CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2012; **00**:1–18
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Pilot-Abstractions for Data-Intensive Cloud Applications

Andre Luckow, Pradeep Mantha, Melissa Romanus, Shantenu Jha*

Radical Research Group, Rutgers University

SUMMARY

The data generated by scientific applications is experiencing an exponential growth. The ability to analyze prodigious volumes of data requires flexible and efficient compute-data placement strategies. This paper explores and establishes the use of Pilot-Abstractions as a runtime capability that supports a range of applications and analytical methods in a flexible and scalable fashion over a variety of execution environments. In particular, we focus on Pilot-Data and the use of clouds as one element in a broader and heterogeneous distributed cyberinfrastructure. To that end, we build upon existing capabilities of integrating MapReduce with Pilot-Abstractions, called Pilot-MapReduce (PMR) which provides a flexible, infrastructure-independent runtime environment for MapReduce. Pilot-Jobs are used to couple the map phase computation to the nearby source data, and Pilot-Data is used to move intermediate data using parallel data transfers to the reduce computation phase. In this paper, we discuss the generalization of the Pilot-Abstraction to clouds and investigate the use of PMR on a variety of cloud systems, as well as interoperably with other distributed resources. We show how using Pilot-Abstractions, PMR enables the efficient processing of distributed data on heterogeneous distributed infrastructure including different academic and commercial clouds (e.g. Amazon EC2, Google Compute Engine and FutureGrid). We investigate the effectiveness of PMR-based applications on cloud infrastructure for next-generation gene sequencing applications. The flexible runtime provided by the Pilot-Abstraction, helped establish PMR's effectiveness for distributed data scenarios on clusters that are nodes on a WAN. Given the different network characteristics within clouds, we investigate whether performance determinants for clusters, are valid for clouds. Our analysis covers multiple scenarios exposing the different (typical) trade-offs: e.g., the overhead times in spawning virtual machines, different storage types, geographic distribution, and establishes that the Pilots provide a powerful abstraction for clouds as well. Copyright © 2012 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Cloud Computing, MapReduce, Grid Computing, Data-Intensive

1. INTRODUCTION

An important trend is the emergence of large-volume of data, rich in semantics, complexity and information. One attribute, often overlooked is that the prodigious amounts of data is fundamentally distributed – often due to source of production or location of consumption/analysis. In general, distributed data has become a critical factor in many science disciplines [1], e. g. in the areas of fusion energy (ITER), bioinformatics (metagenomics), climate (Earth System Grid), and astronomy (LSST) [2]. In addition to efficient algorithms and methods, an increasing number of these applications need to effectively utilize a diverse range of infrastructure, including clouds in addition to traditional cyberinfrastructure. Furthermore, these infrastructure are distributed and even though they are still evolving they are often disjoint.

Interestingly even though clouds have their origins in enterprise computing, they provide novel opportunities for science & engineering applications. As has been extensively documented [3],

*Correspondence to: Journals Production Department, John Wiley & Sons, Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, UK.

clouds offer a relatively simple, easy-to-use environment with respect to resource management, capacity planning capabilities, software environment & control etc. An interesting capability of clouds is the possibility to bootstrap custom runtime environments, which is often difficult with grids. For example, a number of applications are difficult to build, due to runtime dependencies, or complicated non-portable build systems. This provides an additional rationale for cloud environments. After an application environment is created once, it can be loaded on to various systems, working around issues related to portability on the physical systems. Furthermore, clouds provide greater scheduling flexibility, for example, when the set of resources needed to run an application changes (perhaps rapidly), the resources employed can actually be changed (new resources can be added, or existing resources can be removed from the pool used by the job).

We believe there are three primary architectures for data-intensive analytics: (i) localize all data into a cloud (or a large analytical engine), (ii) decompose and distribute data to as many computing/analytical engines are available (this incidentally is the paradigm employed by particle physics for the discovery of the Higgs), and (iii) a hybrid and hierarchy of the above two paradigms, wherein data is decomposed and committed to several infrastructure, which in turn could be a combination of either of the first two paradigms. Although it is obvious that large-volumes of data can be poured into a cloud, there are limitations to the scalability or validity of this model, viz., what if data-volumes are too large to move, or have constraints on the ability to move centrally (say due to security or privacy concerns). In other words, can clouds also be used for the second and third architectural paradigms, and if so, how? This is not unrelated to runtime capabilities which enable applications to utilize any of the three architectures as needed (viz., based upon data size and resource availability).

Even whilst the macroscopic architecture for data-intensive computing and the relationship of clouds to other elements of the distributed cyberinfrastructure is still converging, there is churn within clouds, i.e., the cloud capability and infrastructure landscape has become highly fragmented. Amidst this fragmentation, the case for cloud interoperability and cloud-grid interoperability can be made. How can such interoperability be provided in an extensible and flexible fashion? The answer is partly dependent upon the level and type of interoperability (application-level, service-level etc) desired, but one route to interoperability definitely arises from a unified model of runtime execution. Are there abstractions that can provide such unification?

The high-level motivating question for this work thus emerges: How can clouds be used to support the requirements of distributed data-intensive applications? Whereas any answer to the above, is bound to be complex and constrained, we concentrate on two components: (i) architecture of the data-cyberinfrastructure and (ii) runtime environments. Thus, the motivating question can be positioned as, are there runtime capabilities that provide the abstractions required for flexible execution whilst supporting different data-analytic architectural paradigms? We posit that Pilot-Abstractions provide precisely such a unification of the runtime/execution model with the architectural flexibility required and thus can meet the requirements of data-intensive applications.

Pilot-Jobs have proven to be a successful abstraction for distributed and high-performance applications, whereby they decouple the workload specification from its execution. A Pilot-Job provides the ability to utilize a placeholder job as a container for a dynamically determined set of compute tasks. Recent work on the P* (Pstar) Model [4] has provided a formal theoretical basis for Pilot-Jobs and provided a conceptual framework upon which to analyze and compare distinct implementations. Significantly, the P* Model provided a symmetrical but logical extension to Pilot-Jobs to data, and introduced the concept of Pilot-Data, which akin to Pilot-Jobs for computational tasks, provides an efficient and extensible route to decoupling the ultimate location of data storage and/or consumption from its production.

We have previously discussed the effectiveness of Pilot-Abstractions for the second architectural paradigm [4]. A primary contribution of this work is in establishing Pilot-Abstractions as a valid resource and runtime abstraction for data-intensive applications for clouds for all three architectural paradigms. Traditionally compute-data cyberinfrastructure was integrated, or at least the data cyberinfrastrucutre (such as storage, access mechanisms etc.) were pretty much determined and constrained by the compute environment. Probably in response to increasing data volumes and

DOI: 10.1002/cpe

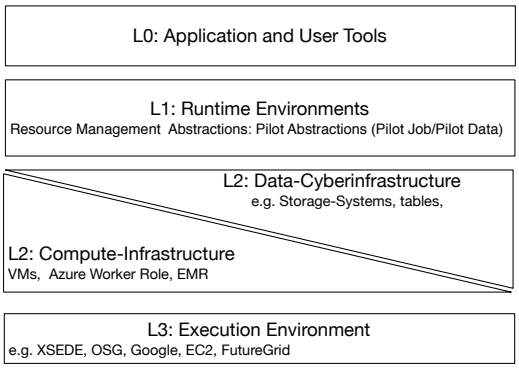

Figure 1. **Pilot-Abstraction and Data-Cyberinfrastructure:** High-level schematic of how different components referred to in the paper relate to each other, and in particular how Pilot-Abstractions place within a broader data-cyberinfrastructure.

challenges inherent, a recent trend is a decoupling between the two, i. e., data-cyberinfrastructure is often an independent degree-of-freedom and importantly often the primary degree-of-freedom, i.e., data-cyberinfrastructure considerations come together if not before compute considerations. This is schematically represented in the level diagram by a logical separation of the same level (L2) in Figure 1. Thus it is important that any effective runtime abstraction be able to support a diverse range of data-cyberinfrastucture and data localization models imposed by them. In this work, we establish that Pilot-Abstractions provide a common runtime/execution model across the range of proliferating compute and data-cyberinfrastructure. Furthermore, we illustrates an interoperable and scalable approaches to providing dynamic resource utilization (often loosely called "elastic resource utilization").

