Project no. 801091

ASPIDE

Research & Innovation Action (RIA)
EXASCALE PROGRAMING MODELS FOR EXTREME DATA PROCESSING

Libraries and tools to support extreme data processing
D2.2

Due date of deliverable: 14th June 2019

Start date of project: June 15^{th}, 2018

Type: Deliverable
WP number: WP2

Responsible institution: University Carlos III of Madrid
Editor and editor's address: David del Rio, University Carlos III of Madrid

Version 2.5

Project co-funded by the European Commission within the Horizon 2020 Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision History

Rev.	Date	Author(s)	Description
1.0	8/05/19	David del Rio (UC3M)	Initial Version.
1.1	30/06/19	Javier Garcia-Blas (UC3M)	Adding related work.
1.2	12/06/19	Adrian Spătaru (IeAT)	Added Chapter 4 introduction and model specification.
1.3	13/06/19	Adrian Spătaru (IeAT)	Added Figure 4.1. Orchestrator Section text.
1.4	13/06/19	Adrian Spătaru (IeAT)	Finished Chapter 4.
1.5	13/06/19	Ioan Drăgan (IeAT)	Rephrasing in Chapter 4.
2.0	14/06/19	Domenico Talia (UNICAL)	Updated the Executive Summary and fixed some typos.
2.1	14/06/19	Javier Garcia-Blas (UC3M)	Adding figure of dependencies.

Main Contributors

Adrian Spătaru (IeAT)
David del Rio (UC3M)
Domenico Talia (UNICAL)
Ioan Drăgan (IeAT)
Javier Garcia-Blas (UC3M)

Executive Summary

This document discusses libraries and software tools for extreme data processing, it is structured as follows. Section 2 covers the state-of-the-art of existing libraries and tools for both data-intensive and data-analytics applications. Section 3 presents a prototype of a parallel programming framework for data-intensive application based on task parallelism and workflow execution. Section 4 presents the ASPIDE Gateway, a collection of components designed to handle application composition and orchestration. Section 5 concludes this document.

The placement of the deliverable is outlined in Figure 1.

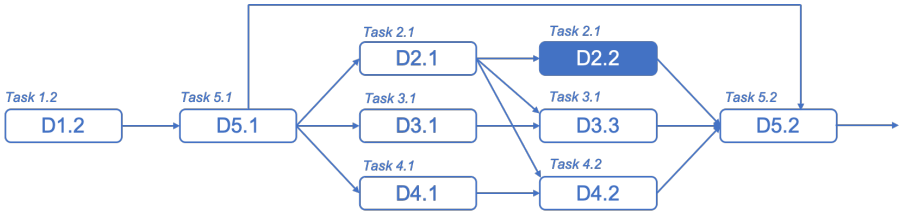

Figure 1: Deliverable dependencies of D2.2.

Contents

Chapter 1

Introduction

The modern computational elements used in distributed heterogeneous platforms provides performance improvements thanks to their parallel capabilities. However, programming efficiently for these architectures demands big efforts in order to transform sequential applications into parallel and to optimize such applications. Compared to sequential programming, designing and implementing parallel applications for operating on modern hardware poses a number of new challenges to developers. Communication overheads, load imbalance, poor data locality, improper data layouts, contention in parallel I/O, deadlocks, starvation or the appearance of data races in threaded environments are just examples of those challenges. Besides, maintaining and migrating such applications to other parallel platforms demands considerable efforts. Thus, it becomes clear that programmers require additional expertise and endeavor to implement parallel applications, apart from the knowledge needed in the application domain.

To tackle this issue, several solutions in the area, such as parallel programming frameworks, have been developed to efficiently exploit parallel computing architectures. Indeed, multiple parallel programming frameworks from the state-of-the-art benefit from shared memory multi-core architectures, such as OpenMP, Cilk or Intel TBB; distributed platforms, such as MPI or Hadoop; and some others especially tailored for accelerators, as e.g., OpenCL and CUDA. Basically, these frameworks provide a set of parallel algorithmic skeletons (e.g. parallel_for or parallel_reduce) representing recurrent algorithmic structures that allow running pieces of source code in parallel. However, users require understanding different frameworks, not only to decide which fits best for their purposes but also to properly use them. Not to mention the migration efforts of applications among frameworks, which becomes as well an arduous task.

In this project, we will desing and develop a data-aware programming model for data intesive applications in terms of data-parallel tasks using a simplified API and DSLs on top of it. This way, the proposed solution maihg help application developers

to access and use resources without the need to manage low-level architectural entities. In the same way, we want to provide a way to easily swich among different execution modes or policies without requiring to modify the applications source code.

This document introduces some already existing tools and libraries for supporting extreme data processing from the state-of-the-art. Additionally, we describe in detail those tools that will be employed or extended in this project for supporting the proposed programming paradigm in D2.1.

Chapter 2

Existing libraries and tool for data intensive applications

2.1 Big Data Analytics Ecosystem

Big Data affects many different ecosystems and areas of research and business, thus there is no unique definition for it and its scope is still a controversial topic among these communities. From the data analysis perspective, the multi-v model reflects a way to define Big Data by describing several of its features, and it keeps evolving over time adding more attributes as needed. The core characteristics included in this model are [1]:

1. Volume of the data sise. Volume is necessary in order to get valuable insight from analytics tools. It is usual to find volumes in the order of peta or terabytes at the enterprise level. These volumes can also be quantified in the order of billions of records, tables, files or transactions depending on the platform used by the system. In order to handle these volumes, Big Data systems and applications must be designed to maximise throughput and resilience.

2. Velocity of data production and processing. Data can be produced and consumed at different rates. Big Data systems can even incorporate diverse source frequencies and processing speeds including batch processing, streaming, near- and real-time speed.

3. Variety of data types. Nowadays a data source an be anything: sensors, third parties, Web applications, etc. Hence data can be highly heterogeneous and may be unstructured, and platforms must be able to understand and integrate this diverse data to aggregate the knowledge from different sources. In addition, data types depend greatly on the application and its domain: we find structured statistical data in business intelligence, time series and

geospatial data in Internet-of-Things, and media, text and graph data in social environments.

In addition, from the business perspective the following features are also key [2]:

1. Veracity of data. The volume of data is key to obtain knowledge, but the derived information would be flawed if the quality of data is low. High-quality Big Data must be reliable in terms of trust and integrity.

2. Value in business terms. The model or analysis that results from processing Big Data must provide enterprise value to make up for the investment expenses necessary to collect and analyse data.

These definitions have a key aspect in common: Big Data focuses on data, in particular, on data that is large in volume as perceived by the communities over time. This paradigm-shift centred towards data has affected all areas of computing from data acquisition, transfer and storage, to data analysis and visualisation. This reflected on traditional areas of business and science –like genomics, climatology, finance, and business intelligence– that were able to obtain better knowledge with existing methods, but also promoted novel areas of research to exploit the intrinsic value of data and improve the system's capability to cope with the requirements of data processing and storage. Areas like Internet-of-things and Big Data analytics developed greatly thanks to the advances in Big Data.

Big Data analytics (BDA) is one of the best examples of how Big Data disrupted an established area like business intelligence, exploiting advanced analytics techniques operating on Big Data to evolve from descriptive tools to predictive and prescriptive models. Today, enterprises are exploring Big Data to incorporate knowledge discovery into their business to detect interrelations among apparently unrelated attributes of datasets [3]. Enterprises can now understand the current state of the business and customer behaviour through complex techniques like predictive analytics, data mining, statistical analysis, data visualisation, artificial intelligence, and natural language processing, paired with support platforms such as map-reduce, in-database analytics, in-memory databases, and columnar data stores. Some of these techniques have been around for years and they have been revamped due to their good adaptability to very large data sets with minimal data preparation. In addition, infrastructures like cloud computing offer the possibility to lower the economical costs of deploying BDA, and building analytics workflows at different levels of abstractions.

