Switch Ethernet	Switch Fiber
Infrastructure	Cloud	Supercomputer

As we are heading towards exascale systems, achieving billion-fold parallelism within energy constraints is an extremely challenging task. The explosion of data being produced at a brisk speed brings many challenges that may include, but are not limited to, minimized data movement, data locality, data storage, effective and efficient searching algorithms, and data analysis.

Reed et al. [18] identified several exascale challenges including (1) High bandwidth and low latency interconnect technologies that also require locality-aware algorithms for efficient data transfer. (2) Advances in memory technology that directly influences data movement and energy constraints. (3) Data management software to handle a massive amount of data and efficient in situ data analysis requires some revolutionary changes in applications and scientific workflows. (4) Programming models for expressing parallelism and data locality, alleviating programmers' burden of expressing billion-fold parallelism and fault-handling.

Data locality thus is coined as one of the major issues the computing research community is facing for exascale systems and is currently managed in petascale systems at the application, file system, or middle level. Parallel file systems, e.g., Lustre [279], GPFS [192], and PVS [280] are bound to be compatible with POSIX (Portable Operating System Interface) for maintaining compatibility between operating systems but compromise data locality. On the other hand, the Google File System (GFS) uses the map-reduce processing framework to avoid being POSIX-constrained. These frameworks represent the best efforts to bring computation to data and are widely used in data-intensive applications. However, these techniques tailored for data-centric computation are not well-suited, and, therefore, not widely adopted, by the HPC community [310]. Runtime performance depends on efficient task scheduling by optimally allocating tasks to the target architecture. Over-provisioning of parallel work leads to threads spending a long time in the waiting queues for required resources and under-provisioning leads to underutilization of resources. Locality-aware performance optimization of an application on multicore architecture is challenging due to the shared, hybrid, and distributed memory architecture with several hierarchies determined by non-uniform communication latencies. Power consumption and bounding the energy for exascale computation is perhaps one of the major challenges. An additional challenge is dealing with sparse resilience, as hardware failure is a norm in an exascale environment. The increasing complexity regarding memory hierarchy and resilience demands opting for different approaches to software development, i.e., domain-specific languages and compilers, auto-tuning software, language constructs and tools to deal with massive parallelism, etc. The following are the key challenges and opportunities that require careful consideration.

6.1. Programming Paradigms

Most of the current programming paradigms in the HPC and big data environments (MPI, OpenMP, OpenCL, map-reduce) do not meet the needs of exascale computing and this issue demands thorough reinvestigation [311]. Hybrid approaches also need some innovation more specifically on locality-based communication to address scalability and billion-fold parallelism. The Partitioned Global Address Space PGAS model offers a rich set of functionalities, and its different implementations present a multi-threaded view, while MPI depicts a fragmented data view. Load balancing, increasing complexity with scalability, and a lack of hierarchical decomposition are some of the constraints of PGAS that limited its growth to address issues related to a converged exascale system [310]. OpenCL can be a potential candidate due to its portability but is criticized for being too low-level, leaving complexity to the programmer to handle data transfer, synchronization, etc. None of the programing paradigms actually fit in the exascale era and there is a need to build a new programming model to overcome the limitations of current programing infrastructures. There is also a need for efficient data placement and data access strategies to reduce communication and the cost of data movement. The complexities of handling these low-level details compelled engineers to focus on high-level optimization by focusing more on minimizing communication.

6.2. Programming Models and Language Support

The evolution of software productivity could not match the speed at which hardware and network technologies have evolved over the years. There is a need for some evolutionary and revolutionary changes at different levels of software ecosystems to address the issues related to the converged HPC and big data environment. Performance optimization in terms of Gflops/sec is no longer viable in today's world as energy consumption is one of the primary concerns for exascale systems. Energy efficiency is directly affected by data locality, which, in turn, can be achieved by bringing computation to data, minimizing data movement by the efficient exploitation of cache hierarchies, reducing communication, locality-aware process/thread mapping, and in situ/in transit data analysis. There is a need to invest considerable effort in investigating locality-aware programming models

with compiler support, runtime environments, high-level languages, and abstraction strategies to build flexible, dynamic systems to capitalize multi/many-core architectures with complex memory hierarchies. There is a trade-off between performance and portability that needs to be clearly understood. Compilers need innovation and runtime systems need intelligence for efficient exploitation of data locality and performance at runtime. At the application level, algorithms must address data locality, load balancing, scalability, and communication [62]. The application and data interoperability issue between different programming models/languages also needs further investigation. Optimizing the performance of exascale systems may require automated approaches to deal with billion-fold parallelism and software support to facilitate application development.

6.3. Programming Abstractions

There is a need for efficient high-level programming, exploiting dynamic parallelism, data locality through the abstraction layer, and complex structure abstractions [27] to facilitate and assist programmers to address data locality issues and resource abstractions (from physical resources, i.e., memory, cores) for effective resource utilization in the converged HPC and big data environment. Data-centric abstraction exploitation is needed for the converged HPC and big data environment to alleviate the programmer's burden to deal with heterogeneous systems.

6.4. Innovations in Data Layout Strategies

Data locality issues must be addressed at different layers of the input/output stack, which demands strategic, dynamic, adaptive, and predictive methodologies to collocate the computation and data [310]. There is also a need for locality-aware data distribution based on the runtime behavior of the application, which is unknown at compile time. One way of addressing this problem is the use of machine learning-based techniques to predict runtime behavior by capitalizing low-overhead profiling tools to extract the runtime features (runtime data mobility from backend storage to application) that can be used to train the predictor for optimal code/core selection to minimize communication. Intelligent data placement algorithms need to be investigated for reducing unnecessary communication.

6.5. Locality-Aware Scheduling

As the number of computing resources is on a rise with complex memory hierarchies and heterogeneity making locality-aware scheduling and resource management a challenging task that cannot be handled efficiently by the current centralized scheduling systems, there is a need for adaptive distributed job scheduling management infrastructure, re-engineering of locality-aware scheduling algorithms, and amalgamation of data locality with an allocation mechanism but without compromising the scalability and energy constraints. AI-assisted approaches such as those described in [57,312,313] would play an important role in this direction.

6.6. Software Hardware Co-Design

There is a need for coordinated efforts in software-hardware co-design to address the challenges of future-generation systems with the ability to handle applications from multi-dimensional domains. The mapping of application and system software should also be aided by co-design [62] so that data-locality issues can be addressed by minimizing communication.

6.7. Innovations in Memory and Storage Technologies

Minimizing data movement demands innovation in memory technologies. Classical DRAM and SRAM may not be suitable for future systems. New non-volatile memory technologies are emerging but are still in their infancy. There is a need for an integrated software stack to address the issues of I/O requirements of data-centric applications via

network/storage aggregation and efficient utilization of client-side cache, along with server-side optimized utilization of storage management.

6.8. High-Speed Interconnects

The scalability of the communication bandwidth of high-speed interconnects must match that of the increasing processing capability of a node with multi/many cores. The compact integration of interconnects must minimize remote data access latency by providing high bandwidth and low-latency efficient interconnection within energy constraints.

Due to the increase in system concurrency, parallel and distributed programing have become a major issue. Programming to achieve billion-fold parallelism faces many challenges including power consumption, memory, communication, fault tolerance, and heterogeneity. These challenges directly or indirectly affect data locality and require innovation at both the hardware and software levels. Data locality abstraction, available in the forms of libraries, runtime systems, data structures, and language, needs innovation to increase productivity without compromising performance.

7. Proposed Future HPC and Big Data Converged System Architecture

As the complexity of computer systems is on the rise with the number of cores per processor, different levels of cache, processors per node, and high-speed interconnects, there is an ever-growing need for new optimization techniques to minimize communication and an efficient way of exploiting parallelism for heterogeneous resources. Also, computation is getting cheaper; there is a need for a paradigm shift from compute-centric to data-centric. The current programming environments do not fully facilitate the abstraction to optimize data locality resulting in programmers having to use other means for locality-aware optimization. There is a potential scope to exploit locality by providing eco-friendly, environmentally responsible computations and innovations in algorithmic research, i.e., communication-avoiding algorithms and a need for automated parallelism mapping on target system software and hardware architecture [62].

Implementing codes with system characteristics in mind (minimizing communication, avoiding cache misses by using blocks that fit in the cache) and mapping processes to specific cores are the two main approaches studied to improve the performance of parallel applications in multicore architectures. Optimal task mapping requires heuristics to find the best mapping strategy but these heuristics require the runtime behavior of these tasks to be known, which, in turn, requires static and dynamic analysis to facilitate automatic mapping. One of the major challenges to mapping the parallel executable tasks is to obtain the needed runtime behavior of these tasks.

One way of looking at data locality issues in converged HPC and big data environments is by effectively and efficiently utilizing resources that should ultimately minimize communication. Once the code is parallelized, there is a need for efficient code mapping to the underlying architecture for efficient exploitation of hardware resources. The optimal mapping decision is non-trivial and depends not only on the parallel algorithm but also on the relative costs of communication and computation. Expert programmers can implement effective mapping, but this manual process is expensive and error-prone. Although manual mapping can be effective on a particular architecture where the programmer is responsible for all the issues, i.e., load-balancing, synchronization, communication, etc., this solution is not portable and needs considerable effort for the code to run on a different platform [158].

Compiler-based mapping techniques can be an alternative to parallelism mapping but manual tuning of compiler-based approaches is complex to handle with the increasing level of hardware complexity and application diversity. Researchers engineered numerous compiler-based heuristics (mostly platform-specific) over the years, based on analytical models to optimize compilation decisions. Feedback-directed or iterative compiling produces multiple versions of a program and empirically measures performance by actually running code on target hardware. This measured performance acts as feedback for the selection of the best among different options. This does require an exhaustive search space

for an optimal solution with an expensive compilation overhead. To overcome these issues, predictive modeling came into the picture but much of the work in the literature is based on sequential programs [158,314]. Predictive techniques predict optimization (without executing them on target hardware or in a simulator) based on previous knowledge generated by offline training with different compilation options, which are then used to predict the best among available options. In contrast to static compilation techniques, dynamic adaption makes runtime decisions based on dynamic environment information, i.e., the number of processing elements and the program's execution environment for dynamic scheduling of parallel programs.

Exascale systems are expected to have millions of components with deep hierarchical structures both horizontally (network interconnection) and vertically (memory architecture, i.e., cache hierarchies, NUMA). To achieve massive scalability, there is a need for the efficient exploitation of resources but within energy constraints that can be achieved by addressing data locality, i.e., how data is placed, accessed, and moved across complex system hierarchies (horizontally and vertically). We believe that future computing systems will require research in three main dimensions: characterization of workloads, characterization of the system resources, and smart ways to map the workloads to the underlying system resources under multiple constraints with configurable preferences. We have highlighted some of the work related to process mapping by categorizing them as algorithmic approaches and machine learning-based techniques. There is a need to dedicate efforts to automated approaches to provide a portable mapping solution, which models the interaction of parallel applications and the underlying architecture effectively and efficiently. Machine/deep learning thus can be applied to automate the process of mapping workload to processing cores for optimal load balancing and scalability, and address the locality issues either by avoiding or minimizing communication. Mapping application tasks on a multi/many-core system involves the assignment and ordering of the tasks and their communications onto the platform resources. The communicating tasks are mapped on the same core or close to each other in order to optimize the communication delay and energy efficiency. There is a need to adapt environmentally friendly green computing practices including green production, recycling, and development and design.

Figure 1 gives an abstract view of machine learning and design patterns-based solution for the converged HPC and big data environment. Design patterns can be seen as repeatable general solutions to commonly occurring problems in software design to allow flexible and robust development. The standardization and organization of design patterns in both HPC and big data environments can be seen as potential candidates for this inevitable convergence. These design patterns serve here as a catalyst to produce a dataset that is used for the training of AI-based models to address convergence challenges, i.e., data locality, energy efficiency, fault tolerance, etc. These challenges and growing demands of HPDA to speed up data analysis requires revolutionary and evolutionary changes at different levels of HPC and big data ecosystems. Generic guidelines provided by these design patterns would help software developers to design robust and energy-efficient solutions. High bandwidth requirements and increasing network complexity further increase energy consumption and heat emission. So, there is a need to shift the focus to computing with renewable energy and green solutions for the converged HPC and big data environment.

