International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:10, No:3, 2016

A Survey on Data-Centric and Data-Aware Techniques for Large Scale Infrastructures

Silvina Caíno-Lores, Jesús Carretero

***Abstract*—** Large scale computing infrastructures have been widely developed with the core objective of providing a suitable platform for high-performance and high-throughput computing. These systems are designed to support resource-intensive and complex applications, which can be found in many scientific and industrial areas. Currently, large scale data-intensive applications are hindered by the high latencies that result from the access to vastly distributed data. Recent works have suggested that improving data locality is key to move towards exascale infrastructures efficiently, as solutions to this problem aim to reduce the bandwidth consumed in data transfers, and the overheads that arise from them. There are several techniques that attempt to move computations closer to the data. In this survey we analyse the different mechanisms that have been proposed to provide data locality for large scale high-performance and high-throughput systems. This survey intends to assist scientific computing community in understanding the various technical aspects and strategies that have been reported in recent literature regarding data locality. As a result, we present an overview of locality-oriented techniques, which are grouped in four main categories: application development, task scheduling, in-memory computing and storage platforms. Finally, the authors include a discussion on future research lines and synergies among the former techniques.

***Keywords*—** Co-scheduling, data-centric, data-intensive, data locality, in-memory storage, large scale.

I. Introduction

LARGE scale infrastructures, such as supercomputers, grids, clouds and clusters, have been widely developed with the core objective of providing a suitable platform for high-performance and high-throughput computing. As these paradigms typically require massive hardware resources and dedicated middleware, large scale computing holds specific challenges in order to achieve sufficient efficiency in terms of memory, CPU, I/O, network latencies, and power consumption, to name a few. These systems are oriented towards supporting resource-demanding and complex applications, which can be found in many scientific and industrial areas, such as bioengineering, physics, climate modelling, and health sciences. Therefore, large scale infrastructures have a key role in many fields of research. This motivates the special attention they get from computer scientists, and the numerous works that are published every year that aim to improve them.

There are several issues that are still not solved by the academia with regard to these infrastructures. In particular, computer scientists have realised that, as problems become larger and more complex, a powerful infrastructure is not sufficient to achieve proper scalability, both in terms of overall performance, resource utilisation, and power efficiency.

S. Caíno-Lores is with the Department of Computer Science and Engineering, University Carlos III of Madrid, Madrid, Spain, 28911 (e-mail: scaino@arcos.inf.uc3m.es).

J. Carretero is with the Department of Computer Science and Engineering, University Carlos III of Madrid, Madrid, Spain, 28911.

Recent works have suggested that improving data locality is key to move towards exascale infrastructures efficiently [1]. With the main goal of summarising the current trends in data locality reinforcement for large scale infrastructures, this survey analyses the most relevant publications in this topic, and provides highlights on promising future research. A proper understanding of the opportunities in this direction would be very useful for the scientific community, but also for system architects, platform designers, application developers and final users, who could take advantage of this knowledge to build more efficient systems and applications. Given the former, this survey provides the following contributions:

- An extensive analysis on works related to data locality for large scale infrastructures.
- An overview of the most relevant locality-oriented techniques.
- A discussion on future research lines and synergies among the former techniques.

The rest of this paper is organized as follows: Section II develops the main research areas detected after the analysis of the selected works, Section III categorises works that are related to programming models and workflows, Section IV analyses the techniques that are related to resource and task scheduling, Section V develops the works focused on moving the computations to the node's local storage and memory, Section VI analyses the works directed towards data-aware storage systems, Section VII discusses future research lines and opportunities in the light of the aggregated results of the selected works, finally, Section VIII provides key ideas as conclusions of this review.