After discussing relevant and prior work in the next section, we present a brief discussion of the landscape of elements comprising cloud cyberinfrastructure. Any effective runtime abstraction must be able to support applications against the rich and dynamic landscape. Section 4 provides a conceptual extension of the Pilot-Abstraction to clouds, we propose that it provides a runtime abstraction which supports interoperability across a diverse and dynamic cyberinfrastructure landscape. Section 4 also provides an overview of Pilot-Data and affinity, and how these concepts apply to cloud infrastructure. Section 5 discusses how the Pilot-Abstraction has been used to support flexible and extensible implementations of MapReduce – a well-known pattern/programing model for data-intensive applications. The integration of MapReduce and Pilot-Abstraction, referred to as PilotMapReduce, paves the way for uniform execution environment for grids and clouds, and thereby sets the stage for common scheduling, localization and optimization techniques. In Section 6, we design and discuss a set of experiments with the aim of establishing Pilot-Abstractions as a runtime abstraction as measured by a set of objectives, viz., validity, interoperability and ability to support application/application-level patterns.

2. RELATED WORK

Related Work: The implementation of pilot jobs has existed on grid infrastructures for some time. Batch queuing systems on the existing grid hardware drove the desire for user-level control of tasks and ease of use of job descriptions for data driven applications. Pilot jobs provide the ability to distribute workload across multiple systems and provide an easy way to schedule many jobs at one time. The first such implementation of a pilot-job was Condor glidein[5]. A glidein daemon can pull Condor binaries to a compute node. This compute node then becomes available in a given Condor pool as a resource. Jobs can then be run on the pool of compute nodes.

Since the notion of pilot jobs saw such promising uptake on grids, it was natural that this work be explored in its extension to clouds. An infrastructure requirement out of CERN drove the creation

DOI: 10.1002/cpe

of CoPilot[6], which acts as a layer on top of existing grid/batch systems to allow job submission to clouds such as Amazon EC2 or Scientific Clouds. The creation of CoPilot was driven from the high energy physics community having requirements for cloud/grid interoprobility. CoPilot uses XMPP messaging to communicate between the VMs and the grid infrastructure. It is based more on the actual submission of jobs and focuses less on user-level control and application-level programming than SAGA BigJob.

The Coaster system[7] is another pilot-job implementation which has been shown to work in both cloud and grid environments. Using the Coaster service, one executes a master on the head node, and the Coaster workers run on compute nodes to execute jobs sounds at least vaguely familiar. Coasters offers a zero-install feature in which it deploys itself and installs itself from the head node and onto the virtual machines without needing any prior installation on the machine. In cloud mode, Coasters does require installation to a head node. While SAGA BigJob uses SAGA to connect to middleware, Coasters uses the CoG kit abstraction library for gram, ssh, Condor, PBS, SGE support. While Coasters has been shown to be effective for clouds, grids, and clusters independently, its notion of pilot-jobs has not been shown interoperably across both clouds and grids.

The Venus-C Generic Worker[8] is a cloud-only approach to pilot-jobs from Microsoft. Venus-C is built upon Microsoft Azure, but in order to create a layer of abstraction above the inner workings of the cloud, the idea of a Generic Worker was born. The idea behind the Generic Worker was to allow scientists to do their science without requiring knowledge of backend HPC systems by offering e-Science as a service. Venus-C has not been shown to work with grids, because its main objective is to get scientists using cloud infrastructure instead. While the notion of moving to the cloud for data-driven science is an important one, many existing cyberinfrastructures still have powerful grid computers that can also be leveraged to assist with the data-driven computations.

Prior Work: Whereas we will discuss in greater Sections 4 and 5 some of the concepts upon which this paper is built, for completeness we briefly outline them here.

Towards a Theoretical Model of Pilot-Jobs and Interoperable Implementations: The seamless uptake of distributed infrastructures by scientific applications has been limited by the lack of pervasive and simple-to-use abstractions at multiple levels at the development, deployment and execution stages. Of all the abstractions proposed to support effective distributed resource utilization, a survey of actual usage suggested that Pilot-Jobs were arguably one of the most widely-used distributed computing abstractions as measured by the number and types of applications that use them, as well as the number of production distributed cyberinfrastructures that support them. Our initial investigation [9] into Pilot-abstractions was motivated by the desire to provide a single Pilot-Job framework that would work over heterogeneous and myriad types of grid middleware.

This led to BigJob [10] – a SAGA-based Pilot-Job, which was used to support a range of applications, ranging from uncoupled ensembles of molecular dynamics (MD) simulations [10], to Ensemble-Kalman filter based applications [11] with global synchronization to loosely-coupled MD simulations with pair-wise synchronization [12]. Although BigJob provided the syntactical uniformity, i.e., a common API and framework for different grids, our experience led us to understand that different Pilot-Jobs had different semantics and capabilities, and made vast if not inconsistent assumptions of applications/users. This motivated our efforts in search of a common minimally complete model of Pilot-Jobs, and resulted in the P* model [4].

Once a common and uniform conceptual model was available, the notion of Pilot-Data was conceived using the power of symmetry, i.e., the notion of Pilot-Data was as fundamental to dynamic data placement and scheduling as Pilot-Jobs was to computational tasks. As a measure of validity, the P* model was amenable and easily extensible to Pilot-Data. The consistent and symmetrical treatment of data and compute in the model led to the generalization of the model as the *P* Model of Pilot Abstractions*.

Pilot-MapReduce: Following on the foundation laid by the P* Model of Pilot abstractions, a natural logical next step was to investigate if the capabilities of Pilot-abstractions could be used to support agile and dynamic execution modes for application and application-patterns? This was in-turn motivated by the observation that the MapReduce programming model is a popular solution

DOI: 10.1002/cpe

for distributed data processing and involves map and reduce (compute) phases, and a shuffle (data movement) phase. However, traditional MapReduce implementations like Hadoop are typically designed for single clusters and when scaled across widely distributed clusters lead to performance problems. Furthermore, when working in a distributed data context, a static resource assignment for MapReduce is very constraining. Could the use of dynamic runtime environment such as that provided by Pilot-abstractions overcome some of the traditional limitations of MapReduce, as well as the policies of the typical clusters they execute on? This led to the development of a Pilot abstractions based MapReduce [13] programming model — PilotMapReduce, and an investigation into its suitability for distributed data analysis. PilotMapReduce decouples the logic/pattern of MapReduce from the actual management of the distributed compute, data and network resources. By decoupling job scheduling and monitoring from the resource management, PilotMapReduce efficiently reuses the resource management and late-binding capabilities of BigJob and BigData, which are SAGA based Pilot-Job and Pilot-Data implementations.

3. CLOUD-BASED INFRASTRUCTURES FOR DATA-INTENSIVE APPLICATIONS

Whereas a cloud can be described on the basis of how (i.e., at what level the cloud as a service is provided), we will focus on the the specific elements that comprise the data-cyberinfrastructure subsystem (see Fig. 1). This is motivated by the fact that different cloud services that aim to support Big Data analytics have emerged. The foundation for many of the different Big Data offerings are different kind of storage services available, typically at the IaaS level. However, in order to appreciate the landscape of these cloud-based data-cyberinfrastructure and how they are provided, for completeness we review some established cloud taxonomy and terminology.

Cloud Taxonomies: There are multiple possible cloud taxonomical classifications. One approach classifies clouds as a service and uses the nature of the service as the basis. Using the "as-a-Service" taxonomy, clouds are classified into three categories: Software as a Service (SaaS), platform as a service (PaaS), and Infrastructure as a Service (IaaS) [3]. The infrastructure-as-a-service (IaaS) layer provides low-level, typically virtualized data and compute resources. Computational resources are typically represented as virtual machines instances. Examples for IaaS services are Amazon EC2 [14] and S3 [15] as well as the new Google Compute Engine (GCE) [16] service. PaaS services provide a higher-level capabilities, such as controlled runtime environments for web applications, task ensembles or MapReduce. Example of PaaS services are: Azure Web and Worker Roles, Google's App Engine, BigQuery [17] and Prediction API [18], and Amazon Elastic MapReduce [19].

Adjunct to the above classification, Clouds can also be classified according to their deployment model into public and private clouds. Frameworks such as OpenStack [20] and Eucalyptus [21] aim to provide a framework for building a private cloud environment which similar capabilities as EC2 and S3. FutureGrid's cloud environment currently supports both framework. Currently, new science cloud infrastructures, e. g. FutureGrid [22], are being developed and existing production infrastructures are being expanded with cloud capabilities, e. g. EGI [23]. While the focus of many science clouds is currently mainly on IaaS, there are several attempts in providing high-level PaaS services: FutureGrid e. g. offers Hadoop appliances [24] and Venus-C [8] provides a generalized worker role service (similar to the Azure Worker Role).