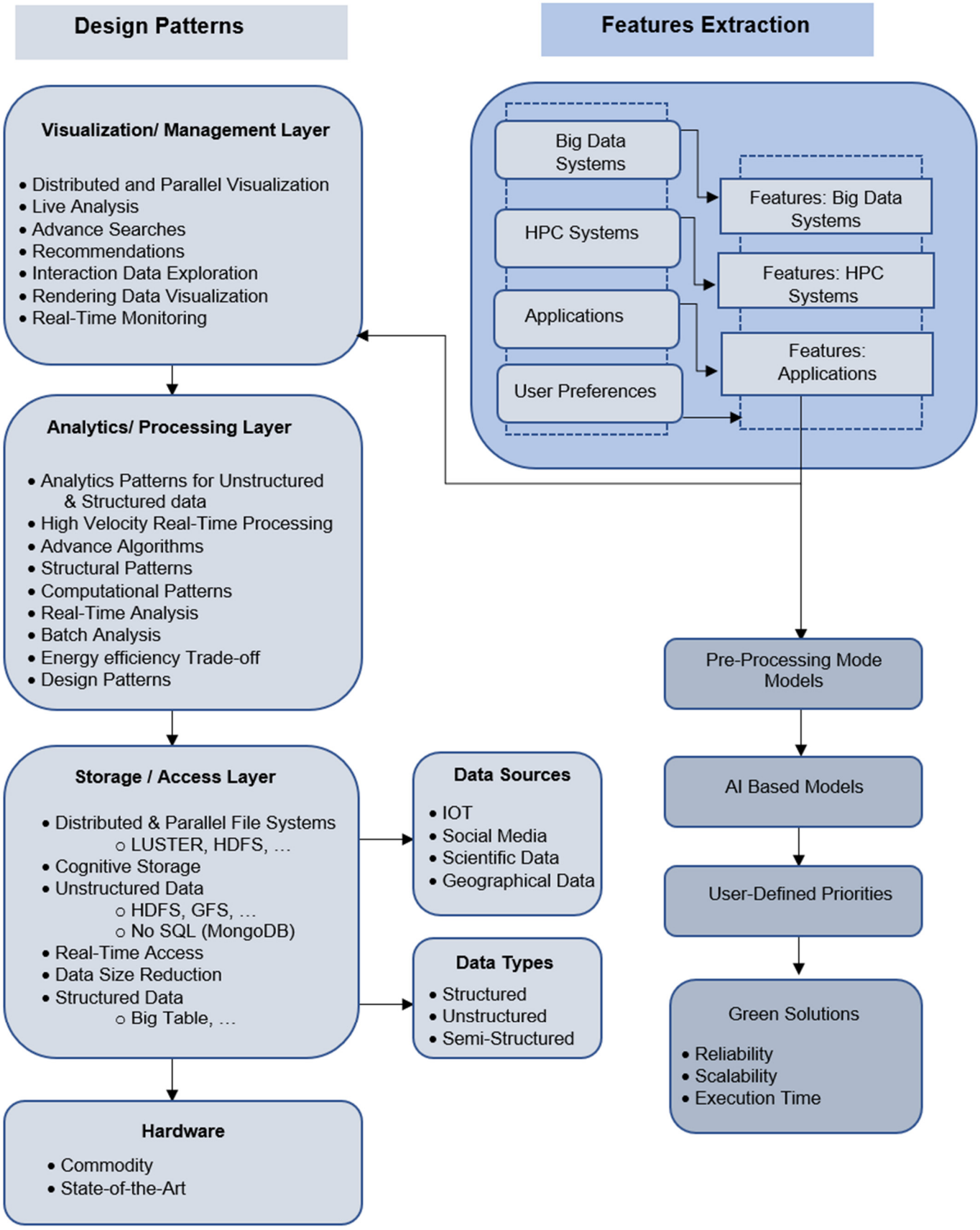

Figure 1. Design patterns and AI-based Architecture for Converged HPC and Big Data Environments.

The feature extractor extracts the required static and dynamic features, which can then be used for preparing training data for AI-based models along with an optimal solution based on the user's preferences. Software processing entities (processes, threads) and their dependencies can be expressed as an application virtual topology, e.g., messages exchanged between processes in message-passing models (MPI) and access to a common memory location (OpenMP). These software-processing entities need to communicate with each other regularly or irregularly, which demands efficient utilization of available resources to facilitate data access and communication. The characteristics of the underlying hardware architecture need to be gathered in a portable way to target a broad range of architectures. The cluster environment demands network topology, and the multi/many-core environment demands intra-node structure information to be accumulated in a comprehensive way, in order to map the application's virtual topology on the targeted physical topology. The memory access behavior for each task needs to be gathered with minimum overhead, as many existing memory tracing techniques are based on simulating applications with a large overhead. Hardware counters are also used for this purpose but have low accuracy due to the sampling of application memory accesses. An alternative way is Dynamic Binary Instrumentation (Valgrind, MemTrace) but this also has a lack of direct support for parallel applications and overhead and accuracy issues. There is a need for more sophisticated tools that provide us with all the necessary information with minimum overhead and maximum accuracy [315]. As discussed above, the efficient use of virtual topology, communication patterns, underlying hardware characteristics, network topology, and memory access behavior, helps us to collect quality data features to prepare training data for the AI-based models.

In a distributed environment, there is a need for innovation in message-passing models to effectively utilize the internal communication pipeline based on the underlying network topology. Current supercomputing systems use job schedulers for resource allocation (SLRUM, PBS) but these do not consider the application communication requirements and lack the intelligence to map tasks to the underlying topology. The complexity will increase further with future systems with millions of cores arranged in multiple levels of hierarchies, multiple sockets with a number of cores, nodes with multiples sockets, blades with multiple nodes, multiple blades arranged in racks, and the whole system arranged in multiple racks, and interconnection of these components with complex network topologies. The issues related to resource contention, routing schemes of underlying network topology, scalability, job scheduling, and process mapping need further investigation to achieve scalable performance. There is a need for topology-aware communication libraries and schedulers to use runtime network information to make intelligent scheduling decisions [316] and also consider the simultaneous mapping of concurrent applications to heterogeneous resources.

The use of machine learning helps developers to automatically engineer dynamic optimization strategies and runtime adaption methodologies to cater to changing program behaviors. Runtime adaptions can be facilitated by the dynamic configuration of hardware using machine-learning techniques. Capturing the static program features along with dynamic features can be used to predict application behavior, which can then be used to configure hardware resources. There is a need to find a synergy between high-level optimizations for parallelism mapping and low-level compiler transformations by carefully considering the trade-offs for an optimized mapping [317].

Design patterns, thus, can be used at different layers of the software ecosystem to address a broad range of challenges including scalability, elasticity, adaptability, robustness, locality, and storage with solutions that are already tested and implemented in similar or closely related environments, e.g., parallel design patterns (Our Pattern Language OPL) [318] and big-data patterns [319]. Both OPL and big-data patterns are organized in a logical architecture of different layers. Figure 1 shows the layered architecture of design patterns, which act as a catalyst to produce high-quality data for AI-based models to address different problems based on the user's preferences including fault tolerance, parallelization mapping, which ultimately improves data locality, and energy efficiency,

thus providing green solutions for the converged HPC and big data environment. Applying multiple design patterns is itself a tedious task due to the diverse nature of these design patterns and there are few efforts to address these challenges using machine-learning techniques [320,321]. These design patterns assist (both HPC and big data) software developers to design and address data locality issues efficiently. Designing software architecture is a complex process and requires the identification of quality attributes that must be in-line with functional requirements. As these processes are done in multiple phases, therefore, applying a systematic approach to designing a software architecture is of fundamental importance. Software architecture patterns are not straight forward to apply and demand an automated process to carefully consider different factors such as stake holder's expectations, functional and nonfunctional requirements, etc.

Design solutions for data visualization and interactive management are hard to assess and reapply. At the abstract level, different design patterns at the visualization and management layer are defined to address distributed, parallel, interactive, and live data visualization and analysis. Most of the available tools don't provide a built-in provision to facilitate data visualization and information distribution at scale. The processing layer gathers structured, unstructured, and semi-structured data from heterogeneous resources and makes it available after processing for the rest of the pipeline. It supports both batch and real-time processing. The processing layer includes design patterns, high-velocity real-time processing, large-scale batch/graph analysis, strategy patterns, data conversion, structural and computational patterns, etc. The trade-off between fault tolerance, performance, and energy efficiency needs careful monitoring by incorporating best practices from both the HPC and big data environments. One of the major issues in the data ingestion process is to keep data in the right place. Data ingestion systems help to transfer data in the form of events or messages to other applications and data stores, which allows reliable data processing. Data storage is critical to facilitate access at scale, as data are stored in numerous formats. Big data analytics and strategic enterprise applications demands/require different varieties of data. Here, structured and unstructured data is sourced from heterogeneous resources including IoT, scientific simulations, social media, etc., all of which have different scalability and availability requirements. Parallel and distributed design patterns for smart storage and efficient retrieval are organized at the storage and access layer, as shown in Figure 1. The diverse storage options provide different characteristics in terms of data availability, scalability, and performance. These design patterns provide cognitive storage (automated purging), data size reduction, and high-volume hierarchical, linked, tabular, and binary storage with indirect and integrated data access.

Limitations of Proposed Solution

Implementing design patterns in applications is proven and tested. They are evolved as reusable components and make our application reliable, scalable, and easily maintainable. At the same time, using a design pattern itself is a tedious task due to the high volume and complexity of these design patterns. This requires developers to learn and test different patterns and choose the optimal one, which is a time-consuming process due to their vague and abstract nature. The use of design-pattern recognition based on their theoretical description results in a poor assignment. The proposed solution is very much dependent on the quality of data being produced by the use of HPC and big data design patterns. Due to the high volume and application complexity of these design patterns, there is a need to focus on the exploration of innovative approaches to automate the process of applying these design patterns to address convergence challenges. At the same time, design solutions for data visualization and interactive management are hard to assess and reapply.

The use of AI techniques, standardization, and efficient exploitation of design patterns can lead to an effective and systematic solution to address the locality and other challenges of converged HPC and big-data environments.

8. Conclusions

This paper presented a review of cutting-edge research on data locality in HPC, big data, and converged systems. The efficient resource utilization and performance of parallel applications demand intelligent parallelism mapping to the target architecture but within energy constraints. Numerous research efforts dedicated to data locality issues at different levels of HPC and big-data software ecosystems are presented. The paper also reviewed research efforts for the inevitable convergence of HPC and big data primarily focusing on locality and highlighted some of the challenges that need further investigation. To address these challenges, we also presented future trends and proposed a solution to be considered as a future direction. The explosion of data more specifically in the edge environment from multidisciplinary domains (IoT, smart cities, and remote sensors, i.e., satellite imagery) demands the investment of research efforts towards "fog or edge" computing infrastructure to provide storage, processing, and communication facilities for the integrated HPC and big data environment. So, there is a need to promote the development of software libraries for intermediate processing. The connected and ubiquitous synergy between HPC and big data demands the exploration of some revolutionary and evolutionary innovations and coordinated efforts at different levels of integrated software ecosystems. The use of design patterns, cognitive computing (machine learning, natural language processing), and intelligent process mapping can be seen as potential candidates to address data locality and other challenges for the integrated HPC and big data environment.

Author Contributions: Conceptualization, S.U. and R.M.; methodology, S.U. and R.M.; validation, S.U., R.M., I.K. and A.A.; formal analysis, S.U., R.M., I.K. and A.A.; investigation, S.U., R.M., I.K. and A.A.; resources, R.M., I.K. and A.A.; data curation, S.U.; writing—original draft preparation, S.U. and R.M.; writing—review and editing, R.M., I.K. and A.A.; visualization, S.U.; supervision, R.M. and I.K.; project administration, R.M., I.K. and A.A.; funding acquisition, R.M., I.K. and A.A. All authors have read and agreed to the published version of the manuscript.

Funding: The authors acknowledge with thanks the technical and financial support from the Deanship of Scientific Research (DSR) at the King Abdulaziz University (KAU), Jeddah, Saudi Arabia, under Grant No. RG-10-611-38.

Data Availability Statement: Not applicable.

Acknowledgments: The work carried out in this paper is supported by the HPC Center at the King Abdulaziz University.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, M.; Mao, S.; Liu, Y. Big Data: A Survey. *Mob. Netw. Appl.* **2014**, *19*, 171–209. [CrossRef]
2. Farber, R. The Convergence of Big Data and Extreme-Scale HPC, HPC Wire. 2018. Available online: https://www.hpcwire.com/2018/08/31/the-convergence-of-big-data-and-extreme-scale-hpc/ (accessed on 1 November 2022).
3. Alam, F.; Almaghthawi, A.; Katib, I.; Albeshri, A.; Mehmood, R. iResponse: An AI and IoT-Enabled Framework for Autonomous COVID-19 Pandemic Management. *Sustainability* **2021**, *13*, 3797. [CrossRef]
4. Alomari, E.; Katib, I.; Albeshri, A.; Yigitcanlar, T.; Mehmood, R. Iktishaf+: A Big Data Tool with Automatic Labeling for Road Traffic Social Sensing and Event Detection Using Distributed Machine Learning. *Sensors* **2021**, *21*, 2993. [CrossRef] [PubMed]
5. Alkhayat, G.; Hasan, S.H.; Mehmood, R. SENERGY: A Novel Deep Learning-Based Auto-Selective Approach and Tool for Solar Energy Forecasting. *Energies* **2022**, *15*, 6659. [CrossRef]
6. Alahmari, N.; Alswedani, S.; Alzahrani, A.; Katib, I.; Albeshri, A.; Mehmood, R. Musawah: A Data-Driven AI Approach and Tool to Co-Create Healthcare Services with a Case Study on Cancer Disease in Saudi Arabia. *Sustainability* **2022**, *14*, 3313. [CrossRef]
7. Alswedani, S.; Mehmood, R.; Katib, I. Sustainable Participatory Governance: Data-Driven Discovery of Parameters for Planning Online and In-Class Education in Saudi Arabia During COVID-19. *Front. Sustain. Cities* **2022**, *4*, 97. [CrossRef]
8. Alaql, A.A.; AlQurashi, F.; Mehmood, R. Data-Driven Deep Journalism to Discover Age Dynamics in Multi-Generational Labour Markets from LinkedIn Media. Mathmatics & Computer Science. *Preprints* **2022**, 2022100472. [CrossRef]
9. Alqahtani, E.; Janbi, N.; Sharaf, S.; Mehmood, R. Smart Homes and Families to Enable Sustainable Societies: A Data-Driven Approach for Multi-Perspective Parameter Discovery Using BERT Modelling. *Sustainability* **2022**, *14*, 13534. [CrossRef]
10. Janbi, N.; Mehmood, R.; Katib, I.; Albeshri, A.; Corchado, J.M.; Yigitcanlar, T. Imtidad: A Reference Architecture and a Case Study on Developing Distributed AI Services for Skin Disease Diagnosis over Cloud, Fog and Edge. *Sensors* **2022**, *22*, 1854. [CrossRef]