II. Data-Locality Techniques

Large scale systems typically present a series of key elements in their architectures. Relevant work improving data locality has been found for most of these components. In order to organise the selected works in a comprehensive manner, we grouped them in four sections, each one related to one of the four core layers that constitute the architecture of a high-performance system. Fig. 1 constitutes a representation of this stack, in which the following layers are reflected:

1) **Applications:** one of the purposes of large scale systems is to support the execution of complex applications with heavy resource requirements. The ability to design and develop applications with a focus on data locality has been an interesting research topic, from which several

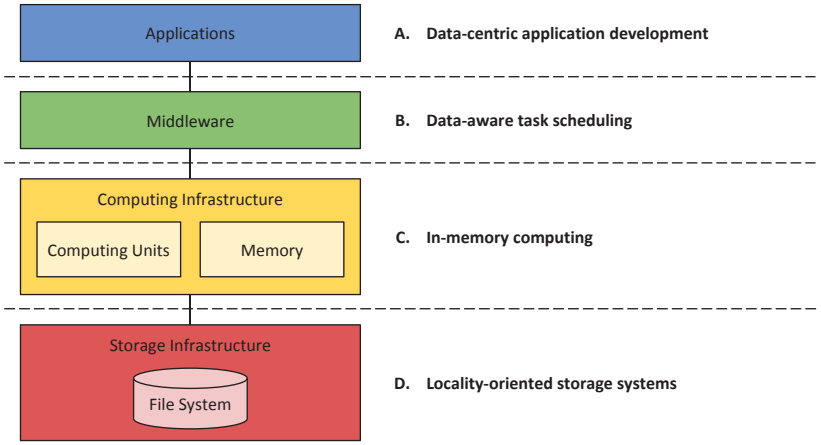

Fig. 1 System layers and their match to the major research topics detected in this survey

programming models, frameworks and workflows have arisen. These works are described in Section III.

2) **Middleware:** complex systems need dedicated platforms that orchestrate tasks and manage resources in order to behave in a coordinated manner, and meet the requirements of the applications. These pieces of software constitute the middleware that permits node intercommunication, data transmission, load balancing, task assignment and fault tolerance, among others. As these platforms control main features within the system, their role is key with regard to data awareness. This is reflected in the numerous works that present data-centric scheduling and load balancing techniques for middleware platforms, which is covered in Section IV.
3) **Computing infrastructure:** the infrastructure that lies beneath the middleware has a major impact in the final performance of the system. In this particular topic, a tailored infrastructure can contribute to the locality of the data or break the whole paradigm provided by upper layers, even in a transparent way. Work in this area has been conducted to improve locality by moving data to the node's memory to minimise interaction with storage nodes. This topic is discussed in Section V.
4) **Storage Infrastructure:** storage systems constitute one of the greatest bottlenecks when dealing with data-intensive computations. Therefore, data awareness in file systems and storage infrastructures can significantly improve the system's overall locality, as other layers can benefit from the system's knowledge of data placement. The works conducted towards this direction are described in Section VI.

The main topics proposed by the authors do not yield a strict classification of the selected works in this survey. In fact, there are synergies and common research lines among them that motivate further discussion for future work. The authors provide a selection of promising research lines and their synergies in Section VII.

III. Data-Centric Programming Models

As previously introduced, minimising data movements is very important for the final performance. At the application development stage, working with programming models that provide a data processing layer able to abstract resource allocation, data management and task execution can result in an improvement in performance and locality.

The map-reduce [2] data processing model is probably the most relevant data-centric model, as it enables analytics on big datasets by parallelising computations for HPC and multi-core environments [3]. Besides the numerous works that took advantage of it to improve performance of a wide range of applications, it had a major impact in subsequent map-reduce-inspired models. A map-reduce-based algorithm consists of a two-phase algorithm that takes as input a set of key-value pairs retrieved from the input files. The input is split across a group of homogeneous *map* functions, which process the data and forward the result to the *reduce* tasks in order to write the final result. The original map-reduce implementation by Google relies on the Google File System (GFS) [4] to achieve locality by block replication, and considers data-aware task scheduling. A similar approach is followed by the open map-reduce implementation, Hadoop [5], and its partner file system Hadoop Distributed File System (HDFS) [6].

