Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

A view of programming scalable data analysis: from clouds to exascale

Domenico Talia

Abstract

Scalability is a key feature for big data analysis and machine learning frameworks and for applications that need to analyze very large and real-time data available from data repositories, social media, sensor networks, smartphones, and the Web. Scalable big data analysis today can be achieved by parallel implementations that are able to exploit the computing and storage facilities of high performance computing (HPC) systems and clouds, whereas in the near future Exascale systems will be used to implement extreme-scale data analysis. Here is discussed how clouds currently support the development of scalable data mining solutions and are outlined and examined the main challenges to be addressed and solved for implementing innovative data analysis applications on Exascale systems.

Keywords: Big data analysis, Cloud computing, Exascale computing, Data mining, Parallel programming, Scalability

Introduction

Solving problems in science and engineering was the first motivation for inventing computers. After a long time since then, computer science is still the main area in which innovative solutions and technologies are being developed and applied. Also due to the extraordinary advancement of computer technology, nowadays data are generated as never before. In fact, the amount of structured and unstructured digital datasets is going to increase beyond any estimate. Databases, file systems, data streams, social media and data repositories are increasingly pervasive and decentralized.

As the data scale increases, we must address new challenges and attack ever-larger problems. New discoveries will be achieved and more accurate investigations can be carried out due to the increasingly widespread availability of large amounts of data. Scientific sectors that fail to make full use of the huge amounts of digital data available today risk losing out on the significant opportunities that big data can offer.

To benefit from the big data availability, specialists and researchers need advanced data analysis tools and applications running on scalable architectures allowing for the extraction of useful knowledge from such huge data sources. High performance computing (HPC) systems and cloud computing systems today are capable platforms for addressing both the computational and data storage needs of big data mining and parallel knowledge discovery applications. These computing architectures are needed to run data analysis because complex data mining tasks involve data- and compute-intensive algorithms that require large, reliable and effective storage facilities together with high performance processors to get results in appropriate times.

Now that data sources became very big and pervasive, reliable and effective programming tools and applications for data analysis are needed to extract value and find useful insights in them. New ways to correctly and proficiently compose different distributed models and paradigms are required and interaction between hardware resources and programming levels must be addressed. Users, professionals and scientists working in the area of big data need advanced data analysis programming models and tools coupled with scalable architectures to support the extraction of useful information from such massive repositories. The scalability of a parallel computing system is a measure of its capacity to reduce program execution time in proportion to the number of its processing elements (The Appendix introduces and discusses in detail scalability in parallel systems). According to scalability definition, scalable data analysis refers to the ability of a hardware/software parallel system to exploit increasing computing resources effectively in the analysis of (very) large datasets.

Correspondence: talia@dimes.unical.it
DIMES, Università della Calabria, Rende, Italy

Today complex analysis of real-world massive data sources requires using high-performance computing systems such as massively parallel machines or clouds. However in the next years, as parallel technologies advance, Exascale computing systems will be exploited for implementing scalable big data analysis in all the areas of science and engineering [23]. To reach this goal, new design and programming challenges must be addressed and solved. The focus of the paper is on discussing current cloud-based designing and programming solutions for data analysis and suggesting new programming requirements and approaches to be conceived for meeting big data analysis challenges on future Exascale platforms.

Current cloud computing platforms and parallel computing systems represent two different technological solutions for addressing the computational and data storage needs of big data mining and parallel knowledge discovery applications. Indeed, parallel machines offer high-end processors with the main goal to support HPC applications, whereas cloud systems implement a computing model in which virtualized resources dynamically scalable are provided to users and developers as a service over the Internet. In fact, clouds do not mainly target HPC applications; they instrument scalable computing and storage delivery platforms that can be adapted to the needs of different classes of people and organizations by exploiting the Service Oriented (SOA) approach. Clouds offer large facilities to many users that were unable to own their parallel/distributed computing systems to run applications and services. In particular, big data analysis applications requiring access and manipulating very large datasets with complex mining algorithms will significantly benefit from the use of cloud platforms.

Although not many cloud-based data analysis frameworks are available today for end users, within a few years they will become common [29]. Some current solutions are based on open source systems, such as Apache Hadoop and Mahout, Spark and SciDB, while others are proprietary solutions provided by companies such as Google, Microsoft, EMC, Amazon, BigML, Splunk Hunk, and InsightsOne. As more such platforms emerge, researchers and professionals will port increasingly powerful data mining programming tools and frameworks to the cloud to exploit complex and flexible software models such as the distributed workflow paradigm. The growing utilization of the service-oriented computing model could accelerate this trend.

From the definition of the big data term, which refers to datasets so large and complex that traditional hardware and software data processing solutions are inadequate to manage and analyze, we can infer that conventional computer systems are not so powerful to process and mine big data [28] and they are not able to scale with the size of problems to be solved. As mentioned before, to face with limits of sequential machines, advanced systems like HPC, clouds and even more scalable architectures are used today to analyze big data. Starting from this scenario, Exascale computing systems will represent the next computing step [1, 34]. Exascale systems refers to high performance computing systems capable of at least one exaFLOPS, so their implementation represents a very significant research and technology challenge. Their design and development is currently under investigation with the goal of building by 2020 high-performance computers composed of a very large number of multi-core processors expected to deliver a performance of 10^18 operations per second. Cloud computing systems used today are able to store very large amounts of data, however they do not provide the high performance expected from massively parallel Exascale systems. This is the main motivation for developing Exascale systems. Exascale technology will represent the most advanced model of supercomputers. They have been conceived for single-site supercomputing centers not for distributed infrastructures as multi-clouds or fog computing systems that are aimed to decentralized computing and pervasive data management that could be interconnected with Exascale systems that could used as backbone for very large scale data analysis.

The development of Exascale systems urges to address and solve issues and challenges both at hardware and software level. Indeed it requires to design and implement novel software tools and runtime systems able to manage a very high degree of parallelism, reliability and data locality in extreme scale computers [14]. New programming constructs and runtime mechanisms able to adapt to the most appropriate parallelism degree and communication decomposition for making scalable and reliable data analysis tasks are needed. Their dependence from parallelism grain size and data analysis task decomposition must be deeply studied. This is needed because parallelism exploitation depends on several features like parallel operations, communication overhead, input data size, I/O speed, problem size, and hardware configuration. Moreover, reliability and reproducibility are two additional key challenges to be addressed. Indeed at programming level, constructs for handling and recovering communication, data access, and computing failures must be designed. At the same time, reproducibility in scalable data analysis asks for rich information useful to assure similar results on environments that dynamically may change. All these factors must be taken into account in designing data analysis applications and tools that will be scalable on exascale systems.

Moreover, reliable and effective methods for storing, accessing and communicating data, intelligent techniques for massive data analysis and software architectures enabling the scalable extraction of knowledge from data, are needed [28]. To reach this goal, models and technologies enabling cloud computing systems and HPC architectures must be extended/adapted or completely changed to be reliable and scalable on the very large number of processors/cores that compose extreme scale platforms and for supporting the implementation of clever data analysis algorithms that ought to be scalable and dynamic in resource usage. Exascale computing infrastructures will play the role of an extraordinary platform for addressing both the computational and data storage needs of big data analysis applications. However, as mentioned before, to have a complete scenario, efforts must be performed for implementing big data analytics algorithms, architectures, programming tools and applications in Exascale systems [24].

Pursuing this objective within a few years, scalable data access and analysis systems will become the most used platforms for big data analytics on large-scale clouds. In a longer perspective, new Exascale computing infrastructures will appear as the platforms for big data analytics in the next decades, and data mining algorithms, tools and applications will be ported on such platforms for implementing extreme data discovery solutions.