Cloud Data-Infrastructure: The storage services typically have different characteristics, i. e. they usually differ in their read and/or write performance, supported data volumes, interfaces & semantics, consistency guarantees, degree of replication, reliability, scalability etc. (see Baron [25] for an overview of Amazon's storage services). While clouds often provide storage services with different semantics and at different levels of abstractions (including e. g. relation databases), in the following we focus on general file-based storage types. In general, file-based cloud storage can be classified as follows: (i) local storage refers to local hard disk directly attached to the compute resource (typically a VM); (ii) remote storage refers to different forms of distributed (possible

DOI: 10.1002/cpe

Storage Type	Azure	Amazon	Google	Open-Stack	Euca-lyptus	XSEDE	OSG
Local	yes	yes	yes	yes	yes	yes	yes
Remote	Azure Drive	EBS	GCE Disks	Nova Volumes	EUCA Block Storage	Lustre, GPFS	no
Distributed Storage	Azure Blob Storage	S3	Google Storage	Swift	Walrus	GFFS	SRM

Table I. **Storage Types Supported by the Different Infrastructures:** Each infrastructure supports various storage options. Most commonly, storage is mounted as a local or remote filesystem. Distributed object storage provides some interesting characteristics for data-intensive applications.

parallel) filesystems than can be attached to a VM either as block storage device or NAS. Both type (i) and (ii) storage is commonly integrated via the Linux virtual filesystem layer and thus, is suitable in particular for legacy, file-based applications.

A novel type of storage introduced by cloud environments are (iii) object stores, which are highly distributed storage systems that can potentially spawn across multiple data centers. Object stores are optimized primarily for "write once, read many" workloads and can support massive volumes of data with their scale-out architectures. For example, Amazon S3 automatically replicates data across multiple data centers within a region. These kind of stores are not suitable for all workloads (e. g. traditional, transactional workloads). On the other hand, typical Big Data workloads that (i) require the storage of large volumes of data and (ii) are characterized by a large amount of reads are particularly suitable for such stores. Access to such storage systems is via a common – often simplified – namespace and API. For example, cloud systems, such as the Azure Blob Storage, Amazon S3 and Google Storage, provide only a namespace with a 1-level hierarchy. This means that applications need to be adapted, in order to benefit from object storage. The most widely used object stores are: Amazon S3 [15], Azure Storage [26] and Google Cloud Storage [27]. In addition, both Eucalyptus and OpenStack provide an object store: Eucalyptus Walrus [28] and OpenStack Swift [29]. A major limiting factor, is the necessity to ingest large volumes of data over the WAN. Large volume data transfer are associated with high costs and unpredictable and/or unacceptable performance.

Table I shows an overview of distributed storage systems. In addition to the described object stores and local storage that is directly attached to the VM, each cloud infrastructure provides several further storage options. The focus of this analysis are file-based storage systems. Structured storage types (e.g. relational databases) and key-/values stores are not considered. Azure Drive [30], Amazon EBS [25] and Google Compute Engine Disks [31] are comparable block-based storage service that provide storage that can directly be attached as device to a VM. Some services support special features, e. g. Azure Drive can be backed by both page and block blobs optimizing for random respectively sequential access; Amazon EBS supports guaranteed I/O on EBS volumes. Similar, services exist for both Eucalyptus and OpenStack: Eucalyptus Block Storage [32] and Nova Volume [33].

On traditional HPC and HTC infrastructure the notion of dynamically provisioned storage does not exist. However, in particular HPC infrastructures, such as XSEDE, typically provides access to a parallel filesystem, such as Lustre [34] or GPFS [35]. On the Open Science Grid (OSG) only local storage is available. In addition, both infrastructures provide or are going to deploy distributed storage services: The Global Federated File System (GFFS) [36] is currently evaluated in the context of XSEDE. GFFS provides a global namespace on top of various XSEDE storage resources. The system handles file movement, replication etc. transparently. OSG deploys several SRM-based storage services [37].

Data Management: As explained before, various strategies for managing data in distributed environments exist. The most common approach in the context of clouds is the localization of all

DOI: 10.1002/cpe

data, i. e. the application data is initially pushed to a cloud storage service, such Amazon S3, Google Storage and Azure Storage. These service, such as S3, are well suited for storing large amounts of data and support typical data-intensive workloads (e. g. with a lot of sequential reads). Often, cloud providers make different type of data sets (e. g. the 1000 genome dataset) directly accessible from their central object store. By eliminating the initial data movement, the entry barrier to running computations on this data in the cloud is significantly decreased.

Often, data is too large to move and thus, must either be pre-processed or completely processed before it can be moved. Even within single clouds, the management of data across different regions is challenging, e. g. Amazon S3 replicates data geographically but only within a single region.

While some cloud services that support a high degree of geographic distribution have emerged, e. g. content delivery services such as Akamai and Amazon CloudFront, dealing with geographically distributed data remains a challenge. However, both Google and Facebook internally deploy systems that supports the management of large data volumes across multiple, geographically dispersed locations. Both Google Spanner [38] and Facebook Prism [39] aim to provide one logical namespace and handle the automatic replication and movement of data. However, these capabilities are currently not available as external cloud services. If an application requires such capabilities, it needs to implement these manually. In this paper we show how Pilot-Abstractions can help manage data dynamically and geographically distributed data.

Cloud Compute Infrastructure: In addition to storage systems, there are other cloud infrastructures that although important in understanding cloud capabilities and applications that can be supported. For example, the availability of *queues* provides agile and effective approaches to distributed coordination. Furthermore, Hadoop [40] has become an important building block for many data-intensive applications in clouds. Amazon e. g. offers with Elastic MapReduce (EMR) [19] a pay per use MapReduce clusters. EMR is based on Amazon EC2 and S3 and supports out-of-the-box both the open source and the MapR Hadoop distribution. It includes, not only the Hadoop core, but also Hive, Pig and HBase. Amazon provides a file system adaptor for S3, i. e. data can be streamed from S3 into Hadoop. Microsoft provides a similar service based on the Hortonworks Hadoop platform [41] called to Hadoop on Azure [42]. Also, various markets for data in the cloud have emerged. Cloud providers, such as Amazon and Microsoft published various scientific data set in these markets. Amazon e. g. provides access to 200 TB of data of the 1000 genome project [43] and the CloudBurst Hadoop application [44].

Cloud Applications: While cloud environments have traditionally deployed in commercial settings, more and more science applications utilize to some extend cloud resources or are natively developed for clouds. While traditional HPC applications – often based on MPI – show limitations in cloud environments [45, 46], there is a broad class of loosely-coupled applications that is well-suited for clouds [47, 48]. There is a large class of data-intensive applications that is loosely coupled and thus very well-suitable for cloud environments, e. g. genome sequencing applications or applications based on the replica-exchange algorithm [49].

4. PILOT-ABSTRACTIONS FOR CLOUDS

Pilot-Abstractions provide a suitable means to marshall heterogeneous sets of both compute and data resources and support the efficient utilization of different kinds of commercial as well science cloud resources. Pilot-Abstraction have been extensively used on both HPC and HTC infrastructures for a range of application scenarios as a resource management abstraction to, (i) improve the utilization of resources, (ii) to reduce wait times of a collection of tasks, (iii) to facilitate bulk or high-throughput simulations where multiple jobs need to be submitted which would otherwise saturate the queuing system, and (iv) as a basis to implement application specific execution, scheduling and policy decisions (see Figure 3).

As alluded to in the introduction, there are many reasons motivating the exploration and extension of Pilot-Abstractions for clouds. Pilot-Jobs map well to IaaS cloud systems, wherein a Pilot can

DOI: 10.1002/cpe

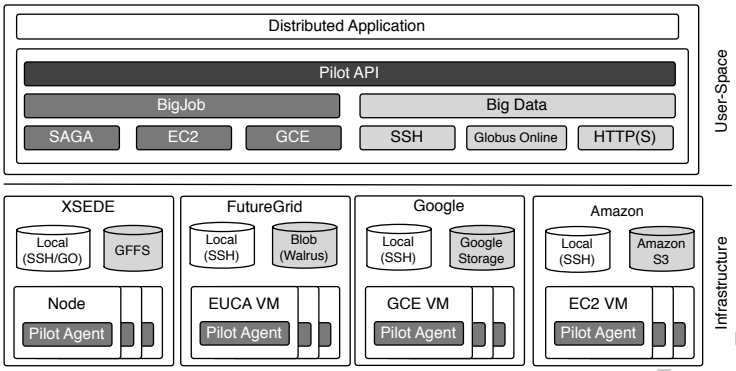

Figure 2. **BigJob Pilot Abstractions and Supported Resource Types:** BigJob and BigData provide a unified abstraction to compute and data resources of both grid and cloud environments. Resources are either accessed via SAGA [50, 51] or via a custom adaptor.

marshall multiple VMs, possibly of different characteristics and performance capabilities; an agent which pulls and executes tasks is deployed on each VM. Ultimately, there is a decoupling between task specification and resource assignment, with the Pilot-Manager or an equivalent cloud-entity carrying out the mapping using dynamic/real-time information. Given the above, not surprisingly, the Pilot-Jobs concept has already been applied to clouds; for example, PaaS cloud systems, such as Venus-C (Azure) [8], support the notion of Generic Workers (worker role) which are conceptually similar to Pilots in that they pull tasks (application workload) from a *central repository* when the environment is available.