Figure 2.1 represents the traditional knowledge discovery workflow for Big Data, which includes dealing with data acquisition from diverse sources, processing and combining data in many ways in order to build a model that can be used for analysis and visualisation, finally incorporating feedback mechanisms to refine data processing and modelling stages. This workflow has been usually combined with the lambda architecture [4] to provide scalable integration and interoperability across different datasets trough real-time analytics. This architecture was proposed

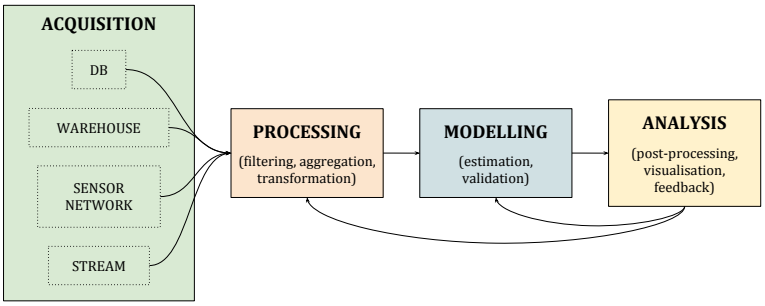

Figure 2.1: Typical workflow of a BDA application.

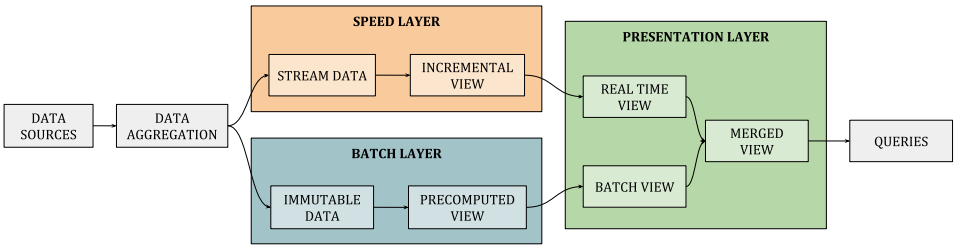

Figure 2.2: Representation of the lambda architecture.

with the goal of providing a generalist platform to serve different applications with diverse latency needs in a streaming environment. As shown in Fig. 2.2, the lambda architecture includes a speed layer for pure stream processing in real-time, a batch layer for storing raw data and processing higher quality views of long-term data, and a presentation layer that manages queries and output visualisation.

Looking ahead, it is expected that areas such as mobile technology, social media, IoT and data-driven sciences will generate data to a global total in the order of dozens of zettabytes [5]. This data will yield valuable information for smart applications, science and decision making processes in business.

2.1.1 Data-Centric Batch and Stream Processing

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 [6] data processing model was the most relevant data-centric model when BDA research took off, as it enables analytics on big datasets by parallelising computations for HPC and multi-core environments [7]. 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) [8] to achieve locality by block replication, and considers data-aware task scheduling. A similar approach is followed by the open map-reduce implementation, Hadoop [9], and its partner file system Hadoop Distributed File System (HDFS) [10]. Applications in map-reduce work with many large files and need to execute fast transfers and operations on a wide and diverse dataset. 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.

One of the models that emerged from map-reduce is map-reduce-merge [11], 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 [12] 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 [13], Haloop [14] and Twister4Azure [15]. 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. HaLoop's 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 work in [16] 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 [17] 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) [18], through multiple parallel operations, built around an acyclic data flow model. It retains, however, the scalability and fault tolerance features of map-reduce. The Spark framework relies heavily in the concept of *resilient distributed dataset* (RDD) [18] 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. However, in most Java-based map-reduce platforms [19] the

deep component stack and its dependence on the JVM yield a significant memory consumption that also affects execution time due to frequent garbage collection operations [20] and serialisation if bindings to other languages are used [21].

Spark has inspired subsequent works like GraphX, 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.

Map-reduce-based programming models [22]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. Moreover, some models evolved into workflow frameworks to support the composition of heterogeneous and coupled components to simulate different aspects of an application model. As these modules interact and exchange significant volumes of data at runtime [23], making these transfers efficient has a major impact in the overall performance. 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 [24] offers scheduling optimisations at the task 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 task 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. Another approach is followed by Dryad [25], 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. The DMCF framework [26] has some similarities with Dryad. In particular, the main contribution of DMCF is the integration of different hardware/software solutions for high-level programming, management and execution of parallel data analysis workflows. The DMCF runtime was designed to enable the parallel execution of data analysis workflows on multiple virtual machines to improve performance and ensure scalability of applications. To this end, the runtime implements data-driven task parallelism that automatically spawns ready-to-run workflow tasks to the Cloud resources, taking into account dependencies among tasks and current availability of data to be processed.

Finally, Nephele [27] 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 Dryad, jobs in Nephele are expressed as a directed acyclic graph (DAG).

Several frameworks have explored the possibility of building rich data SQL-like abstractions for database processing. For example, Pig Latin [28] 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 [29]. Another popular approach is Hive [30], an open-source data warehousing solution that 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. Finally, REX [31] is a programming model built for database management systems (DBMSs) that support recursive SQL. These systems are more efficient because they only propagate the changes in the states that are produced in each step, while accumulating each iteration. REX focuses on supporting these iterative computations in order to update states in an efficient and extensible manner. Additionally, REX's data partitioning and replication scheme ensures that every node can access any non-local data without consulting a central directory server.

In-memory computing has also affected data-base oriented platforms with approaches like Phoenix [32] for shared and distributed memory machines. Shark [33], which supports the Hive warehousing system [34] on Spark, is a popular 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. Similarly, pure map-reduce paradigms have benefited from in-memory trends resulting in platforms for memory-intensive workloads such as Mammoth [19], Piccolo [35], Main-Memory Map-Reduce (M3R) [36], and Hyracks [37].

Some of the former works indicate that 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. New technologies based in multicore processors can improve the performance of applications by favouring locality through intra-node data sharing, which minimises data exchanges across compute nodes [23]. The prospective usage of map-reduce based models at different levels of parallelism within the computing infrastructure, as typically done in HPC systems,

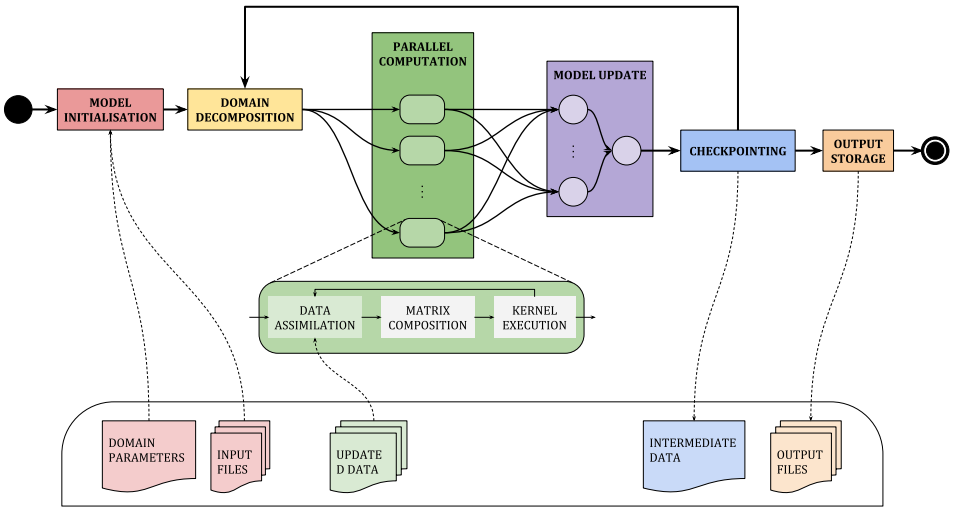

Figure 2.3: Application model for the typical HPC scientific application.

might provide a shared space programming abstraction that replaces existing parallel programming models such as message passing.

2.2 High-Performance Computing Ecosystem

High-performance computing (HPC) refers to the usage of aggregated computing power in order to deliver as much speed as possible to run complex parallel programs efficiently. This terms is tightly related to the concept of *supercomputing*, which pushes HPC to the highest operational rate of the available technology. Nowadays, top modern supercomputers reach perform in the order of one hundred petaflops, and a machine capable of delivering one exaflop is expected to appear around 2020.