One of the models that emerged from map-reduce is map-reduce-merge [7], a model that adds a *merge* phase that can efficiently aggregate the data already partitioned and sorted by the map and reduce modules. Map-reduce does not directly support processing multiple related heterogeneous datasets, limitation that causes efficiency issues when map-reduce is applied in relational operations like joins. The map-reduce-merge model can, on the other hand, express relational algebra operators and implement several join algorithms.

MapIterativeReduce [8] is an alternative model that extends map-reduce to better support reduce-intensive applications, while substantially improving its efficiency by eliminating the implicit barrier between the map and the reduce phases. Among implementations of map-iterative-reduce we can find

Twister [9], Haloop [10] and Twister4Azure [11]. Twister assumes that the intermediate data produced after the map stage of the computation will fit into the distributed memory. Twister4Azure locally caches the loop-invariant input data in the workers' memory and storage to improve scalability in Cloud environment. HaLoops scheduler places on the same physical machines those map and reduce tasks that occur in different iterations but access the same data. With this approach, data can be more easily cached and re-used between iterations. HaLoops maintains three types of caches: reducer input cache, reducer output cache, and mapper input cache.

The Spark [12] programming model supports a wide range of functionalities that enable the development of applications that do not fit nicely the map-reduce paradigm, such as many iterative machine learning algorithms and interactive data analysis tools. Spark reuses a working set of data, known as resilient distributed dataset (RDD) [13], through multiple parallel operations, built around an acyclic data flow model. It retains, however, the scalability and fault tolerance features of map-reduce.

Map-reduce-based programming models have also evolved into language frameworks that provide a data access layer through a set of APIs, thus eliminating the need to re-implement repetitive tasks by working on top of the processing layer. Pig Latin [14] is a high-level data-flow language and execution framework whose compiler produces sequences of batch processing map-reduce programs. Pig offers SQL-style high-level data manipulation constructs, which can be assembled in an explicit dataflow and interleaved with custom map- and reduce-style functions or executables [15].

Another popular approach is Hive [16], an open-source data warehousing solution. Hive supports queries expressed in a SQL-like declarative language known as HiveQL, which are compiled into map-reduce jobs. In addition, HiveQL enables users to plug in custom map-reduce scripts into queries. Hive adds special optimisations to improve data-locality and reduce data-transfer overhead, such as pruning unnecessary files from partitions on the file system, and buffering small tables in the distributed main memory of worker nodes for faster access.

Map-reduce is able to process large amounts of partitioned input datasets by spawning a set of homogeneous map and reduce tasks. To improve sharing of such input, CloudFlow [17] offers scheduling optimisations at the function and job levels, based on the access frequency of different datasets. A shared job data handler identifies multiple jobs of different users, finds the frequently- or partially-shared data items, and copies them to the local file system of the nodes for future use. Additionally, a shared function data handler delivers the shared data to the map functions that need it by means of a data-centric pre-fetching mechanism, instead of following a pull scheme. Therefore, when computation tasks are located away from the data they consume, the data they need can be pushed near the compute node to improve data locality.

New technologies based in multi-core processors can improve the performance of applications by favouring intra-node data sharing, which minimises data exchanges across compute nodes. The work in [18] aims to enhance intra-node data sharing, proposing a distributed data sharing and task execution framework. This tool has two main objectives: map tasks to processor cores to maximise intra-node data sharing and locality, and provide a shared space programming abstraction that replaces existing parallel programming models such as message passing.

Another approach is followed by Dryad [19], a general-purpose distributed execution engine for coarse-grain data-parallel applications. A Dryad application combines computational nodes with communication channels to form a dataflow graph. Dryad runs the application by executing the vertices of this graph on the available nodes of a distributed environment, or in several CPUs for single-node machines. The application can infer the size and placement of data at run time, and modify the graph as the computation progresses to make an efficient use of the available resources.

Finally, Nephele [20] is a data processing framework that aims to exploit the dynamic resource allocation offered by compute clouds for both task scheduling and execution. It allows to assign the particular tasks of a processing job to different types of virtual machines, and takes care of their instantiation and termination during the job execution. Similar to Microsofts Dryad [19], jobs in Nephele are expressed as a directed acyclic graph (DAG). Currently, in Nephele the only way to ensure locality between tasks is to execute them on the same virtual machine in the Cloud.