In this paper we first discuss cloud-based scalable data mining and machine learning solutions, then we examine the main research issues that must be addressed for implementing massively parallel data mining applications on Exascale computing systems. Data-related issues are discussed together with communication, multi-processing, and programming issues. Section II introduces issues and systems for scalable data analysis on clouds and Section III discusses design and programming issues for big data analysis in Exascale systems. Section IV completes the paper also outlining some open design challenges.

Data analysis on clouds

Clouds implement elastic services, scalable performance and scalable data storage used by a large and everyday increasing number of users and applications [2, 12]. In fact, clouds enlarged the arena of distributed computing systems by providing advanced Internet services that complement and complete functionalities of distributed computing provided by the Web, Grid systems and peer-to-peer networks. In particular, most cloud computing applications use big data repositories stored within the cloud itself, so in those cases large datasets are analyzed with low latency to effectively extract data analysis models.

Big data is a new and over-used term that refers to massive, heterogeneous, and often unstructured digital content that is difficult to process using traditional data management tools and techniques. The term includes the complexity and variety of data and data types, real-time data collection and processing needs, and the value that can be obtained by smart analytics. However we should recognize that data are not necessarily important per se but they become very important if we are able to extract value from them; that is if we can exploit them to make discoveries. The extraction of useful knowledge from big digital datasets requires smart and scalable analytics algorithms, services, programming tools, and applications. All these solutions need to find insights in big data will contribute to make them really useful for people.

The growing use of service-oriented computing is accelerating the use of cloud-based systems for scalable big data analysis. Developers and researchers are adopting the three main cloud models, software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS), to implement big data analytics solutions in the cloud [27, 31]. According to a specialization of these three models, data analysis tasks and applications can be offered as services at infrastructure, platform or software level and made available every time form everywhere. A methodology for implementing them defines a new model stack to delivery data analysis solutions that is a specialization of the XaaS (Everything as a Service) stack and is called *Data Analysis as a Service* (*DAaaS*). It adapts and specifies the three general service models (SaaS, PaaS and IaaS), for supporting the structured development of Big Data analysis systems, tools and applications according to a service-oriented approach. The DAaaS methodology is then based on the three basic models for delivering data analysis services at different levels as described here (see also Fig. 1):

- *Data analysis infrastructure as a service (DAIaaS).* This model provides a set of hardware/software virtualized resources that developers can assemble and use as a an integrated infrastructure where storing large datasets, running data mining applications and/or implementing data analytics systems from scratch;
- *Data analysis platform as a service (DAPaaS).* This model defines a supporting software platform that developers can use for programming and running their data analytics applications or extending existing ones without concerning about the underlying infrastructure or specific distributed architecture issues; and
- *Data analysis software as a service (DASaaS).* This is a higher-level model that offers to end users data mining algorithms, data analysis suites or ready-to-use knowledge discovery applications as Internet services that can be accessed and used directly through

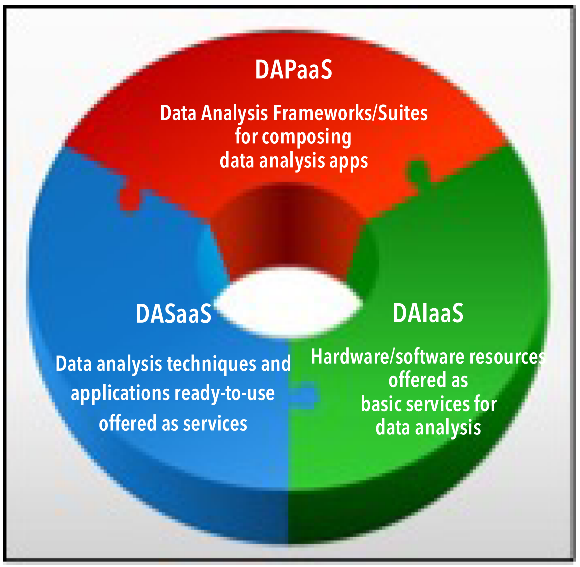

DAPaaS
Data Analysis Frameworks/Suites for composing data analysis apps
DASaaS
Data analysis techniques and applications ready-to-use offered as services
DAIaaS
Hardware/software resources offered as basic services for data analysis

Fig. 1 The three models of the DAaaS software methodology. The DAaaS software methodology is based on three basic models for delivering data analysis services at different levels (application, platform, and infrastructure). The DAaaS methodology defines a new model stack to delivery data analysis solutions that is a specialization of the XaaS (Everything as a Service) stack and is called Data Analysis as a Service (DAaaS). It adapts and specifies the three general service models (SaaS, PaaS and SaaS), for supporting the structured development of Big Data analysis systems, tools and applications according to a service-oriented approach

a Web browser. According to this approach, every data analysis software is provided as a service, avoiding end users to worry about implementation and execution details.

Cloud-based data analysis tools

Using the DASaaS methodology we designed a cloud-based system, the Data Mining Cloud Framework (DMCF) [17], which supports three main classes of data analysis and knowledge discovery applications:

- *Single-task applications*, in which a single data mining task such as classification, clustering, or association rules discovery is performed on a given dataset;
- *Parameter-sweeping applications*, in which a dataset is analyzed by multiple instances of the same data mining algorithm with different parameters; and
- *Workflow-based applications*, in which knowledge discovery applications are specified as graphs linking together data sources, data mining tools, and data mining models.

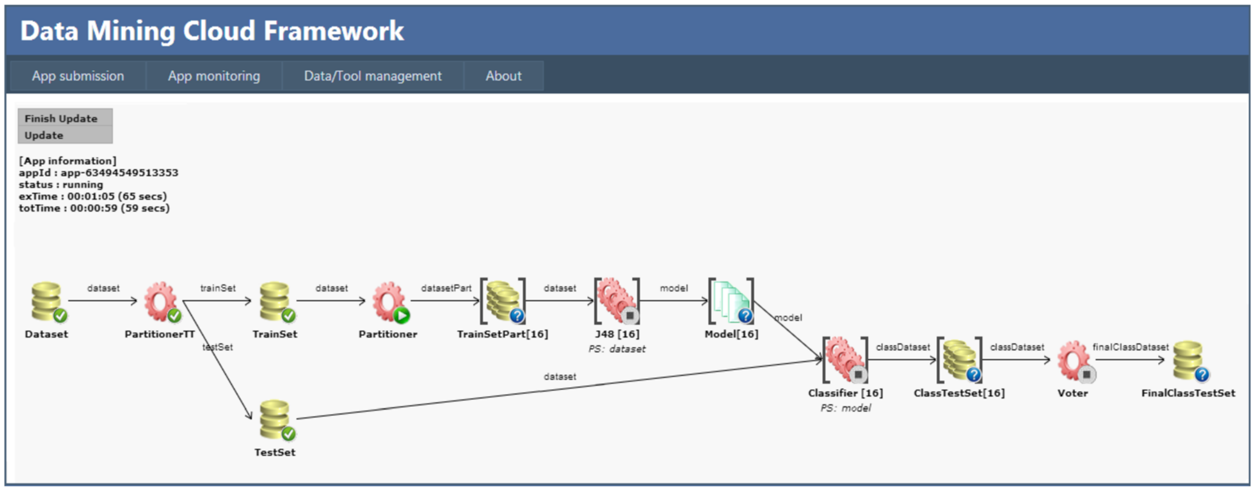
DMCF includes a large variety of processing patterns to express knowledge discovery workflows as graphs whose nodes denote resources (datasets, data analysis tools, mining models) and whose edges denote dependencies among resources. A Web-based user interface allows users to compose their applications and submit them for execution to the Cloud platform, following the data analysis software as a service approach. Visual workflows can be programmed in DMCF through a language called VL4Cloud (Visual Language for Cloud), whereas script-based workflows can be programmed by JS4Cloud (JavaScript for Cloud), a JavaScript-based language for data analysis programming.