11. Arfat, Y.; Usman, S.; Mehmood, R.; Katib, I. Big data tools, technologies, and applications: A survey. In *Smart Infra-Structure and Applications Foundations for Smarter Cities and Societies*; Springer: Cham, Switzerland, 2020; pp. 453–490.
12. Mehmood, R.; Sheikh, A.; Catlett, C.; Chlamtac, I. Editorial: Smart Societies, Infrastructure, Systems, Technologies, and Applications. *Mob. Netw. Appl.* **2022**, *1*, 1–5. [CrossRef]
13. Yigitcanlar, T.; Butler, L.; Windle, E.; DeSouza, K.C.; Mehmood, R.; Corchado, J.M. Can Building "Artificially Intelligent Cities" Safeguard Humanity from Natural Disasters, Pandemics, and Other Catastrophes? An Urban Scholar's Perspective. *Sensors* **2020**, *20*, 2988. [CrossRef] [PubMed]
14. Yigitcanlar, T.; Corchado, J.M.; Mehmood, R.; Li, R.Y.M.; Mossberger, K.; Desouza, K. Responsible Urban Innovation with Local Government Artificial Intelligence (AI): A Conceptual Framework and Research Agenda. *J. Open Innov. Technol. Mark. Complex.* **2021**, *7*, 71. [CrossRef]
15. Yigitcanlar, T.; Mehmood, R.; Corchado, J.M. Green Artificial Intelligence: Towards an Efficient, Sustainable and Equitable Technology for Smart Cities and Futures. *Sustainability* **2021**, *13*, 8952. [CrossRef]
16. Alsaigh, R.; Mehmood, R.; Katib, I. AI Explainability and Governance in Smart Energy Systems: A Review. *arXiv* **2022**, arXiv:arXiv:2211.00069. [CrossRef]
17. Schwartz, R.; Dodge, J.; Smith, N.A.; Etzioni, O. Green AI. *Commun. ACM* **2020**, *63*, 54–63. [CrossRef]
18. Reed, D.A.; Dongarra, J. Exascale computing and big data. *Commun. ACM* **2015**, *58*, 56–68. [CrossRef]
19. Elia, D.; Fiore, S.; Aloisio, G. Towards HPC and Big Data Analytics Convergence: Design and Experimental Evaluation of a HPDA Framework for eScience at Scale. *IEEE Access* **2021**, *9*, 73307–73326. [CrossRef]
20. Brox, P.; Garcia-Blas, J.; Singh, D.E.; Carretero, J. DICE: Generic Data Abstraction for Enhancing the Convergence of HPC and Big Data. In Proceedings of the Latin American High Performance Computing Conference, Guadalajara, Mexico, 6-8 October 2021; pp. 106–119. [CrossRef]
21. Hachinger, S.; Martinovič, J.; Terzo, O.; Levrier, M.; Scionti, A.; Magarielli, D.; Goubier, T.; Parodi, A.; Harsh, P.; Apopei, F.-I.; et al. HPC-Cloud-Big Data Convergent Architectures and Research Data Management: The LEXIS Approach. *Int. Symp. Grids Clouds* **2021**, *378*, 4. [CrossRef]
22. Karagiorgou, S.; Terzo, O.; Martinovič, J. CYBELE: On the Convergence of HPC, Big Data Services, and AI Technologies. In *HPC, Big Data, and AI Convergence Towards Exascale*; CRC Press: Boca Raton, FL, USA, 2022; pp. 240–254.
23. Tzenetopoulos, A.; Masouros, D.; Koliogeorgi, K.; Xydis, S.; Soudris, D.; Chazapis, A.; Kozanitis, C.; Bilas, A.; Pinto, C.; Nguyen, H.; et al. EVOLVE: Towards converging big-data, high-performance and cloud-computing worlds. In Proceedings of the 2022 Design, Automation\& Test in Europe Conference\& Exhibition (DATE), Antwerp, Belgium, 14–23 March 2022; pp. 975–980.
24. Ejarque, J.; Badia, R.M.; Albertin, L.; Aloisio, G.; Baglione, E.; Becerra, Y.; Boschert, S.; Berlin, J.R.; Anca, A.D.; Elia, D.; et al. Enabling dynamic and intelligent workflows for HPC, data analytics, and AI convergence. *Futur. Gener. Comput. Syst.* **2022**, *134*, 414–429. [CrossRef]
25. Sukumar, S.R.; Balma, J.A.; Rickett, C.D.; Maschhoff, K.J.; Landman, J.; Yates, C.R.; Chittiboyina, A.G.; Peterson, Y.K.; Vose, A.; Byler, K.; et al. The Convergence of HPC, AI and Big Data in Rapid-Response to the COVID-19 Pandemic. In *Smoky Mountains Computational Sciences and Engineering Conference*; Springer: Cham, Switzerland, 2021; pp. 157–172.
26. Scionti, A.; Viviani, P.; Vitali, G.; Vercellino, C.; Terzo, O. Enabling the HPC and Artificial Intelligence Cross-Stack Con-vergence at the Exascale Level. In *HPC, Big Data, and AI Convergence Towards Exascale*; CRC Press: Boca Raton, FL, USA, 2022; pp. 37–58.
27. Unat, D.; Dubey, A.; Hoefler, T.; Shalf, J.; Abraham, M.; Bianco, M.; Chamberlain, B.L.; Cledat, R.; Edwards, H.C.; Finkel, H.; et al. Trends in Data Locality Abstractions for HPC Systems. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 3007–3020. [CrossRef]
28. Mohammed, T.; Albeshri, A.; Katib, I.; Mehmood, R. UbiPriSEQ—Deep Reinforcement Learning to Manage Privacy, Security, Energy, and QoS in 5G IoT HetNets. *Appl. Sci.* **2020**, *10*, 7120. [CrossRef]
29. Janbi, N.; Katib, I.; Albeshri, A.; Mehmood, R. Distributed Artificial Intelligence-as-a-Service (DAIaaS) for Smarter IoE and 6G Environments. *Sensors* **2020**, *20*, 5796. [CrossRef] [PubMed]
30. Caragea, C.; Manegold, S. Memory Locality. In *Encyclopedia of Database Systems*; Springer: Boston, MA, USA, 2009; pp. 1713–1714.
31. Snir, M.; Yu, J. *On the Theory of Spatial and Temporal Locality*; University of Illinois ar Urbana-Champaign: Urbana, IL, USA, 2005.
32. Caíno-Lores, S.; Carretero, J. A Survey on Data-Centric and Data-Aware Techniques for Large Scale Infrastructures. *Int. J. Comput. Inf. Eng.* **2016**, *10*, 517–523.
33. Zhang, H.; Chen, G.; Ooi, B.C.; Tan, K.-L.; Zhang, M. In-Memory Big Data Management and Processing: A Survey. *IEEE Trans. Knowl. Data Eng.* **2015**, *27*, 1920–1948. [CrossRef]
34. Dolev, S.; Florissi, P.; Gudes, E.; Sharma, S.; Singer, I. A Survey on Geographically Distributed Big-Data Processing Using MapReduce. *IEEE Trans. Big Data* **2017**, *5*, 60–80. [CrossRef]
35. Senthilkumar, M.; Ilango, P. A Survey on Job Scheduling in Big Data. *Cybern. Inf. Technol.* **2016**, *16*, 35–51. [CrossRef]
36. Idris, M.; Hussain, S.; Ali, M.; Abdulali, A.; Siddiqi, M.H.; Kang, B.H.; Lee, S. Context-aware scheduling in MapReduce: A compact review. *Concurr. Comput. Pr. Exp.* **2015**, *27*, 5332–5349. [CrossRef]
37. Mozakka, M.; Esfahani, F.S.; Nadimi, M.H. Survey on Adaptive Job Schedulers in Mapreduce. *J. Theor. Appl. Inf. Technol.* **2014**, *31*, 661–669.
38. Nagina; Dhingra, S. Scheduling Algorithms in Big Data: A Survey. *Int. J. Eng. Comput. Sci.* **2016**, *5*, 11737–17743.
39. Kasiviswanath, N.; Reddy, P.C. A Survey on Big Data Management and Job Scheduling. *Int. J. Comput. Appl.* **2015**, *130*, 41–49.
40. Akilandeswari, H.; Srimathi, P. Survey on Task Scheduling in Cloud Environment. *IJCTA* **2016**, *9*, 693–698. [CrossRef]

41. Hoefler, T.; Jeannot, E.; Mercier, G.; Jeannot, E.; Žilinskas, J. *High-Performance Computing on Complex Environments*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2014; pp. 73–94.
42. Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. Mapping on multi/many-core systems. In Proceedings of the 50th Annual Design Automation Conference on—DAC '13, New York, NY, USA, 29 May 2013–7 June 2013; p. 1. [CrossRef]
43. Asaadi, H.; Khaldi, D.; Chapman, B. A Comparative Survey of the HPC and Big Data Paradigms: Analysis and Experiments. In Proceedings of the 2016 IEEE International Conference on Cluster Computing (CLUSTER), Taipei, Taiwan, 12–16 September 2016; pp. 423–432. [CrossRef]
44. Jha, S.; Qiu, J.; Luckow, A.; Mantha, P.; Fox, G.C. A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures. In Proceedings of the 2014 IEEE International Congress on Big Data, Anchorage, AK, USA, 27 June–2 July 2014; pp. 645–652. [CrossRef]
45. Asch, M.; Moore, T.; Badia, R.M.; Beck, M.; Beckman, P.; Bidot, T.; Bodin, F.; Cappello, F.; Choudhary, A.; De Supinski, B.; et al. Big data and extreme-scale computing. *Int. J. High Perform. Comput. Appl.* **2018**, *32*, 435–479. [CrossRef]
46. Yin, F.; Shi, F. A Comparative Survey of Big Data Computing and HPC: From a Parallel Programming Model to a Cluster Architecture. *Int. J. Parallel Program.* **2021**, *50*, 27–64. [CrossRef]
47. Golasowski, M.; Martinovič, J.; Levrier, M.; Hachinger, S.; Karagiorgou, S.; Papapostolou, A.; Mouzakitis, S.; Tsapelas, I.; Caballero, M.; Aldinucci, M.; et al. Toward the Convergence of High-Performance Computing, Cloud, and Big Data Domains. In *HPC, Big Data, and AI Convergence Towards Exascale*; CRC Press: Boca Raton, FL, USA, 2022; pp. 1–16.
48. Usman, S.; Mehmood, R.; Katib, I. Big Data and HPC Convergence for Smart Infrastructures: A Review and Proposed Architecture. In *Smart Infrastructure and Applications Foundations for Smarter Cities and Societies*; Springer: Cham, Switzerland, 2020; pp. 561–586.
49. Usman, S.; Mehmood, R.; Katib, I. Big Data and HPC Convergence: The Cutting Edge and Outlook. In *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*; Springer: Cham, Switzerland, 2018; Volume 224, pp. 11–26. [CrossRef]
50. Usman, S.; Mehmood, R.; Katib, I. HPC & Big Data Convergence: The Cutting Edge & Outlook, Poster presented. Proceedings of the first Middle East meeting of the Intel Extreme Performance Users Group, IntelXPUG, King Abdullah University of Science and Technology (KAUST), Jeddah, Saudi Arabia, 22–25 April 2018. Available online: https://epostersonline.com/ixpug-me2018/node/19 (accessed on 1 November 2022).
51. Alotaibi, H.; Alsolami, F.; Abozinadah, E.; Mehmood, R. TAWSEEM: A Deep-Learning-Based Tool for Estimating the Number of Unknown Contributors in DNA Profiling. *Electronics* **2022**, *11*, 548. [CrossRef]
52. Althumairi, A. 'Governmental Communication' launches the visual identity of the 'We are All Responsible' initiative to confront 'COVID 19'. *Int. J. Environ. Res. Public Health* **2021**, *18*, 282. [CrossRef]
53. Muhammed, T.; Mehmood, R.; Albeshri, A.; Alsolami, F. *HPC-Smart Infrastructures: A Review and Outlook on Performance Analysis Methods and Tools*; Springer: Cham, Switzerland, 2020; pp. 427–451.
54. Aqib, M.; Mehmood, R.; Alzahrani, A.; Katib, I.; Albeshri, A.; Altowaijri, S.M. Smarter Traffic Prediction Using Big Data, In-Memory Computing, Deep Learning and GPUs. *Sensors* **2019**, *19*, 2206. [CrossRef]
55. Muhammed, T.; Mehmood, R.; Albeshri, A.; Katib, I. UbeHealth: A Personalized Ubiquitous Cloud and Edge-Enabled Networked Healthcare System for Smart Cities. *IEEE Access* **2018**, *6*, 32258–32285. [CrossRef]
56. AlAhmadi, S.; Muhammed, T.; Mehmood, R.; Albeshri, A. *Performance Characteristics for Sparse Matrix-Vector Multi-Plication on GPUs*; Springer: Cham, Switzerland, 2020; pp. 409–426.
57. Mohammed, T.; Albeshri, A.; Katib, I.; Mehmood, R. DIESEL: A novel deep learning-based tool for SpMV computations and solving sparse linear equation systems. *J. Supercomput.* **2020**, *77*, 6313–6355. [CrossRef]
58. Muhammed, T.; Mehmood, R.; Albeshri, A.; Katib, I. SURAA: A Novel Method and Tool for Loadbalanced and Coalesced SpMV Computations on GPUs. *Appl. Sci.* **2019**, *9*, 947. [CrossRef]
59. Alahmadi, S.; Mohammed, T.; Albeshri, A.; Katib, I.; Mehmood, R. Performance Analysis of Sparse Matrix-Vector Multiplication (SpMV) on Graphics Processing Units (GPUs). *Electronics* **2020**, *9*, 1675. [CrossRef]
60. Alyahya, H.; Mehmood, R.; Katib, I. *Parallel Iterative Solution of Large Sparse Linear Equation Systems on the Intel MIC Architecture*; Springer: Cham, Switzerland, 2019; pp. 377–407. [CrossRef]
61. Mehmood, R.; Crowcroft, J. Parallel Iterative Solution Method for Large Sparse Linear Equation Systems. Technical Report Number UCAM-CL-TR-650, Computer Laboratory, University of Cambridge, Cambridge, UK, 2005. 2005. Available online: https://www.cl.cam.ac.uk/research/srg/netos/papers/MC05.pdf (accessed on 26 February 2016).
62. *Nicole Casal Moore*. Towards a Breakthrough in Software for Advanced Computing. Available online: https://cse.engin.umich.edu/stories/a-breakthrough-for-large-scale-computing (accessed on 24 August 2022).
63. Guest, M. The Scientific Case for High Performance Computing in Europe 2012–2020. *Tech. Rep.* 2012.
64. Matsuoka, S.; Sato, H.; Tatebe, O.; Koibuchi, M.; Fujiwara, I.; Suzuki, S.; Kakuta, M.; Ishida, T.; Akiyama, Y.; Suzumura, T.; et al. Extreme Big Data (EBD): Next Generation Big Data Infrastructure Technologies Towards Yottabyte/Year. *Supercomput. Front. Innov.* **2014**, *1*, 89–107. [CrossRef]
65. ETP4HPC, A. EuropEan Technology platform for High Performance Computing. In *ETp4hpc ETP4HPC*; Barcelona, Spain, 2013. Available online: https://www.etp4hpc.eu/pujades/files/ETP4HPC_book_singlePage.pdf (accessed on 1 November 2022).
66. Hoefler, T.; Jeannot, E.; Mercier, G. *An Overview of Topology Mapping Algorithms and Techniques in High-Performance Computing*; Wiley-IEEE Press: Hoboken, NJ, USA, 2014; pp. 73–94. [CrossRef]