IV. Data-Aware Scheduling

This section discusses techniques and algorithms focused on scheduling tasks in multi-node environments. Scheduling in large-scale systems is a wide concept: it faces the allocation of a set of tasks to multiple processors and the establishment of execution order [21]. From the perspective of this work, schedulers would aim to allocate tasks to a set of compute nodes, assigning tasks to nodes that already have the input data, or at least to those nodes that are close to each other.

The authors of [22] propose an algorithm that allocates tasks to nodes mainly considering the closeness of the input data, and controls the movement of information when the storage node and the compute node are different. The emphasis is on providing a solution where the configuration of the system can change over the time and where the hardware can vary between different nodes, which is known as a heterogeneous system. Firstly, the algorithm allocates a task depending on the input data and data stored in the destination node. If no task meets this criterion, then the algorithm allocates the task which data is closer to the compute node, moving the required data to that node. Here, the algorithm requests tasks with the shorter transmission time, computed based on a constant speed.

A different approach is introduced in [23]. It presents a heuristic task-scheduling algorithm called balance-reduce (BAR). First, it allocates a task and then tunes the initial task allocation in order to reduce the job completion time. BAR adjusts data locality dynamically based on the cluster and network workload. This scheduling algorithm takes into account the amount of data located in different servers, the execution cost of the task and the server workload, but it

requires prior knowledge about the amount of tasks and the load of the servers. When the network is poor, BAR enhances data locality; when the cluster is overloaded, it decreases data locality to start the tasks earlier.

In [24], the authors present a scheduler called Stork. It manages different components of the system, such as the I/O system and CPU. The scheduler uses a job description language used to represent the job's data placement. Stork controls the load of the system, managing the number of simultaneous accesses to each storage node. By applying different policies, the algorithm supports several options to establish an order in data transfers and to reduce overload.

Minimizing the number of movements of files between storage nodes and compute nodes is the main thrust of [25]. The algorithm is based in an hypergraph, which consists in a set of vertices with different weights connected by hyper-edges with different costs. This creates a set of nets between the nodes. The purpose of this hypergraph is to represent the state of the system with its jobs, files and compute and storage nodes. This hypergraph can be partitioned into several groups representing the compute nodes, which allows the algorithm to balance and minimize files transferring costs. Later, the order of the jobs in each compute node is decided. It uses a strategy that prioritises the jobs with the shortest execution time, which is calculated taking into consideration the time to retrieve its input files.

In [26], a scheduler to improve data locality for map-reduce jobs is proposed. The basic idea is to predict the most appropriate nodes to which future map tasks should be assigned. Then input data can be preload to node memory without any delaying on launching new tasks. Thus, data prefetching is carried out concurrently with data processing.

A solution developed for Hadoop is explained in [27]. The authors propose a new job scheduling policy that seeks to take advantage of shared input files among different jobs to improve data locality. Their main goal is to allow tasks that share the same input block to be scheduled sequentially and in the same compute node even, if there are tasks belonging different jobs. When a node requests a new task, the policy looks for a task that handles the same data block used by the task just completed. Regarding Hadoop, delayed scheduling [28] also attempts to achieve high data locality. When an idle node requests a task, jobs are sorted in a queue according to the number of running local tasks in the said node. The algorithm also incorporates a mechanism to avoid starvation, leveraging fairness and data locality.

Other approaches follow decentralised models in which each compute node has its own meta-scheduler, aiming to reduce the overall makespan and improve the resource performance. In [29], the authors propose a model for independent tasks in federated grids. In this work each infrastructure runs a scheduling algorithm, which looks for a reduction in the makespan on the running application. Another perspective is provided in [30]: worker-centric scheduling. Unlike task-centric scheduling, in this technique the worker takes the initiative and request to the scheduler a task, but only when it is idle. One of the advantages of this worker-centric algorithm is the ability to balance task allocation, which yields a reduction in latencies.