Much has been said and assumed about the "elastic" properties of clouds. Clouds do not provide infinite and immediate access to resources, at least not with at the same level of quality/performance. The putative illusion of infinite elasticity is a consequence of system-level multiplexing. We posit that such "elastic" characteristics can also be provided by runtimes which support qualitatively similar dynamic execution modes. Also, one route to interoperability definitely arises from a unified model of runtime execution that Pilots present. In this section, we show how such an unification can be provided.

4.1. Implementing the P Model: BigJob and BigData*

The P* model [4] is an attempt to provide the first minimal but complete model for describing and analyzing Pilot-Jobs. We particularly extended this model to Pilot-Data (PD): PD provides late-binding capabilities for data by separating the allocation of physical storage and application-level Data-Units. Similar to a Pilot-Compute, a Pilot-Data provides the ability to create Pilots and to insert respectively retrieve Data-Units. The Pilot-API [52] provides an abstract interface to both Pilot-Compute and Pilot-Data implementations that adhere to the P* model.

Consistent with the P* model, BigJob (BJ) [10] and BigData (BD) [13] provide a unified runtime environment for Pilot-Computes and Pilot-Data on heterogeneous infrastructures (see Figure 2). For this purpose, BigJob provides a higher-level, unifying interface to heterogeneous and/or distributed data and compute resources. The framework is accessed via the Pilot-API [52], which provides two key abstractions:

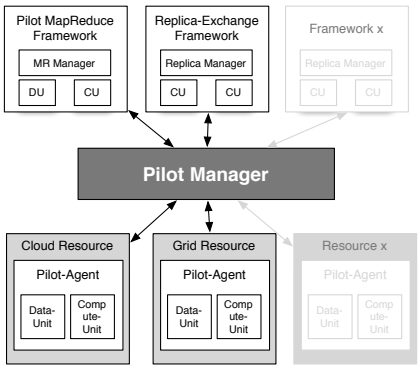

Figure 3. **Pilot-Abstraction for Application-Level Resource Management:** Pilots are a well-proven abstraction for accessing different types of storage and compute resources. Higher-Level framework, such as Pilot-MapReduce or the Replica-Exchange [12] can be built on these capabilities.

DOI: 10.1002/cpe

Pilot-Job and Pilot-Data. Applications can specify their resource requirements using a Pilot description. Pilots are started via the Pilot-Compute Service respectively the Pilot-Data Service.

Pilot-Jobs eliminate the need to interact with different kinds of compute resources, e. g. batch-style HPC/HTC resources as well as cloud resources, and provide a unified abstraction for allocating resources. Similarly, Pilot-Data removing the need to interoperate with different data sources and stores, e. g. repositories, databases etc, by providing a virtualized data service layer, which can be used to allocated and access a heterogeneous set of data sources and stores.

Figure 4 shows the architecture of BigJob/BigData. The Pilot-Manager is the central entity, which manages the actual Pilots via the Pilot-Agent. Each agent is responsible for gathering local information, for pulling Compute-Units from the manager. The framework relies on a central coordination service based on Redis [53] for communication between manager and agent.

As shown in Figure 2 BigJob/BigData supports various storage and compute resource types via a flexible plugin architecture. For Pilot-Computes BJ supports various types of HPC/HTC resources via SAGA/Bliss (e. g. Globus, Torque or Condor resources). Further, various cloud plugins have been developed for supporting EC2-style backends as well as Google Compute Engines. The plugin utilizes the native EC2 and GCE API for launching the VM: for EC2 the Boto library [54] and for GCE the Google Python API client [55] is used. The EC2-API [56] is the de facto standard for managing VMs in clouds and can also be used for accessing different science clouds, e. g. the Eucalyptus and OpenStack cloud of FutureGrid. Google Compute Engine provides modern HTTP/REST/JSON and OAUTH for authentication. Having started the VM, the Pilot-Agent is launched using SAGA and the SSH adaptor of SAGA.

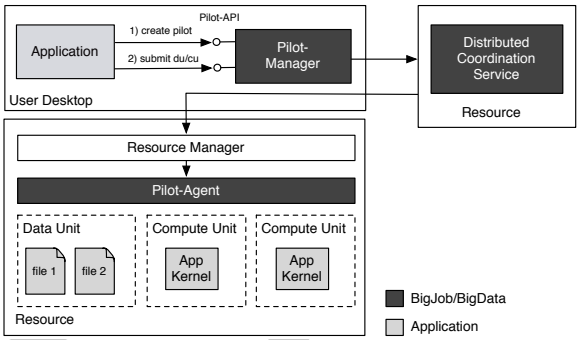

Figure 4. **BigJob and BigData Architecture:** The Pilot-Manager is the central coordinator of the framework, which orchestrates a set of Pilots. Each Pilot is represented by a decentral component referred to as the Pilot-Agent, which manages the set of resources assigned to it.

Similarly, a Pilot-Data can be placed on different types of storage, e. g. a local or remote filesystem or on object stores as often found in cloud environments. Depending on the storage type different access respectively transfer protocols are supported. For example, local/remote storage both on grid and cloud resources can be managed via SSH. On production grid infrastructures, such as XSEDE [57], this kind of storage can also be accessed via Globus Online [58], which underneath relies on GridFTP [59] for file movement etc. A particularity of cloud environments are the described object stores. Object stores are typically accessed via custom API typically based on HTTP/REST. For example, the Amazon S3 API [60] is based on HTTP and XML and is also supported by Eucalyptus Walrus and OpenStack Swift. Google Storage provides two APIs: an S3 compatible and a JSON-based REST API. Again, Boto is used for S3-based storage resources and the Google Python API client for Google Storage.

4.2. Pilot-API: Managing Data and Compute Dependencies

Pilot-Abstractions can marshal the difference between the different cloud storage types and and provide a seamless, unifying environment to the application simplify the usage of distributed infrastructures. Application can acquire resources in form of Pilots. A central Compute-Data Service is used to submit Compute-Units (CUs) and Data-Units (DUs) to these Pilots. Application can declaratively specify CUs and DUs and effectively manage the data flow between them.

Figure 5 shows the interactions and the data flow between Compute-Units and Data-Units. A CU can have both input and output dependencies to a set of DUs. BigJob and BigData will ensure that

DOI: 10.1002/cpe

these dependencies are met when the CU is executed, i. e. either the DUs are move to the CU or the CU is executed at (or close to) the location of the DUs. The output files can be automatically transferred to a set of output DUs. If a CU declares a set of input dependency in the `input_data`, the runtime system will then move the data to the working directory of the CU. Depending on the locality of the DU, this can be a local or remote operation. BigJob/BigData uses an affinity-aware scheduler that ensures that if possible "affine" CUs and DUs are co-located.

4.3. Affinities and Scheduling

Pilot-Abstraction provide the basis for application-level scheduling. Applications can make placement decisions based on the Pilots it has acquired. Additionally, applications can rely on an application-level scheduling service as provided by BigJob and BigData. This service is exposed via the Compute-Data Service interface and accepts both CUs and DUs. The service decides how and when to place data and schedule its movement.

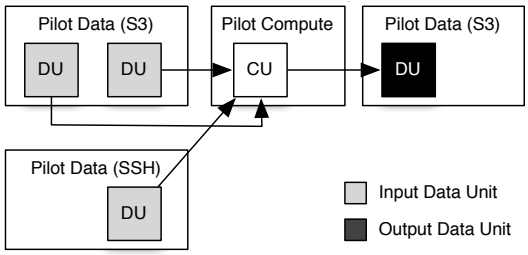

Figure 5. **DU and CU Interactions and Data Flow:** Each CU can specify a set of input DU (residing in potential different PD). The framework will ensure that the DU is staged to the working directory of the CU. Having terminated the job, the specified output is moved to the output DU.