67. Majo, Z.; Gross, T.R. A library for portable and composable data locality optimizations for NUMA systems. In Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming—PPoPP 2015, San Francisco, CA, USA, 7–11 February 2015; volume 50; pp. 227–238. [CrossRef]
68. Lezos, C.; Latifis, I.; Dimitroulakos, G.; Masselos, K. Compiler-Directed Data Locality Optimization in MATLAB. In Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems—SCOPES '16, New York, NY, USA, 23–25 May 2016; pp. 6–9. [CrossRef]
69. Ragan-Kelley, J.; Barnes, C.; Adams, A.; Paris, S.; Durand, F.; Amarasinghe, S. Halide. *ACM SIGPLAN Not.* **2013**, *48*, 519–530. [CrossRef]
70. Chamberlain, B. Parallel Processing Languages: Cray's Chapel Programming. Available online: https://www.cray.com/blog/chapel-productive-parallel-programming/ (accessed on 17 September 2022).
71. Charles, P.; Grothoff, C.; Saraswat, V.; Donawa, C.; Kielstra, A.; Ebcioglu, K.; von Praun, C.; Sarkar, V. X10. In Proceedings of the 20th Annual ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications—OOPSLA '05, New York, NY, USA, 16–20 October 2005; Volume 40, pp. 519–538. [CrossRef]
72. Huang, L.; Jin, H.; Yi, L.; Chapman, B. Enabling locality-aware computations in OpenMP. *Sci. Program.* **2010**, *18*, 169–181. [CrossRef]
73. Gupta, S.; Zhou, H. Spatial Locality-Aware Cache Partitioning for Effective Cache Sharing. In Proceedings of the 2015 44th International Conference on Parallel Processing, Beijing, China, 1–4 September 2015; pp. 150–159. [CrossRef]
74. González, A.; Aliagas, C.; Valero, M. A data cache with multiple caching strategies tuned to different types of locality. In Proceedings of the 9th International Conference on Supercomputing—ICS '95, New York, NY, USA, 3–7 July 1995. [CrossRef]
75. Seshadri, V.; Mutlu, O.; Kozuch, M.A.; Mowry, T.C. The evicted-address filter. In Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques—PACT '12, Minneapolis, MN, USA, 19–23 September 2012; p. 355. [CrossRef]
76. Rivers, J.; Davidson, E. Reducing conflicts in direct-mapped caches with a temporality-based design. In Proceedings of the 1996 ICPP Workshop on Challenges for Parallel Processing, Ithaca, NY, USA, 12 August 2002; Volume 1, pp. 154–163. [CrossRef]
77. Johnson, T.L.; Hwu, W.-M.W. Run-time adaptive cache hierarchy management via reference analysis. In Proceedings of the 24th Annual International Symposium on Computer Architecture—ISCA '97, Boulder, CO, USA, 2–4 June 1997; Volume 25, pp. 315–326. [CrossRef]
78. Jiang, X.; Madan, N.; Zhao, L.; Upton, M.; Iyer, R.; Makineni, S.; Newell, D.; Solihin, Y.; Balasubramonian, R. CHOP: Adaptive filter-based DRAM caching for CMP server platforms. In Proceedings of the HPCA—16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture, Bangalore, India, 9–14 January 2010; pp. 1–12. [CrossRef]
79. Muchnick, S.S. *Advanced Compiler Design and Implementation*; Morgan Kaufmann Publishers: Burlington, MA, USA, 1997.
80. Allen, R.; Kennedy, K. *Optimizing Compilers for Modern Architectures: A Dependence-Based Approach*; Morgan Kaufmann Pub-lishers: Burlington, MA, USA, 2001.
81. Wolfe, M. Loops skewing: The wavefront method revisited. *Int. J. Parallel Program.* **1986**, *15*, 279–293. [CrossRef]
82. Kowarschik, M.; Weiß, C. *An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 213–232. [CrossRef]
83. Xue, J.; Ling, J. *Loop Tiling for Parallelism*; Kluwer Academic: New York, NY, USA, 2000.
84. Bao, B.; Ding, C. Defensive loop tiling for shared cache. In Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Shenzhen, China, 23–27 February 2013; pp. 1–11. [CrossRef]
85. Wolf, M.E.; Lam, M.S. A data locality optimizing algorithm. In Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation—PLDI '91, Toronto, Canada, 26–28 June 1991. [CrossRef]
86. Irigoin, F.; Triolet, R. Supernode partitioning. In Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages—POPL '88, Boston, MA, USA, 15–21 January 1988; pp. 319–329. [CrossRef]
87. Zhou, X.; Giacalone, J.-P.; Garzarán, M.J.; Kuhn, R.H.; Ni, Y.; Padua, D. Hierarchical overlapped tiling. In Proceedings of the Tenth International Symposium on Code Generation and Optimization—CHO '12, Montreal, Canada, 27 February–3 March 2012; pp. 207–218. [CrossRef]
88. Liu, L.; Chen, L.; Wu, C.; Feng, X.-B. Global Tiling for Communication Minimal Parallelization on Distributed Memory Systems. In *Euro-Par 2008—Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 382–391. [CrossRef]
89. Hogstedt, K.; Carter, L.; Ferrante, J. On the parallel execution time of tiled loops. *IEEE Trans. Parallel Distrib. Syst.* **2003**, *14*, 307–321. [CrossRef]
90. Yi, Q. Automated programmable control and parameterization of compiler optimizations. In Proceedings of the International Symposium on Code Generation and Optimization (CGO 2011), Chamonix, France, 2–6 April 2011; pp. 97–106. [CrossRef]
91. Hall, M.; Chame, J.; Chen, C.; Shin, J.; Rudy, G.; Khan, M.M. *Loop Transformation Recipes for Code Generation and Auto-Tuning*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 50–64. [CrossRef]
92. Tavarageri, S.; Pouchet, L.-N.; Ramanujam, J.; Rountev, A.; Sadayappan, P. Dynamic selection of tile sizes. In Proceedings of the 2011 18th International Conference on High Performance Computing, Bengaluru, India, 18–21 December 2011; pp. 1–10. [CrossRef]
93. Kennedy, K.; McKinley, K.S. Optimizing for parallelism and data locality. In Proceedings of the 25th Anniversary International Conference on Supercomputing Anniversary Volume, New York, NY, USA, 2–6 June 2014; pp. 151–162. [CrossRef]

94. Mittal, S. A Survey Of Cache Bypassing Techniques. *J. Low Power Electron. Appl.* **2016**, *6*, 5. [CrossRef]
95. Raicu, I.; Zhao, Y.; Dumitrescu, C.; Foster, I.; Wilde, M. Falkon. In Proceedings of the 2007 ACM/IEEE Conference on Supercomputing—SC '07, New York, NY, USA, 10–16 November 2007; p. 43. [CrossRef]
96. Yoo, A.B.; Jette, M.A.; Grondona, M. *SLURM: Simple Linux Utility for Resource Management*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 44–60. [CrossRef]
97. Gentzsch, W. Sun Grid Engine: Towards creating a compute power grid. In Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, Brisbane, QLD, Australia, 15–18 May 2002. [CrossRef]
98. Thain, D.; Tannenbaum, T.; Livny, M. Distributed computing in practice: The Condor experience: Research Articles. *Concurr. Comput. Pract. Exp.* **2005**, *17*, 323–356. [CrossRef]
99. Ousterhout, K.; Wendell, P.; Zaharia, M.; Stoica, I. Sparrow. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles—SOSP '13, New York, NY, USA, 3–6 November 2013. [CrossRef]
100. Olivier, S.; Porterfield, A.K.; Wheeler, K.B.; Spiegel, M.; Prins, J.F. OpenMP task scheduling strategies for multicore NUMA systems. *Int. J. High Perform. Comput. Appl.* **2012**, *26*, 110–124. [CrossRef]
101. Frigo, M.; Leiserson, C.E.; Randall, K.H. The implementation of the Cilk-5 multithreaded language. *ACM SIGPLAN Not.* **1998**, *33*, 212–223. [CrossRef]
102. Wang, K.; Zhou, X.; Li, T.; Zhao, D.; Lang, M.; Raicu, I. Optimizing load balancing and data-locality with data-aware scheduling. In Proceedings of the 2014 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 27–30 October 2014; pp. 119–128. [CrossRef]
103. Falt, Z.; Kruliš, M.; Bednárek, D.; Yaghob, J.; Zavoral, F. *Locality Aware Task Scheduling in Parallel Data Stream Processing*; Springer: Cham, Switzerland, 2015; pp. 331–342.
104. Muddukrishna, A.; Jonsson, P.A.; Brorsson, M. Locality-Aware Task Scheduling and Data Distribution for OpenMP Programs on NUMA Systems and Manycore Processors. *Sci. Program.* **2015**, *2015*, 1–16. [CrossRef]
105. Ding, W.; Zhang, Y.; Kandemir, M.; Srinivas, J.; Yedlapalli, P. Locality-aware mapping and scheduling for multicores. In Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Shenzhen, China, 23–27 February 2013; pp. 1–12. [CrossRef]
106. Lifflander, J.; Krishnamoorthy, S.; Kale, L.V. Optimizing Data Locality for Fork/Join Programs Using Constrained Work Stealing. In Proceedings of the SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, USA, 16–21 November 2014; pp. 857–868. [CrossRef]
107. Xue, L.; Kandemir, M.; Chen, G.; Li, F.; Ozturk, O.; Ramanarayanan, R.; Vaidyanathan, B. Locality-Aware Distributed Loop Scheduling for Chip Multiprocessors. In Proceedings of the 20th International Conference on VLSI Design Held Jointly with 6th International Conference on Embedded Systems (VLSID'07), Bangalore, India, 6–10 January 2007; pp. 251–258. [CrossRef]
108. Isard, M.; Budiu, M.; Yu, Y.; Birrell, A.; Fetterly, D. Dryad. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007—EuroSys '07, New York, NY, USA, 21–23 March 2007; Volume 41, pp. 59–72. [CrossRef]
109. Maglalang, J.; Krishnamoorthy, S.; Agrawal, K. Locality-Aware Dynamic Task Graph Scheduling. In Proceedings of the 2017 46th International Conference on Parallel Processing (ICPP), Bristol, UK, 14–17 August 2017; pp. 70–80. [CrossRef]
110. Yoo, R.M.; Hughes, C.J.; Kim, C.; Chen, Y.-K.; Kozyrakis, C. Locality-Aware Task Management for Unstructured Par-allelism: A Quantitative Limit Study. In Proceedings of the Twenty-Fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures, New York, NY, USA, 23–25 July 2013.
111. Paudel, J.; Tardieu, O.; Amaral, J.N. On the Merits of Distributed Work-Stealing on Selective Locality-Aware Tasks. In Proceedings of the 2013 42nd International Conference on Parallel Processing, Lyon, France, 1–4 October 2013; pp. 100–109. [CrossRef]
112. Choi, J.; Adufu, T.; Kim, Y. Data-Locality Aware Scientific Workflow Scheduling Methods in HPC Cloud Environments. *Int. J. Parallel Program.* **2016**, *45*, 1128–1141. [CrossRef]
113. Guo, Y. A Scalable Locality-Aware Adaptive Work-StealingScheduler for Multi-Core Task Parallelism. Ph.D. Thesis, Rice University, Houston, TX, USA, 2011.
114. Hindman, B.; Konwinski, A.; Zaharia, M.; Ghodsi, A.; Joseph, A.D.; Katz, R.; Shenker, S.; Stoica, I. Mesos: A platform for fine-grained resource sharing in the data center. In Proceedings of the 8th USENIX conference on Networked systems design and implementation. USENIX Association, Boston, MA, USA, March 30–April 1 2011; pp. 295–308.
115. Isard, M.; Prabhakaran, V.; Currey, J.; Wieder, U.; Talwar, K.; Goldberg, A. Quincy. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles—SOSP '09, Big Sky, MT, USA, 11–14 October 2009. [CrossRef]
116. Valiant, L.G. A bridging model for parallel computation. *Commun. ACM* **1990**, *33*, 103–111. [CrossRef]
117. Cheatham, T.; Fahmyy, A.; Stefanescu, D.C.; Valiant, L.G. Bulk Synchronous Parallel Computing-A Paradigm for transportable Software. *Harv. Comput. Sci. Group Tech. Rep.* **1995**.
118. Malewicz, G.; Austern, M.H.; Bik, A.J.; Dehnert, J.C.; Horn, I.; Leiser, N.; Czajkowski, G. Pregel. In Proceedings of the 2010 International Conference on Management of Data—SIGMOD '10, New York, NY, USA, 6–10 June 2010; pp. 135–146. [CrossRef]
119. Apache Hama Big Data and High-Performance Computing. Available online: https://hama.apache.org/ (accessed on 22 January 2018).
120. Giraph-Welcome To Apache Giraph. Available online: https://giraph.apache.org/ (accessed on 20 October 2022).
121. Hill, J.M.; McColl, B.; Stefanescu, D.C.; Goudreau, M.W.; Lang, K.; Rao, S.B.; Suel, T.; Tsantilas, T.; Bisseling, R.H. BSPlib: The BSP programming library. *Parallel Comput.* **1998**, *24*, 1947–1980. [CrossRef]