Figure 2 shows a sample data mining workflow composed of several sequential and parallel steps. It is just an example for presenting the main features of the VL4Cloud programming interface [17]. The example workflow analyses a dataset by using n instances of a classification algorithm, which work on n portions of the training set and generate the same number of knowledge models. By using the n generated models and the test set, n classifiers produce in parallel n classified datasets (n classifications). In the final step of the workflow, a voter generates the final classification by assigning a class to each data item, by choosing the class predicted by the majority of the models.

Although DMCF has been mainly designed to coordinate coarse grain data and task parallelism in big data analysis applications by exploiting the workflow paradigm, the DMCF script-based programming interface (JS4Cloud) allows also for parallelizing fine-grain operations in data mining algorithms as it permits to program in a JavaScript style any data mining algorithm, such as classification, clustering and others. This can be done because loops and data parallel methods are run in parallel on the virtual machines of a Cloud [16, 26].

Like DMCF, other innovative cloud-based systems designed for programming data analysis applications are: Apache Spark, Sphere, Swift, Mahout, and CloudFlows. Most of them are open source. Apache Spark is an open-source framework developed at UC Berkeley for in-memory data analysis and machine learning [34]. Spark has been designed to run both batch processing and dynamic applications like streaming, interactive queries, and graph analysis. Spark provides developers with a programming interface centered on a data structure called the resilient distributed dataset (RDD), that is a read-only multi-set of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. Differently from other systems and from Hadoop, Spark stores data in memory and queries it repeatedly so as to obtain better performance. This feature can be useful for a future implementation of Spark on Exascale systems.

Swift is a workflow-based framework for implementing functional data-driven task parallelism in data-intensive applications. The Swift language provides a functional



[App information]
appId : app-63494549513353
status : running
exTime : 00:01:05 (65 secs)
totTime : 00:00:59 (59 secs)

Fig. 2 A parallel classification workflow designed by the VL4Cloud programming interface. The figure shows a workflow designed by the VL4Cloud programming interface during its execution. The workflow implements a parallel classification application. Tasks/services included in square bracket are executed in parallel. The results produced by classifiers are selected by a voter task that produces the final classification

programming paradigm where workflows are designed as a set of calls with associated command-line arguments and input and output files. Swift uses an implicit data-driven task parallelism [32]. In fact, it looks like a sequential language, but being a dataflow language, all variables are futures, thus execution is based on data availability. Parallelism can be also exploited through the use of the foreach statement. Swift/T is a new implementation of the Swift language for high-performance computing. In this implementation, a Swift program is translated into an MPI program that uses the Turbine and ADLB runtime libraries for scalable dataflow processing over MPI. Recently a porting of Swift/T on very large cloud systems for the execution of very many tasks has been investigated.

DMCF, differently from the other frameworks discussed here, it is the only system that offers both a visual and a script-based programming interface. Visual programming is a very convenient design approach for high-level users, like domain-expert analysts having a limited understanding of programming. On the other hand, script-based workflows are a useful paradigm for expert programmers who can code complex applications rapidly, in a more concise way and with greater flexibility. Finally, the workflow-based model exploited in DMCF and Swift make these frameworks of more general use with respect to Spark that offers a very restricted set of programming patterns (e.g., map, filter and reduce) so limiting the variety of data analysis applications that can be implemented with it.

These and other related systems are currently used for the development of big data analysis applications on HPC and cloud platforms. However, additional research work in this field must be done and the development of new models, solutions and tools is needed [13, 24]. Just to mention a few, active and promising research topics are listed here ordered by importance factors:

- *Programming models for big data analytics*. New abstract programming models and constructs hiding the system complexity are needed for big data analytics tools. The MapReduce model and workflow models are often used on HPC and clouds, but more research effort is needed to develop other scalable, adaptive, general-purpose higher-level models and tools. Research in this area is even more important for Exascale systems; in the next section we will discuss some of these topics in Exascale computing.
- *Reliability in scalable data analysis*. As the number of processing elements increases, reliability of systems and applications decreases, therefore mechanisms for detecting and handling hardware and software faults are needed. Although in [7] has been proved that no reliable communication protocol can tolerate crashes of processors on which the protocol runs, as stated in the same paper some ways in which systems cope with the impossibility result can be found. Among them, at programming level it is necessary to design constructs for handling communication, data access, and computing failures and for recovering from them. Programming models, languages and APIs must provide general and data-oriented mechanisms for failure detection and isolation, avoiding that an entire application can

fail and assuring its completion. Reliability is an issue much more important in the Exascale domain where the number of processing elements is massive and fault occurrence increases making detection and recovering vital.

- *Application reproducibility*. Reproducibility is another open research issue for designers of complex applications running on parallel systems. Reproducibility in scalable data analysis must face, for example, with data communication, data parallel manipulation and dynamic computing environments. Reproducibility demands that current data analysis frameworks (like those based on MapReduce and on workflows) and the future ones, especially those implemented on Exascale systems, must provide additional information and knowledge on how data are managed, on algorithm characteristics and on configuration of software and execution environments.
- *Data and tool integration and openness*. Code coordination and data integration are main issues in large-scale applications that use data and computing resources. Standard formats, data exchange models and common APIs are needed to support interoperability and ease cooperation among design teams using different data formats and tools.
- *Interoperability of big data analytics frameworks*. The service-oriented paradigm allows running large-scale distributed applications on cloud heterogeneous platforms along with software components developed using different programming languages or tools. Cloud service paradigms must be designed to allow worldwide integration of multiple data analytics frameworks.

Exascale and big data analysis

As we discussed in the previous sections, data analysis gained a primary role because of the very large availability of datasets and the continuous advancement of methods and algorithms for finding knowledge in them. Data analysis solutions advance by exploiting the power of data mining and machine learning techniques and are changing several scientific and industrial areas. For example, the amount of data that social media daily generate is impressive and continuous. Some hundreds of terabyte of data, including several hundreds of millions of photos, are uploaded daily to Facebook and Twitter.

Therefore it is central to design scalable solutions for processing and analysis such massive datasets. As a general forecast, IDC experts estimate data generated to reach about 45 zettabytes worldwide by 2020 [6]. This impressive amount of digital data asks for scalable high performance data analysis solutions. However, today only one-quarter of digital data available would be a candidate for analysis and about 5% of that is actually analyzed. By 2020, the useful percentage could grow to about 35% also thanks to data mining technologies.

Extreme data sources and scientific computing

Scalability and performance requirements are challenging conventional data storages, file systems and database management systems. Architectures of such systems have reached limits in handling very large processing tasks involving petabytes of data because they have not been built for scaling after a given threshold. This condition claims for new architectures and analytics platform solutions that must process big data for extracting complex predictive and descriptive models [30]. Exascale systems, both from the hardware and the software side, can play a key role to support solutions for these problems [23].

An IBM study reports that we are generating around 2.5 exabytes of data per day.[1] Because of that continuous and explosive growth of data, many applications require the use of scalable data analysis platforms. A well-known example is the ATLAS detector from the Large Hadron Collider at CERN in Geneva. The ATLAS infrastructure has a capacity of 200 PB of disk and 300,000 cores, with more than 100 computing centers connected via 10 Gbps links. The data collection rate is very high and only a portion of the data produced by the collider is stored. Several teams of scientists run complex applications to analyze subsets of those huge volumes of data. This analysis would be impossible without a high-performance infrastructure that supports data storage, communication and processing. Also computational astronomers are collecting and producing larger and larger datasets each year that without scalable infrastructures cannot be stored and processed. Another significant case is represented by the Energy Sciences Network (ESnet) is the USA Department of Energy's high-performance network managed by Berkeley Lab that in late 2012 rolled out a 100 gigabits-per-second national network to accommodate the growing scale of scientific data.

