J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Pilot-Data: An abstraction for distributed data

Andre Luckow [a], Mark Santcroos [a,b], Ashley Zebrowski [a], Shantenu Jha [a,*]

[a] RADICAL, ECE, Rutgers University, Piscataway, NJ 08854, USA

[b] Bioinformatics Laboratory, AMC, University of Amsterdam, The Netherlands

HIGHLIGHTS

- Pilot-Data as a unified, interoperable access to heterogeneous infrastructures.
- An infrastructure-agnostic abstraction for coupling distributed compute/data.
- A Pilot-Data-based workload management service for co-scheduling compute/data.
- The use of Pilot-Data for genome sequencing on large-scale, distributed resources.

ARTICLE INFO

Article history:
Received 23 November 2013
Received in revised form
24 May 2014
Accepted 17 September 2014
Available online xxxx

Keywords:
Data-intensive
Distributed computing
Pilot-jobs
Grid
HTC
HPC
Cloud
Bigdata

ABSTRACT

Scientific problems that depend on processing large amounts of data require overcoming challenges in multiple areas: managing large-scale data distribution, controlling co-placement and scheduling of data with compute resources, and storing, transferring, and managing large volumes of data. Although there exist multiple approaches to address each of these challenges and the complexity of distributed environments, an integrative approach is missing; furthermore, extending existing functionality or enabling interoperable capabilities remains difficult at best. We propose the concept of *Pilot-Data* to address the fundamental challenges of co-placement and scheduling of data and compute in heterogeneous and distributed environments with interoperability and extensibility as first-order concerns. Pilot-Data is an extension of the Pilot-Job abstraction for supporting the management of data in conjunction with compute tasks. Pilot-Data separates logical data units from physical storage, thereby providing the basis for efficient compute/data placement and scheduling. In this paper, we discuss the design and implementation of the Pilot-Data prototype, demonstrate its use by data-intensive applications on multiple production distributed cyberinfrastructure and illustrate the advantages arising from flexible execution modes enabled by Pilot-Data. Our experiments utilize an implementation of Pilot-Data in conjunction with a scalable Pilot-Job (BigJob) to establish the application performance that can be enabled by the use of Pilot-Data. We demonstrate how the concept of Pilot-Data also provides the basis upon which to build tools and support capabilities like affinity which in turn can be used for advanced data–compute co-placement and scheduling.

1. Introduction

Data generated by scientific applications, instruments and sensors is experiencing an exponential growth in volume, complexity and scale of distribution, and has become a critical factor in many science disciplines [38]. The ability to analyze prodigious volumes of data requires flexible and novel ways to manage distributed data and computations. Furthermore, analytical insight is increasingly dependent on integrating different and distributed data sources, computational methods and computing resources.

Scientific data is often inherently distributed either because it is generated by geographically dispersed data sources (e.g. instruments and/or simulations) or it needs to be distributed to facilitate further processing of the data (as done e.g. in the LHC grid [43]). Working with distributed data involves many challenges beyond its storage and management. A specific challenge is that of data and compute (processing) and the difficulty in being able to effectively and reliably manage co-placement and scheduling is in part due to the challenges inherent in coordination in distributed environments; it is compounded by an increasingly rich, but complex heterogeneous data-cyberinfrastructure, characterized by diverse

* Corresponding author.
E-mail address: shantenu.jha@rutgers.edu (S. Jha).

http://dx.doi.org/10.1016/j.jpdc.2014.09.009
0743-7315/

storage, data management systems and multiple transfer protocols/mechanisms.

Furthermore, tools and data-cyberinfrastructure have not been able to address the need to integrate distributed compute and data resources and capabilities [54,37]. Many solutions focus on either data or compute aspects, leaving it to the application to integrate compute and data; in addition, most currently available scientific applications still operate in legacy modes, in that they often require manual data management (e.g. the stage-in and out of files) and customized or application-specific scheduling.

Although these challenges have existed for a while, they are having progressively greater impact on the performance and scalability of scientific applications. For example, Climate Modeling as performed by the Earth System Grid Federation [17] is inherently a distributed data problem. The overall data generated and stored is 2–10 PB, of which the most frequently used data is 1–2 PB in size. The data is generated by a distributed set of climate centers, and it is stored in a distributed set of federated archives. It is used by a distributed set of users, who either run data analyses on a climate center with which they are associated, or they gather data from the ESGF to a local system for their analyses. Furthermore, data which is generated over time causes real-time changes—spatial and temporal; the scheduling of data analysis jobs needs to be responsive to these spatio-temporal data changes.

To alleviate barriers to scalability and dynamic execution modes, and impediments from an increasingly diverse and heterogeneous infrastructure, some of the questions that must be addressed include: (i) what are the right abstractions for coupling compute and data that hold for a range of application types and infrastructures? (ii) How can these utilize existing and well-known abstractions and not require whole-scale refactoring of applications and tools? (iii) How can the inherent heterogeneity and complexity of distributed cyberinfrastructure be managed? In addition to addressing the challenge of providing interoperable, uniform access to heterogeneous distributed cyberinfrastructure, how can these abstractions also be used to provide the ability to reason about "what" and "when" to distribute as well as "how?" Can these abstractions also enable effective and novel execution modes for data-intensive applications? Whereas no single abstraction (or paper) can answer all of these questions, in this paper we introduce *Pilot-Data (PD)* as a novel abstraction for data-intensive applications that addresses the first three questions, and equally importantly, outlines a research path to understanding other questions.

Pilot-Data (PD) is an extension of the Pilot-Job abstraction and supports the management of data in conjunction with compute tasks. It provides flexible placement and scheduling capabilities for data by separating the allocation of physical storage and application-level data. Pilot-Data is based on the abstraction provided by Pilot-Jobs, which has a demonstrable record of effective distributed resource utilization and supporting a broad range of application types [47,46]. We explore how Pilot-Jobs and Pilot-Data can be used to efficiently manage distributed data and compute on a dynamic set of heterogeneous resources. For this purpose, Pilot-Data provides a simple and useful notion of distributed logical location that from an application's perspective is invariant over the lifetime; thus it supports both a decoupling in time and space (i.e., allowing late-binding) between actual physical infrastructure and the application usage of that infrastructure. The suggestion that Pilot-Data is a conceptual abstraction for distributed data is predicated upon the fact, that like any valid abstraction, it must provide a range of applications with a unifying programming model and usage mode. Pilot-Data must thus retain the flexibility to be used with different CI whilst not constrained to different specific modes of execution or usage. As we will discuss, Pilot-Data provides a general approach to data–compute coupling, in that it is not constrained to a specific scheduling algorithm or infrastructure. Pilot-Data defines a minimal interface for resource acquisition and usage providing applications and frameworks full control and thus, a high flexibility, on how these resources are used. We will utilize Pilot-Data to examine the general challenges and issues in the specific context of BWA—a well known Next-Generation Sequencing (NGS) analysis application [44].

Our focus is on addressing the compute–data management and scheduling problem in the context of *production* distributed computing infrastructure (DCI) and not research infrastructures. Whether it be OSG/EGI (infrastructure with O(1000) sites) or XSEDE/PRACE (infrastructure with O(10) sites) or clouds, the ability to reason about data placement strategies (data replication and/or partitioning) and resource allocation strategies (compute-to-data vs. data-to-compute), when to offload/distribute is required. The realization and solution to these high-level questions however vary significantly between infrastructures. This points to the role of conceptual abstractions which enable reasoning without having to worry about implementation details for a given capability. We acknowledge that there exist multiple other challenges viz., data security, data access rights and policy, and data semantics and consistency. These are all important determinants of the ultimate usability and usage modes but we will not consider them to be in scope of the work of this paper. Our decision is in part explained by the fact that our work is ultimately aimed towards the development of abstractions and middleware for production distributed cyberinfrastructure (DCI) such as EGI [23], PRACE [59], XSEDE [75], and OSG [58], which will be agnostic to specific security and data-sharing policies.

This paper is structured as follows: in Section 2, we provide the reader with a better appreciation for the scope and context of our work with production distributed infrastructures in mind. We discuss related work in Section 3. Section 4 presents a detailed overview of Pilot-Data—the concept, its relation to Pilot-Job and its implementation in BigJob. Section 4 also introduces the Pilot-API as means of providing a common interface to Pilot-Jobs and Pilot-Data and exposing the joint capabilities to support data–compute placement. In Section 5 we present the design and implementation of a Pilot-based workload management service specifically designed for data-intensive applications. We design and conduct a series of experiments in Section 6 in order to establish and evaluate Pilot-Data as an abstraction for distributed data. We conclude with a discussion of the main lessons learned as well as relevant and future issues.

2. Infrastructure for distributed data

The landscape of solutions that have been devised over the years to address the challenges and requirements of distributed data is vast. In this section we provide a brief discussion of relevant cyberinfrastructure for supporting data-intensive applications. Further, we briefly survey data management in production DCI, such as XSEDE, OSG and EGI.

2.1. Software infrastructure for data management

Traditionally HPC systems provided separated storage and compute systems, which led to various inefficiencies in particular with the increasing scale mainly due to the fact that data always needs to get moved out of the storage system in order to facilitate processing. Commonly, parallel file systems, e.g. Lustre [56] and GPFS [68], have been used to manage data in conjunction with parallel applications. While these systems enable applications to access data via a standard POSIX interface, a drawback is that they do not expose data locality via these interfaces, i.e. commonly an application does not have control about file system internal data movements and caching.

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

With the emergence of distributed computing, different remote interfaces to storage resources and data transfer have been developed, such as the Storage Resource Manager (SRM) [70], GridFTP [2] or Globus Online [26]. SRM is a type of storage service that provides dynamic file management capabilities for shared storage resources via a standardized interface. SRM is primarily designed as an access layer with a logical namespace on top of different site-specific storage services. SRM aims to hide the complexity of different low-level storage services, but does not allow applications to control and reason about geographically distributed data. Various implementations of SRM – each optimized for a particular use case – exist: dCache [28], Castor [15], StoRM [18] and DPM [21] to name a few. While Castor and dCache e.g. support the management of tape and disk storage hierarchies, DPM and StoRM are more lightweight and focus on disk-based storage. SRM is heavily utilized in HTC environments to accommodate storage and access to larger volumes of data.