We use *affinities* to describe resource relationship between CUs, DUs, and resources (i. e. Pilots). A simple model for describing such relationships is used: data centers and machines are represented as a tree (see Figure 7). The further the distance between two resources, the smaller the affinity. The framework relies on this affinity description to efficiently manage the different kinds of storage as well as distribution of data/compute across WAN.

As explained before, even within a single cloud, managing data locality is a great challenge due the various storage options. Figure 6 summarized the different kinds of storage options an application has to manage. Pilot-Abstractions not only provide a uniform interface to these different kinds of storage, the also support application-level scheduling. The core of the Pilot-Manager is an affinity-aware scheduler. BigJob and BigData support different forms of data/compute affinities via the Compute-Data Service. The internal scheduler of the Compute-Data Service relies on affinity as first entity to describe relationships between DUs, CUs and resources. Affinities are an essential tool for modeling the different storage characteristics and to allow an effective reasoning about different storage topologies and data/compute placements. BJ relies on affinity label for describing resource topology.

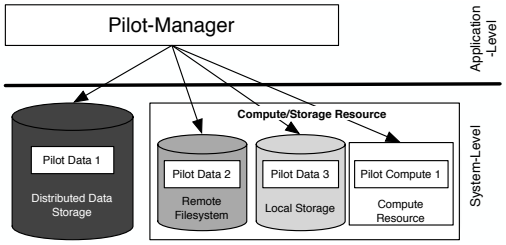

Figure 6. **Pilot-Compute and Pilot-Data on Different Types of Storage Resources:** BigJob/BigData supports various types of resources. Affinities allow the user to describe the relationship between different types of resources enabling the scheduler to efficiently make placement decisions.

Every Pilot is assigned an affinity label in the resource description. Affinities are organized in an hierarchical way. Figure 7 shows e. g. how a distributed system consisting of different types of cloud and grid resources can be modeled. The shorter the distance between two nodes, the higher the affinity. Commercial clouds, such as Amazon and Google, are typically organized into geographic regions. Within each region multiple availability zones exist. Commonly, data movements between the lower-level availability zones are cheap and fast, while movements between regions costs some significant amount on money and time. Even worse are movements between infrastructure, e. g.

DOI: 10.1002/cpe

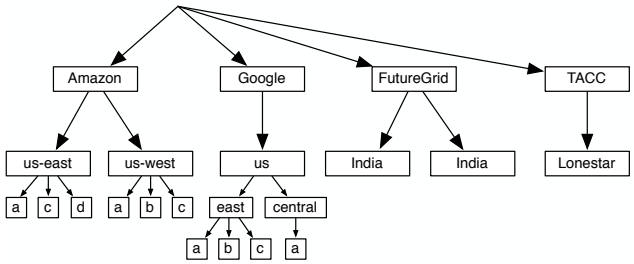

Figure 7. **Managing Affinities between Cloud and Grid Resources:** Pilot-Data assigns each resource an affinity based on a simple hierarchical model. The smaller the distance between two resources, the larger the affinity.

between Amazon, Google and FutureGrid. Both CUs and DUs can request a certain affinity. The runtime system then ensures that the data and compute affinity requirements of the CU/DU are met.

Each of the storage types in Figure 6 can be assigned an affinity. The model is able to deal with subtle differences between clouds and grids/clouds. For example, at Amazon S3 is service that is confined by a single region, while Google Storage spawns multiple regions.

The Compute-Data Service scheduler matches the affinity requirements of a CU/DU with the available set of resources before assigning it to a Pilot. CUs are placed closed to a DU. Before a CU is actually run, the DU is exported to the working directory of the CU. In the cloud cases e. g. this means that the data is copied from an object store to the local filesystem where the CU runs.

Having discussed how Pilots are a good abstraction for resource effective application-level resource management. Using this high-level abstraction to computing and storage resources, higher-level frameworks can as depicted in Figure 3 request resources (Pilots), on which they then can run tasks (Compute-Units). Thus, Pilots can be used as basis for building capabilities on grids/clouds; Pilot-MapReduce is a good example for this (see section 5).

5. PILOT-MAPREDUCE FOR CLOUDS

Pilot-MapReduce (PMR) [13] is a Pilot-based implementation of the MapReduce programming model, which decouples the logic/pattern of MapReduce from the actual management of the compute, data and network resources. For this purpose, PMR efficiently reuses the resource management and late-binding capabilities of BigJob and BigData via the Pilot-API interface. Another key feature of the Pilot-API is the ability to declaratively define the data flow between the Data-Units and Compute-Units. Based on the defined affinities, the scheduler will then attempt to place data and compute as close as possible. If necessary, data will be moved.

Using the Pilot-abstractions, applications such as PMR can transparently use different kinds of grid and cloud resources, e. g. Amazon EC2/S3, Google Compute Engine/Storage and Eucalyptus-based science clouds. Details specific to the respective resource type are hidden behind the Pilot interface enabling the application to focus on the definition and management of the workload in form of CUs and DUs.

PMR exposes an easy-to-use interface, which provides the complete functionality needed by any MapReduce-based application, while hiding the more complex functionality, such as chunking of the input, sorting the intermediate results, managing and coordinating the map and reduce tasks, etc. The user solely has to provide an implementation of the chunk, map and reduce function in form of a executable script.

Architecture and Interactions: Figure 8 shows the architecture of the PilotMapReduce framework. The flow of a MapReduce run involves the (i) the chunking of the data, (ii) the execution of the map compute tasks, and (iii) the execution of the reduce task. Between the stages data must be moved. Between stage (ii) and (iii) the data is sorted and moved; this stage is also referred to as shuffling.

DOI: 10.1002/cpe

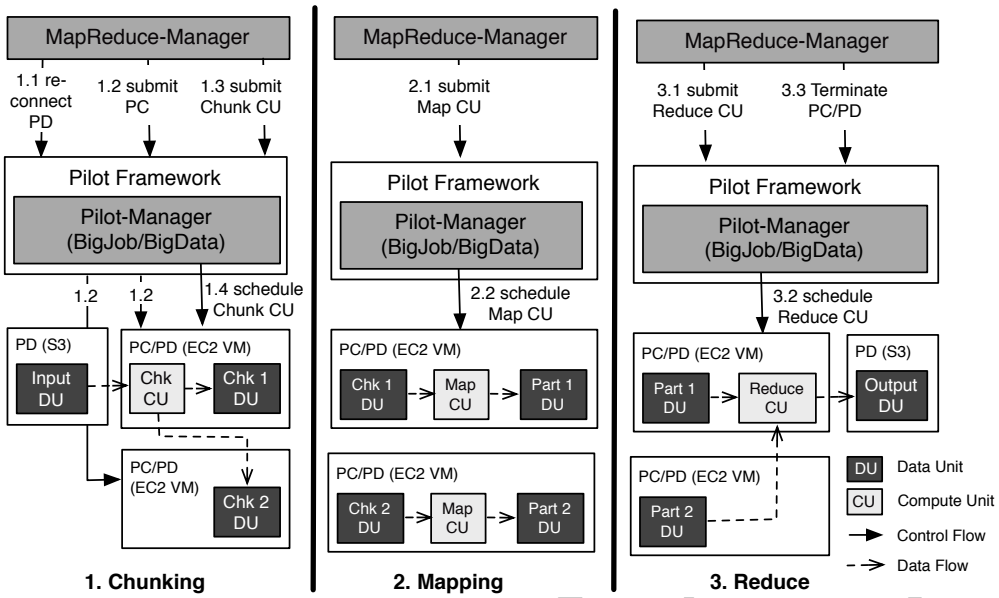

Figure 8. **Pilot-MapReduce Control and Data Flow:** PMR allocates both storage and compute resources via the Pilot-API. The three stage MapReduce run is then managed by submitting Data-Units and Compute-Unit to the Pilot-Manager. The Pilot-framework takes care of moving the data in and out the different stages.

PMR allocates both storage and compute resources using BigJob/BigData. This enables PMR to support a heterogeneous set of cloud/grid resources (cmp. Figure 2). Having spawned a set ofPilots, PMR can flexibly utilize them for executing chunking, mapping and reduce tasks as well as for managing the data flow between the phases. For this purpose, the framework solely needs to declare the input and output data dependencies for a Compute-Unit. BigJob/BigData will then attempt to place the Compute-Unit close to the input data. This practice is applied to the input as well as the intermediate data of the different stages.