If we go from science to society, social data and e-health are good examples to discuss. Social networks, such as Facebook and Twitter, have become very popular and are receiving increasing attention from the research community since, through the huge amount of user-generated data, they provide valuable information concerning human behavior, habits, and travels. When the volume of data to be analyzed is of the order of terabytes or petabytes (billions of tweets or posts), scalable storage and computing solutions must be used, but no clear solutions today exist for the analysis of Exascale datasets. The same occurs in the e-health domain, where huge amounts of patient data are available and can be used for improving therapies, for forecasting and tracking of health data, for the management of hospitals and health centers. Very complex data analysis in this area will need novel hardware/software solutions, however Exascale computing is still promising in other scientific fields where scalable storages and databases are not used/required. Examples of scientific disciplines where future Exascale computing will be extensively used are quantum chromodynamics, materials simulation, molecular dynamics, materials design, earthquake simulations, subsurface geophysics, climate forecasting, nuclear energy, and combustion. All those applications require the use of sophisticated models and algorithms to solve complex equation systems that will benefit from the exploitation of Exascale systems.

Programming models features for exascale data analysis

Implementing scalable data analysis applications in Exascale computing systems is a very complex job and it requires high-level fine-grain parallel models, appropriate programming constructs and skills in parallel and distributed programming. In particular, mechanisms and expertise are needed for expressing task dependencies and inter-task parallelism, for designing synchronization and load balancing mechanisms, handling failures, and properly manage distributed memory and concurrent communication among a very large number of tasks. Moreover, when the target computing infrastructures are heterogeneous and require different libraries and tools to program applications on them, the programming issues are even more complex. To cope with some of these issues in data-intensive applications, different scalable programming models have been proposed [5].

Scalable programming models may be categorized by

i. Their level of abstraction: expressing high-level and low-level programming mechanisms, and
ii. How they allow programmers to develop applications: using visual or script-based formalisms.

Using high-level scalable models, a programmer defines only the high-level logic of an application while hides the low-level details that are not essential for application design, including infrastructure-dependent execution details. A programmer is assisted in application definition and application performance depends on the compiler that analyzes the application code and optimizes its execution on the underlying infrastructure. On the other hand, low-level scalable models allow programmers to interact directly with computing and storage elements composing the underlying infrastructure and thus define the applications parallelism directly.

Data analysis applications implemented by some frameworks can be programmed through a visual interface, which is a convenient design approach for high-level users, for instance domain-expert analysts having a limited understanding of programming. In addition, a visual representation of workflows or components intrinsically captures parallelism at the task level, without the need to make parallelism explicit through control structures [14]. Visual-based data analysis typically is implemented by providing workflows-based languages or component-based paradigms (Fig. 3). Also dataflow-based approaches, that share with workflows the same application structure, are used. However, in dataflow models, the grain of parallelism and the size of data items are generally smaller with respect to workflows. In general, visual programming tools are not very flexible because they often implement a limited set of visual patterns and provide restricted manners to configure them. For addressing this issue, some visual

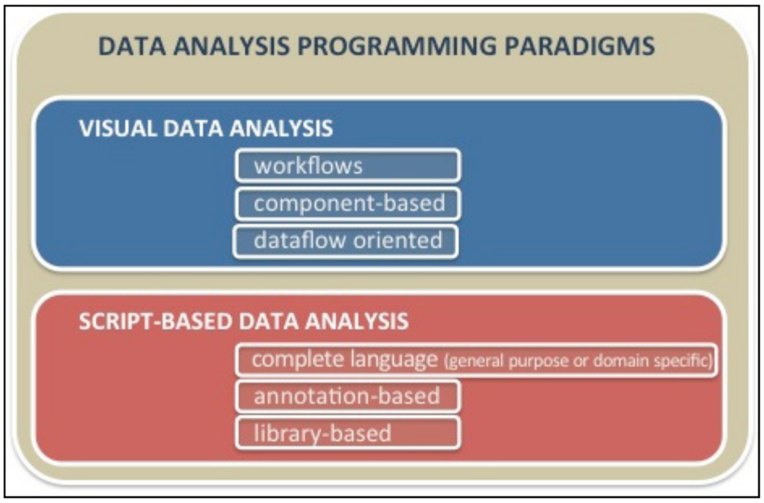

DATA ANALYSIS PROGRAMMING PARADIGMS
VISUAL DATA ANALYSIS
workflows
component-based
dataflow oriented
SCRIPT-BASED DATA ANALYSIS
complete language (general purpose or domain specific)
annotation-based
library-based

Fig. 3 Main visual and script-based programming models used today for data analysis programming

languages provide users with the possibility to customize the behavior of patterns by adding code that can specify operations executed a specific pattern when an event occurs.

On the other hand, code-based (or script-based) formalism allows users to program complex applications more rapidly, in a more concise way, and with higher flexibility [16]. Script-based applications can be designed in different ways (see Fig. 3):

- With a complete language or a language extension that allows to express parallelism in applications, according to a general purpose or a domain specific approach. This approach requires the design and implementation of a new parallel programming language or a complete set of data types and parallel constructs to be fully inserted in an existing language.
- With annotations in the application code that allow the compiler to identify which instructions will be executed in parallel. According to this approach, parallel statements are separated from sequential constructs and they are clearly identified in the program code because they are denoted by special symbols or keywords.
- Using a library in the application code that adds parallelism to the data analysis application. Currently this is the most used approach since it is orthogonal to host languages. MPI and MapReduce are two well-known examples of this approach.

Given the variety of data analysis applications and classes of users (from skilled programmers to end users) that can be envisioned for future Exascale systems, there is a need for scalable programming models with different levels of abstractions (high-level and low-level) and different design formalisms (visual and script-based), according to the classification outlined above.

As we discussed, data-intensive applications are software programs that have a significant need to process large volumes of data [9]. Such applications devote most of their processing time to run I/O operations and exchanging and moving data among the processing elements of a parallel computing infrastructure. Parallel processing in data analysis applications typically involves accessing, pre-processing, partitioning, distributing, aggregating, querying, mining, and visualizing data that can be processed independently.

The main challenges for programming data analysis applications on Exascale computing systems come from potential scalability, network latency and reliability, reproducibility of data analysis, and resilience of mechanisms and operations offered to developers for accessing, exchanging and managing data. Indeed, processing very large data volumes requires operations and new algorithms able to scale in loading, storing, and processing massive amounts of data that generally must be partitioned in very small data grains, on which thousands to millions of simple parallel operations do analysis.

Exascale programming systems

Exascale systems force new requirements on programming systems to target platforms with hundreds of homogeneous and heterogeneous cores. Evolutionary models have been recently proposed for Exascale programming that extend or adapt traditional parallel programming models like MPI (e.g., EPiGRAM [15] that uses a library-based approach, Open MPI for Exascale in the ECP initiative), OpenMP (e.g., OmpSs [8] that exploits an annotation-based approach, the SOLLVE project), and MapReduce (e.g., Pig Latin [22] that implements a domain-specific complete language). These new frameworks limit the communication overhead in message passing paradigms or limit the synchronization control if a shared-memory model is used [11].

As Exascale systems are likely to be based on large distributed memory hardware, MPI is one of the most natural programming systems. MPI is currently used on over about one million cores, therefore is reasonable to have MPI as one programming paradigm used on Exascale systems. The same possibility occurs for MapReduce-based libraries that today are run on very large HPC and cloud systems. Both these paradigms are largely used for implementing Big Data analysis applications. As expected, general MPI all-to-all communication does not scale well in Exascale environments, thus to solve this issue new MPI releases introduced neighbor collectives to support sparse "all-to-some" communication patterns that limit the data exchange on limited regions of processors [11].

Ensuring the reliability of Exascale systems requires a holistic approach including several hardware and software technologies for both predicting crashes and keeping systems stable despite failures. In the runtime of parallel APIs, like MPI and MapReduce-based libraries like Hadoop, if do not want to behave incorrectly in case of processor failure, a reliable communication layer must be provided using the lower unreliable layer by implementing a correct protocol that work safely with every implementation of the unreliable layer that cannot tolerate crashes of the processors on which it runs. Concerning MapReduce frameworks, reference [18] reports on an adaptive MapReduce framework, called P2P-MapReduce, which has been developed to manage node churn, master node failures, and job recovery in a decentralized way, so as to provide a more reliable MapReduce middleware that can be effectively exploited in dynamic large-scale infrastructures.