Several distributed data management systems have been built on top of these low-level storage systems to facilitate the management of geographically dispersed storage resources. The Global Federated Filesystem (GFFS) [32] for example provides a global namespace on top of a heterogeneous set of storage resources. Storage systems can be accessed via different mechanisms, e.g. the virtual filesystem layer in Linux or a transfer protocol, such as GridFTP.

iRODS is a comprehensive distributed data management solution designed to operate across geographically distributed, federated storage resources. iRODS [62] combines storage services with services for metadata, replica, transfer management and scheduling. Central to iRODS are the so called micro-services, i.e. the user defined control logic. Micro-services are automatically triggered and handle pre-defined tasks, e.g. the replication of a dataset to a set of resources. Also, different services covering singular aspects such as replica management (e.g. the Replica Location Service (RLS) [16] or the LCG File Catalogue (LFC) [10]) exist.

A main limitation of current infrastructures is the fact that they treat data and compute differently and require the user to (often painfully) manage data and compute resources separately. A reason for the limited number of higher-level services for data management and integrated compute/data capabilities is the complexity and variety of distributed applications that make it difficult to foresee a particular data access pattern. Thus, file placement is mostly handled by the application and at best supported by application-level services. Also, the available systems do not provide defined quality-of-service and applications are typically unaware of what throughput and latencies to expect. Both limitations emphasize the importance of higher-level application abstractions for distributed data/compute placements that enable applications to trade-off different aspects at runtime.

However, it also must be noted that some systems have emerged that attempt to blur the lines between compute and data. Hadoop [6] for example aims to address this issue by providing an integrated system for compute and data. Hadoop is optimized for data-intensive, write-once/read-many and sequential read workloads at the cost of Posix compliance. Also, it tightly couples compute and data, i.e. the compute framework MapReduce is directly linked to the underlying Hadoop Distributed File System (HDFS). A main limitation of Hadoop is the fact, that it is constrained to localized clusters and does not support distributed data very well [14].

2.2. Production cyberinfrastructure

Data management has become an increasingly important task on production infrastructures. In this section we explore the status of data capabilities in "distributed" HPC and HTC infrastructure. HPC infrastructures, such as XSEDE [75], are primarily concerned with compute-intensive tasks and thus, lack some distributed data/compute management services that are provided in HTC environments, such OSG [3] and EGI [23].

Table 1 summarizes the data cyberinfrastructures deployed by the different production DCIs. The landscape of data cyberinfrastructure is very heterogeneous with most applications only utilizing local capabilities, e.g. parallel filesystems. With the increasing need for supporting distributed data and compute, infrastructures started to deploy more sophisticated capabilities, e.g. XSEDE and OSG provide iRODS support for some resources. In particular, on high-throughput infrastructures distributed data access is essential since distributed filesystems are commonly not provided. Thus, a myriad of data management options emerged on HTC infrastructures. Historically, EGI and OSG introduced SRM as unified access layer for storage resource pools co-located to their compute resources. In addition, OSG provides with iRODS a higher-level data management services, which supports simple means of managing data in a highly distributed environment of compute resources. For example, data replication can be used to replicate datasets to a group of resources to facilitate distributed computation at a later stage. However, the application is still required to manually manage the mapping between these compute and storage resources.

Further infrastructures provide service for managing meta-data and information for data/compute resources. EGI and OSG e.g. offer services for replica management and for resource information (BDII [71]). However, the application is then required to combine these building blocks and to manually construct a resource topology. A reason for this is that the complexity and variety of distributed applications make it difficult to foresee a particular data access pattern. Thus, file placement is mostly handled by the application and at best supported by application-level services. A notable exception is domain specific infrastructures, e.g. the LHC Grid. Nevertheless, this emphasizes the importance of generic, higher-level application abstractions for distributed data/compute placements that can serve broader communities.

In addition to the described services, several domain-specific and higher-level approaches for distributed compute/data management emerged. For example, in context of the Atlas collaboration [49], various tools for managing distributed data have been developed on top of OSG. XRootD [20] e.g. is a distributed storage system that is capable of managing data across geographically dispersed resources using a hierarchical management structure. Based on this lower-level infrastructure various tools for managing computed data exist: the ROOT and PROOF systems e.g. enable the analysis of data stored in XRootD. Further, as part of this infrastructure the Pilots (e.g. PanDA [49]) are used to manage compute in conjunction with datasets residing on SRM or XRootD (see Section 3).

In the cloud space a separate ecosystem of storage services has emerged. A novel type of storage introduced by cloud environments is object stores, a form of highly distributed storage that can potentially be distributed 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 [4] automatically replicates data across multiple data centers within a region. These kinds 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 [74], Amazon S3 [4] and Google Cloud Storage [36], provide only a namespace with a 1-level hierarchy. This means that applications need to be adapted, in order to benefit from object storage. In addition, both Eucalyptus and OpenStack

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

Table 1
Data-cyberinfrastructure.

	XSEDE	OSG	EGI	Atlas/OSG
Storage	Local, Parallel Filesystems	Local, SRM, iRODS	Local, SRM	Local, SRM
Data access	SSH, GridFTP, Globus Online	SSH, SRM, iRODS	SSH, SRM	SSH, SRM
Management	Manual	Manual, iRODS, BDII	Manual, BDII	XRootD, PD2P

provide an object store: Eucalyptus Walrus [22] and OpenStack Swift [55]. A major limiting factor is the necessity to ingest large volumes of data to the cloud storage over the WAN. Large volume data transfers are associated with high costs and unpredictable and/or unacceptable performance. Also, data typically has to be moved to the compute resource (usually a VM) for processing.

While there are various useful services and building blocks for data-intensive application available, they typically require the application to utilize specialized access libraries and tools as well as to manually manage compute/data co-placements by providing the right resource constraints to the scheduler. Higher-level abstractions for compute and data and smart data/compute placement services are missing capabilities of existing infrastructures. While data placement strategies are extensively investigated (see e.g. [48]), currently most production DCI do not support distributed data placements. Having integrated compute/data capabilities and the ability to manage dynamic compute/storage resources is an essential requirement for effectively supporting and scaling dynamic and distributed applications on production infrastructures and thereby overcoming the currently prevailing inflexible execution model.

3. Related work

In this section, we explore the related work with respect to (i) existing systems and algorithms for managing distributed data and compute, (ii) abstractions and programming models for data-intensive applications, (iii) data management in the context of Pilot-Jobs.

Distributed data/compute management systems and algorithms: Managing distributed data and compute has been an ongoing research topic. For grid environments for example, the Stork [42] data-aware batch scheduler provides advanced data and compute placement for Condor and DAGMan. Stork supports multiple transfer protocols like, SRM, (Grid)FTP, HTTP and SRB. Romosan et al. [65] present another data–compute co-scheduling approach on top of Condor and SRM. Both approaches build on top of existing job scheduling and data-transfer and storage solutions. Further frameworks for other distributed environments have been proposed. FRIEDA [33] for example provides a data management framework for cloud-environments.

Different researches on when to (potentially dynamically) distribute and replicate data has been conducted: for example, Foster [63] and Bell [12] investigate different data replication management systems and dynamic replication algorithms in the context of scientific data grids. A limitation of the previous approaches is that the systems and algorithms are usually constrained to system-level replication, making it difficult for the user to control replication on application-level and employ dynamic replication strategies. Glatard et al. [48] attempt to provide a classification of data placement and replications algorithms and systems for distributed environments.

Abstractions and programming models: Various abstractions for optimizing access and management of distributed data have been proposed: Filecule [1] is an abstraction that groups a set of files that are often used together, allowing an efficient management of data using bulk operations. This includes the scheduling of data transfers and/or replications. Similar file grouping mechanisms have been proposed by Amer et al. [5], Ganger et al. [30] and BitDew [24]. Further several higher-level, less resource-oriented abstractions for enabling data analysis on large volumes of data have been proposed. A well-known example is the MapReduce programming model [19] for which various implementations exist [6,52]. Another example is DataCutter [13], a framework that enables exploration and querying of large datasets while minimizing the necessary data movements. While various abstractions for data-intensive applications exist, these are typically bound to a specific infrastructure. For example, Hadoop – the most-widely used MapReduce implementation – intermingles resource management, programming abstraction in a monolithic solution sacrificing flexibility and extensibility with respect to other kinds of data-intensive workloads.

Data management and Pilot-Jobs: Pilot-Jobs have been successful abstractions in distributed computing as evidenced by a plethora of Pilot-Job (PJ) frameworks. With the increasing importance of data, Pilot-Jobs have been also used to process and analyze large data. However, in most Pilot-Job frameworks the support for data movement and placement is insufficient [47]. Only a few of them provide integrated compute/data capabilities, and where they exist, they are often non-extensible and bound to a particular infrastructure. In general, one can distinguish two kinds of data management: (i) the ability to stage-in/stage-out files from another compute node or a storage backend, such as SRM and (ii) the provisioning of integrated data/compute management mechanisms. An example for (i) is Condor-G/Glide-in, which provides a basic mechanism for file staging and also supports access to SRM. Another example is Swift [73], which provides a data management component called Collective Data Management (CDM). DIANE provides in-band data transfer functionality over its CORBA channel.

In the context of the LHC Grid several type (ii) Pilot-Job frameworks that support access to the vast amounts of experimental data created by the Large Hadron Collider have been developed. DIRAC [72] is an example of such a system. It interfaces to SRM storage resources and enables the application to stage-in/out data to this system. AliEn [9] also provides the ability to tightly integrate storage and compute resources and is also able to manage file replicas. While all data can be accessed from anywhere, the scheduler is aware of data localities and attempts to schedule compute close to the data. Similarly, PanDA [51] provides support for the retrieval of data from the XRootD storage infrastructure. The PanDA Dynamic Data Placement component [50] provides a demand-based replication system, which can replicate popular datasets to underutilized resources for later computations. However, this capability is provided on system-level and constrained to official Atlas datasets, i.e. it cannot be applied to user-level datasets. The data/compute management capabilities of AliEn and PanDA are built on top of Condor-G/Glide-in. In addition to this strong coupling to the underlying infrastructure, these frameworks are tightly bound to their specific applications.

Another example for a type (ii) system is Falkon [60], which provides a data-aware scheduler on top of a pool of dynamically acquired compute and data resources [61]. The so called data diffusion mechanism automatically caches data on Pilot-level enabling the efficient re-use of data. Falkon provides limited interoperability and is constrained to Globus-based grid environments.

As alluded before, MapReduce is a popular abstraction for expressing data-intensive, analytical applications. Hadoop is the most widely-used implementation of MapReduce and provides a

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

vertical stack consisting of a distributed filesystems, a data-aware job scheduler, the MapReduce framework as well as various other frameworks. With the increasing variety of Hadoop-based applications and frameworks, the requirements with respect to resource management increased, e.g. it became a necessity to support batch, streaming and interactive data processing. YARN [7] is the new central resource manager of Hadoop 2.0 that was developed to address these needs. While YARN solves some of these problems, it has some limitations: it provides e.g. only a very low-level abstraction for resource management; data locality needs to be manually managed by the application by requesting resources at the location of a file chunk. With the need for supporting even more heterogeneous workloads, Pilot-like frameworks for Hadoop emerged, e.g. Llama [45] and Tez [8]. Another scheduler proposed for Hadoop is Mesos [39]: in contrast to YARN, Mesos is a two level scheduler, i.e. similar to Pilots, resources are initially requested and afterwards directly managed by the application.