The scenario depicted in Figure 8 shows a MapReduce flow on Amazon's cloud infrastructure. Typically, in cloud environments, the data is initially stored in a low-cost object storage, such as S3. In step 1.1 the framework re-connects to the S3-based Pilot-Data. Then, the MR-Manager spawns a set of Pilots for executing the computational tasks. Each Pilot is represented by a VM. In stage 1, the initial data is downloaded to the VM and then chunked into multiple DUs. For this purpose, a chunk DU is executed on a Pilot. The chunk DU downloads the declared input data from the S3 Pilot-Data and splits it into a set of output DUs (which are in this scenario directly located on the VM). Then a map CU is executed on each of the DUs. Each CU creates an output DU containing the result of the map phase, i. e. the partitioning files. Finally, the reduce CU is executed and the output is written back to a defined S3-based Pilot-Data.

On grids, the MR-Manager allocates a set of compute and data resources by starting one (or most often a set of) Pilot-Data and Pilot-Compute on different resources mainly using SAGA/Bliss [51] as abstraction to the local queuing system. In general, on each resource one Pilot-Compute and one Pilot-Data are co-located. In contrasts to clouds, most grid resources have a shared file systems that simplifies in particular the management of the input data.

Distributed MapReduce: Often, there are MR-based applications that to operate on distributed data, including but not limited to distributed scenarios as represented by different clouds or regions within a single cloud, clusters connected over WAN, or production distributed cyberinfrastructure such as XSEDE. Data distribution can be difficult to manage for performance, as latencies and bandwidths may vary unpredictably.

DOI: 10.1002/cpe

PilotMapReduce provides a flexible, extensible implementation of MR: Pilot-Data can be spawned to various (possibly distributed) resources. The affinity-aware scheduler attempts to co-locate compute/data using the minimal amounts of data movements necessary. We previously showed [13] how these capabilities can be used to support different distributed MapReduce topologies, e. g. distributed and hierarchical MapReduce configurations. A distributed PMR utilizes multiple resources often to run map tasks close to the data to avoid costly data transfers; the intermediate data is then moved to another resource for running the reduce tasks. In a hierarchical PMR the outputs of the first complete MapReduce run are moved to a central aggregation resource. A complete MapReduce run is then executed on this resource to combine the results.

6. EXPERIMENTS

The aim of this section is to demonstrate how BigJob/BigData can be used to interoperable access different types of compute and storage resources. We show how BigJob/BigData can support interoperability in a highly fragmented cloud environment consisting of various commercial and science cloud infrastructures with different interfaces and characteristics. In particular, we focus on aspect related to infrastructure for data-intensive applications and novel services such as object stores (that are commonly not available in traditional grid resp. HPC environments). Further, we illustrate representative usage modes of Pilot-Abstractions and show how they can support frameworks such as Pilot-MapReduce. For this purpose, we run several experiments on FutureGrid (FG) and various cloud infrastructures, including the Eucalyptus cloud on India/FutureGrid, Amazon EC2 and Google Compute Engine. Each experiment is repeated at least five times.

6.1. Objective 1: Interoperability

To validate the effectiveness and usability of the Pilot-API/BigJob/BigData, we conducted a series of experiments on various cloud infrastructures. In particular, we focus on evaluating the interoperability of the different storage and compute services offered by the different infrastructures. We show how the Pilot-API allows the effective management of data and data transfers. Further, it provides application a powerful tool to trade-off different infrastructure characteristics.

In the following, we focus on the most common usage modes of clouds: Initially, files are moved to a cloud-based storage services. From there the data is made available to the VMs for running computations. Commonly, files are moved only once to the cloud. As part of this evaluation, we investigate both the transfer both to and within the cloud infrastructure. We use the FutureGrid machines Hotel and India as submission machine, i. e. the machine where the initial data resides and the Pilots are submitted from. We investigate different scenarios: In scenario (a) we investigate the influence of the input data size on the overall runtime; in scenario (b) we analyze different infrastructures configurations.

Figure 9 shows the results on scenario (a). For this scenario, we utilize the Eucalyptus VM and Walrus service on India. The initial dataset resides on India. As expected, the runtime increases with the increasing amount of data. Both the data upload time to cloud and download time within the Eucalyptus cloud increase linearly. As all transfers are done in a local network this is an expected result. Also, we observed a high fluctuations in the VM startup times, which can be observed in the Pilot queue times. This is caused by externally controlled factors, most likely the current utilizations of the Eucalyptus cluster (the data was collected over a period of week). In general, the Pilot queue time varies on India between 1 and 10 min.

In scenario (b) we investigate the execution of different workload/infrastructure configurations with a constant input DU of size 1 GB. Figure 10 shows the results. Scenario (b1) - (b3) show different configurations on FG/India: in (b1) and (b2) we compare different Pilot-Data backends (Eucalyptus Walrus and SSH) on the same resources. In both cases we upload the 1 GB input file to the Walrus resp. SSH-based storage. We then execute 1 CU that downloads the data prior its execution. While the initial upload to the Walrus respectively SSH Pilot-Data is comparable, the download to the VM time with SSH is considerable slower than for Walrus. In general SSH is

DOI: 10.1002/cpe

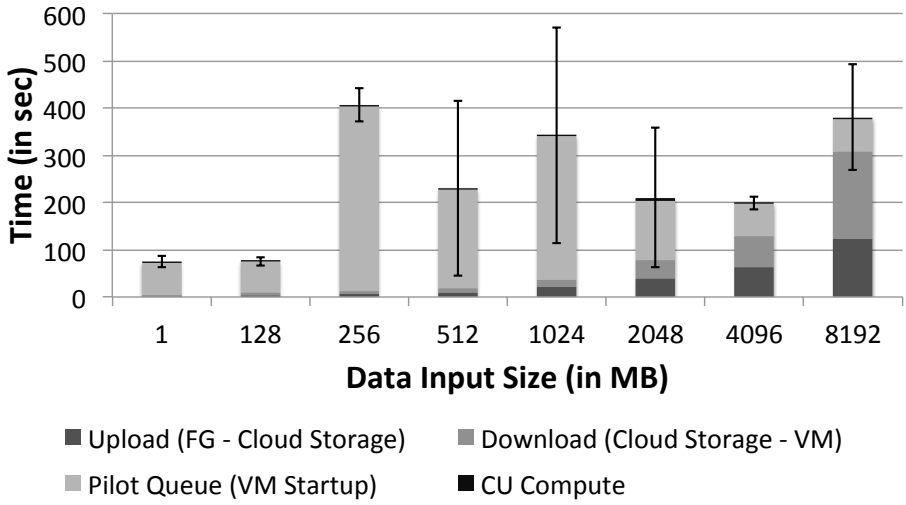

Figure 9. **Dependence on Input Data Size (FutureGrid/Eucalyptus/Walrus on India):** With the increasing amount of data both the time to upload the files to the cloud and the the time to download to the VM increases. Note that the VM startup time (observable in the Pilot queue time) depends on external factors, in this case the current utilization of the Eucalyptus cluster.

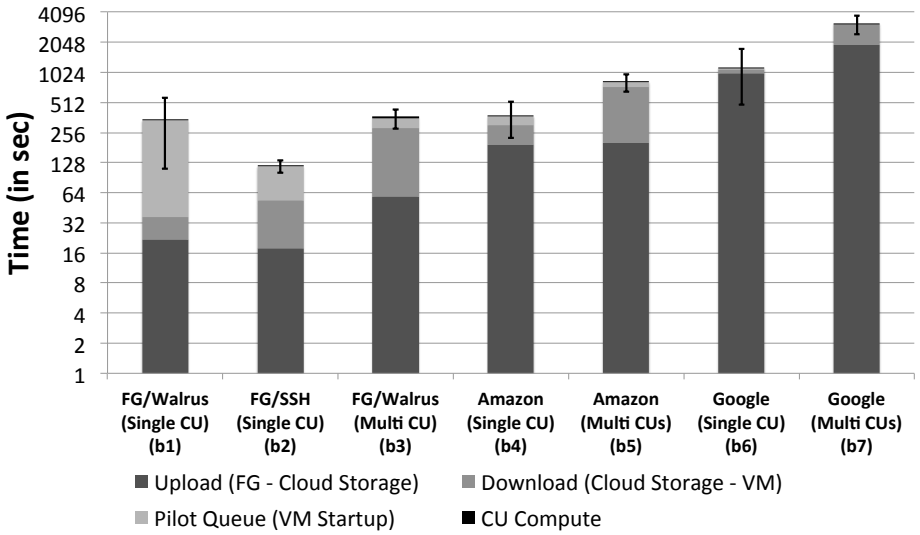

Figure 10. **BigJob/BigData on Different Infrastructures:** BigJob/BigData supports different commercial and science cloud infrastructures each with its own types of storage and compute services. In scenario (b1), (b2), (b4), (b6) we execute 1 CU; in scenario (b3), (b5) and (b7) we run 5 CUs per VM core to test the available connectivity between VM and cloud storage. A critical factor in particular for the commercial clouds of Amazon and Google is the available bandwidth for the data ingest.

associated with additional overheads caused e. g. by the encryption, in particular when compared to the HTTP-based Walrus protocol. Again, the Pilot queue was highly unpredictable and dominate the overall runtime in these scenarios. In (b3) we execute in total execute 20 CUs (up to 4 concurrently) each of them downloads the input data from the Walrus PD. While the initial data is only uploaded once, it obviously leads to a significant increase in the download time indicating that the available bandwidths to the cloud storage might have been saturated.