V. In-Memory Computing for Data Locality Enforcement

As we have previously described, the actual infrastructure that lies beneath a large scale application has a key role in its final performance. This also applies to data awareness and data locality support, as moving data to the node's memory, and reusing already existent or processed data, minimises the interaction with storage nodes at the infrastructure level.

The work in [31] suggests that iterative and interactive applications are the ones that could take the highest advantage of in-memory data storage for fast reuse. The Spark framework relies heavily in the concept of *resilient distributed dataset* (RDD) [13] to provide this functionality. RDDs are in-memory collections of data, and the operations on them are tracked in order to provide significant fault tolerance. According to its authors, the system has proven to be highly scalable, fault tolerant and fast.

Spark has inspired subsequent works like GraphX [32], which extends the framework to support graph parallel computing. Working with graphs has, as indicated by the authors, specific challenges and requirements that were not fully addressed by previous works. GraphX aims to introduce fault tolerant, parallel data processing to graph processing, with a focus on in-memory computing for effective distribution of the work-load.

Shark [33], which supports the Hive warehousing system [34] on Spark, offers a similar approach, but oriented towards SQL-based data analytics by means of machine learning. These algorithms are typically iterative, thus in-memory computing suits well the need for cached data to be reused. Nevertheless, the authors emphasise that coarse-grained distributed shared-memory yields better performance that fine-grained memory in this case. This is due to the huge size of the data sets, which makes impossible to update records individually.

The work in [35] follows a different approach, yet it aims to improve data locality by means of data reuse and replication, especially for loosely-coupled data-intensive applications. This work overlaps with the scheduling section as well, as its goal is to define a scheduling heuristic to expose data reuse and replication patterns across the system, which are exploited by the scheduler.

Piccolo [36] is a data-centric programming model for writing parallel in-memory applications in data centers. Piccolo allows computation running on different machines to share distributed, mutable state via a key-value table interface. Piccolo enables efficient application implementations. In particular, applications can specify locality policies to exploit the locality of shared state access. Then, Piccolo's run-time engine automatically resolves write-write conflicts using user defined accumulation functions.

The former works indicate that, even if it seems counter-intuitive, in-memory databases and computing are able to scale to petascale systems. No further work has found indicating whether this could hold for exascale systems though.

VI. Locality-Oriented Storage Platforms

The storage platform beneath an application and its supporting platform can be tailored to help with data locality provisioning. In this section we include relevant techniques that aim to increase the level of data-awareness in data storage abstractions.

To achieve efficient data locality, one would expect a degree of coordination between the service provided by the storage layer and the application's processes, as these could compete for resources and create I/O bottlenecks. The work by Chen et al. [37] proposes data layout awareness to avoid the contention caused by process interruption on I/O transactions. This is built by allowing a specific process to be serviced continuously, instead of interrupted randomly by other processes by combining the I/O requests via aggregators that issue them on behalf of all processes. In this strategy, these aggregated requests and the partitions of file domains are rearranged in such a way that each aggregator is able to access data contiguously. Additionally, multiple aggregators can access file servers concurrently, thus exploiting better locality and reducing contention.

Regarding coordination, VIDAS (Virtualized DAta Sharing) [38] constitutes an object-based and virtualised data store that aims to minimise intra-node data movement. It relies on the integration of data sharing, storage I/O coordination, and data locality awareness to provide efficient data sharing among co-located virtual machines.

In ROMIO [39], such coordination is achieved by means of a two phase approach for accessing data. The first stage allows the processes to perform non-contiguous I/O requests according to the data distribution across the disks, which results in each process making a single, large and contiguous data retrieval. In the second phase, processes redistribute data among themselves to match the application's desired data localisation. The advantage of this method is that, by making all file accesses large and contiguous, the I/O time is reduced significantly to levels that match independent requests. The added cost of inter-process communication for data redistribution is small compared with the savings in I/O time.

As parallel file systems can significantly improve locality if designed with such purpose, there are examples of no-SQL database systems that aim to improve the overall efficiency of the data analysis process. To maintain performance when the amount of data grows to the levels of large data-intensive applications, these databases need a data-awareness component to efficiently manipulate such input.