4. Pilot-Data: a unified abstraction for compute and data

Pilot-Data was conceived as a unified abstraction to distributed data management in conjunction with Pilot-Jobs—a capability that has been neglected by many Pilot-Job frameworks. *A Pilot-Job is defined as an abstraction that generalizes the reoccurring concept of utilizing a placeholder job as a container for a set of compute tasks* [47]. From a practical point-of-view, data management and movement for most Pilot-Jobs – if it exists at all – is at best ad hoc and not generic. Consequently most Pilot-Jobs rely on application-level data management, i.e. data needs to be pre-staged or each task is responsible for pulling in the data.

Pilot-Data (PD) is an extension of the Pilot-Job abstraction for supporting the management of data in conjunction with compute tasks. PD separates logical compute and data from physical resource enabling efficient compute/data placements using various strategies (e.g. moving compute to data and vice versa, opportunistic replication, partitioning) independent from the underlying infrastructure. Pilot-Data provides a well-defined semantic for data movement, storage and access in conjunction with compute carried out through Pilot-Jobs. Pilot-Data separates application-level data and compute tasks from infrastructure-level storage/compute enabling efficient compute/data placements. Like Pilot-Jobs, it allows for application-level control, and the logical decoupling of physical storage/data locations from the production and consumption of data. In summary, Pilot-Data provides the following key capabilities:

1. *Dynamic resource management:* It supports the management and access to (dynamic) storage resources in conjunction with Pilot-Jobs. For this purpose, it provides a unified access layer to different heterogeneous data cyberinfrastructures, such as: SRM, iRODS, Globus Online and S3. Pilot-Data facilitates and utilizes late binding of data and physical resources for optimal coupling and management. The framework is agnostic to the type of data and can be used to manage arbitrary data in distributed, dynamic environments.
2. *Distributed namespace for data:* Pilot-Data provides a simple, two-level distributed, global namespace spawning heterogeneous storage resources that can be accessed from any resource.
3. *Higher-level abstraction for compute/data coupling:* The Pilot-Data abstraction aims to support the coupling of different application components and the management of the data flow between the different application stages, e.g. the components of a (distributed) workflow. The Data-Unit abstraction provides the ability to group files. The Data-Unit URL serves as a single level namespace independent of the actual physical location of the Data-Unit, which can be e.g. replicated across multiple geographically distributed Pilot-Data. Further, the application is able to organize files into an application-level hierarchical namespace within a Data-Unit.
4. *Compute/data scheduling:* Pilot-Data supports the co-scheduling of compute and data into co-located Pilot-Compute and Pilot-Data using a data-aware workload management service. Also, applications can utilize lower-level primitives (i.e. the Pilot-Abstraction) to manually optimize data and compute placements.

Pilot-Data combines a unified abstraction for compute and data with an interoperable Pilot-based access layer to heterogeneous resources bridging often disperse compute and data cyberinfrastructures allowing applications to reason about data/compute universally.

4.1. Design objectives

In this section we derive the design objectives for Pilot-Data based on common data and processing patterns. Commonly applications deal with two types of data: static and dynamic data. The majority of data is static and resides in archives to which it is written once, but never modified; most of this data remains unanalyzed [31]. Commonly this type of data is shared between multiple users and applications. Pilot-Data can provide a unified abstraction for accessing these datasets using compute resources allocated via a Pilot. Dynamic data arises when some property of the input data or its delivery changes, for example in terms of arrival rate, provenance, burstiness, or source. There may be variability in the structure of the data, for example, data schema, file formats, ontologies, etc. [40]. A good example is data feeds generated by scientific instruments, experiments, simulations or dynamic workflows. With the dynamic nature of data the requirements with respect to compute grow: dynamic data comes with dynamic and potentially real-time processing requirements. In this case the lifecycle of the data is tightly bound to the lifecycle of compute. Thus, an integrative approach of compute/data management is essential. Pilot-Data can be used to allocate appropriate storage resources to meet the space and I/O requirements for dynamic data and facilitates the effective processing of this data in a Pilot-Compute.

Typically, scientific applications involve multiple steps of data generation and processing. Examples of application patterns are: ensembles, coupled ensembles, more complex pipelines possible comprising different kinds of raw and derived data [37], MapReduce-based applications and workflows. Often, input data is partitioned to facilitate the data-parallel processing of data, e.g. by an ensemble of tasks. In this case one can differentiate between (i) partitioned data, i.e. data that is divided in a way so that each task consumes a unique part of the data, and (ii) shared data that is required by all tasks. Dynamic data often arisen in multi-stage workflows where it is often difficult to predict the output of the previous stage.

Pilot-Jobs have been shown to be highly effective in supporting fine-grained ensemble tasks and proved particularly useful for applications with dynamic resource requirements. Pilot-Data aims to address (i) the data management challenge arising from data-intensive applications and (ii) the potentially dynamic compute/data requirements associated with the needs of dynamic and distributed compute/data trading of properties such as storage type (and associated IOPS), network bandwidth, compute capacity and data locality. Assuming a dynamic data feed from a scientific instrument as an example, if sufficient storage and bandwidth is available, data can be cached and then processed in a Pilot-Compute. If the application has realtime requirements, i.e. alatency requirement with respect to availability of results relative to data volume and computational complexity, resource management becomes even more challenging. Usually, this requires the usage of readily available compute resources from a pool of Pilots. In summary, Pilot-Data is designed to address the following usage modes:

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

1. Manage input and output for Pilot-based applications and provide access to user-owned as well as community datasets available on many infrastructures. Pilot-Data supports applications in exploring data parallelism, e.g. by allowing them to efficiently partition and/or replicate data. This way it allows applications to optimize data movements, e.g. to create a Pilot-local replica of the dataset to facilitate the faster processing of this data.
2. Manage *dynamic* data in different scenarios, e.g. within data-intensive, dynamic workflows or the intermediate data within MapReduce. In this cases it is necessary to create short-term, transient "storage space" for intermediate data, which can be removed after the end of the application run. Pilot-Data enables the reasoning about application and resources, such as data/compute localities and placements on top of a dynamic compute/data overlay.
3. Support common data processing patterns, such as data-partitioning, parallel processing and output gathering.

While there are many commonalities between compute and data, depending on the type of data there are distinct differences: the lifecycle of static data differs extremely from compute—data commonly outlives computes and is often consumed by different kinds of compute. Nevertheless, the management of such datasets is a challenge: commonly, these datasets are partitioned, filtered and replicated in the user space using a myriad of scripts making it difficult to track transformations and results. The lifecycle of transient, dynamic data is commonly strongly coupled to the lifecycle of the associated compute. Pilot-Data enables Pilot-based applications to acquire appropriate data resources needed for processing their datasets. It further enables them to manage file movements to and from these resources. By doing so, Pilot-Data allows Pilot-based applications to utilize data locality in conjunction with their Pilot-Computes. For example, applications can create dynamic caches in conjunction with their Pilot-Computes, which can be used for intermediate data or for the fan-out additional compute tasks.

4.2. BigJob: a Pilot-Compute and data implementation

Architecture and design: Consistent with our aims of providing complete Pilot-Job capabilities, we implemented Pilot-Data as an extension of BigJob (BJ) [46,52], which is a SAGA-based Pilot-Job implementation. BigJob provides a unified runtime environment for Pilot-Computes and Pilot-Data on heterogeneous infrastructures. The framework offers a higher-level interface – the Pilot-API – to heterogeneous and/or distributed data and compute resources. Fig. 1 shows the high-level architecture of BigJob. The Pilot-Manager is the central entity of the framework, which is responsible for managing the lifecycle of a set of Pilots (both Pilot-Computes and Pilot-Data). For this purpose BigJob relies on a set of resource adaptors (see adaptor pattern [29]).

A resource adaptor encapsulates the different infrastructure-specific semantics of the backend system, e.g. in the case of Pilot-Compute different resource management systems and in the case of Pilot-Data different storage types (e.g. file vs. object storage), access and transfer protocols. Using this architecture, BigJob eliminates the need for application developers to interact directly with different kinds of compute and storage resources, such as the batch queue of HPC/HTC resources or the VM management system of cloud resources.

As shown in Fig. 2 BJ supports various types of HPC/HTC resources via SAGA-Python [66] (e.g. Globus, Torque or Condor resources). Further, adaptors for cloud resources (Amazon EC2 and Google Compute Engine) exist. A Pilot-Data backend is defined by (i) the storage resource and (ii) the access protocol to this storage. On XSEDE, storage resources such as parallel filesystems (commonly Lustre or GPFS) can be remotely accessed using different

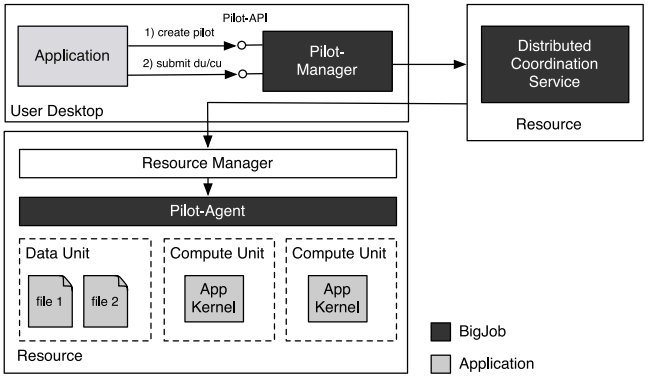

Fig. 1. *BigJob high-level architecture:* The Pilot-Manager is the central coordinator of the framework, which orchestrates a set of Pilots. Each Pilot is represented by a decentral component referred to as the Pilot-Agent, which manages the set of resources assigned to it.

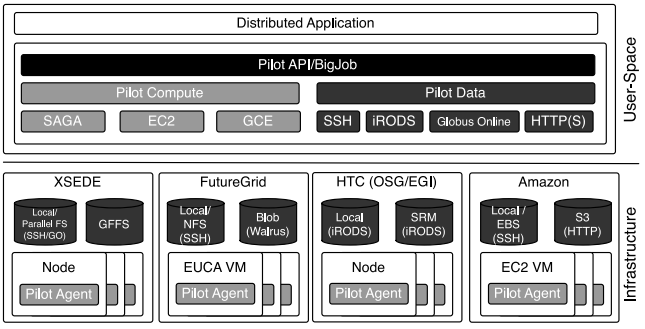

Fig. 2. *BigJob Pilot abstractions and supported resource types:* BigJob provides a unified abstraction to a heterogeneous set of distributed compute and data resources. Resources are either accessed via SAGA [34,66] or via a custom adaptor.

protocols and services, e.g. SSH, GSISSH, GridFTP [2] and Globus Online [26]. Other storage types, e.g. cloud object stores as S3 or iRODS, tightly integrate storage and access protocol and provide additional features such as data replication. Each Pilot-Data adaptor encapsulates a particular storage type and access protocol.