On the other hand, new complete languages such as X10 [29], ECL [33], UPC [21], Legion [3], and Chapel [4] have been defined by exploiting in them a data-centric approach. Furthermore, new APIs based on a revolutionary approach, such as GA [20] and SHMEM [19], have been implemented according to a library-based model. These novel parallel paradigms are devised to address the requirements of data processing using massive parallelism. In particular, languages such as X10, UPC, and Chapel and the GA library are based on a partitioned global address space (PGAS) memory model that is suited to implement data-intensive Exascale applications because it uses private data structures and limits the amount of shared data among parallel threads.

Together with different approaches, such as Pig Latin and ECL, those programming models, languages and APIs must be further investigated, designed and adapted for providing data-centric scalable programming models useful to support the reliable and effective implementation of Exascale data analysis applications composed of up to millions of computing units that process small data elements and exchange them with a very limited set of processing elements. PGAS-based models, data-flow and data-driven paradigms, local-data approaches today represent promising solutions that could be used for Exascale data analysis programming. The APGAS model is, for example, implemented in the X10 language where it is based on the notions of places and asynchrony. A place is an abstraction of shared, mutable data and worker threads operating on the data. A single APGAS computation can consist of hundreds or potentially tens of thousands of places. Asynchrony is implemented by a single block-structured control construct async. Given a statement ST, the construct async ST executes ST in a separate thread of control. Memory locations in one place can contain references to locations at other places. To compute upon data at another place, the

```
at(p)ST
```

statement must be used. It allows the task to change its place of execution to p, executes ST at p and returns, leaving behind tasks that may have been spawned during the execution of ST.

Another interesting language based on the PGAS model is Chapel [4]. Its locality mechanisms can be effectively used for scalable data analysis where light data mining (sub-)tasks are run on local processing elements and partial results must be exchanged. Chapel data locality provides control over where data values are stored and where tasks execute so that developers can ensure parallel data analysis computations execute near the variables they access, or vice-versa for minimizing the communication and synchronization costs. For example, Chapel programmers can specify how domains and arrays are distributed amongst the system nodes. Another appealing feature in Chapel is the expression of synchronization in a data-centric style. By associating synchronization constructs with data (variables), locality is enforced and data-driven parallelism can be easily expressed also at very large scale. In Chapel, *locales* and *domains* are abstractions for referring to machine resources and map tasks and data to them. *Locales* are language abstractions for naming a portion of a target architecture (e.g., a GPU, a single core or a multicore node) that has processing and storage capabilities. A *locale* specifies where (on which processing node) to execute tasks/statements/operations. For example, in a system composed of 4 *locales*

```
const Locs : [4]locale;
```

for executing the method *Filter(D)* on the first *locale*, we can use

```
on Locs[0] do Filter(D);
```

and to execute the K-means algorithm on the 4 *locales* we can use

```
forall lc in Locs(i) do on lc do Kmeans();
```

Whereas locales are used to map tasks to machine nodes, domain maps are used for mapping data to a target architecture. Here is a simple example of a declaration of a rectangular domain

```
const D: domain(2) = {1..n, 1..n};
```

Domains can be also mapped to locales. Similar concepts (logical regions & mapping interfaces) are used in the Legion programming model [3, 5].

Exascale programming is a strongly evolving research field and it is not possible to discuss in details all programming models, languages and libraries that are contributing to provide features and mechanisms useful for exascale data analysis application programming. However, the next section introduces, discusses and classifies current programming systems for Exascale computing according to the most used programming and data management models.

Exascale programming systems comparison

As mentioned, several parallel programming models, languages and libraries are under development for providing high-level programming interfaces and tools for implementing high-performance applications on future Exascale computers. Here we introduce the most significant proposals and discuss their main features. Table 1 lists and classifies the considered systems and in it some pros and fallacies of different classes are summarized.

Since Exascale systems will be composed of millions of processing nodes, distributed memory paradigms, and message passing systems in particular, are candidate tools to be used as programming systems for such class of systems. In this area, MPI is currently the most used and studied system. Different adaptations of this well-known model are under development such as, for example, Open MPI for Exascale. Other systems based on distributed memory programming are Pig Latin, Charm++, Legion, PaRSEC, Bulk Synchronous Parallel (BSP), AllScale API, and Enterprise Control Language (ECL). Just considering Pig Latin, we can notice that some of its parallel operators such as FILTER, which selects a set of tuples from a relation based on a condition, and SPLIT, which partitions a relation into two or more relations, can be very useful in many highly parallel big data analysis applications.

On the other side, we have shared-memory models where the major system is OpenMP that offers a simple parallel programming model although it does not provide mechanisms to explicitly map and control data distribution and includes non-scalable synchronization operations that are making very challenging its implementation on massively parallel systems. Other programming systems in this area are Threading Building Blocks (TBB), OmpSs, and Cilk++. The OpenMP synchronization model based on locks, atomic and sequential sections that limit parallelism exploitation in Exascale systems are going to be modified and integrated in recent OpenMP implementations with new techniques and routines that increase asynchronous operations and parallelism exploitation. A similar approach is used in Cilk++ that supports parallel loops and hyperobjects, a new construct designed to solve data race problems created by parallel accesses to global variables. In fact, a hyperobject allows multiple tasks to share state without race conditions and without using explicit locks.

As a tradeoff between distributed and shared memory organizations, the Partitioned Global Address Space (PGAS) model has been designed for implementing a global memory address space that is logically partitioned and portions of it are local to single processes. The main goal of the PGAS model is to limit data exchange and isolate failures in very large-scale systems. Languages and libraries based on PGAS are Unified Parallel C (UPC), Chapel, X10, Global Arrays (GA), Co-Array

Table 1 Exascale programming systems classification

Programming Models	Languages	Libraries/APIs	Pros and Fallacies
Distributed memory	Charm++, Legion, High Performance Fortran (HPF), ECL, PaRSEC	MPI, BSP, Pig Latin, AllScale,	Distributed memory languages/APIs are very close to the Exascale hardware model. Systems in this class consider and deal with communication latency however data exchange costs are the main source of overhead. Except AllScale, and some MPI version, systems in this class do not manage network and CPU failures.
Shared memory	TBB, Cilk++	OpenMP, OmpSs	Shared memory models do not map efficiently on Exascale systems, extensions have been proposed to perform better dealing with synchronization and network failures. No single convincing solution till now exists.
Partitioned memory	UPC, Chapel, X10, CAF	GA, SHMEM, DASH, OpenSHMEM, GASPI	The local memory model is very useful but combination with global/shared memory mechanisms introduce too much overhead. GASPI is the only system in this class enabling applications to recover from failures.
Hybrid models	UPC + MPI, C++/MPI,	MPI + OpenMP, Spark-MPI, FLUX, EMPI4Re, DPLASMA,	Hybrid models facilitate the mapping to the hardware architectures, however the different programming routines compete for resources making hard to control concurrency and contention. Resilient mechanisms are harder to implement because of the mixing of different constructs and data models.

Fortran (CAF), DASH, and SHMEM. PGAS appears to be suited for implementing data-intensive exascale applications because it uses private data structures and limits the amount of shared data among parallel threads. Its memory-partitioning model facilitates failure detection and resilience. Another programming mechanism useful for decentralized data analysis is related to data synchronization. In the SHMEM library it is implemented through the shmem_barrier operation that performs a barrier operation on a subset of processing elements, then enables them to go further by sharing synchronized data.