122. BSPonMPI. Available online: https://bsponmpi.sourceforge.net/ (accessed on 20 January 2022).
123. Yzelman, A.N.; Bisseling, R.H.; Roose, D.; Meerbergen, K. MulticoreBSP for C: A High-Performance Library for Shared-Memory Parallel Programming. *Int. J. Parallel Program.* **2013**, *42*, 619–642. [CrossRef]
124. Yzelman, A.; Bisseling, R.H. An object-oriented bulk synchronous parallel library for multicore programming. *Concurr. Comput. Pr. Exp.* **2011**, *24*, 533–553. [CrossRef]
125. Abello, J.M.; Vitter, J.S. *External memory algorithms: DIMACS Workshop External Memory and Visualization, May 20–22, 1998*; American Mathematical Society: Providence, RI, USA, 1999.
126. Kwiatkowska, M.; Mehmood, R. *Out-of-Core Solution of Large Linear Systems of Equations Arising from Stochastic Modelling*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 135–151. [CrossRef]
127. Mehmood, R. Disk-Based Techniques for Efficient Solution of Large Markov Chains. PhD Thesis, School of Computer Science, University of Birmingham,, Birmingham, UK, 2004.
128. Jung, M.; Wilson, E.H.; Choi, W.; Shalf, J.; Aktulga, H.M.; Yang, C.; Saule, E.; Catalyurek, U.V.; Kandemir, M. Exploring the future of out-of-core computing with compute-local non-volatile memory. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on—SC '13, Denver, CO, USA, 17–22 November 2013, 17–22 November 2013; pp. 1–11. [CrossRef]
129. Koller, R.; Marmol, L.; Rangaswami, R.; Sundararaman, S.; Talagala, N.; Zhao, M. Write policies for host-side flash caches. In Proceedings of the 11th USENIX Conference on File and Storage Technologies. USENIX Association, San Jose, CA, USA, 12–15 February 2013; pp. 45–58.
130. Saxena, M.; Swift, M.M.; Zhang, Y. FlashTier. In Proceedings of the 7th ACM European Conference on Computer Systems—EuroSys '12, New York, NY, USA, 10–13 April 2012; p. 267. [CrossRef]
131. Byan, S.; Lentini, J.; Madan, A.; Pabon, L.; Condict, M.; Kimmel, J.; Kleiman, S.; Small, C.; Storer, M. Mercury: Host-side flash caching for the data center. In Proceedings of the 012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), Monterey, CA, USA, 19–20 April 2012; pp. 1–12. [CrossRef]
132. Saule, E.; Aktulga, H.M.; Yang, C.; Ng, E.G.; Çatalyürek, Ü.V. An Out-of-Core Task-based Middleware for Da-ta-Intensive Scientific Computing. In *Handbook on Data Centers*; Springer: New York, NY, USA, 2015; pp. 647–667.
133. Rothberg, E.; Schreiber, R. Efficient Methods for Out-of-Core Sparse Cholesky Factorization. *SIAM J. Sci. Comput.* **1999**, *21*, 129–144. [CrossRef]
134. Mandhapati, P.; Khaitan, S. High Performance Computing Using out-of-core Sparse Direct Solvers. *World Acad. Sci. Eng. Technol.* **2009**, *3*, 377–383.
135. Geist, A.; Lucas, R. Whitepaper on the Major Computer Science Challenges at Exascale. 2009. Available online: https://exascale.org/mediawiki/images/8/87/ExascaleSWChallenges-Geist_Lucas.pdf (accessed on 1 November 2022).
136. Das, B.V.D.; Kathiresan, N.; Ravindran, R. Process Mapping Parallel Computing. US8161127B2, 28 November 2011.
137. Hursey, J.; Squyres, J.M.; Dontje, T. Locality-Aware Parallel Process Mapping for Multi-core HPC Systems. In Proceedings of the 2011 IEEE International Conference on Cluster Computing, Austin, TX USA, 26–30 September 2011; pp. 527–531. [CrossRef]
138. Rodrigues, E.R.; Madruga, F.L.; Navaux, P.O.A.; Panetta, J. Multi-core aware process mapping and its impact on communication overhead of parallel applications. In Proceedings of the 2009 IEEE Symposium on Computers and Communications, Sousse, Tunisia, 5–8 July 2009; pp. 811–817. [CrossRef]
139. Rashti, M.J.; Green, J.; Balaji, P.; Afsahi, A.; Gropp, W. *Multi-core and Network Aware MPI Topology Functions*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 50–60. [CrossRef]
140. Hestness, J.; Keckler, S.W.; Wood, D.A. A comparative analysis of microarchitecture effects on CPU and GPU memory system behavior. In Proceedings of the 2014 IEEE International Symposium on Workload Characterization (IISWC), Raleigh, NC, USA, 26–28 October 2014; pp. 150–160. [CrossRef]
141. Chen, H.; Chen, W.; Huang, J.; Robert, B.; Kuhn, H. MPIPP. In Proceedings of the 20th annual international conference on Supercomputing—ICS '06, Cairns, QLD, Australia, 28 June–1 July 2006. [CrossRef]
142. Zhang, J.; Zhai, J.; Chen, W.; Zheng, W. *Process Mapping for MPI Collective Communications*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 81–92. [CrossRef]
143. Pilla, L.L.; Ribeiro, C.P.; Coucheney, P.; Broquedis, F.; Gaujal, B.; Navaux, P.O.; Méhaut, J.-F. A topology-aware load balancing algorithm for clustered hierarchical multi-core machines. *Futur. Gener. Comput. Syst.* **2014**, *30*, 191–201. [CrossRef]
144. Zarrinchian, G.; Soryani, M.; Analoui, M. *A New Process Placement Algorithm in Multi-Core Clusters Aimed to Reducing Network Interface Contention*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 1041–1050. [CrossRef]
145. Mercier, G.; Clet-Ortega, J. *Towards an Efficient Process Placement Policy for MPI Applications in Multicore Environments*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 104–115.
146. Balaji, P.; Gupta, R.; Vishnu, A.; Beckman, P. Mapping communication layouts to network hardware characteristics on massive-scale blue gene systems. *Comput. Sci. Res. Dev.* **2011**, *26*, 247–256. [CrossRef]
147. Smith, B.E.; Bode, B. *Performance Effects of Node Mappings on the IBM BlueGene/L Machine*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1005–1013. [CrossRef]
148. Yu, H.; Chung, I.-H.; Moreira, J. Topology Mapping for Blue Gene/L Supercomputer. In Proceedings of the ACM/IEEE SC 2006 Conference (SC'06), Cairns, QLD, Australia, 28 June–1 July 2006; p. 52. [CrossRef]

149. Ito, S.; Goto, K.; Ono, K. Automatically optimized core mapping to subdomains of domain decomposition method on multicore parallel environments. *Comput. Fluids* **2013**, *80*, 88–93. [CrossRef]
150. Traff, J. Implementing the MPI Process Topology Mechanism. In Proceedings of the ACM/IEEE SC 2002 Conference (SC'02), Baltimore, MD, USA, 16–22 November 2002; p. 28. [CrossRef]
151. Dümmler, J.; Rauber, T.; Rünger, G. Mapping Algorithms for Multiprocessor Tasks on Multi-Core Clusters. In Proceedings of the 2008 37th International Conference on Parallel Processing, Washington, DC, USA, 9–11 September 2008; pp. 141–148. [CrossRef]
152. Hoefler, T.; Snir, M. Generic topology mapping strategies for large-scale parallel architectures. In Proceedings of the International Conference on Supercomputing—ICS '11, Tucson, AZ, USA, 31 May–4 June 2011; pp. 75–84. [CrossRef]
153. Kale, L.V.; Krishnan, S. *CHARM++: A Portable Concurrent Object Oriented System Based on C++*; Technical Report; University of Illinois at Urbana-Champaign: Champaign, IL, USA, 1993.
154. El-Ghazawi, T. *UPC: Distributed Shared Memory Programming*; Wiley: Hoboken, NJ, USA, 2005.
155. Castro, M.; Goes, L.F.W.; Ribeiro, C.P.; Cole, M.; Cintra, M.; Mehaut, J.-F. A machine learning-based approach for thread mapping on transactional memory applications. In Proceedings of the 2011 18th International Conference on High Performance Computing, New York, NY, USA, 12–18 November 2011; pp. 1–10. [CrossRef]
156. Grewe, D.; O'Boyle, M.F.P. *A Static Task Partitioning Approach for Heterogeneous Systems Using OpenCL*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 286–305. [CrossRef]
157. Tournavitis, G.; Wang, Z.; Franke, B.; O'Boyle, M.F.P. Towards a holistic approach to auto-parallelization. *ACM SIGPLAN Not.* **2009**, *44*, 177–187. [CrossRef]
158. Wang, Z.; O'Boyle, M.F. Mapping parallelism to multi-cores. In Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming—PpoPP '09, Raleigh, NC, USA, 14–18 February 2008; Volume 44, p. 75. [CrossRef]
159. Long, S.; Fursin, G.; Franke, B. *A Cost-Aware Parallel Workload Allocation Approach Based on Machine Learning Techniques*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 506–515. [CrossRef]
160. Pinel, F.; Bouvry, P.; Dorronsoro, B.; Khan, S.U. Savant: Automatic parallelization of a scheduling heuristic with machine learning. *Nat. Biol.* **2013**, 52–57. [CrossRef]
161. Emani, M.K.; O'Boyle, M. Celebrating diversity: A mixture of experts approach for runtime mapping in dynamic environments. In Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation—PLDI 2015, Portland, OR, USA, 13–17 June 2015. [CrossRef]
162. Emani, M.K.; O'Boyle, M. *Change Detection Based Parallelism Mapping: Exploiting Offline Models and Online Adaptation*; Springer International Publishing: Cham, Switzerland, 2015; pp. 208–223.
163. Luk, C.-K.; Hong, S.; Kim, H. Qilin. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture—Micro-42, New York, NY, USA, 12–16 December 2009; pp. 45–55. [CrossRef]
164. González-Domínguez, J.; Taboada, G.L.; Fraguela, B.B.; Martín, M.J.; Touriño, J. Automatic mapping of parallel applications on multicore architectures using the Servet benchmark suite. *Comput. Electr. Eng.* **2012**, *38*, 258–269. [CrossRef]
165. Tiwari, D.; Vazhkudai, S.S.; Kim, Y.; Ma, X.; Boboila, S.; Desnoyers, P.J. Reducing Data Movement Costs using Ener-gy-Efficient, Active Computation on SSD. In Proceedings of the 2012 Workshop on Power-Aware Computing and Systems, Hollywood, CA, USA, 7 October 2012.
166. Zheng, F.; Yu, H.; Hantas, C.; Wolf, M.; Eisenhauer, G.; Schwan, K.; Abbasi, H.; Klasky, S. GoldRush. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on—SC '13, Denver, CO, USA, 17–22 November 2013. [CrossRef]
167. Sewell, C.; Heitmann, K.; Finkel, H.; Zagaris, G.; Parete-Koon, S.T.; Fasel, P.K.; Pope, A.; Frontiere, N.; Lo, L.-T.; Messer, B.; et al. Large-scale compute-intensive analysis via a combined in-situ and co-scheduling workflow approach. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis—SC '15, Atlanta, GA, USA, 9 November 2015; p. 50. [CrossRef]
168. Lakshminarasimhan, S.; Shah, N.; Ethier, S.; Klasky, S.; Latham, R.; Ross, R.; Samatova, N.F. Compressing the Incompressible with ISABELA: In-Situ Reduction of Spatio-temporal Data. Springer: Berlin/Heidelberg, Germany, 2011; pp. 366–379. [CrossRef]
169. Zou, H.; Zheng, F.; Wolf, M.; Eisenhauer, G.; Schwan, K.; Abbasi, H.; Liu, Q.; Podhorszki, N.; Klasky, S.; Wolf, M. Quality-Aware Data Management for Large Scale Scientific Applications. In Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, 24–29 June 2012; pp. 816–820. [CrossRef]
170. Kim, J.; Abbasi, H.; Chacon, L.; Docan, C.; Klasky, S.; Liu, Q.; Podhorszki, N.; Shoshani, A.; Wu, K. Parallel in situ indexing for data-intensive computing. In Proceedings of the 2011 IEEE Symposium on Large Data Analysis and Visualization, Providence, RI, USA, 23–24 October 2011; pp. 65–72. [CrossRef]
171. Lakshminarasimhan, S.; Boyuka, D.A.; Pendse, S.V.; Zou, X.; Jenkins, J.; Vishwanath, V.; Papka, M.E.; Samatova, N.F. Scalable in situ scientific data encoding for analytical query processing. In Proceedings of the 22nd international symposium on High-performance parallel and distributed computing, New York, NY, USA, 17–21 June 2013; pp. 1–12. [CrossRef]
172. Su, Y.; Wang, Y.; Agrawal, G. In-Situ Bitmaps Generation and Efficient Data Analysis based on Bitmaps. In *24th International Symposium on High-Performance Parallel and Distributed Computing—HPDC '15*; ACM: New York, NY, USA, 2015; pp. 61–72. [CrossRef]