Runtime interactions: Pilots are described using a JSON-based description (see Pilot-API in Section 4.3), which is submitted to the Pilot-Manager. The description contains various attributes that are used for expressing the resource requirements of the Pilot. An important attribute is the backend URL of the resource manager (for Pilot-Computes) or the storage/transfer service (for Pilot-Data). The URL scheme is used to select an appropriate BigJob adaptor. Once an adaptor is instantiated, it is bound to the respective Pilot object; all resource specific aspects for this Pilot are then handled by this adaptor.

Fig. 3 shows the typical interactions between the components of the BigJob/Pilot-Data framework after the submission of the application workload (i.e. the CUs and DUs). The core of the framework is the Pilot-Manager. The Pilot-Manager is able to manage multiple Pilot-Agents. The application workload is submitted to the Pilot-Manager via the Compute-Data Service interface of the Pilot-API (see Section 4.3). After submission, DUs and CUs are put into an in-memory queue of the distributed coordination service, which is continuously processed by the scheduler component. This asynchronous interface ensures that the application can continue without needing to wait for BigJob to finish the placement of a CU or DU.

Distributed coordination and control management: The main task of the coordination and communication service is to facilitate control flow and data exchange between distributed components of the framework, i.e. the Pilot-Manager and Pilot-Agent. BigJob uses a shared in-memory data store, Redis [64], for this purpose.

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

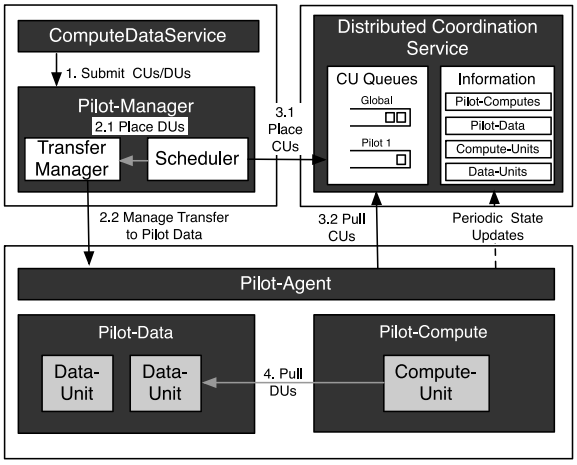

Fig. 3. *BigJob application workload management:* The figure illustrates the typical steps involved for placing and managing the application workload, i.e. the DUs and CUs.

Both manager and agent exchange various types of control data via a defined set of Redis data structures and protocols: (i) the Pilot-Agent collects various information about the local resource, which is pushed to the Redis server and used by the Pilot-Manager to conduct e.g. placement decisions; (ii) CU is stored in several queues. Each Pilot-Agent generally pulls from two queues: its agent-specific queue and a global queue. Since the Redis server is globally available, it also serves as central repository that enables the seamless usage of BigJob from distributed locations. That means that application can easily re-connect to a Pilot and Compute-Unit, via a unique URL.

Data management: BigJob supports two forms of data management: (i) in the push-based mode all data transfers are handled by the Pilot manager, (ii) in the pull mode the Pilot-Agent downloads the data before running a Compute-Unit. Further, there are two types of data: (i) data associated with a Pilot and (ii) data associated with a Compute-Unit. For each Pilot instance a sandbox is created; every Compute-Unit is assigned a directory in this sandbox. For every Compute-Unit both Pilot and Compute-Unit data files are made available in the sandbox of the Compute-Unit and can be accessed by the application via their I/O subsystem (e.g. the Posix API or a specialized I/O library, such as HDF5).

Fault tolerance: Ensuring fault tolerance in distributed environments is a challenging task [25]. BigJob is designed to support a basic level of fault tolerance. Failures can occur on many levels: on hardware, network, and software level. The complete state of BigJob is maintained in the distributed coordination service Redis, which stores the state both in-memory and on the filesystem to ensure durability and recoverability. Both the application and the Pilot-Manager can disconnect from running Pilot-Agent and re-connect later using the state within Redis. Also, the agent and manager are able to survive transient Redis failures. To address permanent Redis failures additional pre-cautions are required, e.g. a redundant Redis server setup with failover. Nevertheless, in most cases the ability to quickly restart the Redis server (if necessary on another resource) is sufficient. Another error source is file movements: Pilot-Data currently relies on the built-in reliability features of the transfer service; Globus Online e.g. automatically restarts failed transfers. In the future, we will provide fault tolerance also for non-benign faults, e.g. network partitions, resource slowdowns, etc.

4.3. Pilot-API: an abstraction for distributed data and compute

The Pilot-API [57] is an *implementation abstraction* providing a well-defined control and programming interface to Pilot-Jobs and

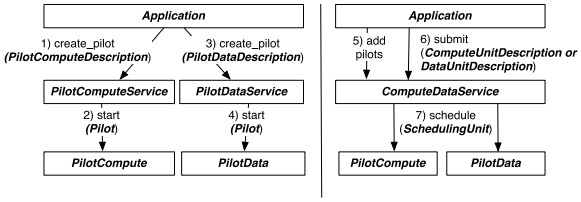

Fig. 4. The Pilot-API exposes two primary functionalities: the PilotComputeService and PilotDataService are used for the management of Pilot-Computes and Pilot-Data. The application workload is submitted via the Compute-Data Service.

Pilot-Data. It builds upon earlier work in the context of P* [47]—a *conceptual* model and abstraction that enabled the reasoning about Pilot-Jobs and Pilot-Data in a semantically consistent way. The Pilot-API is designed to be interoperable and extensible and exposes the core functionalities of a Pilot framework and can be used across multiple distinct production cyberinfrastructures.

Defining the right abstraction for managing computational, data-intensive tasks and distributed resources is a challenging task and requires trading-off contradictory objectives, such as simplicity vs. flexibility. One of the best-known resource management abstractions is a *process*. A process encapsulates an executing program providing the abstraction of a virtual CPU and memory [11]. While the illusion of infinite resources simplifies application development, distributed environments are generally too complex to maintain this abstraction. Thus, in a distributed environment a program commonly consists of multiple processes. Programs are executed via a resource manager using the job abstraction. A *job* denotes to the batch execution of a program without user intervention on a set of (possibly distributed) resources. In contrast to local processes, an application typically is required to specify the resources requirements, i.e. the number of cores/resource slots, memory, etc. Commonly, the job abstraction is used for managing compute and data-intensive applications in HPC and HTC environments. It also provides the basis for the SAGA job model [35]. The Pilot-API relies on a similar model; however, it separates resource allocation (i.e. the start of the Pilot) from the actual execution of the workload providing applications with the ability to use late-binding when assigning compute/data to resources. This approach is also referred to as multi-level scheduling. To support multi-level scheduling the Pilot-API provides two packages: (i) one for resource allocation and Pilot management and (ii) one for application workload management (see Fig. 4). In this section we describe the fundamental abstractions and usage models of both parts of the Pilot-API.

4.3.1. Resource allocation and pilot management

The first part of the Pilot-API is concerned with the management of the lifecycle of Pilots, i.e. Pilot-Computes and Pilot-Data. A Pilot-Compute allocates a set of computational resources (e.g. cores). A Pilot-Data is conceptually similar and represents a physical storage resource that is used as a logical container for dynamic data placement, e.g. for compute-local data replicas or for caching intermediate data.

A Pilot-Compute marshals the job running the Pilot-Agent; it is responsible for managing a set of resource slots acquired from the local resource manager. The instantiation of Pilot-Computes is done via a factory class, the Pilot-Compute Service, using a description object containing the resource requirements of the application, the Pilot-Compute Description. The description comprises a service URL referring to the resource manager used for instantiating the Pilot, a process count specifying the number of required resource slots and several optional (potentially backend-specific) attributes. Further, the Pilot-Compute API provides methods for managing the lifecycle of the agent job, i.e. for querying its current state and runtime attributes and for canceling it.

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

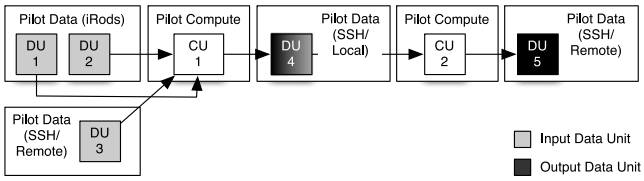

Fig. 5. *DU and CU interactions and data flow:* Each CU can specify a set of input DU. The framework will ensure that the DU is transferred to the CU. Having terminated the job, the specified output is moved to the output DU.

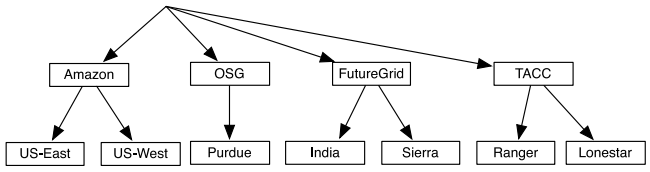

Fig. 6. *Affinities between distributed resources:* Pilot-Data assigns each resource an affinity based on a simple hierarchical model. The smaller the distance between two resources, the larger the affinity.

Similarly, Pilot-Data objects are created via the Pilot-Data Service class. A Pilot-Data refers to a physical storage location, e.g. a directory on a local or remote filesystem or a bucket in a cloud storage service. Similarly, lifecycle methods for the management and querying of the storage resource are provided.

Using Pilot-Abstractions different types of distributed compute resources, storage infrastructures and transport protocols can be marshaled into an application-specific resource overlay. Once an application has started a set of Pilot-Computes and Pilot-Data—the control of these resources is delegated to the application, which can then utilize them accordingly and optimize execution with respect to computational/memory requirements of its tasks and/or data locality.