All this computational power and sophisticated infrastructures involve massive investment in hardware development, runtime design, and daily operational costs. As a consequence, they have been put to the service of strategic areas of science and industry that rely on complex numerical applications that cannot be run on commodity machines due to their performance requirements. This includes sectors like aviation, energy, pharmaceutical, oil and gas, and automotive; and high-end scientific research on climate, medicine, bioinformatics, and physics.

To exploit the scalability and performance of supercomputers, HPC applications rely heavily on parallelism techniques to maximise the usage of resources. Supported by advanced runtimes, these applications coordinate parallel processing on many cores and nodes with network-intensive data transfers between compute and storage nodes. In addition, some applications need to iterate to refine their results, modify the underlying model, or incorporate new data. Figure 2.3 depicts these relationships

which form the structure of many HPC applications. The core simulated models are typically initialised with a combination of input data and base environmental conditions as parameters, and the simulation domain is distributed so that kernel computations can be conducted in parallel. Ideally, these simulations are pleasingly parallel and computations can be executed independently while incorporating partial new data. Once kernels converge, the resulting data is merged with the results coming from the other processing units in order to update the model, leading to a communication-intensive process that results in the input that will be fed to the following step. As a fault-tolerance measure, some simulations include checkpointing procedures to store intermediate models and restore the simulation from them in case of failure. Finally, simulation results are written to storage.

HPC is not unaffected by current data-centric trends, and scientists are already tackling how HPC can benefit from the availability of Big Data. High-performance data analytics, data-intensive scientific computing, visualisation and machine learning are areas of research that currently inherit the performance and scalability aspirations of traditional HPC, while incorporating new challenges that affect how data is managed and transmitted at all levels of the system and software stack.

2.2.1 Supercomputers and Data-Intensive Clusters

Large scale HPC 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 with heavy resource requirements, thus they need dedicated platforms that orchestrate tasks and manage resources in order to behave in a coordinated manner, and meet the application requirements. These pieces of software constitute the middleware that permits node intercommunication, data transmission, load balancing, task assignment and fault tolerance, among others.

Traditional HPC infrastructures are built in such way that storage and computation are not located in the same nodes, following the schema depicted in 2.4. Networks are also isolated to avoid the interference of I/O operations to the parallel file system with computation communications. Parallel file systems maintain a logical space view and provide an efficient access to data, which can be distributed through several sites and among multiple I/O servers and disks to deliver higher degree of parallelism.

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

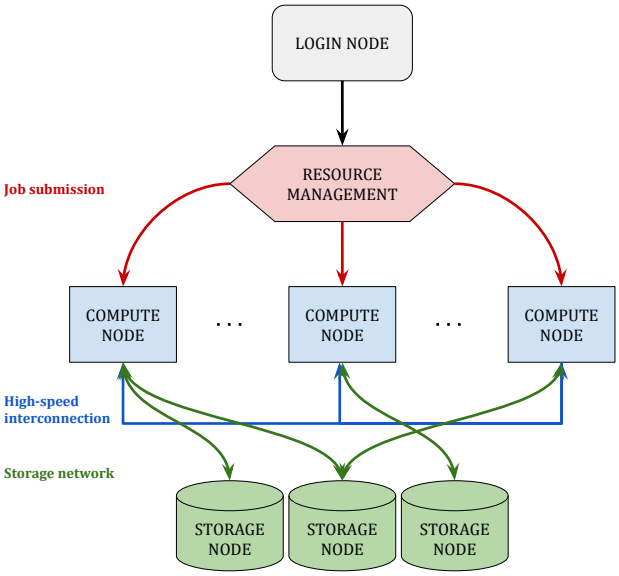

Figure 2.4: Traditional architecture of a HPC infrastructure with isolated storage and computation networks.

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. With the advent of data-centric trends, recent works have suggested that improving data locality across all layers of the system stack is key to move towards exascale infrastructures efficiently [38].

Some authors claim that current architecture of high-end computing systems is inefficient because storage is completely segregated from the compute resources, thus further network interconnections are needed to access storage [39]. 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. To avoid the drawbacks of traditional parallel file systems, a new generation of distributed file systems has emerged as support layers for data-centric frameworks like map-reduce. The most relevant parallel file systems that have a focus on data locality are the Hadoop File System (HDFS) [40], the Google File System (GFS) [8]. Work in this area has also been conducted to improve locality by moving data to the node's memory to minimise interaction with storage nodes. This resulted in new infrastructure architectures that incorporate deeper memory hierarchies and local storage in compute nodes, following the model of cloud-oriented datacentres [CITE SAN DIEGO Y LOS 2020].

The influence of Big Data and analytics in supercomputing is also reflected in the

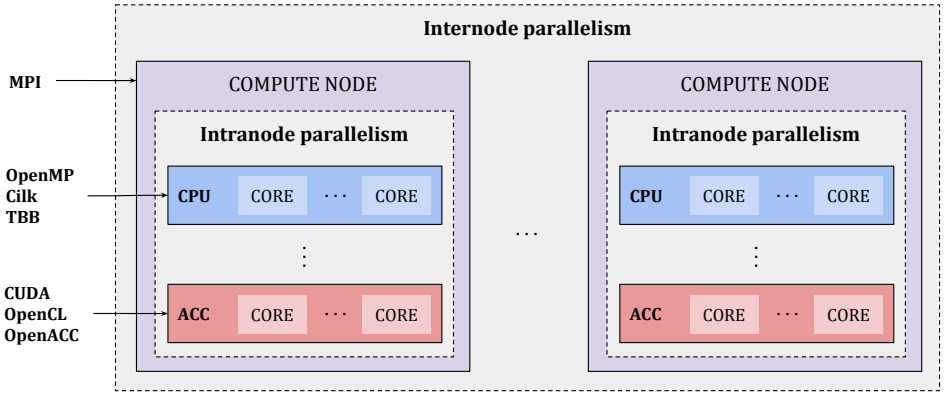

Figure 2.5: Parallelism layers in HPC programming models, including inter- and intra-node parallelism.

incorporation of new hardware architectures tailored for deep learning and data-intensive computing [41], resulting in dedicated accelerators like vector processors [CITE], tensor processing units (TPUs) [CITE], general purpose graphical processing units (GPGPUs) [CITE], and field programmable gate arrays (FPGAs). These new technologies provide further computational power for applications, but it is still unclear which areas will require the full exascale power that will be provided by impending heterogeneous infrastructures, as the bottleneck might remain at upper layers of the software stack like monitoring, resource management, data management, and communications [42]. In addition, applications are also evolving towards complex workflows involving iterative analytics, data-intensive operations and compute-intensive computations. Making an efficient usage of supercomputers in this landscape will require algorithm, runtime and data management refinements to support applications with mixed requirements, without diminishing usability.

2.2.2 Parallel Programming Models and Runtimes

HPC applications aim to run at the maximum level of parallelism provided by supercomputers in order to reduce execution time and increase scalability. On submission, applications are provided with a set of allocated processing units distributed across several nodes, and optionally different types of accelerators might be assigned if present in the infrastructure. Figure 2.5 represents these diverse processing units.

The message passing interface (MPI) standard is the most common procedure to exploit inter-node parallelism in HPC environments, and is the basis for numerous runtimes and workflows for scientific computing [CITE]. The implementations of MPI allow the execution of standard operations comprising multiple processes on distributed memory platforms, which provided coarse-grained parallelism sufficient

for terascale applications.

Thread-level parallelism is the basis for fine-grained intra-node parallelism for multi-core CPUs. Developers can choose from a wide range of threading libraries like POSIX threads [CITE], Intel's threading building blocks (TBB) [CITE], and Microsoft's parallel patterns library (PPL) [CITE]. Nowadays, the open specification for multi-processing (OpenMP) [CITE] is still one of the most used tools for parallelisation, mostly because its annotation-based nature minimises the impact on sequential code. As machines reached petascale, combining MPI and OpenMP became a common procedure to reach massive parallelism on machines supporting distributed and shared memory [43–45].