Starting from those three main programming approaches, hybrid systems have been proposed and developed to better map application tasks and data onto hardware architectures of Exascale systems. In hybrid systems that combine distributed and shared memory, message-passing routines are used for data communication and inter-node processing whereas shared-memory operations are used for exploiting intranode parallelism. A major example in this area is given by the different MPI + OpenMP systems recently implemented. Hybrid systems have been also designed by combining message passing models, like MPI, with PGAS models for restricting data communication overhead and improving MPI efficiency in execution time and memory consumption. The PGAS-based MPI implementation EMPI4Re, developed in the EPiGRAM project, is an example of this class of hybrid systems.

Associated to the programming model issues, a set of challenges concern the design of runtime systems that in exascale computing systems must be tightly integrated with the programming tools level. The main challenges for runtime systems obviously include parallelism exploitation, limited data communication, data dependence management, data-aware task scheduling, processor heterogeneity, and energy efficiency. However, together with those main issues, other aspects are addressed in runtime systems like storage/memory hierarchies, storage and processor heterogeneity, performance adaptability, resource allocation, performance analysis, and performance portability. In addressing those issues the currently used approaches aim at providing simplified abstractions and machine models that allow algorithm developers and application programmers to generate code that can run and scale on a wide range of exascale computing systems.

This is a complex task that can be achieved by exploiting techniques that allow the runtime system to cooperate with the compiler, the libraries and the operating system to find integrated solutions and make smarter use of hardware resources by efficient ways to map the application code to the exascale hardware. Finally, due to the specific features of exascale hardware, runtime systems need to find methods and techniques that allow bringing the computing system closer to the application requirements. Research work in this area is carried out in projects like XPRESS, StarPU, Corvette DEGAS, libWater [10], Traleika-Glacier, OmpSs [8], SnuCL, D-TEC, SLEEC, PIPER, and X-TUNE that are proposing innovative solutions for large-scale parallel computing systems that can be used in exascale machines. For instance a system that aims at integrating the runtime with the language level is OmpSs where mechanisms for data dependence management (based on DAG analysis like in libWater) and for mapping tasks to computing nodes and handling processor heterogeneity (the *target* construct) are provided. Another issue to be taken into account in the interaction between the programming level and the runtime is performance and scalability monitoring. In the StarPU project, for example, performance feedback through task profiling and trace analysis is provided.

In very large-scale high performance machines and in Exascale systems, the runtime systems are more complex than in traditional parallel computers. In fact, performance and scalability issues must be addressed at the inter-node runtime level and they must be appropriately integrated with intra-node runtime mechanisms [25]. All these issues relate to system and application scalability. In fact, vertical scaling of systems with multicore parallelism within a single node must be addressed. Scalability is still an open issue in Exascale systems also because speed-up requirements for system software and runtimes are much higher than in traditional HPC systems and different portions of code in applications or runtimes can generate performance bottlenecks.

Concerning application resiliency, the runtime of Exascale systems must include mechanisms for restarting task and accessing data in case of software or hardware faults without requiring developer involvement. Traditional approaches for providing reliability in HPC include: checkpointing and restart (see for instance *MPI_Checkpoint*), reliable data storage (through file and in-memory replication or double buffering), and message logging for minimizing the checkpointing overhead. In fact, whereas the global checkpointing/restart technique is the most used to limit system/application faults, in the Exascale scenario new mechanisms with low overhead and highly scalability must be designed. These mechanisms should limit task and data duplication through smart approaches for selective replication. For example, silent data corruption (SDC) is recognized to be a critical problem in Exascale computing. However, although replication is useful, their inherent inefficiency must be limited. Research work is carried out in this area to define technique that limit replication costs while offering protection from SDC. For application/task checkpointing, instead of checkpointing the entire address space of the application, as occurs in OpenMP and MPI, the minimal state of the tasks needed to be checkpointed for the fault recovery must be identified thus limiting data size and recovery overhead.

Requirements of exascale runtime for data analysis

One of the most important aspect to ponder in applications that run on Exascale systems and analyze big datasets is the tradeoff between sharing data among processing elements and computing things locally to reduce communication and energy costs, while keeping performance and fault-tolerance levels. A scalable programming model founded on basic operations for data intensive/data-driven applications must include mechanisms and operations for

- Parallel data access that allows increasing data access bandwidth by partitioning data into multiple chunks, according to different methods, and accessing several data elements in parallel to meet high throughput requirements.
- Fault resiliency that is a major issue as machines expand in size and complexity. On Exascale systems with huge amount of processes, non-local communication must be prepared for a potential failure of one of the communication sides; runtimes must features failure handing mechanisms for recovering from node and communication faults.
- Data-driven local communication that is useful to limit the data exchange overhead in massively parallel systems composed of many cores; in this case data availability among neighbor nodes dictates the operations taken by those nodes.
- Data processing on limited groups of cores allows concentrating data analysis operations involving limited sets of cores and large amount of data on localities of Exascale machines facilitating a type of data affinity co-locating related data and computation.
- Near-data synchronization to limit the overhead generated by synchronization mechanisms and protocols that involve several far away cores in keeping data up-to-date.
- In-memory querying and analytics needed to reduce query response times and execution of analytics operations by caching large volumes of data in the computing node RAMs and issuing queries and other operation in parallel on the main memory of computing nodes.
- Group-level data aggregation in parallel systems is useful for efficient summarization, graph traversal and matrix operations, therefore it is of great importance in programming models for data analysis on massively parallel systems.
- Locality-based data selection and classification for limiting the latency of basic data analysis operations running in parallel on large scale machines in a way that the subset of data needed together in a given phase are locally available (in a subset of nearby cores).

A reliable and high-level programming model and its associated runtime must be able to manage and provide implementation solutions for those operations together with the reliable exploitation of a very large amount of parallelism.

Real-world big data analysis applications cannot be practically solved on sequential machines. If we refer to real-world applications, each large-scale data mining and machine learning software that today is under development in the areas of social data analysis and bioinformatics will certainly benefit from the availability of Exascale computing systems and from the use of Exascale programming environments that will offer massive and adaptive-grain parallelism, data locality, local communication and synchronization mechanisms, together with the other features discussed in the previous sections that are needed for reducing execution time and making feasible the solution of new problems and challenges. For example, in bioinformatics applications parallel data partitioning is a key feature for running statistical analysis or machine learning algorithms on high performance computing systems. After that, clever and complex data mining algorithms must be run on each single core/node of an Exascale machine on subsets of data to produce data models in parallel. When partial models are produced, they could be checked locally and must be merged among nearby processors to obtain, for example, a general model of gene expression correlations or of drug-gene interactions. Therefore for those applications, data locality, highly parallel correlation algorithms, and limited communication structures are very important to reduce execution time from several days to a few minutes. Moreover, fault tolerance software mechanisms are also useful in long-running bioinformatics applications to avoid restarting them from the beginning when a software/hardware failure occurs.

Moving to social media applications, nowadays the huge volume of user-generated data in social media platforms, such as Facebook, Twitter and Instagram, are very precious sources of data from which to extract insights concerning human dynamics and behaviors. In fact, social media analysis is a fast growing research area that will benefit form the use of Exascale computing systems. For example, social media users moving through a sequence of places in a city or a region may create a huge amount of geo-referenced data that include extensive knowledge about human dynamics and mobility behaviors. A methodology for discovering behavior and mobility patterns of users from social media posts and tweet includes a set of steps such as collection and pre-processing of geotagged items, organization of the input dataset, data analysis and trajectory mining algorithm execution, and results visualization. In all those data analysis steps, the utilization of scalable programming techniques and tools is vital to obtain practical results in feasible time when massive datasets are analyzed. The Exascale programming features and requirements discussed here and in the previous sections will be very useful in social data analysis for executing parallel tasks like concurrent data acquisition (thus data items are collected exploiting parallel queries from different data sources), parallel data filtering and data partitioning by the exploitation of local and in-memory algorithms, classification, clustering and association mining algorithms that are very computing intensive and need a large number of processing elements working asynchronously to produce learning models from billions of posts containing text, photos and videos. The management and processing of Terabytes of data that are involved in those applications cannot be done efficiently without solving issues like data locality, near-data processing, large asynchronous execution and the other ones addressed in Exascale computing systems.