4.3.2. Application workload management

The Pilot-API provides the Data-Units (DU) and Compute-Unit (CU) classes as the primary abstraction for expressing and managing application workloads. Using these two primitives, applications can specify computational tasks including their input and output files. A CU represents a self-contained piece of work, while a DU represents a self-contained, related set of data. A CU encapsulates an application task, i.e. a certain executable task to be executed with a set of parameters and input files. A DU is defined as an `immutable` container for a logical group of "affine" data files, e.g. data that is often accessed together e.g. by multiple Compute-Units. This simplifies distributed data management tasks, such as data placement, replication and/or partitioning of data and abstracting low-level details, such as storage service specific access details. A DU is completely decoupled from its physical location and can be stored in different kinds of backends, e.g. on a parallel filesystem, cloud storage or in-memory. Replicas of a DU can reside in different Pilot-Data.

A Compute-Unit is a computational task that operates on a set of input data represented by one or more Data-Units. Further, Data-Units can be bound to a Pilot-Compute facilitating the reuse of data between a set of Compute-Units, e.g. to efficiently support iterative applications. The output of a Compute-Unit can be written to a set of Data-Units. The runtime system ensures that the logical references to a DU will be resolved and ensures that the files are made available in the sandbox of the CU, i.e. if necessary the files corresponding to the DU will be moved. Using these two core abstractions for application workloads an application can compose complex application scenarios consistent of multiple Compute-Units and Data-Units. Both CUs and DUs are described by the use of a Compute-Unit-Description (CUDs) and Data-Unit-Description (DUDs) objects defined in the JSON format. A DUD contains all references to the input files that should be used to initially populate the DU.

Further, applications can express data/compute dependencies on an abstract, high level using the Pilot-API. Fig. 5 shows an example of a data flow between multiple phases of compute. As described, applications are required to organize their data in the form of DUs, which represents a logical group of files. A DU can be potentially placed in multiple Pilots to facilitate fault tolerance or a faster access. Applications can declaratively specify CUs and DUs and effectively manage the data flow between them using the Pilot-API. A CU can have input and output dependencies to a set of DUs. For this purpose, the API declares two fields in the Compute-Unit Description: `input_data` and `output_data` that can be populated with a reference to a DU. The runtime system ensures that these dependencies are met when the CU is executed, i.e. either the DUs are moved to a Pilot that is close to the CU or the CU is executed in a Pilot close to the DU's Pilot. In the best case, the Pilot-Data of the dependent DUs is co-located on the same resource as the CU, i.e. the data can be directly accessed via a logical filesystem link. Otherwise, the data is moved via a remote transfer. Further, a CU can constrain its execution location to a certain resource. Ultimately, the input data is made available in the working directory of the CU.

The Pilot-API provides various levels of control on how the application workload is managed: (i) applications can either bind their workload (i.e. CUs and DUs) directly to a Pilot (a Pilot-Compute or a Pilot-Data) using their own application-level scheduling mechanisms or (ii) applications can utilize a workload management service, such as the Compute-Data Service introduced in the next section. CU and DU descriptions are submitted to one of these services, which returns a Compute-Unit/Data-Unit instance. This instance can then be used for state queries and lifecycle management (e.g. canceling a CU).

5. Compute-Data Service: a workload management service based on affinities

Typically network bandwidth within clusters and even more in WAN settings are oversubscribed by a significant factor, ignoring the locality thus, can have a severe impact on an application's performance. Different investigations (e.g. [53,63]) have shown that considering data/compute entities equally while making placement decisions leads to performance gains. An important consequence of data and computation as equal first-class entities is that either data can be provisioned where computation is scheduled to take place (as is done traditionally), or compute can be provisioned where data resides. This equal assignment of Compute-Units leads to a richer set of possible correlations between the involved DUs and CUs; correlations can be either spatial and/or temporal. These correlations arise either as a consequence of constraints of localization (e.g. data is fixed, compute must move, or vice-versa), or as temporal ordering imposed on the different Data-Units and Compute-Units.

Resource affinities describe the relationship between a set of compute and/or storage resources. We use a simple model for describing resource affinities: data centers and machines are organized in a logical topology tree. The further the distance between two resources, the smaller their affinity. Fig. 6 shows how a distributed system consisting of different types of cloud and grid resources can be modeled. Using such a resource topology, the runtime system can deduce the connectivity between two resources, to estimate e.g. the costs induced by a potential data transfer. While this model is currently very coarse grained, it can be enhanced by assigning weights to each edge to reflect dynamical changes in

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

factors that contribute to connectivity. The affinity of a Pilot is determined based on the resource it is located on, i.e. the proximity of two Pilots is deduced from the distance of their resource in the resource topology tree. Currently the mapping between a resource and a Pilot is done by assigning each Pilot a logical location using a user-defined affinity label in the Pilot-Description. This logical location assignment is utilized by the scheduler to create the resource topology tree.

Compute/data affinities describe the relationships between DUs and CUs. A DU is the primary abstraction for the logical grouping of data. CUs can have input and output dependencies to a set of DUs, i.e. the data of these DUs is required for the computational phase of the CU. The output data is automatically written to one or more output DUs. The framework utilizes these affinities to place DUs and CUs into a suitable Pilot-Compute or Pilot-Data. Further, CUs and DUs can constrain their execution resource to a particular affinity (e.g. to a certain location or sub-tree in the logical resource topology). The runtime system then ensures that the data and compute affinity requirements of the CU/DU are met.

Pilot-based scheduling: BigJob provides a rudimentary but an important proof-of-concept affinity-aware scheduler that attempts to minimize data movements by co-locating affine CUs and DUs to Pilots with a close proximity. The scheduler is a plug-able component of the runtime system and can be replaced if desired. The default implementation relies on the resource topology and affinity attributes provided via the Pilot-API to reason about the relationships between DUs, CUs, Pilots and resources to optimize data localities and movements. The affinity-aware scheduler currently implements a simple strategy based on earlier research [53] that suggests that considering both data and compute during placement decisions leads to a better performance. As shown in Fig. 3, BJ relies on two queues for managing CUs. CUs without any affinity are assigned to the global queue from where they can be pulled from multiple Pilot-Agents. If there is affinity to a certain Pilot because the input data resides in this Pilot-Data, the CU can be placed in a Pilot specific queue. For each CU the following steps are executed:

1. The Pilot-Manager attempts to find a Pilot that best fulfills the requirements of the CU with respect to (i) the requested affinity and (ii) the location of the input data.
2. If a Pilot with the same affinity exists and Pilot has an empty slot, the CU is placed in this pilots queue.
3. If delayed scheduling is active, wait for *n* s and recheck whether Pilot has a free slot.
4. If no Pilot is found, the CU is placed in a global queue and pulled by the first Pilot which has an available slot.

The Pilot-Agent that pulls the CU from a queue is responsible for ensuring that the input DU is staged to the correct location, i.e. before the CU is run, the DU is made available in the working directory of the CU either via remote transfer or a logical link.

6. Experiments

It is important to appreciate that experiments that aim to characterize the performance of an abstraction are by their very nature difficult. We cite two primary reasons: the first is that an abstraction is only as good as the infrastructure that it is implemented on. Furthermore, what Pilot-Data provides is a uniform way of reasoning about compute–data distribution and implementing them, not necessarily new capabilities in and of themselves. Not surprisingly, our experiments do not aim to understand the performance of Pilot-Data per se, but the application performance that can be enabled by the use of Pilot-Data.

Before we discuss experiments in the next sub-section, we develop some minimal terminology that enables such reasoning across different modes of distribution and infrastructure, as well as understand the primary components and trade-offs to determine compute–data placement. We continue with the description of several experiments aimed at understanding three different aspects of Pilot-Data: (i) in Section 6.2 we demonstrate a proof of existence and correctness of Pilot-Data via the ability to provide uniformity of access and usage modes for different infrastructures (e.g. XSEDE and OSG); (ii) in Section 6.3 we discuss how Pilot-Data provides a conceptually simple and uniform framework to reason about how and when to distribute over very different and architecturally distinct infrastructures, and (iii) in Section 6.4 we discuss some advanced capabilities and performance advantages arising from the ability to use Pilot-Data to select "optimal" usage modes and support scalability of large-scale data-intensive applications.

6.1. Reasoning about compute–data placement

A question that arises in the design of systems and that distributed data-intensive applications have to address, is whether to assign and move computational tasks to where data resides, or to move data to where computational tasks can be executed. An associated question is when to commit to a given approach. Additionally, if replication is an option, applications and systems have to determine what the degree of replication of data should be, and possibly where to replicate. As an abstraction for distributed data, Pilot-Data must provide the ability to answer the above questions and implement the results. To programmatically determine the best approach and to understand Pilot-Data based experiments, the value of several parameters have to be considered:

- T_Q is defined as the queue waiting time at a given resource. $T_{Q_{\text{Pilot}}}$ is the queue time of the Pilot-Job. $T_{Q_{\text{Task}}}$ is defined as the Pilot-internal queuing time.
- T_C is defined as the compute time.
- T_S is the staging time, which is defined as the transfer time T_X plus the time to register the data into the system.
- $T_R(R)$ is defined as the time to replicate data, where R is the number of sites that data is replicated over.
- T_D is the time at which data will be accessible across all distributed resources. When replication is involved, it is defined as the sum of the $T_R(R)$ and T_S.

The relative values of the parameters above, provide the basis to reason about whether to process/compute where the data already resides, or whether to move data to where the processing power lies; they also provide insight on how to possibly distribute data or not. To a first approximation, which of the two approaches should be employed is given by the relative values of T_D and the typical value of T_Q. The appropriate mode is amongst other things, strongly dependent upon data volumes in consideration, as well as the capabilities of the tools and middleware in use.

When both data and compute can be scheduled, the decision about which entity to place first and which to move – compute to data or data to compute – is determined via a simple trade-off between T_Q and T_X. If the expected T_X is larger than the T_Q, then the compute is assigned to a site first, and subsequently data is placed. When data is already distributed, compute resources have to be chosen in response to this distribution. Resources co-located with data replicas, with the lowest queue waiting time present optimal choice. However, it is important to appreciate that there is an overhead in ensuring that data is replicated in a distributed fashion.

In practice hybrid modes can be employed. As an example, distributed data replication can initially be set to be partial, viz., only over a subset of possible distributed sites. In other words, replication might commence over a subset of suitably chosen nodes, followed by a sequential increase in the replication (factor) if compute resources close to the replica do not have sufficient compute

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

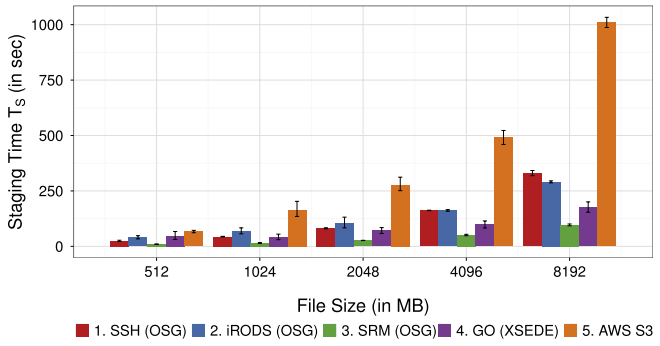

Fig. 7. *Pilot-Data on different infrastructures:* Time to instantiate a Pilot-Data with a dataset of given size. For iRODS, we measure only the staging time and not the time required to replicate the data. SRM performs best mainly due to the reliance on GridFTP as data transfer protocol. SSH and iRODS show an acceptable performance for smaller datasets. Globus Online is associated with some overheads due to its service-based nature, which is particularly visible for smaller data sizes. S3 is constrained by the limited bandwidth available to the Amazon data center.