Current HPC infrastructures have incorporated different types of accelerators to enhance the performance of specific applications. Programming models adapted accordingly to ease the access to further finer-grain intra-node parallelism. GPGPUs are the most widely adopted accelerator in current HPC machines given they power efficiency and their many-core architecture, which pushes forward massive parallelism to the order of thousands of cores in a single chip. There are several libraries that enable the interaction with GPGPUs, such as OpenCL [CITE], Nvidia CUDA [CITE], and OpenACC [CITE], supporting data offload to the accelerators, kernel operator definition, direct execution of such code on the device, and result retrieval back to the host CPU. Accelerator runtimes have also been integrated with intra-node parallelism through OpenMP [46], and inter-node parallelism via MPI [47, 48]. The mechanisms to build hybrid runtimes exploting both intra- and inter-node parallelism had major influence in subsequence advances in further parallelism integration, and they are expected to be present in future exascale systems to cope with the need for adaptive hybrid programming models for heterogeneous extreme scale machines [49].

2.3 Current Trends in HPDA Convergence

Instances of HPDA convergence are evident in the literature of computer science, physical sciences, and business. One example of using HPC to accelerate BDA tools is Malitsky et al. [50], who developed Spark workflows over MPI to parallelise and visualise reconstructions of synchrotron light source X-ray microscopy. Another example, from the commercial sector, of HPC for BDA is PayPal, who rely on the high concurrency and low latency of HPC systems for fraud detection [51]. Evidence of convergence in the opposite direction –BDA tools for HPC applications– also appears. Nashed et al. [52] demonstrated that machine learning libraries, specifically TensorFlow, could be applied to HPC ptychographic reconstruction, and parallelised over DIY [53]. Bicer et al. created a map-reduce framework over MPI called Trace for tomographic reconstruction [54].

Because Spark underlies many BDA tools, the performance of Spark for scientific

computing has been studied in several works. Kira [55], a flexible and distributed astronomy image processing toolkit using Apache Spark, was used to implement a source extractor application called Kira SE for astronomy images. The study shows that Spark may be an alternative to an equivalent C program for many-task applications. Another interesting study was shown in [56], where the performance of a Spark implementation of a classification algorithm in the domain of High Energy Physics (HEP) was evaluated. The results showed that the implementation scaled well, but the performance was poor compared with the results of an untuned MPI implementation of the same algorithm.

To overcome the former problems, three main approaches have been proposed: developing tailored frameworks, implementing a map-reduce framework using MPI or executing the Spark framework using MPI as the communication engine, and promoting storage converge between BD and HPC platforms.

2.3.1 Tailored Map-Reduce frameworks

Several tailored map-reduce and data analytics frameworks have been developed. These environments target a particular family of applications or processor architecture, but they are not generalised for reuse in other contexts. A preliminary work was ROOT [57], an object-oriented C++ high-energy physics (HEP) framework designed for storing and analysing petabytes of data efficiently by using a TTree object container optimised for statistical data analysis over very large data sets. A proposal to accelerate spark communication was presented in [58], which used a high-performance RDMA-accelerated data shuffle in the Spark framework on high-performance networks and provided a performance improvement of 80%. An adaptation of the map-reduce framework for specific heterogeneous architectures has been proposed in MrPhi [59], an optimised framework for Intel Xeon Phi co-processors. A similar solution for Spark on GPUs was IBMSparkGPU [60], but it is valid for local tasks only. Trace [54], mentioned earlier, is a high-throughput tomographic reconstruction engine for large-scale datasets using both (thread-level) shared memory and (process-level) distributed memory parallelisation using a special data structure called a replicated reconstruction object. Fox et al. studied in [61] various frameworks for deep learning networks that can scale across multiple machines with full parallel support and distributed execution, such us Tensorflow, CNTK, Deeplearning4j, MXNet, H2O, Caffe, Theano, and Torch.

An approach for running data-centric applications on MPI was proposed in [62], where the authors proposed an event-driven pipeline and in-memory shuffle using DataMPI-Iteration, which provided overlapping of computation and communication for iterative BDA computing and showed a speedup of 9X-21X over Apache Hadoop, and 2X-3X over Apache Spark for PageRank and K-means. In [63], Anderson et al. proposed a system for integrating MPI with Spark by offloading computation to an MPI environment from within Spark. The evaluation made with four distributed

graph and machine learning applications shows speedups between 3X and 17X, including all of the overheads.

Several works have addressed the opportunities of shifting scientific workflows from traditional HPC and HTC infrastructures to BDA computing platforms. In particular, authors have focused on exploring data-intensive workflows, since they are the most tightly related to conventional BDA applications in terms of data volumes [64, 65]. Experimentation with well known workflows, like Montage, shows that running costs could be significantly decreased with BDA infrastructures, but performance would suffer from virtualisation and latency overheads [66–68].

Other works have been devoted to the application of BDA frameworks, specially map-reduce, to the HPC world. Most attempts have tried to adapt map-reduce to use distributed file systems available in HPC environments. For instance, the Ceph file system provides support for Hadoop, and Maltzahn et al. [69] studied the combination of both systems. The result of running map-reduce with HDFS and GPFS was studied by Ananthanarayanan et al. [70] and Wang et al. [71]. These works investigated the data-centric analytics framework running on a compute-centric HPC environment that relies on high-performance file systems.

The performance of executing HPC applications on data-centric cloud has also been studied in several works. The results of executing a scientific HPC application on Amazon EC2 were presented by Evangelinos et al. [72] analysed, showing that the performance of network in cloud is worse than that of HPC by one to two orders of magnitude, which was also proved by Gupta et al. [73] on different cloud platforms. Both showed that raw performance difference between compute centric HPC and cloud is pronounced.

2.3.2 Map-Reduce frameworks using MPI

Instances of BDA and HPC convergence are evident in the literature of computer science, physical sciences, and business. Malitsky et al. [50] developed Spark workflows over MPI to parallelise and visualise reconstructions of synchrotron light source X-ray microscopy. PayPal relies on the high concurrency and low latency of HPC systems for fraud detection in BDA [51]. Convergence in the opposite direction—BDA tools for HPC applications—also appears, for example, in the usage of machine learning libraries, specifically TensorFlow, for HPC ptychographic reconstruction by Nashed et al. [52] or in the creation of a map-reduce framework over MPI, called Trace, for tomographic reconstruction [54].

Because Spark underlies many BDA tools, the performance of Spark for scientific computing has been studied in several works. A study on Kira [55], a flexible and distributed astronomy image processing toolkit using Apache Spark, showed that Spark may be an alternative to an equivalent C program for many-task applications. The performance of a Spark implementation of a classification algorithm in the

domain of High Energy Physics (HEP) was evaluated in [56], showing good scalability, but poor performance was compared with the results of an untuned MPI implementation of the same algorithm.

To overcome the former problems, two main approaches have been proposed: MPI-based framework implementations, using MPI as the communication engine without re-implementing the framework.

There are some implementations of map-reduce frameworks using MPI. Plimpton et al [74] created a parallel library written with message-passing (MPI) calls that allows algorithms to be expressed in the map-reduce paradigm, simplifying programming by using map and reduce operations callable from C++, C, Fortran, or scripting languages such as Python . Wang et al. [75] proposed a map-reduce-like framework, called Smart, to execute data analytics algorithms online alongside computational simulations (in situ analytics) in time-sharing or space-sharing modes. A more recent map-reduce framework over MPI is Mimir [76], which provides a redesign of the execution model with optimisation techniques to increase performance and to reduce memory usage, thus increasing scalability to allow significantly larger problems to be executed. Another variant is FT-MRMPI [77], an extension to provide a fault tolerant map-reduce framework on MPI for HPC clusters. The main limitation of these solutions is that significant reimplementation effort is required to modify tools, libraries and applications to use these frameworks, which can impede adoption.