173. Karimabadi, H.; Loring, B.; O'Leary, P.; Majumdar, A.; Tatineni, M.; Geveci, B. In-situ visualization for global hybrid simulations. In Proceedings of the Conference on Extreme Science and Engineering Discovery Environment Gateway to Discovery—XSEDE '13, Atlanta, GA, USA, 13–18 July 2013; p. 1. [CrossRef]
174. Yu, H.; Wang, C.; Grout, R.W.; Chen, J.H.; Ma, K.-L. In Situ Visualization for Large-Scale Combustion Simulations. *IEEE Comput. Graph. Appl.* **2010**, *30*, 45–57. [CrossRef]
175. Zou, H.; Schwan, K.; Slawinska, M.; Wolf, M.; Eisenhauer, G.; Zheng, F.; Dayal, J.; Logan, J.; Liu, Q.; Klasky, S.; et al. FlexQuery: An online query system for interactive remote visual data exploration at large scale. In Proceedings of the 2013 IEEE International Conference on Cluster Computing (CLUSTER), Indianapolis, IN, USA, 23–27 September 2013; pp. 1–8. [CrossRef]
176. Woodring, J.; Ahrens, J.; Tautges, T.J.; Peterka, T.; Vishwanath, V.; Geveci, B. On-demand unstructured mesh translation for reducing memory pressure during in situ analysis. In Proceedings of the 8th International Workshop on Ultrascale Visualization—UltraVis '13, Denver, CO, USA, 17–22 November 2013. [CrossRef]
177. Nouanesengsy, B.; Woodring, J.; Patchett, J.; Myers, K.; Ahrens, J. ADR visualization: A generalized framework for ranking large-scale scientific data using Analysis-Driven Refinement. In Proceedings of the 2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV), Paris, France, 9–10 November 2014; pp. 43–50. [CrossRef]
178. Landge, A.G.; Pascucci, V.; Gyulassy, A.; Bennett, J.C.; Kolla, H.; Chen, J.; Bremer, P.-T. In-Situ Feature Extraction of Large Scale Combustion Simulations Using Segmented Merge Trees. In Proceedings of the SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, USA, 16–21 November 2014; pp. 1020–1031. [CrossRef]
179. Zhang, F.; Lasluisa, S.; Jin, T.; Rodero, I.; Bui, H.; Parashar, M. In-situ Feature-Based Objects Tracking for Large-Scale Scientific Simulations. In Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, 24–29 June 2012; pp. 736–740. [CrossRef]
180. Mehmood, R.; Meriton, R.; Graham, G.; Hennelly, P.; Kumar, M. Exploring the influence of big data on city transport operations: A Markovian approach. *Int. J. Oper. Prod. Manag.* **2017**, *37*, 75–104. [CrossRef]
181. Mehmood, R.; Graham, G. Big Data Logistics: A health-care Transport Capacity Sharing Model. *Procedia Comput. Sci.* **2015**, *64*, 1107–1114. [CrossRef]
182. AlOmari, E.; Katib, I.; Mehmood, R. Iktishaf: A Big Data Road-Traffic Event Detection Tool Using Twitter and Spark Machine Learning. *Mob. Networks Appl.* **2020**, 1–16. [CrossRef]
183. Alotaibi, S.; Mehmood, R.; Katib, I.; Rana, O.; Albeshri, A. Sehaa: A Big Data Analytics Tool for Healthcare Symptoms and Diseases Detection Using Twitter, Apache Spark, and Machine Learning. *Appl. Sci.* **2020**, *10*, 1398. [CrossRef]
184. Aqib, M.; Mehmood, R.; Alzahrani, A.; Katib, I. Aqib, M.; Mehmood, R.; Alzahrani, A.; Katib, I. A smart disaster management system for future cities using deep learning, GPUs, and in-memory computing. In *Smart Infrastructure and Applications*; Springer: Cham, Switzerland, 2020; pp. 159–184.
185. Aqib, M.; Mehmood, R.; Alzahrani, A.; Katib, I.; Albeshri, A.; Altowaijri, S.M. Rapid Transit Systems: Smarter Urban Planning Using Big Data, In-Memory Computing, Deep Learning, and GPUs. *Sustainability* **2019**, *11*, 2736. [CrossRef]
186. Suma, S.; Mehmood, R.; Albeshri, A. Automatic Detection and Validation of Smart City Events Using HPC and Apache Spark Platforms. In *Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 55–78. [CrossRef]
187. Alotaibi, S.; Mehmood, R. Big Data Enabled Healthcare Supply Chain Management: Opportunities and Challenges. In *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (LNICST)*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 224, pp. 207–215. [CrossRef]
188. Ahmad, I.; Alqurashi, F.; Abozinadah, E.; Mehmood, R. Deep Journalism and DeepJournal V1.0: A Data-Driven Deep Learning Approach to Discover Parameters for Transportation. *Sustainability* **2022**, *14*, 5711. [CrossRef]
189. Arfat, Y.; Usman, S.; Mehmood, R.; Katib, I. Big data for smart infrastructure design: Opportunities and challenges. In *Smart Infrastructure and Applications Foundations for Smarter Cities and Societies*; Springer: Cham, Switzerland, 2020; pp. 491–518.
190. Singh, D.; Reddy, C.K. A survey on platforms for big data analytics. *J. Big Data* **2014**, *2*, 1–20. [CrossRef] [PubMed]
191. Dean, J.; Ghemawat, S. MapReduce. *Commun. ACM* **2008**, *51*, 107. [CrossRef]
192. Ghemawat, S.; Gobioff, H.; Leung, S.-T. The Google file system. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles—SOSP '03, Bolton Landing, NY, USA, 19–22 October 2003; Volume 37, p. 29. [CrossRef]
193. White, T. *Hadoop: The Definitive Guide*, 4th ed.; Yahoo Press: Sunnyvale, CA, USA, 2009.
194. Shvachko, K.; Kuang, H.; Radia, S.; Chansler, R. The Hadoop Distributed File System. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Vilage, NV, USA, 3–7 May 2010; pp. 2–10.
195. Borthakur, D.; Rash, S.; Schmidt, R.; Aiyer, A.; Gray, J.; Sarma, J.S.; Muthukkaruppan, K.; Spiegelberg, N.; Kuang, H.; Ranganathan, K.; et al. Apache hadoop goes realtime at Facebook. In Proceedings of the 2011 International Conference on Management of Data–SIGMOD '11, Athens, Greece, 12–16 June 2011; pp. 1071–1080. [CrossRef]
196. Apache Tez. Available online: https://tez.apache.org/ (accessed on 18 June 2022).
197. Ekanayake, J.; Li, H.; Zhang, B.; Gunarathne, T.; Bae, S.-H.; Qiu, J.; Fox, G. Twister. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing—HPDC '10, New York, NY, USA, 21–25 June 2010; pp. 810–818. [CrossRef]

198. Padhy, R.P. Big Data Processing with Hadoop-MapReduce in Cloud Systems. *IJ-CLOSER Int. J. Cloud Comput. Serv. Sci.* **2012**, *2*, 233–245. [CrossRef]
199. Singh, K.; Kaur, R. Hadoop: Addressing challenges of Big Data. In Proceedings of the 2014 IEEE International Advance Computing Conference (IACC), New Delhi, India, 21–22 February 2014; pp. 686–689. [CrossRef]
200. Yang, H.-C.; Dasdan, A.; Hsiao, R.-L.; Parker, D.S. Map-reduce-merge. In Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data—SIGMOD '07, Beijing, China, 12–14 June 2007; pp. 1029–1040. [CrossRef]
201. Katal, A.; Wazid, M.; Goudar, R.H. Big data: Issues, challenges, tools and Good practices. In Proceedings of the 2013 Sixth International Conference on Contemporary Computing (IC3), Noida, India, 8–10 August 2013; pp. 404–409.
202. Tudoran, R.; Costan, A.; Antoniu, G. MapIterativeReduce. In Proceedings of the Third International Workshop on MapReduce and Its Applications Date—MapReduce '12, Delft, the Netherlands, 18–19 June 2012; pp. 9–16. [CrossRef]
203. Bu, Y.; Howe, B.; Balazinska, M.; Ernst, M.D. HaLoop. *Proc. VLDB Endow.* **2010**, *3*, 285–296. [CrossRef]
204. Zaharia, M.; Chowdhury, M.; Das, T.; Dave, A.; Ma, J.; McCauley, M.; Franklin, M.; Shenker, S.; Stoica, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, San Jose, CA, USA, 25–27 April 2012; p. 2.
205. Chen, C.L.P.; Zhang, C.-Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Inf. Sci.* **2014**, *275*, 314–347. [CrossRef]
206. Olston, C.; Reed, B.; Srivastava, U.; Kumar, R.; Tomkins, A. Pig latin. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data—SIGMOD '08, Vancouver, BC, Canada, 10–12 June 2008; p. 1099. [CrossRef]
207. Lin, Z.; Cai, M.; Huang, Z.; Lai, Y. *SALA: A Skew-Avoiding and Locality-Aware Algorithm for MapReduce-Based Join*; Springer: Cham, Switzerland, 2015; pp. 311–323.
208. Ibrahim, S.; Jin, H.; Lu, L.; Wu, S.; He, B.; Qi, L. LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud. In Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, Indianapolis, IN, USA, 30 November–3 December 2010; pp. 17–24. [CrossRef]
209. Rhine, R.; Bhuvan, N.T. *Locality Aware MapReduce*; Springer: Cham, Switzerland, 2015; pp. 221–228. [CrossRef]
210. Eltabakh, M.Y.; Tian, Y.; Özcan, F.; Gemulla, R.; Krettek, A.; McPherson, J. CoHadoop. *Proc. VLDB Endow.* **2011**, *4*, 575–585. [CrossRef]
211. Yu, X.; Hong, B. Grouping Blocks for MapReduce Co-Locality. In Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium, Hyderabad, India, 29 May 2015; pp. 271–280. [CrossRef]
212. Tan, J.; Meng, S.; Meng, X.; Zhang, L. Improving ReduceTask data locality for sequential MapReduce jobs. In Proceedings of the 2013 Proceedings IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 1627–1635. [CrossRef]
213. Wang, J.; Xiao, Q.; Yin, J.; Shang, P. DRAW: A New Data-gRouping-AWare Data Placement Scheme for Data Intensive Applications With Interest Locality. *IEEE Trans. Magn.* **2013**, *49*, 2514–2520. [CrossRef]
214. Xie, J.; Yin, S.; Ruan, X.; Ding, Z.; Tian, Y.; Majors, J.; Manzanares, A.; Qin, X. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), Atlanta, GA, USA, 19–23 April 2010; pp. 1–9. [CrossRef]
215. Arasanal, R.M.; Rumani, D.U. *Improving MapReduce Performance through Complexity and Performance Based Data Placement in Heterogeneous Hadoop Clusters*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 115–125. [CrossRef]
216. Lee, C.-W.; Hsieh, K.-Y.; Hsieh, S.-Y.; Hsiao, H.-C. A Dynamic Data Placement Strategy for Hadoop in Heterogeneous Environments. *Big Data Res.* **2014**, *1*, 14–22. [CrossRef]
217. Ubarhande, V.; Popescu, A.-M.; Gonzalez-Velez, H. Novel Data-Distribution Technique for Hadoop in Heterogeneous Cloud Environments. In Proceedings of the 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems, Santa Catarina, Brazil, 8–10 July 2015; pp. 217–224. [CrossRef]
218. Sujitha, S.; Jaganathan, S. Aggrandizing Hadoop in terms of node Heterogeneity & Data Locality. In Proceedings of the IEEE International Conference on Smart Structures and Systems (ICSSS)'13, Chennai, India, 28–29 March 2013; pp. 145–151. [CrossRef]
219. Guo, Z.; Fox, G.; Zhou, M. Investigation of Data Locality in MapReduce. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012); Institute of Electrical and Electronics Engineers (IEEE), Ottawa, ON, Canada, 13–16 May 2012; pp. 419–426.
220. Chen, Y.; Liu, Z.; Wang, T.; Wang, L. *Load Balancing in MapReduce Based on Data Locality*; Springer: Cham, Switzerland, 2014; pp. 229–241. [CrossRef]
221. Chen, T.-Y.; Wei, H.-W.; Wei, M.-F.; Chen, Y.-J.; Hsu, T.-S.; Shih, W.-K. LaSA: A locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment. In Proceedings of the 2013 International Conference on Collaboration Technologies and Systems (CTS), San Diego, CA, USA, 20–24 May 2013; pp. 342–346. [CrossRef]
222. Park, J.; Lee, D.; Kim, B.; Huh, J.; Maeng, S. Locality-aware dynamic VM reconfiguration on MapReduce clouds. In Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing—HPDC '12, New York, NY, USA, 18–22 June 2012; pp. 27–36. [CrossRef]
223. Zaharia, M.; Borthakur, D.; Sarma, J.S.; Elmeleegy, K.; Shenker, S.; Stoica, I. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In Proceedings of the 5th European conference on Computer systems, New York, NY, USA, 13–16 April 2010.