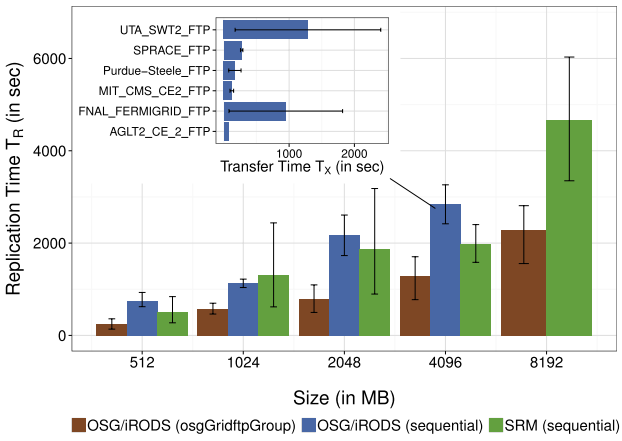

Fig. 8. *Using replication on OSG: T_R* on OSG: in scenario osgGridFTPGroup data input to 9 iRODS resources that are members of this group, in the sequential scenario 6 iRODS respectively SRM resources are used. The inset shows the distribution of T_R with respect to the different hosts for the 4 GB and OSG/iRODS scenario.

capacity. Currently these decisions are made manually, but eventually such scheduling and placement decisions will be driven by both application and system information; the affinity model discussed earlier provides one way of providing this information to the "scheduling engine".

As with any model, it is important to recognize its practical limitations. In many cases applications contest for shared resources, such as the network or shared storage systems. Thus, T_X e.g. will be dependent on additional external factors (such as the current network utilization).

6.2. Understanding Pilot-Data

The objective of the first set of experiments is to demonstrate the ability of Pilot-Data to marshal different storage backend infrastructures. We then characterize the performance of Pilot-Data on different cyberinfrastructures (e.g. XSEDE and OSG) by investigating the different components of the data distribution time T_D; some of these experiments involve investigating the impact of replication. For joint submission to XSEDE and OSG resources, we utilize GW68—a gateway node located at Indiana University and part of the XSEDE infrastructure.

In the first experiment we investigate T_S on different Pilot-Data backends. Fig. 7 illustrates T_S for different backends, i.e. the time necessary to populate a Pilot-Data on different infrastructures: in scenario 1 the PD is mapped to a directory on an OSG submission machine, in scenario 2 on an iRODS collection on the OSG iRODS infrastructure, in scenario 3 on an SRM directory, in scenario 4 on a directory on Lonestar accessed via Globus Online and in scenario 5 on an Amazon S3 bucket.

The performance primarily depends on the infrastructure used and in particular the available bandwidth between the submission machine and the storage backend. T_S is dominated by T_X, i.e. the time necessary to transfer files to the Pilot-Data location. Experiments with smaller data sizes have shown that T_{register} is negligible. Thus, the runtime is directly influenced by the available bandwidth and the characteristics of the respective transfer protocol. SRM on OSG clearly shows the best performance: SRM is a highly optimized storage backend which is in this scenario used with GridFTP a highly efficient data transfer protocol. Globus Online particularly performs well for larger data volumes: the service also utilizes GridFTP however adds an additional management layer. For smaller data volumes SSH is a better choice. The initialization for setting up an SSH connection is significantly lower than for the creation of a Globus Online request. With larger data volumes the initialization overhead becomes insignificant and Globus Online benefits from the more efficient GridFTP transfer protocol. T_S for iRODS behaves comparable to T_S for SSH. T_S for S3 increases linearly—an indicator that the bandwidth to the AWS site is a limiting factor. It is noteworthy that Pilot-Data can support various combinations of different storage and transfer protocols giving the application the possibility to chose an appropriate backend with respect to its requirements; e.g. while Globus Online is the best choice for large volume transfers within XSEDE, data-intensive applications are required to use iRODS.

In the second experiment, we evaluate the impact of replication (T_R) on T_D. If system-level support for replication is provided, e.g. by a distributed data management middleware such as iRODS, Pilot-Data can utilize this capability as a dynamic caching mechanism (to be contrasted with the usage of iRODS for data storage and management). In the following experiment, we investigate T_R for different infrastructures and configurations: (1) iRODS/OSG with group-based replication (osgGridFTPGroup), (2) iRODS/OSG with sequential replication in which one replica is created after the other and (3) SRM with sequential replication.

Fig. 8 illustrates the results: on OSG the T_R for the group-based replication with iRODS is significantly better than for the sequential replication. Note that the sites used for iRODS and SRM experiments do not fully overlap, so the results should not be considered as a performance comparison between the protocols, but merely give an impression about the scalability. In both cases of iRODS, the frequency of failures was very high. While the `osgGridFtpGroup` group consisted of 9 nodes, the average number of resources that actually received a replica was ~7.5. In general, the overall performance is determined by the available bandwidth between the central iRODS server (located at Fermilab near Chicago) and the individual sites. For 4 GB case in scenario (2), the individual T_X is depicted in the inset of Fig. 8.

OSG provides the user with a variety of storage services and thus, options to organize their compute and data. Abstraction such as Pilot-Data is important to provide a unified access to these services and to enable the application to reason about the distribution of their compute and data. While e.g. sequential replication is well suited for creating a small number of replicas, it is only beneficial when a small amount of additional compute resources for a dataset is required. In other cases, e.g. in order to support larger amounts of compute, an OSG wide replication using iRODS internal concurrent replication or an orchestrated concurrent SRM replication is beneficial. Another important observation is the fact that different sites have very different performance characteristics. Thus, the ability for applications to optimize data/compute

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

placements with respect to their computational and data requirement presents both plenty of opportunities but also is somewhat challenging. Pilot-Data provides a unified interface which allows applications to trade-off different infrastructure capabilities and characteristics to enable an efficient execution of its workload.

6.3. Understanding Pilot-Abstractions on heterogeneous infrastructure

Infrastructures significantly differ in the way they manage data and compute; e.g., on XSEDE resources it is generally possible to place data on the distributed filesystem available to all compute nodes. On OSG this is simply not possible since users generally cannot access compute nodes without Condor; however, the iRODS service on OSG enables the application to push data to the different OSG resources. These different kinds of semantics increase the complexity for applications to deal with data. Experiments in this section show how Pilot-Data provides a unified, logical resource abstraction, which allows applications to reason about trade-offs and pursue different compute/data placements strategies as laid out before, e.g. bringing compute to data versus data to compute.

For this purpose we use the Pilot-API to manage the input/output data in conjunction with the computational tasks of the BWA genome sequencing application [44]. The application requires two kinds of input data: (i) the reference genome and index files (∼8 GB), and (ii) the short read file(s) obtained from the sequencing machines. The alignment process can be parallelized by partitioning the read files and processing them using multiple BWA tasks. The reference genome and index files are shared between all tasks. The experimental configuration consists of 2 GB read files, which are partitioned to 8 tasks each processing 256 MB.

In scenarios 1 and 2 we conduct baseline experiments using *simple* data management, i.e. each task pulls in all input data from the submission machine (GW68). Tasks are distributed across 8 Pilots on OSG (scenario 1) and across a single Pilot marshaling 24 cores on Lonestar/XSEDE (scenario 2); note that in HTC environments such as OSG, a Pilot typically marshals only a single core (up to a maximum of one entire node). We restrict OSG resources to a set of 9 machines, which are supported by the OSG iRODS installation. The resources are distributed across the eastern and central US including resources at TACC, Purdue and Cornell. The OSG Pilot-Computes are submitted using the SAGA-Python Condor adaptor [66] and GlideinWMS [69], a workload management system built on top of the Pilot capabilities of Condor-G/Glide-in [27] provided as a service on OSG.

In scenarios 3–5 we use the ability to co-locate Pilot-Computes and Pilot-Data. For this purpose, the DU containing the input data is placed in a Pilot-Data close to the Pilot-Compute. In scenario 3, the data is placed and replicated into an iRODS-based PD (9 machines); in scenario 4 an SSH Pilot-Data on the shared Lustre scratch filesystem of Lonestar is used. In both scenarios the Pilot-Computes and Pilot-Data are co-located. Finally, we investigate the ability to use Pilot-Computes and Pilot-Data across multiple OSG and XSEDE resources in scenario 5. In this scenario, the input dataset resides on a Pilot-Data on Lonestar. Two Pilot-Computes are used; One Pilot-Compute allocating one node with 12 cores is submitted to Lonestar and four Pilot-Computes are spawned on OSG.

Figs. 9 and 10 analyze the different scenarios. The insert describes T_D, i.e. the time for uploading, inserting and in case of iRODS replicating 8.3 GB of input data. In general, the Pilot queuing times, i.e. the time until a Pilot becomes active, are higher on OSG than on XSEDE resources. The queuing time mainly depends on three factors: the current utilization of the resource, the allocation of the user and the overhead induced by the queuing system.

Scenarios 1 and 2 clearly show the limitations of simple data management approaches. The necessity for each task to pull in

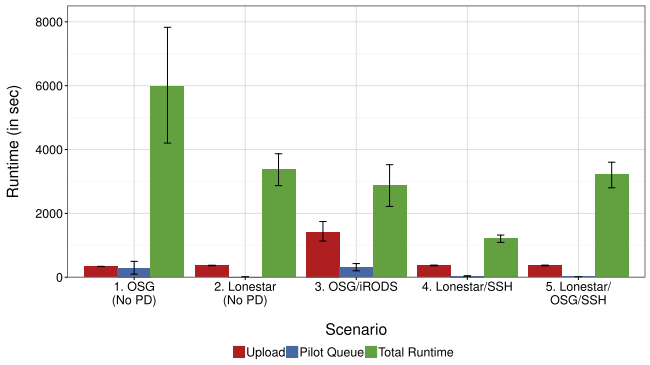

Fig. 9. *Genome sequencing using Pilot-Data on different infrastructures:* Runtimes for running BWA on 2 GB of sequence read files using 8 tasks for five different infrastructure configurations on XSEDE and OSG. The usage of PD (scenarios 3–5) led to a performance improvement compared to a naive data management (scenario 1–2).