Due to the aforementioned limitations, executing the Spark framework using MPI as the communication engine is becoming the most feasible way to bridge the gap between HPC and BDA frameworks. This approach allows users to benefit from efficient MPI libraries—such as DIY and others—in Spark with little effort on their parts. In [62], Liang and Lu proposed an event-driven pipeline and in-memory shuffle using DataMPI-Iteration, showing a speedup of 9X - 21X over Apache Hadoop, and 2X - 3X over Apache Spark for PageRank and k-means clustering. Anderson et al. [63] proposed a system for integrating MPI with Spark by offloading computation to an MPI environment from within Spark. The evaluation made with four distributed graph and machine learning applications shows speedups between 3X and 17X. Alchemist [78] is another effort in this direction, focusing on the ability to call MPI-based libraries from Spark. Using Alchemist with Spark helps accelerate linear algebra, machine learning, and related computations, while still retaining the benefits of working within the Spark environment. The authors report speed-up of up to 8x for select application such as SVD (singular value decomposition).

2.3.3 Storage converge between BD and HPC platforms

Scientific workflows are composed of heterogeneous and coupled components that simulate different aspects of the domain they model [26]. These modules interact

and exchange significant volumes of data at runtime, hence making these transfers efficient has a potential major impact in the overall performance of the resulting application [79]. As a consequence, both the storage infrastructure and the logical file system abstractions could affect performance and scalability, thus making data management a key aspect in workflow design and implementation [80].

In order to support the degree of scalability and performance required by modern simulators, one of the key elements to take into consideration in a data-centric transformation procedure is the avoidance of I/O bottlenecks [81]. Given the workflow nature of many state-of-the-art simulators for scientific computing, Srirama et al. [82] proposed a workflow-partitioning strategy to reduce the data communication in the resulting deployment. The aim of this work is similar to ours: migrating scientific workflows to the cloud with a focus on data locality. However, it focuses on task scheduling and the underlying peer-to-peer data-sharing mechanisms required for their efficient communication, while we attempt to provide data locality by design in the resulting applications.

A first attempt to optimise map-reduce storage on HPC clusters by utilising Lustre as the storage provider for RDMA intermediate data was presented in [83]. Matri et al. [84] analyses the applicability of blobs (binary large objects) and object storage systems to solve the problems with POSIX-IO-compliant file systems and as a mechanism to replace distributed file systems for BDA analytics. Their characterisation of the I/O profiles of several HPC and BDA applications shows that most of these workloads execute read/write operations to files, with little need of metadata. Sherish et al. recently showed in [85] how BDA tools can be used for HEP data analysis because extremely large HEP datasets can be represented and held in memory across the system and accessed interactively by encoding an analysis using the high-level programming abstractions in Spark.

BDA analytics tools –like Hadoop and Spark– are being explored to provide straightforward data distribution and caching mechanisms in data-intensive HPC applications. Their data-centric nature permits reasoning about tasks over distributed data abstractions without worrying about task scheduling, which is managed by the middleware to enforce data locality and minimise transfers. The inherent parallelism of these tools has resulted in positive experimental results showing their suitability for massively parallel workloads like MTC-like workflows [86]. Nevertheless, challenges remain with respect to workflows built with a pure HPC focus, which rely on MPI and traditional storage infrastructures [87].

These tools have already been used to create workflow execution engines and scientific workflow management systems (SWfMS) for current state-of-the-art workflows [86, 88]. However, this topic is still fairly new and the experience with applying these techniques is still limited. Previous works have contributed with guidelines and methodological approaches to make the design of scientific workflows easier and more efficient, with a user-centric and visual perspective [89]. A theoretical analysis of migrating common HPC-oriented workflows to

a BD processing platform (i.e., Apache Hadoop) was made in [90]. Authors implemented six representatives of common scientific workflow patterns in Apache Hadoop environment and discussed implementation challenges as well as Hadoop environment applicability for each of the basic patterns.

Chapter 3

Generic Parallel Pattern Interface

In this section we introduce GrPPI, a generic and reusable parallel pattern interface for C++ applications [91]. This interface takes full advantage of modern C++ features, generic programming, and metaprogramming concepts to act as common interface between different programming models hiding away the complexity behind the use of concurrency mechanisms. Furthermore, the modularity of GrPPI allows to easily integrate new patterns, as well as to combine them to arrange more complex constructions. Thanks to this property, GrPPI can be used to implement a wide range of existing stream-processing and data-intensive applications with relative small efforts, having as a result portable codes that can be executed on multiple platforms.

Basically, this library provides three main components in order to provide a unified interface for the supported frameworks: *i)* pattern classes and interfaces; *ii)* execution policies and *iii)* communication channels.

Firstly, GrPPI provides a set of classes that represent each of the supported patterns in order to allow compositions. These objects store references to the necessary functions and information (e.g. concurrency degree) related to the pattern configuration. Thus, they can be composed in order to express complex constructions that can not be represented by making use of a single pattern. Additionally, GrPPI provides a set of functions for each supported parallel pattern and offers two different alternatives for pattern execution and composition. Thanks to these interfaces, pattern constructions can be represented using a functional programming way. In this project, we plan to extend these interfaces for supporting new parallel patterns such as map-by-key.

On the other hand, key point of GrPPI is the ability to easily switch between different programming framework implementations for a given pattern. This is achieved

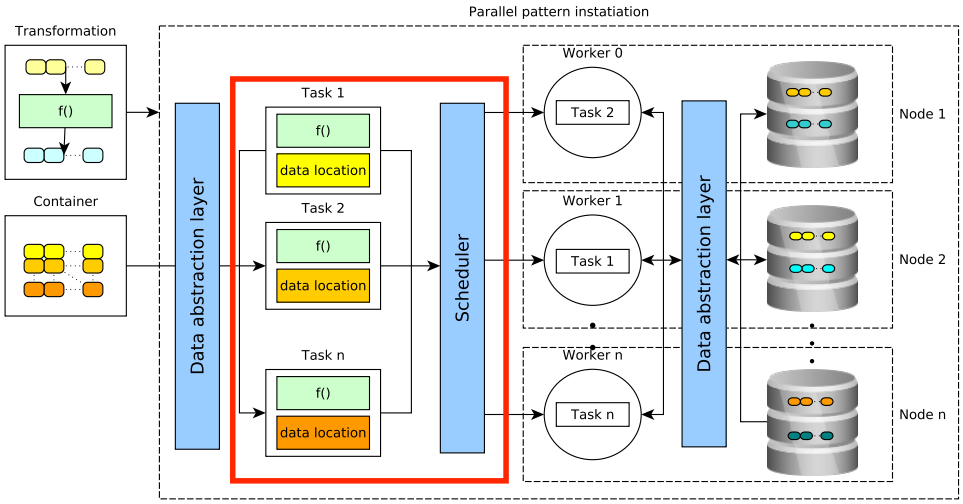

Figure 3.1: Task-generation and scheduling schema. Red box depicts the working progress on tasks generation under GrPPI.

by providing a set classes that encapsulate the actual pattern implementations in a specific framework. This way, by only changing the execution policy provided as first argument to the pattern call, the framework used underneath is selected accordingly. The current GrPPI version provides support for sequential, C++ threads, OpenMP, Intel TBB, FastFlow and MPI frameworks. In this project, we plan to provide a task-distributed backend for data-intensive applications.

Finally, GrPPI provides a set communications channels for transfering data among workers. In the current version, GrPPI supports different First-In-First-Out (FIFO) queues for multiple-producer/multiple consumers in shared memory, using atomics and locks, and distributed memory, using MPI one-sided and two-sided primitives. In this project, we plan to extend these communications channels for supporting data and task communications using two different existing protocols MPI and ZeroMQ.

Figure 3.1 depicts task generation and scheduling schema of the proposed GrPPI task-based backend and highlights the current efforts. Basically, we have implemented a first version that is able to generate the task corresponding to a pattern composition for shared-memory platform. Afterwards, tasks are processed by a set of worker entities using a FIFO scheduler. Currently, we are working on the new communication channels using ZeroMQ for communicating tasks in a distributed environment.

3.1 Repository

The current release version can be found in https://github.com/arcosuc3m/grppi and the prototype task-based backend in the GrPPI "task-backend" branch.

3.2 Requirements

GrPPI is a header-only library that only requires a C++14 compliant compiler. GrPPI has been tested with the following compilers and versions:

- g++: 6.3, 7.2.
- clang++: 3.9, 4.0, 5.0.