Bigtable [40] provides a data model that allows dynamic control over the data layout an format. Additionally, it permits the users to manipulate the properties related to locality in the data structure, so that their applications can benefit from a tailored schema. Following the trend of increasingly relying in memory, Bigtable supports both disk-based and in-memory storage.

Nowadays, key-value stores are increasingly used to manipulate large amounts of data. The work in [41] proposes a database that is able to self-organise data replicas according to data access locality patterns. The system is decentralised in order to optimise object placement. The authors also investigate how to detect the optimal number of replicas with respect to look-up efficiency.

Finally, systems like HBase [42], built on top of HDFS, enhance data locality for binary large objects (BLOBs) [43]. This yields several challenges, as the huge size of the database makes mandatory to distribute it in order to make it efficient. HBase combines the HDFS block-based file system, which is in charge of storing the data, with a series of metadata that permits to retrieve the location of the queried information. Data locality is achieved both by the underlying file system and by the ability to detect where the data is stored.

VII. Discussion

This section finally analyses the common aspects of the selected references, and exposes the future lines of work that arise from the reviewed literature.

In the case of the scheduling algorithms, dynamic scheduling depending on the system's conditions constitutes the main open challenge in this topic. Since the objective is to use resources as efficiently as possible, this scheduling approaches yield the need for real-time monitoring.

Moving storage from disks to memory is a current trend in large scale computing due to the reduced response time this provides. Nevertheless, memory performance and fault-tolerance is key to achieve the level of robustness that traditional storage systems have attained. Additionally, the development of efficient persistence mechanisms is an issue that must be tackled in order to make in-memory computing reliable enough for large scale production systems.

In-memory computing can be immediately related to data reuse and replication, as it minimises transfer times between the computing and storage units. Moreover, it has major synergies with data-centric application development techniques, in a similar way as distributed file systems helped with workflow input data awareness.

Data-aware scheduling is an area that could greatly benefit in-memory computing, as locating a task in the node that holds its input in memory could improve the overall system's performance. We have also detected that merging in-memory datasets with no-SQL storage systems is a promising trend that already shows interesting results, but its scalability and reliability for ultra scale systems must be assessed in future work.

There is a need for further research in parallel file systems and storage architectures to support data awareness. Past research has focused on merging workflows, programming models and parallel file systems. However, data awareness should be provided in a transparent and smart manner at the storage layer, which would benefit the entire distributed system stack, and not just some of its applications.

Some works have introduced the promising possibility to perform intelligent data replication by monitoring the overall system's behaviour. This could be further developed by integrating adaptive scheduling with the storage infrastructure,

seeing both stored data and tasks as a whole. In this context, scalability would be a major concern due to the introduced overhead.

VIII. Conclusions

Large scale computing infrastructures have a major impact in science and society. Nevertheless, several scalability issues arise with large and complex problems, especially affecting performance, resource utilisation and power efficiency. In this context, these systems must implement algorithms and policies that, with the existing devices and hardware, are able to reap the highest possible output.

This work reviews and summarises the most relevant publications related to data locality techniques, which are promising methods that could improve scalability and sustainability in a significant manner. After the analysis of these works we were able to discuss future research lines that could be particularly promising in this area. Dynamism and adaptability in data transfer, task scheduling and data replication are key to build smart systems that could exploit the relationship between data and processes. Nonetheless, this requires efficient and reliable real-time monitoring, which constitutes one of the major challenges in large scale computing in general.

With the increasing amount of available memory, in-memory datasets seem to be the main technique to support fast large scale storage systems, especially no-SQL databases. However, it is required further research in the scalability and fault-tolerance of current memory systems. These techniques could be combined with transparent data-aware persistence supported by parallel file systems.

Acknowledgment

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness, under the grant TIN2013-41350-P-2014-2016 (High-end computing systems scalable data management techniques).

References

[1] K. Yelick, S. Coghlan, B. Draney, R. S. Canon *et al.*, "The magellan report on cloud computing for science," *US Department of Energy, Washington DC, USA, Tech. Rep*, 2011.