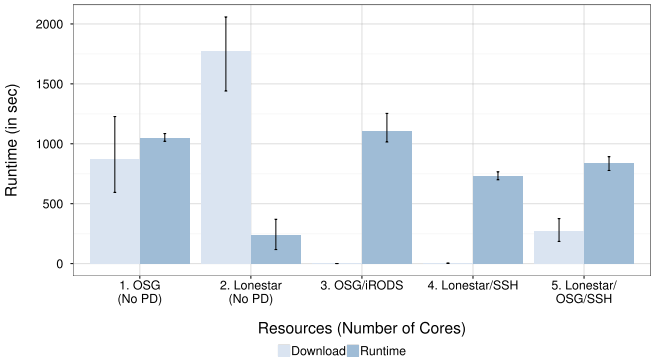

Fig. 10. *Staging and task runtimes:* When comparing the individual task staging and runtimes, it becomes obvious that file staging quickly becomes a bottleneck. By using Pilot-Data the file staging time (Download) can be significantly reduced. In scenario 5 half of the tasks are required to download the files, thus, a small file staging time remains.

8.3 GB data remotely creates a bottleneck. In scenario 3 we utilize the data/compute co-location capabilities of the OSG Condor and iRODS installation. The runtime T is significantly improved compared to scenario 1 mainly due to the elimination of data transfers. However, the upfront costs for creating the PD and replicating the data across OSG are higher—$T_{D_{iRODS}}$ is ∼1418 s, $T_{D_{SSH}}$ is only ∼338 s for scenario 4/5, but does not have a replication component (see the inset of Fig. 9). Thus, $T_{S_{CU}}$ is significantly higher for SSH than for iRODS. However, even after including T_D, the performance of iRODS is still 30% better than the SSH scenario.

In scenario 5, as the input data resides in a PD on Lonestar, the staging time for tasks on Lonestar is significantly reduced. Since the Pilot queuing time on Lonestar was shorter than on OSG, the majority of the tasks were executed on Lonestar; on average 4.5 out of the 8 tasks were run on Lonestar. Finally, scenario 5 shows that if sufficient compute resources are available close to the data, it is beneficial to execute tasks close to the data. In particular this scenario demonstrates the power of the Pilot-Data abstraction, which enables the effective and interoperable use of multiple, heterogeneous infrastructures – OSG and XSEDE – via a unified API.

6.4. Understanding scalability, distribution and replication

In this section we investigate the usage of Pilot-Data to manage distributed data and compute. Reasons for using distributed resources are manifold: often data is pre-distributed or the available resources (cores, I/O) on single machines are not sufficient. For example, while the overall I/O throughputs on HPC machines,

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

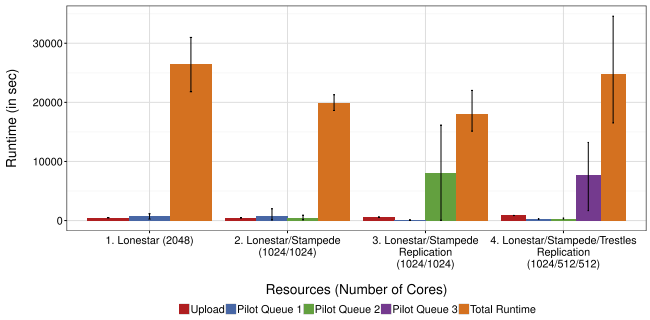

Fig. 11. *Large-scale, distributed genome sequencing on XSEDE (overall scenario runtime):* Running 1024 tasks each consuming 9 GB data on up to three XSEDE machines. The experiment shows that using multiple resources can improve T despite the overhead introduced for data movements. The usage of replication can further improve the runtime T and is essential if the bandwidths to the remote resource are limited as in scenario 4.

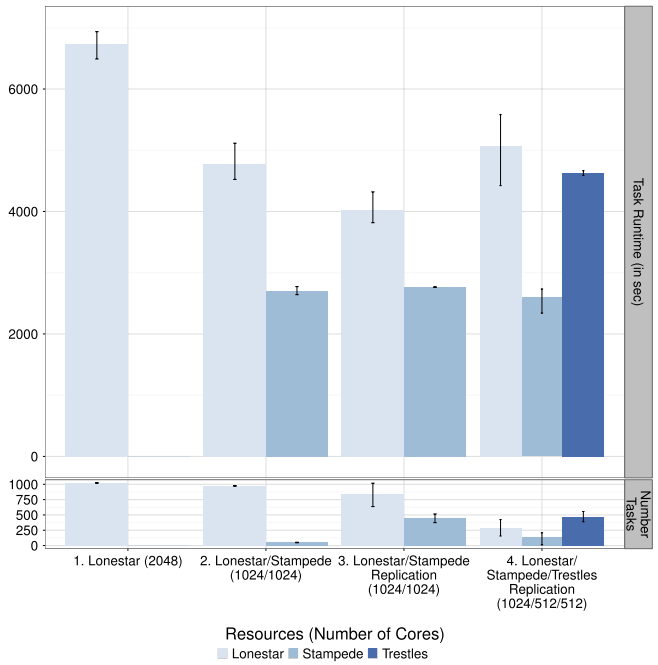

Fig. 12. *Large-scale, distributed genome sequencing on XSEDE (task runtime and distribution):* The task runtimes indicate a sensitivity to the number of concurrent tasks on Lonestar, e.g. when running a majority of the 1024 tasks using 2 cores each (scenario 1). In scenario 2 the necessity to move file limits the number of tasks that are executed non-data local on Stampede. The usage of data replication improves the distribution and runtime of the tasks (scenario 3). Fig. 13 analyzes scenario 4.

such as Lonestar or Stampede, and parallel filesystems, such as Lustre or GPFS, are impressive, we will show that I/O scaling may be constrained for data-intensive applications at large scale. Further, many resources provide significantly less I/O capacity than these flagship machines. Also, actual I/O speeds are highly dependent on the current utilization of the machine. Pilot-Data provides the ability to overcome some of these constraints by distributing compute/data to multiple distributed resources, potentially avoiding situations where disk access speeds are the main constraint in improving performance.

Offloading tasks to distributed/remote resources is a viable strategy to minimize bottlenecks, such as high queuing times, insufficient compute or I/O resources for a certain workload and data placed on distributed resources. As alluded to in Section 6.1 there are different parameters to consider when distributing compute and/or data. The main barrier for distributing large-scale data-intensive applications across multiple resources is the necessity to move data. We evaluate four scenarios using Pilot-Data with different data placement strategies (with and without up-front data replication) on up to three XSEDE machines. For this purpose, we use a larger BWA ensemble consisting of 1024 tasks each processing a read file of 1 GB size on different distributed XSEDE configurations. In total each task consumes 9 GB and the ensemble 9200 GB of data. For each task two cores are requested.

Figs. 11 and 12 summarize the result. As shown in Fig. 11, the runtime improves with the number of resources used. Using a single machine, such as Lonestar in scenario 1, does not yield in an optimal performance. The long runtime of the individual CUs depicted in Fig. 12 indicates a bottleneck on this machine (very likely the I/O capacity of the Lustre filesystem is insufficient). Thus, in scenario 2 we distribute the workload to two machines in close proximity: Lonestar and Stampede (both located at TACC). On each machine we request a Pilot with 1024 cores. The overall runtime of the 1024 CUs and also the individual CU runtimes improve in this scenario. However, the necessity of moving the data led to another bottleneck: tasks that are executed on the remote machine Stampede are required to move 9 GB of input data. Moving this data from Lonestar to Stampede required on average 450 s per task. Thus, in total only about 5% of the tasks are executed on Stampede (see lower part of Fig. 12).

To optimize data placements, we deploy data replication in scenario 3: before the Pilot-Computes and tasks are started a replica of the input Data-Unit is created and placed on Stampede. In average the creation of the replica takes 130 s and is negligible in contrast to the overall compute time. A reason for this is the optimized replication mechanism, which utilizes the replica closest to the target site. Thus, an improvement in T is observable despite the fact that the queuing time on Stampede during the time of the experiment was very long (in average 8100 s and thus, about 20 times as long as in scenario 2). Considering this, the overall runtime could have been even better at a different time. In this scenario, the distribution of the CUs improves; despite the longer queuing time at Stampede, about 40% of the tasks are executed on this machine.

Finally, we explore the distribution of the workload across 3 machines in a wide area network in scenario 4. Again, we utilize the DU replication capability of Pilot-Data. Several attempts of conducting the experiments without replication failed. As seen in Fig. 11, the runtime T is about 6000 s longer than in the best case (scenario 3). Nevertheless, it is still shorter than in the single resource scenario. Fig. 13 shows the timeline of an example run. As indicated by the large error bars, the runtime of each CU fluctuated strongly: in general, CUs started later on a machine run longer. As seen, the number of active CUs is constrained by the non-availability of resources. After Pilot 3 becomes active the number of active CUs peaks. Overall, we experienced a high fluctuation in the queue time on Trestles, which also impacts both the distribution of CUs as well as the CU runtime. The more CUs that are allocated to Trestles, the slower the average runtime of each CU. As expected, with the degree of distribution the predictability of the run decreases; minor differences (e.g. in the queue time) can significantly alter the overall runtime. Thus, it is critical to deploy application-level routines to react to dynamic changes in the resource availability.

In summary, Pilot-Abstractions enable large-scale applications to use various strategies for allocating distributed data and compute resources. The usage of distributed resources enables applications to exploit additional resources in a flexible manner, avoiding queuing times on a single resource. However, it must be noted that in particular for long-running CUs, the first available resource may not be the best one. As seen in scenario 3, it can be beneficial to wait for the faster machines even though there is a significant queuing time. Also, particularly with larger ensembles, fault tolerance becomes a challenge. During the runs we observed

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

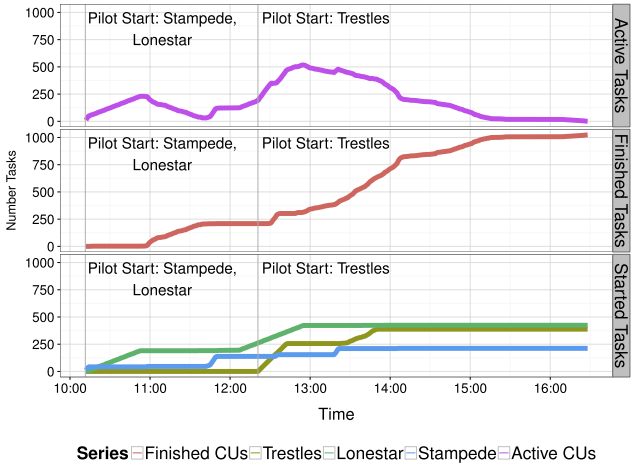

Fig. 13. *Time series for a single run on Lonestar/Stampede/Trestles:* During this experiment Stampede represented a significant bottleneck. Only 212 out of the 256 slots were claimed despite the fact that the Pilot was active from the beginning. Trestles in contrast claimed only 36 CUs less than Lonestar despite the significantly larger Pilot queuing time.

failures due to high loads (e.g. in scenario 1), wall time limits and file transfer errors. In the future we will require enhancements to BigJob in order to bolster fault tolerance (e.g. supporting a reliable way to restart transfers). Also, more advanced strategies, such as data replication and more fine granular partitioning, will improve the distributability of large-scale applications.