However, even if there is no mandatory need for external libraries, some additional libraries are needed for using their corresponding GrPPI backends.

- Intel Threading Building Blocks (TBB).
- FastFlow.
- Boost v1.68 and OpenMPI v3.1.2.

Chapter 4

ASPIDE Gateway

In this chapter, we introduce the ASPIDE Gateway, a collection of components designed to handle application composition and orchestration. In order to facilitate the configuration and accelerate application development and tuning for the user inexperienced with exascale systems configuration, this component will allow for visual representation and composition of tasks, data, and infrastructure for an application.

The software is built around the open-source platform Alien4Cloud [1], which provides a set of core services:

- a REST API for storing TOSCA [92] specifications
- a visual interface for composing applications based on service requirements and exposed interfaces
- a plugin interface for additional functionalities such as Orchestrators and UI elements.

The TOSCA files will provide the basis for task composition and scheduling policies, defining requirements and relationships between DCEx entities. Using the Drag and Drop Interface, one is able to link together tasks, data and computing infrastructure into an Application and experiment with different configurations for the parameters defined in the specification.

The Orchestrator resolves task dependencies and communicates with several components of the DCEx platform for the purpose of scheduling. It is then responsible to check the status of running tasks and start pending tasks as soon as their inputs become available.

Several UI interfaces need to be added in to present the user with information gathered by the tools provided by the ASPIDE project, such as the auto-tuning tool

[1] https://http://alien4cloud.github.io

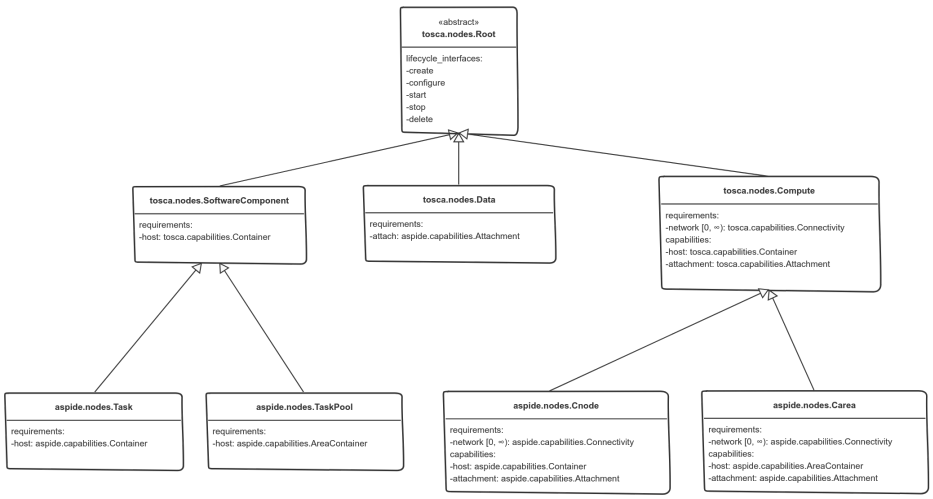

Figure 4.1: Class diagram for DCEx programming model and relation with TOSCA normative types

E-TUNER.

4.1 Programming Model Specification

TOSCA specifications define application component requirements, capabilities, the relationships between them, as well as defining attributes that will be filled during the lifecycle of a component. The root TOSCA node defines five lifecycle interfaces for the existence of any component: create, configure, start, stop and delete. All other components are derived from this and therefore inherit the lifecycle interfaces. TOSCA specifications files are developed for the DCEx programming model: *Cnode*, *Carea*, *Data*, *Task*, and *TaskPool*. A class diagram is presented in Figure 4.1 to show the relationship between the DCEx constructs and the TOSCA normative types.

The *Cnode* and *Carea* components are derived from `tosca.nodes.Compute` and inherit its requirements (i.e., network, which may be absent), and capabilities: `host` (other components can be hosted on it) and **attachment** (used for attaching storage requirements). The two types of nodes will have different implementations for those capabilities.

The *Data* component is derived from the root TOSCA node and imposes a requirement which can be fulfilled by the `attachment` capability of a Cnode or Carea. The `DCExAttachTo` relationship between the Data component and Com-

pute component defines the replication or partition strategies. Additionally, the Data component will have a `consume` endpoint, which the task can use to gain information about data type, path, and others.

The *Task* component is derived from `tosca.nodes.SoftwareComponent` and inherits the requirement to be hosted. This requirement will be matched by any further derivation of Cnode. The *TaskPool* is derived from Task and requires to be hosted on a Carea. Additionally, it has a property which can be set by the user defining the pool size. These two components can be instantiated through the standard lifecycle interfaces, providing scripts for each state.

Subsequent definitions will be derived from Task and TaskPool, enhancing them with further capabilities. Several *GrPPITask*s will have the capability to be pipelined together with others hosted on the same Cnode or Carea. This, in turn, will generate a single GrPPI pipeline application, taking advantage of the underlying communication mechanism. Similarly, a *StreamTask* will expose an endpoint for output data and will require a relationship with a `consume` endpoint of a Data or Task component.

4.2 Plugins

The **DCEx Orchestrator** plugin contains the logic for decomposing a TOSCA application in its task and data components and apply the lifecycle stages corresponding to dependencies between tasks, data, and computing infrastructure. In order to achieve this it will make use of:

- the monitoring APIs to determine the state of a given task
- the data management APIs to query data state, request data transfers, and improve scheduling decisions
- the scheduling runtime APIs in order to start/stop/clear a Task or TaskPool.

Several **UI plugins** will enhance the development of Exascale Applications presenting the operator with insights gathered from the Event Detection System and Decision Support System, part of the E-TUNER component. Additional plugins will be implemented in order to present the analytics obtained from monitoring components.

4.3 Repository

The ASPIDE Gateway consists of a set of Java Components and TOSCA specifications, bundled as git submodules. The current development version can be found at `https://github.com/adispataru/aspide-gateway`

4.4 Requirements

The components require Java, version 1.8 or above. Maven is used for managing external library dependencies and is therefore required to compile the software. A binary version will be provided as well.

The visual interface can be accessed using a Web Browser, but all actions can also be performed by interacting with the REST API.

The Orchestrator and custom UI plugins need to have access to the core ASPIDE services for monitoring and scheduling applications.

Chapter 5

Conclusions

In this chapter, we have proposed a generic and reusable parallel pattern interface that acts as a switch between existing frameworks. This way, developers can benfit from this interface to diminish the required efforts to select the most suitable framework and to migrate parallel code from one framework to another.

Bibliography

[1] Doug Laney. 3d data management: Controlling data volume, velocity and variety. *META group research note*, 6(70):1, 2001.

[2] Paul Zikopoulos, Chris Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[3] Philip Russom et al. Big data analytics. *TDWI best practices report, fourth quarter*, 19(4):1–34, 2011.

[4] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable real-time data systems*. New York; Manning Publications Co., 2015.

[5] Big Data Value Association. European big data value strategic research and innovation agenda. Technical report, October 2017.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] Joerg Fritsch and Coral Walker. The problem with data. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 708–713. IEEE, 2014.

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. 37(5):29–43, 2003.

[9] Tom White. *Hadoop: The Definitive Guide: The Definitive Guide*. O'Reilly Media, 2009.

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

[11] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D Stott Parker. Mapreduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040. ACM, 2007.

[12] Radu Tudoran, Alexandru Costan, and Gabriel Antoniu. Mapiterativereduce: a framework for reduction-intensive data processing on azure clouds. In *Proceedings of third international workshop on MapReduce and its Applications Date*, pages 9–16. ACM, 2012.

[13] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[14] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D Ernst. Haloop: efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment*, 3(1-2):285–296, 2010.

[15] Thilina Gunarathne, Bingjing Zhang, Tak-Lon Wu, and Judy Qiu. Scalable parallel computing on clouds using twister4azure iterative mapreduce. *Future Generation Computer Systems*, 29(4):1035–1048, 2013.

[16] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: A runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM.

[17] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010.