[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[3] J. Fritsch and C. Walker, "The problem with data," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*. IEEE, 2014, pp. 708–713.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[5] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.

[6] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, May 2010, pp. 1–10.

[7] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-reduce-merge: simplified relational data processing on large clusters," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 1029–1040.

[8] R. Tudoran, A. Costan, and G. Antoniu, "Mapiterativereduce: a framework for reduction-intensive data processing on azure clouds," in *Proceedings of third international workshop on MapReduce and its Applications Date*. ACM, 2012, pp. 9–16.

[9] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 810–818.

[10] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.

[11] T. Gunarathne, B. Zhang, T.-L. Wu, and J. Qiu, "Scalable parallel computing on clouds using twister4azure iterative mapreduce," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1035–1048, 2013.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10, Berkeley, CA, USA, 2010, pp. 10–10.

[13] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "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, 2012, pp. 2–2.

[14] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 1099–1110.

[15] C. Dobre and F. Xhafa, "Parallel programming paradigms and frameworks in big data era," *International Journal of Parallel Programming*, vol. 42, no. 5, pp. 710–738, 2014.

[16] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive-a petabyte scale data warehouse using hadoop," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*. IEEE, 2010, pp. 996–1005.

[17] F. Zhang, Q. M. Malluhi, T. Elsayed, S. U. Khan, K. Li, and A. Y. Zomaya, "Cloudflow: A data-aware programming model for cloud workflow applications on modern hpc systems," *Future Generation Computer Systems*, 2014.

[18] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi, "Enabling in-situ execution of coupled scientific workflow on multi-core platform," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 1352–1363.

[19] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.

[20] D. Warneke and O. Kao, "Nephele: efficient parallel data processing in the cloud," in *Proceedings of the 2nd workshop on many-task computing on grids and supercomputers*. ACM, 2009, p. 8.

[21] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 260–274, 2002.

[22] X. Zhang, Y. Feng, S. Feng, J. Fan, and Z. Ming, "An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments," in *Cloud and Service Computing (CSC), 2011 International Conference on*. IEEE, 2011, pp. 235–242.

[23] V. W. Thawari, S. D. Babar, N. A. Dhawas, and others, "An efficient data locality driven task scheduling algorithm for cloud computing," *International Journal in Multidisciplinary and Academic Research (SSIJMAR)*, vol. 1, no. 3, 2012.

[24] T. Kosar and M. Balman, "A new paradigm: Data-aware scheduling in grid computing," *Future Generation Computer Systems*, vol. 25, no. 4, pp. 406–413, 2009. (Online). Available: http://www.sciencedirect.com/science/article/pii/S0167739X08001520

[25] G. Khanna, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz, "A data locality aware online scheduling approach for I/O-intensive jobs with file sharing," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2007, pp. 141–160.

[26] M. Sun, H. Zhuang, X. Zhou, K. Lu, and C. Li, "HPSO: Prefetching Based Scheduling to Improve Data Locality for MapReduce Clusters," in *Algorithms and Architectures for Parallel Processing*. Springer, 2014, pp. 82–95.

[27] A. Bezerra, P. Hernandez, A. Espinosa, and J. C. Moure, "Job scheduling for optimizing data locality in hadoop clusters," in *Proceedings of the 20th European MPI Users' Group Meeting*. ACM, 2013, pp. 271–276.

[28] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10.