Together with an accurate modeling of basic operations and of the programming languages/APIs that include them, supporting correct and effective data-intensive applications on Exascale systems will require also a significant programming effort of developers when they need to implement complex algorithms and data-driven applications such that used, for example, in big data analysis and distributed data mining. Parallel and distributed data mining strategies, like

- collective learning,
- meta-learning, and
- ensemble learning,

must be devised using fine grain parallel approaches to be adapted on Exascale computers. Programmers must be able to design and implement scalable algorithms by using the operations sketched above specifically adapted to those new systems. To reach this goal, a coordinated effort between the operation/language designers and the application developers would be very fruitful.

In Exascale systems, the cost of accessing, moving, and processing data across a parallel system is enormous [24, 30]. This requires mechanisms, techniques and operations for capable data access, placement and querying. In addition, scalable operations must be designed in such a way to avoid global synchronizations, centralized control and global communications. Many data scientists want to be abstracted away from these tricky, lower level, aspects of HPC until at least they have their code working and then potentially to tweak communication and distribution choices in a high level manner in order to further tune their code. Interoperability and integration with the MapReduce model and MPI must be investigated with the main goal of achieving scalability on large-scale data processing.

Different data-driven abstractions can be combined for providing a programming model and an API that allow the reliable and productive programming of very large-scale heterogeneous and distributed memory systems. In order to simplify the development of applications in heterogeneous distributed memory environments, large-scale data-parallelism can be exploited on top of the abstraction of n-dimensional arrays subdivided in partitions, so that different array partitions are placed on different cores/nodes that will process in parallel the array partitions. This approach can allow the computing nodes to process in parallel data partitions at each core/node using a set of statements/library calls that hide the complexity of the underlying process. Data dependency in this scenario limits scalability, so it should be avoided or limited to a local scale.

Abstract data types provided by libraries, so that they can be easily integrated in existing applications, should support this abstraction. As we mentioned above, another issue is the gap between users with HPC needs and experts with the skills to make the most of these technologies. An appropriate directive-based approach can be to design, implement and evaluate a compiler framework that allows generic translations from high-level languages to Exascale heterogeneous platforms. A programming model should be designed at a level that is higher than that of standards, such as OpenCL, including also checkpointing and fault resiliency. Efforts must be carried out to show the feasibility of transparent checkpointing of Exascale programs and quantitatively evaluate the runtime overhead. Approaches like CheCL show that it is possible to enable transparent checkpoint and restart, also in high-performance and dependable GPU computing including support for process migration among different processors such as a CPU and a GPU.

The model should enable the rapid development with reduced effort for different heterogeneous platforms. These heterogeneous platforms need to include low energy architectures and mobile devices. The new model should allow a preliminary evaluation of results on the target architectures.

Concluding remarks and future work

Cloud-based solutions for big data analysis tools and systems are in an advanced phase both on the research and the commercial sides. On the other hand, new Exascale hardware/software solutions must be studied and designed to allow the mining of very large-scale datasets on those new platforms.

Exascale systems raise new requirements on application developers and programming systems to target architectures composed of a very large number of homogeneous and heterogeneous cores. General issues like energy consumption, multitasking, scheduling, reproducibility, and resiliency must be addressed together with other data-oriented issues like data distribution and mapping, data access, data communication and synchronization. Programming constructs and runtime systems will play a crucial role in enabling future data analysis programming models, runtime models and hardware platforms to address these challenges, and in supporting the scalable implementation of real big data analysis applications.

In particular, here we summarize a set of open design challenges that are critical for designing Exascale programming systems and for their scalable implementation. The following design choices, among others, must be taken into account:

- *Application reliability*: Data analysis programming models must include constructs and/or mechanisms for handling task and data access failures and for recovering. As new data analysis platforms appear ever larger, the fully reliable operations cannot be implicit and this assumption becomes less credible, therefore explicit solutions must be proposed.
- *Reproducibility requirements*. Big data analysis running on massively parallel systems demands for reproducibility. New data analysis programming frameworks must collect and generate metadata and provenance information about algorithm characteristics, software configuration and execution environment for supporting application reproducibility on large-scale computing platforms.
- *Communication mechanisms*: Novel approaches must be devised for facing network unreliability [7] and network latency, for example by expressing asynchronous data communications and locality-based data exchange/sharing.
- *Communication patterns*: A correct paradigm design should include communication patterns allowing application dependent features and data access models, limiting data movement and simplify the burden on Exascale runtimes and interconnection.
- *Data handling and sharing patterns*: Data locality mechanisms/constructs, like near-data computing must be designed and evaluated on big data applications when subsets of data are stored in nearby processors and by avoiding that locality is imposed when data must be moved. Other challenges concern data affinity control data querying (NoSQL approach), global data distribution and sharing patterns.
- *Data-parallel constructs*: Useful models like data-driven/data-centric constructs, dataflow parallel operations, independent data parallelism, and SPMD patterns must be deeply considered and studied.
- *Grain of parallelism*: from very fine-grain to process-grain parallelism must be analyzed also in combination with the different parallelism degree that Exascale hardware supports. Perhaps different grain size should be considered in a single model to address hardware needs and heterogeneity.

Finally, since big data mining algorithms often require the exchange of raw data or, better, of mining parameters and partial models, to achieve scalability and reliability on thousands of processing elements, metadata-based information, limited-communication programming mechanisms, and partition-based data structures with associated parallel operations must be proposed and implemented.

Endnotes

[1]https://www.ibm.com/annualreport/2013/bin/assets/2013_ibm_annual.pdf

Appendix

Scalability in parallel systems

Parallel computing systems aim at exploiting the capacity of usefully employing all its processing elements during application execution. Indeed, only an ideal parallel system can do that fully because of its sequential times that cannot be parallelized (As the Amdahl's law suggests [35]) and due to several sources of overhead such as sequential operations, communication, synchronization, I/O and memory access, network speed, I/O system speed, hardware and software failures, problem size and program input. All these issues related to the ability of parallel systems to fully exploit their resources are referred as system or program scalability [36].

The scalability of a parallel computing system is a measure of its capacity to reduce program execution time in proportion to the number of its processing elements. According to this definition, scalable computing refers to the ability of a hardware/software parallel system to exploit increasing computing resources effectively in the execution of a software application [37].

Despite the difficulties that can be faced in the parallel implementation of an application, a framework or a programming system, a scalable parallel computation can always be made cost-optimal if the number of processing elements, the size of memory, the network bandwidth and the size of the problem are chosen appropriately.

For evaluation and measuring scalability of a parallel program some metrics have been defined and are largely used: parallel runtime *T(p)*, speedup *S(p)* and efficiency *E(p)*. Parallel runtime is the total processing time of the program using *p* processor (with $p > 1$). Speedup is the ratio between the total processing time of the program on 1 processor and the total processing time on p processors: $S(p) = T(1)/T(p)$. Efficiency is the ratio between speedup and the total number of used processors: $E(p) = S(p)/p$.

Application scalability is influenced by the available hardware and software resources, their performance and reliability, and by the sources of overhead discussed before. In particular, scalability of data analysis applications are tight related to the exploitation of parallelism in data-driven operations and the overhead generated by data management mechanisms and techniques. Moreover, application scalability also depends on the programmer ability to design the algorithms reducing sequential time and exploiting parallel operations. Finally, the instruction designers and the runtime implementers contribute to exploitation of scalability [38]. All these arguments mean that for realizing exascale computing in practice many issues and aspects must ne taken into account by considering all the layers of hardware/software stack involved in the execution of Exascale programs.