DOI: 10.1002/cpe

The cloud scenarios (b4) - (b7) are dominated by the data ingest phase (i. e. the upload time). Typically, data movement into the cloud are slower and more expensive than intra-cloud data movements. In this particular scenario the data ingest time for clouds varies between ∼4 minutes for Amazon and ∼18 minutes for Google. The primary determinant of this time is the available Internet bandwidth. While the upload performance is determined by external factors (currently utilization of the Internet connection), in the investigated scenario the data transfer time is consistently higher for Google Compute Engine than for Amazon. The external dependency on Internet bandwidth is the high standard deviation (that is mainly caused by the data ingest phase). Also, it must be noted that Amazon gives you more option to select a location for your data – in this scenario the region us-east, which is close to the hotel resource, was used. For Google Storage it is only possible to select between the US and Europe. Once the data is uploaded to the cloud, it can be re-used multiple times. Usually, the cloud providers ensure that a suitable link between VM and cloud storage exists.

The Pilot queue time (which includes the VM startup time), GCE is faster than Amazon EC2. Also, the available intra-cloud bandwidths are better for GCE: The data download time for GS/GCE is about 40 % faster than for S3/EC2 in the single CU case. However, the bandwidth available at GCE saturates fast with 8 CUs concurrently downloading the input data: the download time in the multi CU scenario is 12 times slower on GCE compared to a slowdown of only 5 times on Amazon. Also, the CU execution time are significantly lower on GCE. In both cases the smallest possible VM was used; however, it must be noted that Amazon's t1.micro instance type doesn't provide any guaranteed CPU time. Also the memory is with 613 MB considerable smaller than for Google's smallest VM n1-standard-1, which has 3.75 GB memory.

6.2. Objective 2: Support of Higher-Level Frameworks

In this section, we show how higher-level frameworks – in this case Pilot-MapReduce – can benefit from the usage of Pilot-Abstractions. For this purpose, we utilize the Amazon EC2/S3 services for running a genome sequencing MapReduce application with PMR (GS/PMR).

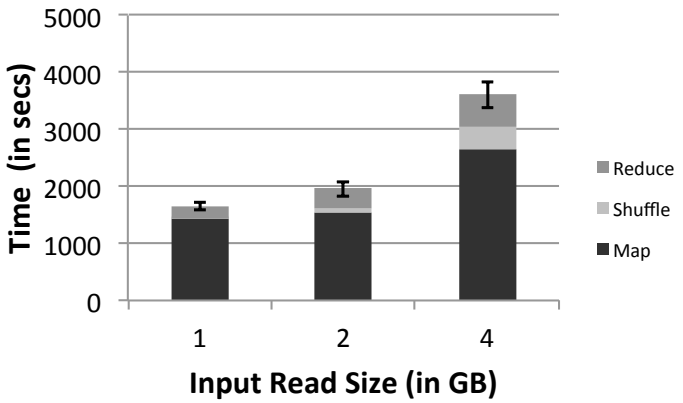

Figure 11. **PMR on Clouds:** The figures show the results of different PMR runs on EC2. The input data has been initially moved to S3. As expected, the runtime increases with the input size, which is consistent with earlier results [13]. The observed fluctuations are generally very low (compared to the overall runtime).

The GS/PMR is a Pilot-MapReduce based Seqal [61, 13] application. It performs an important part of the sequencing workflow. The application implements the read alignment in the mapping phase using BWA aligner [61] and duplicate read removal in reduce phase using Picards rmdup [62]. The GS/PMR reducer removes duplicate reads based on the key fields-chromosome, position and strand of the mapper output.

For experiments demonstrating the utilization of Pilot-abstractions for PilotMapReduce, we utilize a large Amazon VM (type: m2.4xlarge) as well as different sets (1 GB, 2 GB, 4 GB) of input data, which has been initially transferred to Amazon S3. PMR also uses a S3-PD to store

DOI: 10.1002/cpe

in the necessary chunk, reducer and mapper scripts. The application re-connect at the beginning to this data using the PD URL. Consistent we earlier experiments, the upload to the S3 PD requires between ∼2.5 minutes for the 1 GB and ∼10 minutes for the 4 GB scenario.

We run a total of 8 MapReduce workers and utilized a chunk size of 475740 short reads. Results are plotted in Fig. 11. The experiment is repeated three times, with experiment-to-experiment fluctuations in time relatively small (compared to actual values). The time for launching the Pilot, which includes the VM startup time, is about ∼3 minutes. Consistent with Ref. [13], the reduce phase increases essentially linearly with input data size. The intermediate data that needs to be moved to the reduce task is about 1.8 as large as the input data.

7. CONCLUSION AND FUTURE WORK

Pilot-Abstractions enable the efficient interoperable usage of different kinds of resources. We show how a large set of heterogeneous set of both commercial and scientific cloud resources can be accessed via the Pilot-Compute and Pilot-Data Abstraction. Further, we established affinities as a powerful abstraction for describing relationships between compute, data and resources. We showed that affinities can be used to model distributed resource topologies consisting of heterogeneous cloud resources. Pilot-Abstractions provide the basis for implementing higher-level, distributed frameworks, which can support different programming, e. g. PMR provides a Pilot-based implementation of the MapReduce programming model.

This paper contributes to the practice and experience of distributed systems for data-intensive computing by validating and extending the Pilot-abstraction — a highly successful, but arguably under utilized runtime environment in distributed computing, to cloud infrastructures. We believe there remain a plethora of important research questions that we do not touch upon here, but for which we believe that pilot-abstractions will provide a useful vehicle to research.

ACKNOWLEDGEMENTS

This work is funded by NSF CHE-1125332 (Cyber-enabled Discovery and Innovation), HPCOPS NSF-OCI 0710874 award, NSF-ExTENCI (OCI-1007115). Important funding for SAGA has been provided by the UK EPSRC grant number GR/D0766171/1 (via OMII-UK) and the Cybertools project (PI Jha) NSF/LEQSF (2007-10)-CyberRII-01, NSF EPSCoR Cooperative Agreement No. EPS-1003897 with additional support from the Louisiana Board of Regents. SJ acknowledges the e-Science Institute, Edinburgh for supporting the research theme. “Distributed Programming Abstractions” & 3DPAS. This work has also been made possible thanks to computer resources provided by TeraGrid TRAC award TG-MCB090174 (Jha). This document was developed with support from the US NSF under Grant No. 0910812 to Indiana University for “FutureGrid: An Experimental, High-Performance Grid Test-bed”. We thank FutureGrid support, in particular Fugang Wang for exceptional support. Last but not least, we thank our fellow Radicalistas for their support and input.

REFERENCES

1. Hey AJG, Tansley S, Tolle KM. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009. URL http://research.microsoft.com/en-us/collaboration/fourthparadigm/.
2. Jha S, Hong NC, Dobson S, Katz DS, Luckow A, Rana O, Simmhan Y, Weissman J. 3DPAS: Distributed Dynamic Data-intensive Programming Abstractions and Systems. 2011. http://www.cct.lsu.edu/~sjha/select_publications/3dpas.pdf.
3. Jha S, Katz DS, Luckow A, Merzky A, Stamou K. *Cloud Computing: Principles and Paradigms*, chap. Understanding Scientific Applications for Cloud Environments. Rajkumar Buyya and James Broberg, Wiley, 2010.
4. Luckow A, Santcroos M, Weidner O, Merzky A, Mantha P, Jha S. P*: A Model of Pilot-Abstractions. *8th IEEE International Conference on e-Science 2012*, 2012.
5. Frey J, Tannenbaum T, Livny M, Foster I, Tuecke S. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing* Jul 2002; **5**(3):237–246, doi:10.1023/A:1015617019423. URL http://dx.doi.org/10.1023/A:1015617019423.
6. ”Co-Pilot: The Distributed Job Execution Framework”, Predrag Buncic, Artem Harutyunyan, Technical Report, Portable Analysis Environment using Virtualization Technology (WP9), CERN.
7. Hategan M, Wozniak J, Maheshwari K. Coasters: Uniform resource provisioning and access for clouds and grids. *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*, UCC