New York, NY, USA: ACM, 2010, pp. 265–278. (Online). Available: http://doi.acm.org/10.1145/1755913.1755940
[29] K. Leal, E. Huedo, and I. M. Llorente, "A decentralized model for scheduling independent tasks in Federated Grids," *Future Generation Computer Systems*, vol. 25, no. 8, pp. 840–852, Sep. 2009. (Online). Available: http://www.sciencedirect.com/science/article/pii/S0167739X09000211
[30] S. Y. Ko, R. Morales, and I. Gupta, "New worker-centric scheduling strategies for data-intensive grid applications," in *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*. Springer-Verlag New York, Inc., 2007, pp. 121–142. (Online). Available: http://dl.acm.org/citation.cfm?id=1516134
[31] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 810–818. (Online). Available: http://doi.acm.org/10.1145/1851476.1851593
[32] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management Experiences and Systems*, ser. GRADES '13. New York, NY, USA: ACM, 2013, pp. 2:1–2:6.
[33] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Fast data analysis using coarse-grained distributed memory," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 689–692.
[34] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.
[35] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima, "Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Springer Berlin Heidelberg, 2005, vol. 3277, pp. 210–232.
[36] R. Power and J. Li, "Piccolo: Building fast, distributed programs with partitioned tables." in *OSDI*, vol. 10, 2010, pp. 1–14.
[37] Y. Chen, X.-H. Sun, R. Thakur, H. Song, and H. Jin, "Improving parallel i/o performance with data layout awareness," in *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, Sept 2010, pp. 302–311.
[38] P. Llopis, J. Blas, F. Isaila, and J. Carretero, "Vidas: object-based virtualized data sharing for high performance storage i/o," in *Proceedings of the 4th ACM workshop on Scientific cloud computing*. ACM, 2013, pp. 37–44.
[39] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective i/o in romio," in *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*. IEEE, 1999, pp. 182–189.
[40] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, Jun. 2008. (Online). Available: http://doi.acm.org/10.1145/1365815.1365816
[41] J. Paiva, P. Ruivo, P. Romano, and L. Rodrigues, "Autoplacer: Scalable self-tuning data placement in distributed key-value stores," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX, 2013, pp. 119–131. (Online). Available: https://www.usenix.org/conference/icac13/technical-sessions/presentation/paiva
[42] M. Vora, "Hadoop-hbase for large-scale data," in *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, vol. 1, Dec 2011, pp. 601–605.
[43] M. Shapiro and E. Miller, "Managing databases with binary large objects," in *Mass Storage Systems, 1999. 16th IEEE Symposium on*, 1999, pp. 185–193.

Silvina Caíno-Lores obtained her M.Sc. in Computer Science and Technology from the Carlos III University of Madrid (Spain) in 2015, and she is currently working towards her Ph.D. under the direction of Prof. Jesús Carretero Prez. Her B.Sc. thesis was awarded by the IT Service Management Forum (itSMF). Her research interests include, but are not limited to, cloud computing, high-performance and high-throughput computing, parallel and distributed systems, in-memory computing and storage, and scientific computing and simulation. She is currently a teaching and research assistant in the Computer Science Department at the Carlos III University of Madrid (Spain), a member of the network for sustainable ultrascale computing (NESUS ICT COST Action IC1305).

Jesús Carretero is a Full Professor of Computer Architecture and Technology at Universidad Carlos III de Madrid (Spain), where he is responsible for that knowledge area since 2000. His research activity is centered on high-performance computing systems, large-scale distributed systems and real-time systems. He is Action Chair of the IC1305 COST Action "Network for Sustainable Ultrascale Computing Systems (NESUS)", and he is also currently involved in the FP7 program REPARA "Reengineering and Enabling Performance And poweR of Applications". Prof. Carretero is Associated Editor of the journal Computer and Electrical Engineering and International Journal of Distributed Sensor Networks. He has published more than 180 papers in journals and international conferences, editor of several books of proceedings, and guest editor for special issues of journals as International Journal of Parallel Processing, Cluster Computing, Computers and Electrical Engineering, and New Generation Computing, and he is coauthor of several text books related to Operationg Systems and Computer Architecture. He has participated in many conference organization committees, and he has been General chair of HPCC 2011 and MUE 2012, and Program Chair of ISPA 2012, EuroMPI 2013, C4Bio 2014, and ESAA 2014. Prof. Carretero is a senior member of the IEEE Computer Society and member of the ACM. He was a visiting scholar at the NorthWestern University of Chicago (Ill, USA). He works currently at Universidad Carlos III de Madrid, where he has been teaching since 2000.