[18] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion 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*, pages 2–2. USENIX Association, 2012.

[19] X. Shi, M. Chen, L. He, X. Xie, L. Lu, H. Jin, Y. Chen, and S. Wu. Mammoth: Gearing hadoop towards memory-intensive mapreduce applications. *IEEE Transactions on Parallel and Distributed Systems*, 26(8):2300–2315, Aug 2015.

[20] Ionel Gog, Jana Giceva, Malte Schwarzkopf, Kapil Vaswani, Dimitrios Vytiniotis, Ganesan Ramalingan, Manuel Costa, Derek Murray, Steven Hand, and Michael Isard. Broom: Sweeping out garbage collection from big data systems. *Young*, 4:8, 2015.

[21] Luca Salucci, Daniele Bonetta, and Walter Binder. Lightweight multi-language bindings for apache spark. In *Proceedings of the 22Nd International Confer-*

ence on Euro-Par 2016: Parallel Processing - Volume 9833, pages 281–292, New York, NY, USA, 2016. Springer-Verlag New York, Inc.

[22] Ahsan Javed Awan, Mats Brorsson, Vladimir Vlassov, and Eduard Ayguade. *How Data Volume Affects Spark Based Data Analytics on a Scale-up Server*, pages 81–92. Springer International Publishing, Cham, 2016.

[23] Fan Zhang, Ciprian Docan, Manish Parashar, Scott Klasky, Norbert Podhorszki, and Hasan Abbasi. Enabling in-situ execution of coupled scientific workflow on multi-core platform. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 1352–1363. IEEE, 2012.

[24] Fan Zhang, Qutaibah M Malluhi, Tamer Elsayed, Samee U Khan, Keqin Li, and Albert Y Zomaya. Cloudflow: A data-aware programming model for cloud workflow applications on modern hpc systems. *Future Generation Computer Systems*, 2014.

[25] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.

[26] Domenico Talia, Paolo Trunfio, and Fabrizio Marozzo. *Data analysis in the cloud: models, techniques and applications*. Elsevier, 2015.

[27] Daniel Warneke and Odej Kao. Nephele: efficient parallel data processing in the cloud. In *Proceedings of the 2nd workshop on many-task computing on grids and supercomputers*, page 8. ACM, 2009.

[28] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[29] Ciprian Dobre and Fatos Xhafa. Parallel programming paradigms and frameworks in big data era. *International Journal of Parallel Programming*, 42(5):710–738, 2014.

[30] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Hive-a petabyte scale data warehouse using hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 996–1005. IEEE, 2010.

[31] Svilen R Mihaylov, Zachary G Ives, and Sudipto Guha. Rex: recursive, delta-based data-centric computation. *Proceedings of the VLDB Endowment*, 5(11):1280–1291, 2012.

[32] A. Al-Badarneh, H. Najadat, M. Al-Soud, and R. Mosaid. Phoenix: A mapreduce implementation with new enhancements. In *2016 7th International Conference on Computer Science and Information Technology (CSIT)*, pages 1–5, July 2016.

[33] Cliff Engle, Antonio Lupher, Reynold Xin, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: Fast data analysis using coarse-grained distributed memory. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 689–692, New York, NY, USA, 2012. ACM.

[34] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[35] Russell Power and Jinyang Li. Piccolo: Building fast, distributed programs with partitioned tables. In *OSDI*, volume 10, pages 1–14, 2010.

[36] Avraham Shinnar, David Cunningham, Vijay Saraswat, and Benjamin Herta. M3r: Increased performance for in-memory hadoop jobs. *Proc. VLDB Endow.*, 5(12):1736–1747, August 2012.

[37] Vinayak Borkar, Michael Carey, Raman Grover, Nicola Onose, and Rares Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 1151–1162, Washington, DC, USA, 2011. IEEE Computer Society.

[38] Katherine Yelick, Susan Coghlan, Brent Draney, Richard Shane Canon, et al. The magellan report on cloud computing for science. *US Department of Energy, Washington DC, USA, Tech. Rep*, 2011.

[39] Ioan Raicu, Ian T Foster, and Pete Beckman. Making a case for distributed file systems at exascale. In *Proceedings of the third international workshop on Large-scale system and application performance*, pages 11–18. ACM, 2011.

[40] Dhruba Borthakur. Hdfs architecture guide. *Hadoop Apache Project*, page 53, 2008.

[41] Daniel A Reed and Jack Dongarra. Exascale computing and big data. *Communications of the ACM*, 58(7):56–68, 2015.

[42] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *International Conference on High Performance Computing for Computational Science*, pages 1–25. Springer, 2010.

[43] R. Rabenseifner, G. Hager, and G. Jost. Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In *2009 17th Euromicro*

International Conference on Parallel, Distributed and Network-based Processing, pages 427–436, Feb 2009.

[44] Pablo D. Mininni, Duane Rosenberg, Raghu Reddy, and Annick Pouquet. A hybrid mpi–openmp scheme for scalable parallel pseudospectral computations for fluid turbulence. *Parallel Computing*, 37(6):316 – 326, 2011.

[45] Suchuan Dong and George Em Karniadakis. Dual-level parallelism for high-order cfd methods. *Parallel Computing*, 30(1):1 – 20, 2004.

[46] J. Guan, S. Yan, and J. Jin. An openmp-cuda implementation of multilevel fast multipole algorithm for electromagnetic simulation on multi-gpu computing systems. *IEEE Transactions on Antennas and Propagation*, 61(7):3607–3616, July 2013.

[47] P.S. Rakić, D.D. Milašinović, Ž. Živanov, Z. Suvajdžin, M. Nikolić, and M. Hajduković. Mpi–cuda parallelization of a finite-strip program for geometric nonlinear analysis: A hybrid approach. *Advances in Engineering Software*, 42(5):273 – 285, 2011. PARENG 2009.

[48] S. J. Pennycook, S. D. Hammond, S. A. Jarvis, and G. R. Mudalige. Performance analysis of a hybrid mpi/cuda implementation of the naslu benchmark. *SIGMETRICS Perform. Eval. Rev.*, 38(4):23–29, March 2011.

[49] M. U. Ashraf, F. Alburaei Eassa, A. Ahmad Albeshri, and A. Algarni. Performance and power efficient massive parallel computational model for hpc heterogeneous exascale systems. *IEEE Access*, 6:23095–23107, 2018.

[50] N. Malitsky, A. Chaudhary, S. Jourdain, M. Cowan, P. O'Leary, M. Hanwell, and K. K. Van Dam. Building near-real-time processing pipelines with the spark-mpi platform. In *2017 New York Scientific Data Summit (NYSDS)*, pages 1–8, Aug 2017.

[51] Isaac Lopez. Idc talks convergence in high performance data analysis, 2013. *URL: http://www. datanami. com/2013/06/19/idc_talks_convergence_in_high_performance_data_analysis/(visited on 02/14/2016)*, 2013.

[52] Youssef SG Nashed, Tom Peterka, Junjing Deng, and Chris Jacobsen. Distributed automatic differentiation for ptychography. *Procedia Computer Science*, 108:404–414, 2017.

[53] Youssef SG Nashed, David J Vine, Tom Peterka, Junjing Deng, Rob Ross, and Chris Jacobsen. Parallel ptychographic reconstruction. *Optics express*, 22(26):32082–32097, 2014.

[54] Tekin Bicer, Doğa Gürsoy, Vincent De Andrade, Rajkumar Kettimuthu, William Scullin, Francesco De Carlo, and Ian T. Foster. Trace: a high-throughput tomographic reconstruction engine for large-scale datasets. *Advanced Structural and Chemical Imaging*, 3(1):6, Jan 2017.

[55] Z. Zhang, K. Barbary, F. A. Nothaft, E. Sparks, O. Zahn, M. J. Franklin, D. A. Patterson, and S. Perlmutter. Scientific computing meets big data technology: An astronomy use case. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 918–927, Oct 2015.

[56] Saba Sehrish, Jim Kowalkowski, and Marc Paterno. Exploring the performance of spark for a scientific use case. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 1653–1659. IEEE, 2016.