DOI: 10.1002/cpe

'11, IEEE Computer Society: Washington, DC, USA, 2011; 114–121, doi:10.1109/UCC.2011.25. URL http://dx.doi.org/10.1109/UCC.2011.25.
8. Geuer-Pollmann C. The venus c generic worker. SICS Cloud Day 2011.
9. Luckow A, Jha S, Kim J, Merzky A, Schnor B. Distributed replica-exchange simulations on production environments using saga and migol. *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ESCIENCE '08, IEEE Computer Society: Washington, DC, USA, 2008; 253–260, doi:10.1109/eScience.2008.20. URL http://dx.doi.org/10.1109/eScience.2008.20.
10. Luckow A, Lacinski L, Jha S. SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems. *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2010; 135–144. URL http://doi.ieeecomputersociety.org/10.1109/CCGRID.2010.91.
11. Khamra YE, Jha S. Title: Developing Autonomic Distributed Scientific Applications: A Case Study From History Matching Using Ensemble Kalman-Filters. *GMAC '09: Proceedings of the 6th international conference industry session on Grids meets autonomic computing*, IEEE: New York, NY, USA, 2009; 19–28, doi:http://doi.acm.org/10.1145/1555301.1555304.
12. Thota A, Luckow A, Jha S. Efficient large-scale replica-exchange simulations on production infrastructure. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 2011; **369**(1949):3318–3335, doi:10.1098/rsta.2011.0151. URL http://rsta.royalsocietypublishing.org/content/369/1949/3318.abstract.
13. Mantha PK, Luckow A, Jha S. Pilot-MapReduce: An Extensible and Flexible MapReduce Implementation for Distributed Data. *Proceedings of third international workshop on MapReduce and its Applications*, MapReduce '12, ACM: New York, NY, USA, 2012; 17–24, doi:http://doi.acm.org/10.1145/2287016.2287020.
14. Amazon EC^2 Web Service. URL http://ec2.amazonaws.com.
15. Amazon S3 Web Service. URL http://s3.amazonaws.com.
16. Google Compute Engine. http://cloud.google.com/products/compute-engine.html.
17. Google Big Query. https://developers.google.com/bigquery/docs/overview.
18. Google Prediction API. https://developers.google.com/prediction/.
19. Amazon elastic mapreduce. http://aws.amazon.com/elasticmapreduce/.
20. Openstack. http://www.openstack.org/.
21. Eucalyptus. URL http://www.eucalyptus.com/.
22. Future Grid. URL http://www.futuregrid.org/.
23. Newhouse S, Drescher M. Egi cloud integration profile. http://indico.egi.eu/indico/getFile.py/access?resId=0&materialId=1&confId=415 2011.
24. Figueiredo RJ, Wolinsky DI, Chuchaisri P. Educational virtual clusters for on-demand mpi/hadoop/condor in futuregrid. *TG '11: Proceedings of the 2011 TeraGrid Conference*, ACM: New York, NY, USA, 2011; 1–2, doi:http://doi.acm.org/10.1145/2016741.2016793.
25. Baron J, Schneider R. Storage options in the aws cloud. http://media.amazonwebservices.com/AWS_Storage_Options.pdf 2010.
26. Windows Azure Blob Storage. https://www.windowsazure.com/en-us/develop/net/fundamentals/cloud-storage/.
27. Google Cloud Storage. https://developers.google.com/storage/.
28. Eucalyptus Walrus. URL http://open.eucalyptus.com/wiki/EucalyptusStorage_v1.4.
29. OpenStack Swift. http://swift.openstack.org.
30. Calder B, Edwards A. Windows azure drive. http://download.microsoft.com/download/9/7/0/97097468-CD68-4FD6-967D-C6C99B39A498/Windows%20Azure%20Drive%20-%20February%202010v2.pdf 2010.
31. Google Compute Engine Disks. https://developers.google.com/compute/docs/disks.
32. Eucalyptus Block Storage. http://open.eucalyptus.com/wiki/EucalyptusBlockStorage_v2.0.
33. Nova Volume. http://docs.openstack.org/essex/openstack-compute/starter/content/Nova-volume-d1e2000.html.
34. Oracle. Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System (White Paper) 2007.
35. Schmuck F, Haskin R. Gpfs: A shared-disk file system for large computing clusters. *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, USENIX Association: Berkeley, CA, USA, 2002. URL http://dl.acm.org/citation.cfm?id=1083323.1083349.
36. GFFS Global Federated File System. http://genesis2.virginia.edu/wiki/Main/GFFS.
37. ASim et al. GFD.154: The Storage Resource Manager Interface Specification V2.2. *Technical Report* 2008. Global Grid Forum.
38. Dean J. Designs, lessons and advice from building large distributed systems. http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf 2009.
39. Metz C. Facebook tackles (really) big data with project prism. http://www.wired.com/wiredenterprise/2012/08/facebook-prism/ 2012.
40. Apache Hadoop. http://hadoop.apache.org/ 2011.
41. Hortonworks. http://www.hortonworks.com/.
42. Hadoop on Azure. https://www.hadooponazure.com/.
43. 1000 Genomes Project and AWS. http://aws.amazon.com/1000genomes/.
44. Schatz MC. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics* Jun 2009; **25**(11):1363–1369, doi:10.1093/bioinformatics/btp236. URL http://dx.doi.org/10.1093/bioinformatics/btp236.
45. Evangelinos C, Hill CN. Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. Cloud Computing and Its Applications, 2008. URL http://www.cca08.org/speakers/evangelinos.php.

46. Mehrotra P, Djomehri J, Heistand S, Hood R, Jin H, Lazanoff A, Saini S, Biswas R. Performance evaluation of amazon ec2 for nasa hpc applications. *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, ScienceCloud '12, ACM: New York, NY, USA, 2012; 41–50, doi:10.1145/2287036.2287045. URL http://doi.acm.org/10.1145/2287036.2287045.
47. Gunarathne T, Wu TL, Qiu J, Fox G. Cloud computing paradigms for pleasingly parallel biomedical applications. *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ACM: New York, NY, USA, 2010; 460–469, doi:http://doi.acm.org/10.1145/1851476.1851544.
48. Sehgal S, Erdelyi M, Merzky A, Jha S. Understanding application-level interoperability: Scaling-out mapreduce over high-performance grids and clouds. *Future Generation Computer Systems* 2011; **27**(5):590–599, doi:DOI:10.1016/j.future.2010.11.001. URL http://www.sciencedirect.com/science/article/B6V06-51KT861-2/2/0092927785fbf5e065a7788af8b65b72.
49. Luckow A, Jha S. Abstractions for loosely-coupled and ensemble-based simulations on azure. *Cloud Computing Technology and Science, IEEE International Conference on* 2010; :550–556doi:http://doi.ieeecomputersociety.org/10.1109/CloudCom.2010.85.
50. Goodale T, Jha S, Kaiser H, Kielmann T, Kleijer P, Merzky A, Shalf J, Smith C. A Simple API for Grid Applications (SAGA). *OGF Recommendation Document, GFD.90*, Open Grid Forum 2007. http://ogf.org/documents/GFD.90.pdf.
51. SAGA Bliss. https://github.com/saga-project/bliss/wiki.
52. Pilot API. http://saga-project.github.com/BigJob/apidoc/.
53. Redis. http://redis.io/ 2012.
54. boto: A Python interface to Amazon Web Services. http://docs.pythonboto.org/.
55. Google APIs Client Library for Python. https://developers.google.com/api-client-library/python/.
56. Amazon ec2 api. http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/.
57. XSEDE: Extreme Science and Engineering Discovery Environment. https://www.xsede.org/.
58. Foster I. Globus online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Computing* 2011; **15**:70–73, doi:http://doi.ieeecomputersociety.org/10.1109/MIC.2011.64.
59. Allcock W. GridFTP: Protocol Extensions to FTP for the Grid. *OGF Recommendation Document, GFD.20*, Open Grid Forum 2003. http://ogf.org/documents/GFD.20.pdf.
60. Amazon s3 api. http://aws.amazon.com/archives/Amazon-S3/.
61. Li H, Durbin R. Fast and accurate long-read alignment with burrows wheeler transform. *Bioinformatics* March 2010; **26**:589–595, doi:http://dx.doi.org/10.1093/bioinformatics/btp698. URL http://dx.doi.org/10.1093/bioinformatics/btp698.
62. Picard. http://picard.sourceforge.net 2012.