224. Zhang, X.; Feng, Y.; Feng, S.; Fan, J.; Ming, Z. An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments. In Proceedings of the 2011 International Conference on Cloud and Service Computing, Hong Kong, China, 12–14 December 2011; pp. 235–242. [CrossRef]
225. Hsu, C.-H.; Slagter, K.D.; Chung, Y.-C. Locality and loading aware virtual machine mapping techniques for optimizing communications in MapReduce applications. *Futur. Gener. Comput. Syst.* **2015**, *53*, 43–54. [CrossRef]
226. Xue, R.; Gao, S.; Ao, L.; Guan, Z. BOLAS: Bipartite-Graph Oriented Locality-Aware Scheduling for MapReduce Tasks. In Proceedings of the 2015 14th International Symposium on Parallel and Distributed Computing, Washington, DC, USA, 29 June–2 July 2015; pp. 37–45. [CrossRef]
227. Sadasivam, G.S.; Selvaraj, D. A novel parallel hybrid PSO-GA using MapReduce to schedule jobs in Hadoop data grids. In Proceedings of the 2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC), Fargo, ND, USA, 12–14 August 2010; pp. 377–382. [CrossRef]
228. Zhang, X.; Wu, Y.; Zhao, C. MrHeter: Improving MapReduce performance in heterogeneous environments. *Clust. Comput.* **2016**, *19*, 1691–1701. [CrossRef]
229. Guo, L.; Sun, H.; Luo, Z. *A Data Distribution Aware Task Scheduling Strategy for MapReduce System*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 694–699. [CrossRef]
230. Hammoud, M.; Sakr, M.F. Locality-Aware Reduce Task Scheduling for MapReduce. In Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, Athens, Greece, 29 November–1 December 2011; pp. 570–576. [CrossRef]
231. Ahmad, F.; Chakradhar, S.T.; Raghunathan, A.; Vijaykumar, T.N. Tarazu. In Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems—ASPLOS '12, London, UK, 3–7 March 2012; Volume 40, p. 61. [CrossRef]
232. Kumar, K.A.; Konishetty, V.K.; Voruganti, K.; Rao, G.V.P. CASH. In Proceedings of the International Conference on Advances in Computing, Communications and Informatics—ICACCI '12, Chennai, India, 3–5 August 2012; p. 52. [CrossRef]
233. Zhao, Y.; Wang, W.; Meng, D.; Lv, Y.; Zhang, S.; Li, J. TDWS: A Job Scheduling Algorithm Based on MapReduce. In Proceedings of the 2012 IEEE Seventh International Conference on Networking, Architecture, and Storage, Fujian, China, 28–30 June 2012; pp. 313–319. [CrossRef]
234. Hammoud, M.; Rehman, M.S.; Sakr, M.F. Center-of-Gravity Reduce Task Scheduling to Lower MapReduce Network Traffic. In Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 49–58. [CrossRef]
235. Ibrahim, S.; Jin, H.; Lu, L.; He, B.; Antoniu, G.; Wu, S. Maestro: Replica-Aware Map Scheduling for MapReduce. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, Canada, 13–16 May 2012; pp. 435–442. [CrossRef]
236. Sethi, K.K.; Ramesh, D. *Delay Scheduling with Reduced Workload on JobTracker in Hadoop*; Springer: Cham, Switzerland, 2015; pp. 371–381. [CrossRef]
237. Yang, Y.; Xu, J.; Wang, F.; Ma, Z.; Wang, J.; Li, L. A MapReduce Task Scheduling Algorithm for Deadline-Constraint in Homogeneous Environment. In Proceedings of the 2014 Second International Conference on Advanced Cloud and Big Data, Huangshan, China, 20–22 November 2014; pp. 208–212. [CrossRef]
238. Bezerra, A.; Hernández, P.; Espinosa, A.; Moure, J.C. Job scheduling for optimizing data locality in Hadoop clusters. In Proceedings of the 20th European MPI Users' Group Meeting on—EuroMPI '13, Madrid, Spain, 15–18 September 2013; pp. 271–276. [CrossRef]
239. Sun, M.; Zhuang, H.; Li, C.; Lu, K.; Zhou, X. Scheduling algorithm based on prefetching in MapReduce clusters. *Appl. Soft Comput.* **2016**, *38*, 1109–1118. [CrossRef]
240. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster computing with working sets. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. USENIX Association, Boston, MA, USA, 22–25 June 2010; p. 10.
241. Hess, K. Hadoop vs Spark: Comparison, Features & Cost. Available online: https://www.datamation.com/data-center/hadoop-vs-spark/ (accessed on 16 June 2022).
242. Marr, B. Spark Or Hadoop—Which Is The Best Big Data Framework? Available online: https://www.forbes.com/sites/bernardmarr/2015/06/22/spark-or-hadoop-which-is-the-best-big-data-framework/?sh=33f70d3c127e (accessed on 5 June 2021).
243. Li, S.; Amin, T.; Ganti, R.; Srivatsa, M.; Hu, S.; Zhao, Y.; Abdelzaher, T. Stark: Optimizing In-Memory Computing for Dynamic Dataset Collections. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 103–114. [CrossRef]
244. Engle, C.; Lupher, A.; Xin, R.; Zaharia, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Shark. In Proceedings of the 2012 International Cconference on Management of Data—SIGMOD '12, Scottsdale, AZ, USA, 20–24 May 2012; p. 689. [CrossRef]
245. Santos-Neto, E.; Cirne, W.; Brasileiro, F.; Lima, A. *Exploiting Replication and Data Reuse to Efficiently Schedule Da-ta-Intensive Applications on Grids*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 210–232.
246. Xin, R.S.; Gonzalez, J.E.; Franklin, M.J.; Stoica, I. GraphX. In Proceedings of the First International Workshop on Graph Data Management Experiences and Systems—GRADES '13, New York, NY, USA, 24 June 2013; pp. 1–6. [CrossRef]
247. Goldstein, J.; Ramakrishnan, R.; Shaft, U. Compressing relations and indexes. In Proceedings of the 14th International Conference on Data Engineering, Orlando, FL, USA, 6 August 2002; pp. 370–379. [CrossRef]
248. Larus, J.; Hill, M.; Chilimbi, T. Making pointer-based data structures cache conscious. *Computer* **2000**, *33*, 67–74. [CrossRef]

249. Abadi, D.J.; Madden, S.R.; Hachem, N. Column-stores vs. row-stores. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data—SIGMOD '08, Vancouver, BC, Canada, 10–12 June 2008; pp. 967–980. [CrossRef]
250. Plattner, H. A common database approach for OLTP and OLAP using an in-memory column database. In Proceedings of the 35th SIGMOD International Conference on Management of Data—SIGMOD '09, New York, NY, USA, 29 June–2 July 2009; pp. 1–2. [CrossRef]
251. Copeland, G.P.; Khoshafian, S.N. A decomposition storage model. In Proceedings of the 1985 ACM SIGMOD international conference on Management of data—SIGMOD '85, Austin, TX, USA, 1 May 1985; Volume 14, pp. 268–279. [CrossRef]
252. Kim, C.; Chhugani, J.; Satish, N.; Sedlar, E.; Nguyen, A.D.; Kaldewey, T.; Lee, V.W.; Brandt, S.A.; Dubey, P. Designing fast architecture-sensitive tree search on modern multicore/many-core processors. *ACM Trans. Database Syst.* **2011**, *36*, 1–34. [CrossRef]
253. Leis, V.; Kemper, A.; Neumann, T. The adaptive radix tree: ARTful indexing for main-memory databases. In Proceedings of the 2013 IEEE 29th International Conference on Data Engineering (ICDE), Brisbane, Australia, 8–12 April 2013; pp. 38–49. [CrossRef]
254. Maas, L.M.; Kissinger, T.; Habich, D.; Lehner, W. BUZZARD. In Proceedings of the 2013 International Conference on Management of Data—SIGMOD '13, New York, NY, USA, 22–27 June 2013; pp. 1285–1286. [CrossRef]
255. Albutiu, M.-C.; Kemper, A.; Neumann, T. Massively parallel sort-merge joins in main memory multi-core database systems. *Proc. VLDB Endow.* **2012**, *5*, 1064–1075. [CrossRef]
256. Leis, V.; Boncz, P.; Kemper, A.; Neumann, T. Morsel-driven parallelism. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data—SIGMOD '14, Snowbird, UT, USA, 19 September 2014; pp. 743–754. [CrossRef]
257. Li, Y.; Pandis, I.; Mueller, R.; Raman, V.; Lohman, G. NUMA-aware algorithms: The case of data shuffling. In Proceedings of the Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, 6–9 January 2013.
258. Burr, G.W.; Breitwisch, M.J.; Franceschini, M.; Garetto, D.; Gopalakrishnan, K.; Jackson, B.; Kurdi, B.; Lam, C.; Lastras, L.A.; Padilla, A.; et al. Phase change memory technology. *J. Vac. Sci. Technol. B* **2010**, *28*, 223–262. [CrossRef]
259. Yang, J.J.; Williams, S. Memristive devices in computing system. *ACM J. Emerg. Technol. Comput. Syst.* **2013**, *9*, 1–20. [CrossRef]
260. Apalkov, D.; Khvalkovskiy, A.; Watts, S.; Nikitin, V.; Tang, X.; Lottis, D.; Moon, K.; Luo, X.; Chen, E.; Ong, A.; et al. Spin-transfer torque magnetic random access memory (STT-MRAM). *ACM J. Emerg. Technol. Comput. Syst.* **2013**, *9*, 1–35. [CrossRef]
261. Shi, X.; Chen, M.; He, L.; Xie, X.; Lu, L.; Jin, H.; Chen, Y.; Wu, S. Mammoth: Gearing Hadoop Towards Memory-Intensive MapReduce Applications. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *26*, 2300–2315. [CrossRef]
262. Power, R.; Li, J. Piccolo: Building fast, distributed programs with partitioned tables. In Proceedings of the 9th USENIX conference on Operating systems design and implementation, Vancouver, BC, Canada, 4–6 October 2010; pp. 293–306.
263. Neumeyer, L.; Robbins, B.; Nair, A.; Kesari, A. S4: Distributed stream computing platform. In Proceedings of the IEEE International Conference on Data Mining, ICDM, Sydney, NSW, Australia, 13 December 2010; pp. 170–177. [CrossRef]
264. Condie, T.; Conway, N.; Alvaro, P.; Hellerstein, J.M.; Elmeleegy, K.; Sears, R. MapReduce online. In Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, San Jose, CA, USA, 28–30 April 2010; p. 21.
265. Sikka, V.; Färber, F.; Goel, A.; Lehner, W. SAP HANA. *Proc. VLDB Endow.* **2013**, *6*, 1184–1185. [CrossRef]
266. Lahiri, T.; Neimat, M.-A.; Folkman, S. Oracle TimesTen: An In-Memory Database for Enterprise Applications. *IEEE Data Eng. Bull.* **2013**, *36*, 6–13.
267. Lindström, J.; Lindström, J.; Raatikka, V.; Ruuth, J.; Soini, P.; Vakkila, K. IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability. *IEEE Data Eng. Bull.* **2013**, *36*, 14–20.
268. Raman, V.; Attaluri, G.; Barber, R.; Chainani, N.; Kalmuk, D.; KulandaiSamy, V.; Leenstra, J.; Lightstone, S.; Liu, S.; Lohman, G.M.; et al. DB2 with BLU acceleration. *Proc. VLDB Endow.* **2013**, *6*, 1080–1091. [CrossRef]
269. Zhang, H.; Chen, G.; Ooi, B.C.; Wong, W.-F.; Wu, S.; Xia, Y. Anti-Caching-based elastic memory management for Big Data. In Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, Seoul, Republic of Korea, 13–17 April 2015; pp. 1268–1279. [CrossRef]
270. Gandhi, R.; Gupta, A.; Povzner, A.; Belluomini, W.; Kaldewey, T. Mercury. In Proceedings of the 6th International Systems and Storage Conference on—SYSTOR '13, Haifa, Israel, 2–4 June 2013; p. 1. [CrossRef]
271. Bishop, B.; Kiryakov, A.; Ognyanoff, D.; Peikov, I.; Tashev, Z.; Velkov, R. OWLIM: A family of scalable semantic repositories. *Semantic Web* **2011**, *2*, 33–42. [CrossRef]
272. Memcached A distributed memory object caching system. Available online: https://memcached.org/ (accessed on 18 July 2022).
273. Ananthanarayanan, G.; Ghodsi, A.; Wang, A.; Borthakur, D.; Kandula, S.; Shenker, S.; Stoica, I. PACMan: Coordinated memory caching for parallel jobs. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, San Jose, CA, USA, 25–27 April 2012; p. 20.
274. Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W.C.; Wallach, D.A.; Burrows, M.; Chandra, T.; Fikes, A.; Gruber, R.E. Bigtable: A Distributed Storage System for Structured Data. In Proceeding of the 7th Symposium on Operating Systems Design and Implementation, Seattle, WA, USA, 6–8 November 2006. [CrossRef]
275. Martinec, J.; Rango, A.; Major, E. *The Snowmelt-Runoff Model (SRM) User's Manual*; New Mexico State University: Las Cruces, NM, USA, 1983.
276. Rajasekar, A.; Moore, R.; Hou, C.-Y.; Lee, C.A.; Marciano, R.; de Torcy, A.; Wan, M.; Schroeder, W.; Chen, S.-Y.; Gilbert, L.; et al. iRODS Primer: Integrated Rule-Oriented Data System. *Synth. Lect. Inf. Concepts Retr. Serv.* **2010**, *2*, 1–143. [CrossRef]