In addressing parallel system scalability it must be also tackled system dependability. As the number of processors and network interconnection increases and as tasks, threads and message exchanges increase, the rate of failures and faults increases too [39]. As discussed in reference [40], the design of scalable parallel systems requires assuring system dependability. Therefore understanding of failure characteristics is a key issue to couple high performance and reliability in massive parallel systems at Exascale size.

Abbreviations

APGAS: Asynchronous partitioned global address space; BSP: Bulk Synchronous Parallel; CAF: Co-Array Fortran; DAaaS: Data Analysis as a Service; DAIaaS: Data analysis infrastructure as a service; DAPaaS: Data analysis platform as a service; DASaaS: Data analysis software as a service; DMCF: Data Mining Cloud Framework; ECL: Enterprise Control Language; ESnet: Energy Sciences Network; GA: Global Array; HPC: High Performance Computing; IaaS: Infrastructure as a service; JS4Cloud: JavaScript for Cloud; PaaS: Platform as a service; PGAS: Partitioned global address space; RDD: resilient distributed dataset; SaaS: Software as a service; SOA: Service Oriented Computing; TBB: Threading Building Blocks; VL4Cloud: Visual Language for Cloud; XaaS: Everything as a Service

Acknowledgments
Not applicable.

Funding
This work has been partially funded by the ASPIDE Project funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 801091.

Availability of data and materials
Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Authors' contributions
DT carried out all the work presented in the paper. The author read and approved the final manuscript.

Authors information
Not applicable.

Competing interests
The author declare that he/she has no competing interests.

Publisher's Note

Received: 23 October 2018 Accepted: 29 January 2019
Published online: 11 February 2019

References

1. Amarasinghe S et al (2009) Exascale software study: software challenges in extreme-scale systems. In: Defense Advanced Research Projects Agency. Arlington, VA, USA
2. Armbrust M et al (2010) A view of cloud computing. Commun ACM 53(4 (April 2010)):50–58
3. Bauer M et al (2012) Legion: Expressing locality and independence with logical regions. In: Proc. International Conference on Supercomputing. IEEE CS Press
4. Bradford L, Chamberlain CD, Zima HP (2007) Parallel programmability and the chapel language. International Journal of High Performance Computing Applications 21(3):291–312
5. Diaz J, Munoz-Caro C, Nino A (2012) A survey of parallel programming models and tools in the multi and many-core era. IEEE Trans Parallel Distributed Systems 23(8):1369–1386
6. http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm
7. Fekete A, Lynch N, Mansour Y, Spinelli J (1993) The impossibility of implementing reliable communication in the face of crashes. Journal of ACM 40(5):1087–1107
8. Fernandez A et al (2014) Task-based programming with OmpSs and its application. In: Proc. euro-par 2014: parallel processing workshops, pp 602–613
9. Gorton I, Greenfield P, Szalay AS, Williams R (2008) Data-intensive computing in the 21st century. Computer 41(4):30–32
10. Grasso I, Pellegrini S, Cosenza B, Fahringer T (2013) LibWater: heterogeneous distributed computing made easy. *Procs of the 27th international ACM conference on International conference on supercomputing (ICS '13)*, New York, USA, pp 161–172
11. Gropp W, Snir M (2013) Programming for exascale computers. Computing in Science & Eng 15(6):27–35
12. Gu Y., Grossman R. L., Sector and Sphere: the design and implementation of a high-performance data cloud. Philosophical transactions Series A, Mathematical, physical, and engineering sciences 367 1897, 2429–2445, 2009
13. Lucas et al (2014) Top ten Exascale research challenges. In: Office of Science, U.S. Department of Energy, Washington, D.C
14. Maheshwari K, Montagnat J (2010) Scientific workflow development using both visual and script-based representation. *Proc. of the Sixth World Congress on Services SERVICES '10*, Washington, DC, USA, pp 328–335
15. Markidis S et al (2016) The EPiGRAM project: preparing parallel programming models for exascale. In: Proc. ISC high performance 2016 international workshops, pp 56–58
16. Marozzo F, Talia D, Trunfio P (2015) JS4Cloud: script-based workflow programming for scalable data analysis on cloud platforms. Concurrency and Computation: Practice and Experience 27(17):5214–5237

17. Marozzo F, Talia D, Trunfio P (2013) A cloud framework for big data analytics workflows on azure. In: Catlett C, Gentzsch W, Grandinetti L, Joubert G, Vazquez-Poletti J (eds) Cloud Computing and Big Data. IOS press, advances in Parallel Computing, pp 182–191
18. Marozzo F, Talia D, Trunfio P (2012) P2P-MapReduce: parallel data processing in dynamic cloud environments. J Comput Syst Sci 78(5):1382–1402
19. Meswani MR et al (2012) Tools for benchmarking, tracing, and simulating SHMEM applications. In: Proceedings Cray user group conference (CUG)
20. Nieplocha J (2006) Advances, applications and performance of the global arrays shared memory programming toolkit. International Journal of High Performance Computing Applications 20(2):203–231
21. Nishtala R et al (2011) Tuning collective communication for partitioned global address space programming models. Parallel Comput 37(9):576–591
22. Olston C et al (2008) Pig Latin: a not-so-foreign language for data processing. In: Proceedings SIGMOD '08. Vancouver, Canada, pp 1099–1110
23. Pectu, D. et al., On processing extreme data, Scalable Computing: Practice and Experience, 16, 4, 467–489, 2015
24. Reed DA, Dongarra J (2015) Exascale computing and big data. Commun ACM 58(7):56–68
25. Sarkar V, Budimlic Z, Kulkarni M (eds) (2016) Runtime systems report - 2104 runtime systems summit, U.S. Dept of Energy, USA
26. Talia D (2013) Clouds for scalable big data analytics. Computer 46(5):98–101
27. Talia D, Trunfio P, Marozzo F (2015) Data Analysis in the Cloud - Models, Techniques and Applications. Elsevier, Amsterdam, Netherlands
28. Talia D (2015) Making knowledge discovery services scalable on clouds for big data mining. In: Proc. 2nd IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM), IEEE computer society press, pp 1–4
29. Tardieu O et al (2014) X10 and APGAS at petascale. In: Proc. of the ACM SIGPLAN symposium on principles and practice of parallel programming (PPoPP'14)
30. U.S (2013) Department of Energy, Synergistic Challenges in Data-Intensive Science and Exascale Computing. In: Report of the advanced scientific computing advisory committee subcommittee
31. Hwang K (2017) Cloud computing for machine learning and cognitive applications, MIT Press
32. Wozniak JM, Wilde M, Foster IT Language features for scalable distributed-memory dataflow computing. In: . Proceedings of the workshop on data-flow execution models for extreme-scale computing at PACT, Edmonton, Canada, p 2014
33. Yoo A, Kaplan Y (2009) Evaluating use of data flow systems for large graph analysis. In: Proceedings of the 2nd workshop on many-task computing on grids and supercomputers (MTAGS)
34. Zaharia M et al (2014) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65
35. Amdahl GM (1967) Validity of single-processor approach to achieving large-scale computing capability. *Proc. of AFIPS Conference*, Reston, VA, pp 483–485
36. Bailey D (1991) Twelve ways to fool the masses when giving performance results on parallel computers, RNR technical report, RNR-90-020, NASA Ames Research Center
37. Grama, A. et al., *Introduction to Parallel Computing*, Addison Wesley, 2003
38. Gustafson JL (1988) Reevaluating Amdahl's Law. Commun ACM 31(5):532–533
39. Shi JY et al (2012) Program scalability analysis for HPC cloud: applying Amdahl's law to NAS benchmarks. In: Proc. High Performance Computing, Networking, Storage and Analysis (SCC), IEEE, CS, pp 1215–1225
40. Schroeder B, Gibson G (2006) A large-scale study of failures in high-performance computing systems. Proc. of the International Conference on Dependable Systems and Networks (DSN2006), IEEE CS, Philadelphia, PA, pp 25–28