[57] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, Ph. Canal, D. Casadei, O. Couet, V. Fine, L. Franco, G. Ganis, A. Gheata, D. Gonzalez Maline, M. Goto, J. Iwaszkiewicz, A. Kreshuk, D. Marcos Segura, R. Maunder, L. Moneta, A. Naumann, E. Offermann, V. Onuchin, S. Panacek, F. Rademakers, P. Russo, and M. Tadel. Root — a c++ framework for petabyte data storage, statistical analysis and visualization. *Computer Physics Communications*, 180(12):2499–2512, 2009.

[58] X. Lu, M. W. U. Rahman, N. Islam, D. Shankar, and D. K. Panda. Accelerating spark with rdma for big data processing: Early experiences. In *2014 IEEE 22nd Annual Symposium on High-Performance Interconnects*, pages 9–16, Aug 2014.

[59] M. Lu, Y. Liang, H. P. Huynh, Z. Ong, B. He, and R. S. M. Goh. Mrphi: An optimized mapreduce framework on intel xeon phi coprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 26(11):3066–3078, Nov 2015.

[60] IBM. GPU Enabler for Spark, 2017.

[61] James Fox, Yiming Zou, and Judy Qiu. Software frameworks for deep learning at scale. *Internal Indiana University Technical Report*, 2016.

[62] Fan Liang and Xiaoyi Lu. Accelerating iterative big data computing through mpi. *Journal of Computer Science and Technology*, 30(2):283–294, 2015.

[63] Michael Anderson, Shaden Smith, Narayanan Sundaram, Mihai Capotă, Zheguang Zhao, Subramanya Dulloor, Nadathur Satish, and Theodore L Willke. Bridging the gap between hpc and big data frameworks. *Proceedings of the VLDB Endowment*, 10(8):901–912, 2017.

[64] Y. Zhao, X. Fei, I. Raicu, and S. Lu. Opportunities and challenges in running scientific workflows on the cloud. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*, pages 455–462, Oct 2011.

[65] G. Lin, B. Han, J. Yin, and I. Gorton. Exploring cloud computing for large-scale scientific applications. In *2013 IEEE Ninth World Congress on Services*, pages 37–43, June 2013.

[66] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: The montage example. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press.

[67] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the use of cloud computing for scientific workflows. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 640–645, Dec 2008.

[68] G. Bruce Berriman, Ewa Deelman, Gideon Juve, Mats Rynge, and Jens-S. Vöckler. The application of cloud computing to scientific workflows: a study of cost and performance. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 371(1983), 2012.

[69] Carlos Maltzahn, Esteban Molina-Estolano, Amandeep Khurana, Alex J Nelson, Scott A Brandt, and Sage Weil. Ceph as a scalable alternative to the hadoop distributed file system. *login: The USENIX Magazine*, 35:38–49, 2010.

[70] Rajagopal Ananthanarayanan, Karan Gupta, Prashant Pandey, Himabindu Pucha, Prasenjit Sarkar, Mansi Shah, and Renu Tewari. Cloud analytics: Do we really need to reinvent the storage stack? In *HotCloud*, 2009.

[71] Y. Wang, R. Goldstone, W. Yu, and T. Wang. Characterization and optimization of memory-resident mapreduce on hpc systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 799–808, May 2014.

[72] Constantinos Evangelinos and C Hill. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. *ratio*, 2(2.40):2–34, 2008.

[73] A. Gupta and D. Milojicic. Evaluation of hpc applications on cloud. In *2011 Sixth Open Cirrus Summit*, pages 22–26, Oct 2011.

[74] Steven J. Plimpton and Karen D. Devine. Mapreduce in mpi for large-scale graph algorithms. *Parallel Comput.*, 37(9):610–632, September 2011.

[75] Yi Wang, Gagan Agrawal, Tekin Bicer, and Wei Jiang. Smart: A mapreduce-like framework for in-situ scientific analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 51. ACM, 2015.

[76] T. Gao, Y. Guo, B. Zhang, P. Cicotti, Y. Lu, P. Balaji, and M. Taufer. Mimir: Memory-efficient and scalable mapreduce for large supercomputing systems. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1098–1108, May 2017.

[77] Yanfei Guo, Wesley Bland, Pavan Balaji, and Xiaobo Zhou. Fault tolerant mapreduce-mpi for hpc clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 34:1–34:12, New York, NY, USA, 2015. ACM.

[78] Alex Gittens, Kai Rothauge, Shusen Wang, Michael Mahoney, Lisa Gerhardt, Prabhat, Jey Kottalam, Michael Ringenburg, and Kristyn Maschhoff. Accelerating large-scale data analysis by offloading to high-performance computing libraries using alchemist. In *SIGKDD'18: 24th ACM International Conference on Knowledge Discovery and Data Mining*, London, UK, 2018.

[79] Zhuoyao Zhang. Processing data-intensive work ows in the cloud, 2012.

[80] K. Vahi, M. Rynge, G. Juve, R. Mayani, and E. Deelman. Rethinking data management for big data scientific workflows. In *Big Data, 2013 IEEE International Conference on*, pages 27–35, Oct 2013.

[81] V. Nuthula and N. R. Challa. Cloudifying apps - a study of design and architectural considerations for developing cloudenabled applications with case study. In *Cloud Computing in Emerging Markets (CCEM), 2014 IEEE International Conference on*, pages 1–7, Oct 2014.

[82] Satish Narayana Srirama and Jaagup Viil. Migrating scientific workflows to the cloud: Through graph-partitioning, scheduling and peer-to-peer data sharing. In *High Performance Computing and Communications, 2014 IEEE Intl Conf on*, pages 1105–1112. IEEE, 2014.

[83] M. Wasi ur Rahman, X. Lu, N. S. Islam, R. Rajachandrasekar, and D. K. Panda. High-performance design of yarn mapreduce on modern hpc clusters with lustre and rdma. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 291–300, May 2015.

[84] Pierre Matri, Yevhen Alforov, Alvaro Brandon, Michael Kuhn, Philip Carns, and Thomas Ludwig. Could blobs fuel storage-based convergence between hpc and big data? In *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*, pages 81–86. IEEE, 2017.

[85] Saba Sehrish, Jim Kowalkowski, and Marc Paterno. Spark and hpc for high energy physics data analyses. In *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*, pages 1048–1057. IEEE, 2017.

[86] Z. Zhang, K. Barbary, F. A. Nothaft, E. Sparks, O. Zahn, M. J. Franklin, D. A. Patterson, and S. Perlmutter. Scientific computing meets big data technology: An astronomy use case. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 918–927, Oct 2015.

[87] Andre Luckow, Pradeep Mantha, and Shantenu Jha. Pilot-abstraction: A valid abstraction for data-intensive applications on hpc, hadoop and cloud infrastructures? *arXiv preprint arXiv:1501.05041*, 2015.

[88] Marc Bux, Jörgen Brandt, Carsten Lipka, Kamal Hakimzadeh, Jim Dowling, and Ulf Leser. Saasfee: Scalable scientific workflow execution engine. *Proc. VLDB Endow.*, 8(12):1892–1895, August 2015.

[89] R. Etemadpour, M. Bomhoff, E. Lyons, P. Murray, and A. Forbes. Designing and evaluating scientific workflows for big data interactions. In *Big Data Visual Analytics (BDVA), 2015*, pages 1–8, Sept 2015.

[90] Elif Dede, Madhusudhan Govindaraju, Daniel Gunter, and Lavanya Ramakrishnan. Riding the elephant: Managing ensembles with hadoop. In *Proceedings of the 2011 ACM International Workshop on Many Task Computing on Grids and Supercomputers*, MTAGS '11, pages 49–58, New York, NY, USA, 2011. ACM.

[91] David del Rio Astorga, Manuel F. Dolz, Javier Fernández, and J. Daniel García. A generic parallel pattern interface for stream and data processing. *Concurrency and Computation: Practice and Experience*, Online:e4175–n/a, April 2017.

[92] Tobias Binz, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. Tosca: portable automated deployment and management of cloud applications. In *Advanced Web Services*, pages 527–549. Springer, 2014.