277. Plimpton, S.J.; Devine, K.D. MapReduce in MPI for Large-scale graph algorithms. *Parallel Comput.* **2011**, *37*, 610–632. [CrossRef]
278. Mantha, P.K.; Luckow, A.; Jha, S. Pilot-MapReduce. In Proceedings of the third international workshop on MapReduce and its Applications Date - MapReduce '12, Delft, The Netherlands, 18–19 June 2012; pp. 17–24. [CrossRef]
279. Schwan, P.; Schwan, P. Lustre: Building a file system for 1000-node clusters. *PROC. 2003 LINUX Symp.* **2003**, *2003*, 380–386.
280. Owre, S.; Shankar, N.; Rushby, J.M.; Stringer-Calvert, D.W.J. PVS System Guide. *SRI Int.* **2001**, *1*, 7.
281. Jeannot, E.; Mercier, G.; Tessier, F. Process Placement in Multicore Clusters:Algorithmic Issues and Practical Techniques. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 993–1002. [CrossRef]
282. Wang, Y. Smart: A MapReduce-Like Framework for In-Situ Scientific Analytics. In Proceedings of the SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, TX, USA, 15–20 November 2015.
283. Xu, C.; Goldstone, R.; Liu, Z.; Chen, H.; Neitzel, B.; Yu, W. Exploiting Analytics Shipping with Virtualized MapReduce on HPC Backend Storage Servers. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *27*, 185–196. [CrossRef]
284. Mimi, L. OLCF Group to Offer Spark On-Demand Data Analysis. Available online: https://www.olcf.ornl.gov/2016/03/29/olcf-group-to-offer-spark-on-demand-data-analysis/ (accessed on 15 June 2022).
285. Apache Hadoop C API libhdfs. Available online: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/LibHdfs.html (accessed on 5 May 2022).
286. Jin, H.; Ji, J.; Sun, X.-H.; Chen, Y.; Thakur, R. CHAIO: Enabling HPC Applications on Data-Intensive File Systems. In Proceedings of the 2012 41st International Conference on Parallel Processing, Pittsburgh, PA, USA, 10–13 September 2012; pp. 369–378. [CrossRef]
287. Hoefler, T.; Lumsdaine, A.; Dongarra, J. *Towards Efficient MapReduce Using MPI*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 240–249. [CrossRef]
288. Matsunaga, A.; Tsugawa, M.; Fortes, J. CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. In Proceedings of the 2008 IEEE Fourth International Conference on eScience, Indianapolis, IN, USA, 7–12 December 2008; pp. 222–229. [CrossRef]
289. HTCondor—High Throughput Computing. Available online: https://research.cs.wisc.edu/htcondor/ (accessed on 20 June 2022).
290. Zhang, Z.; Barbary, K.; Nothaft, F.A.; Sparks, E.; Zahn, O.; Franklin, M.J.; Patterson, D.A.; Perlmutter, S. Scientific computing meets big data technology: An astronomy use case. In Proceedings of the 2015 IEEE International Conference on Big Data (Big Data), Santa Clara, CA, USA, 29 October–1 November 2015; pp. 918–927. [CrossRef]
291. Lu, X.; Wang, B.; Zha, L.; Xu, Z. Can MPI Benefit Hadoop and MapReduce Applications? In Proceedings of the 2011 40th International Conference on Parallel Processing Workshops, Taipei City, Taiwan, 13–16 September 2011; pp. 371–379. [CrossRef]
292. Veiga, J.; Exp, R.R.; Taboada, G.L.; Touri, J. Analysis and Evaluation of MapReduce Solutions on an HPC Cluster. *Comput. Electr. Eng.* **2015**, *50*, 200–2016. [CrossRef]
293. Mohamed, H.; Marchand-Maillet, S. Enhancing MapReduce Using MPI and an Optimized Data Exchange Policy. In Proceedings of the 2012 41st International Conference on Parallel Processing Workshops, Pittsburgh, PA, USA, 10–13 September 2012; pp. 11–18. [CrossRef]
294. Ranger, C.; Raghuraman, R.; Penmetsa, A.; Bradski, G.; Kozyrakis, C. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, Scottsdale, AZ, USA, 10–14 February 2007; pp. 13–24. [CrossRef]
295. Lu, X.; Rahman, W.U.; Islam, N.; Shankar, D.; Panda, D.K. Accelerating Spark with RDMA for Big Data Processing: Early Experiences. In Proceedings of the 2014 IEEE 22nd Annual Symposium on High-Performance Interconnects, Mountain View, CA, USA, 26–28 August 2014; pp. 9–16. [CrossRef]
296. Lu, X.; Liang, F.; Wang, B.; Zha, L.; Xu, Z. DataMPI: Extending MPI to Hadoop-Like Big Data Computing. In Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, 19–23 May 2014; pp. 829–838. [CrossRef]
297. Wang, Y.; Jiao, Y.; Xu, C.; Li, X.; Wang, T.; Que, X.; Cira, C.; Wang, B.; Liu, Z.; Bailey, B.; et al. *Assessing the Performance Impact of High-Speed Interconnects on MapReduce*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 148–163. [CrossRef]
298. Yu, W.; Wang, Y.; Que, X. Design and Evaluation of Network-Levitated Merge for Hadoop Acceleration. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 602–611. [CrossRef]
299. Woodie, A. Does InfiniBand Have a Future on Hadoop? *HPC Wire* **2015**.
300. Unstructured Data Accelerator (UDA). Available online: https://format.com.pl/site/wp-content/uploads/2015/09/sb_hadoop.pdf (accessed on 4 January 2022).
301. Mellanox Technologies: End-to-End InfiniBand and Ethernet Interconnect Solutions and Services. Available online: http://www.mellanox.com/ (accessed on 23 November 2022).
302. Chu, V.K.J. Transmission of IP over InfiniBand (IPoIB). Available online: https://www.rfc-editor.org/rfc/rfc4391.html (accessed on 25 November 2021).
303. Woodie, A. Unravelling Hadoop Performance Mysteries. Available online: https://www.enterpriseai.news/2014/11/20/unravelling-hadoop-performance-mysteries/ (accessed on 17 June 2022).

304. Islam, N.S.; Lu, X.; Rahman, W.U.; Panda, D.K. Can Parallel Replication Benefit Hadoop Distributed File System for High Performance Interconnects? In Proceedings of the 2013 IEEE 21st Annual Symposium on High-Performance Interconnects, San Jose, CA, USA, 21–23 August 2013; pp. 75–78. [CrossRef]
305. Rahman, W.U.; Islam, N.S.; Lu, X.; Jose, J.; Subramoni, H.; Wang, H.; Panda, D.K.D. High-Performance RDMA-based Design of Hadoop MapReduce over InfiniBand. In Proceedings of the 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, 20–24 May 2013; pp. 1908–1917. [CrossRef]
306. Islam, N.S.; Rahman, M.W.; Jose, J.; Rajachandrasekar, R.; Wang, H.; Subramoni, H.; Murthy, C.; Panda, D.K. High performance RDMA-based design of HDFS over InfiniBand. In Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis, Atlanta, GA, USA, 14–19 November 2012; pp. 1–12. [CrossRef]
307. Lu, X.; Islam, N.S.; Rahman, W.U.; Jose, J.; Subramoni, H.; Wang, H.; Panda, D.K. High-Performance Design of Hadoop RPC with RDMA over InfiniBand. In Proceedings of the 2013 42nd International Conference on Parallel Processing, Lyon, France, 1–4 October 2013; pp. 641–650. [CrossRef]
308. Turilli, M.; Santcroos, M.; Jha, S. A Comprehensive Perspective on Pilot-Job Systems. *ACM Comput. Surv.* **2019**, *51*, 1–32. [CrossRef]
309. Jones, M.; Nelson, M. Moving ahead with Hadoop YARN. Available online: https://www.ibm.com/developerworks/library/bd-hadoopyarn/ (accessed on 16 May 2018).
310. Petcu, D.; Iuhasz, G.; Pop, D.; Talia, D.; Carretero, J.; Prodan, R.; Fahringer, T.; Grasso, I.; Doallo, R.; Martin, M.J.; et al. On Processing Extreme Data. *Scalable Comput. Pr. Exp.* **2016**, *16*, 467–490. [CrossRef]
311. Da Costa, G.; Fahringer, T.; Gallego, J.A.R.; Grasso, I.; Hristov, A.; Karatza, H.D.; Lastovetsky, A.; Marozzo, F.; Petcu, D.; Stavrinides, G.L.; et al. Exascale Machines Require New Programming Paradigms and Runtimes. *Supercomput. Front. Innov.* **2015**, *2*, 6–27. [CrossRef]
312. Usman, S.; Mehmood, R.; Katib, I.; Albeshri, A.; Altowaijri, S.M. ZAKI: A Smart Method and Tool for Automatic Per-formance Optimization of Parallel SpMV Computations on Distributed Memory Machines. *Mob. Networks Appl.* **2019**. [CrossRef]
313. Usman, S.; Mehmood, R.; Katib, I.; Albeshri, A. ZAKI+: A Machine Learning Based Process Mapping Tool for SpMV Computations on Distributed Memory Architectures. *IEEE Access* **2019**, *7*, 81279–81296. [CrossRef]
314. Emani, M.K.; Wang, Z.; O'Boyle, M.F.P. Smart, adaptive mapping of parallelism in the presence of external workload. In Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Washington, DC, USA, 23–27 February 2013; pp. 1–10. [CrossRef]
315. Diener, M. *Automatic Task and Data Mapping in Shared Memory Architectures*; Technische Universität Berlin: Berlin, Germany, 2015.
316. Subramoni, H. *Topology-Aware MPI Communication and Scheduling for High Performance Computing Systems*; Computer Science and Engineering; Ohio State University: Columbus, OH, USA, 2013.
317. Kulkarni, M.; Pingali, K.; Walter, B.; Ramanarayanan, G.; Bala, K.; Chew, L.P. Optimistic parallelism requires abstractions. In Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation—PLDI, New York, NY, USA, 10–13 June 2007; pp. 211–222. [CrossRef]
318. Keutzer, K.; Mattson, T. *Our Pattern Language—Our Pattern Language*; WordPress, 2016.
319. Mysore, S.J.D.; Khupat, S. Big data architecture and patterns, Part 1: Introduction to big data classification and architecture. *IBM* **2013**.
320. Zanoni, M.; Fontana, F.A.; Stella, F. On applying machine learning techniques for design pattern detection. *J. Syst. Softw.* **2015**, *103*, 102–117. [CrossRef]
321. Dwivedi, A.K.; Tirkey, A.; Ray, R.B.; Rath, S.K. Software design pattern recognition using machine learning techniques. In Proceedings of the 2016 IEEE Region 10 Conference (TENCON), Singapore, 22–25 November 2017; pp. 222–227. [CrossRef]