7. Discussion and future work

An increasing number of scientific applications are data-driven, which is associated with a new set of challenges for existing infrastructures and tools. Science domains and applications greatly differ in the ways they generate, store and use data. Managing applications with different characteristics on top of heterogeneous resources at scale represents a serious challenge—different application workloads require different resource allocation and workload placement strategies. Commonly, there are three primary compute–data placement paradigms: (i) when the data is essentially localized, either from being "poured" into a single storage backend or because the volumes of data allow for small-scale localization; (ii) where the data is decomposed and distributed (with multi-tier redundancy and caching) to an appropriate number of computing/analytical engines as available, e.g., as employed by the EGI/OSG for particle physics (that led to the discovery of the Higgs boson), and (iii) a hybrid of the above two paradigms, wherein data is decomposed and committed to several infrastructures, which could then result in a combination of either of the first two paradigms. Even though specific realizations and backends vary, we have shown Pilot-Data supports reasoning over different compute–data placement paradigms and infrastructures. Pilot-Abstractions allow applications to map system-level capabilities, to a unified, logical resource topology, which enables the application to reason about trade-off and optimize placement decisions accordingly based on the information provided by the affinity model, such as resource localities, and dynamic information, such as resource and bandwidth availabilities.

Pilot-Data enhances the utility and usability of Pilot-Jobs by extending the use of Pilot-Abstractions to data and thus, providing a missing critical component in conventional systems. In conjunction with the fact that Pilot-Jobs provide a well-defined abstraction for distributed resource management independent of infrastructure-specific details, the combined Pilot-Abstraction of Pilot-Data and Pilot-Job is a powerful approach to manage the compute and data challenges in heterogeneous and highly dynamic distributed environments. With increasing heterogeneity (e.g. when using multiple infrastructures as HPC, cloud, and HTC environments), the unpredictability and dynamisms increase as well. Thus, it is critical to provide the right level of control that enables the application to respond to this kind of dynamism, e.g. by acquiring additional compute resources close to the replica of the dataset. Our genome sequencing application for example successfully demonstrates how Pilot-Data provides the right primitives for expressing tasks and their data dependencies and for exploring trade-offs such as data replication in distributed and dynamic environments. We also successfully showed that the Pilot-Data efficiently supports other application patterns, e.g. dynamic workflows [67] or MapReduce [52].

The above functional and qualitative attributes exposed via Pilot-Abstractions enable a simple method of managing complexity (inherent to working with data across diverse infrastructures), thereby supporting the claim that Pilot-Data provides a unifying abstraction for distributed data and compute. The advantages of Pilot-Data, however, extend well beyond the conceptual: through a series of experiments that cover a range of often-realized distributed configurations and scenarios, we have seen how Pilot-Data provides an abstraction and a powerful tool for managing distributed data. We reiterate that the application-scenarios investigated as well implementations are production-grade and used on production DCI such as XSEDE, EGI and OSG, along with their inherent complexity. We have shown that Pilot-Data effectively can distribute data and compute across these infrastructure utilizing system-level features, such as iRODS-based replication on OSG, helping the application to optimize data/compute placement, e.g. by utilizing system-level support for replication where available (e.g. on OSG) and deploy Pilot-Data-level replication in other cases (such as XSEDE).

In the future, we will explore the Pilot-Abstraction as a basis for building higher-level capabilities and frameworks (e.g. for workload and workflow management) to provide further productivity gains. Multi-level scheduling has been demonstrated to be an effective means to address complex and diverse application characteristics and associated requirements with respect to resource management. Our discussion of affinities suggested that they are a good abstraction for capturing relationships between computational tasks and associated data and help to map these dependencies to Pilots. Our prototype workload management service (the Compute-Data Service), which is based on a simple affinity model, will become the basis for Pilot-Data's enhanced scheduling capabilities enabling dynamic execution decisions based on incoming data or varying infrastructure conditions. We will explore different, heterogeneous workloads, e.g. ensembles, data-intensive tasks, workflows to extract the characteristics of these workloads to derive important parameters such as runtime and data characteristics for optimized scheduling. Also, we will investigate high-level abstractions for re-occurring data/compute usage patterns, e.g. data partitioning, filtering or merging of datasets, etc., further improving developer productivity. Another important trend is the ongoing convergence of HPC/HTC and Big Data infrastructures based on Apache Hadoop [41]. We believe that Pilot-Abstractions will prove useful in unifying access to these different types of infrastructures while maintaining the expressiveness needed for data-intensive applications.

Acknowledgments

This work is primarily funded by NSF CAREER Award (OCI-1253644), as well as by NSF Cyber-enabled Discovery and Innovation Award (CHE-1125332), NSF-ExTENCI (OCI-1007115), NSF EarthCube (SCIHM, OCI-1235085) and Department of Energy Award (ASCR, DE-FG02-12ER26115). Pradeep Mantha (Berkeley)

contributed to early associated work on Pilot-Data based applications. We thank our fellow RADICAL members for their support and input, in particular Andre Merzky, Matteo Turilli and Melissa Romanus. We thank Tanya Levshina for help with iRODS configurational issues on OSG. SJ acknowledges useful related discussions with Jon Weissman (Minnesota) and Dan Katz (Chicago). SJ acknowledges UK EPSRC for supporting the e-Science Research themes "Distributed Programming Abstractions" and 3DPAS and earlier support of the SAGA project. AL and SJ acknowledge NSF-OCI 1059635. This work has also been made possible thanks to computer resources provided by TeraGrid TRAC award TG-MCB090174. We thank Yaakoub El-Khamra of TACC for exceptional support on XSEDE systems. EGI experiments were performed on resources provided by SURFsara's Dutch e-Science Grid.

Author contributions

The primary experiments were performed primarily by AL; MS performed experiments with SRM (Fig. 7). The presented experiments were designed by AL and SJ. Data was analyzed by AL and SJ, with contributions from MS. AZ helped in the software implementation for experiments involving iRODS. SJ determined the scope and structure of the paper. AL wrote the paper, with editorial input from SJ, who also co-wrote the introduction and conclusion (with AL).

References

[1] A. Aamnitchi, S. Doraimani, G. Garzoglio, Filecules in high-energy physics: Characteristics and impact on resource management, in: High Performance Distributed Computing, 2006 15th IEEE International Symposium on, 0–0 2006, pp. 69–80.
[2] W. Allcock, GridFTP: Protocol extensions to FTP for the grid, Open Grid Forum, OGF Recommendation Document, GFD.20, 2003. http://ogf.org/documents/GFD.20.pdf.
[3] M. Altunay, P. Avery, K. Blackburn, B. Bockelman, M. Ernst, D. Fraser, R. Quick, R. Gardner, S. Goasguen, T. Levshina, M. Livny, J. Mcgee, D. Olson, R. Pordes, M. Potekhin, A. Rana, A. Roy, C. Sehgal, I. Sfiligoi, F. Wuerthwein, A science driven production cyberinfrastructure—the open science grid, J. Grid Comput. 9 (2) (2011) 201–218. [Online]. Available: http://dx.doi.org/10.1007/s10723-010-9176-6.
[4] Amazon S3 Web Service. http://s3.amazonaws.com.
[5] A. Amer, D. Long, R. Burns, Group-based management of distributed file caches, in: Distributed Computing Systems, 2002, Proceedings, 22nd International Conference on, 2002, pp. 525–534.
[6] Apache Hadoop, 2013. http://hadoop.apache.org/.
[7] Apache Hadoop NextGen MapReduce (YARN), 2013. http://hadoop.apache.org/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/YARN.html.
[8] Apache Tez, 2013. http://hortonworks.com/hadoop/tez/.
[9] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, C. Cirstoiu, C. Grigoras, A. Hayrapetyan, A. Harutyunyan, A.J. Peters, P. Saiz, AliEn: ALICE environment on the grid, J. Phys. Conf. Ser. 119 (6) (2008) 062012. [Online]. Available: http://stacks.iop.org/1742-6596/119/i=6/a=062012.
[10] J.-P. Baud, J. Casey, S. Lemaitre, C. Nicholson, Performance analysis of a file catalog for the LHC computing grid, in: High Performance Distributed Computing, 2005, HPDC-14, Proceedings, 14th IEEE International Symposium on, July 2005, pp. 91–99.
[11] C.G. Bell, Fundamentals of time shared computers, Comput. Des. 7 (1968) 44–59.
[12] W.H. Bell, D.G. Cameron, A.P. Millar, L. Capozza, K. Stockinger, F. Zini, Optorsim: A grid simulator for studying dynamic data replication strategies, Int. J. High Perform. Comput. Appl. 17 (4) (2003) 403–416.
[13] M.D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, J. Saltz, Distributed processing of very large datasets with datacutter, Parallel Comput. 27 (11) (2001) 1457–1478. [Online]. Available: http://dx.doi.org/10.1016/S0167-8191(01)00099-0.
[14] M. Cardosa, C. Wang, A. Nangia, A. Chandra, J. Weissman, Exploring mapreduce efficiency with highly-distributed data, in: Proceedings of the Second International Workshop on MapReduce and its Applications, ser. MapReduce '11, ACM, New York, NY, USA, 2011, pp. 27–34. [Online]. Available: http://doi.acm.org/10.1145/1996092.1996100.
[15] Castor: Cern Advanced Storage Manager. http://castor.web.cern.ch/ (last accessed 2013).
[16] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, R. Schwartzkopf, Performance and scalability of a replica location service, in: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, ser. HPDC '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 182–191. [Online]. Available: http://dx.doi.org/10.1109/HPDC.2004.27.
[17] L. Cinquini, D. Crichton, C. Mattmann, G. Bell, B. Drach, D. Williams, J. Harney, G. Shipman, F. Wang, P. Kershaw, S. Pascoe, R. Ananthakrishnan, N. Miller, E. Gonzalez, S. Denvil, M. Morgan, S. Fiore, Z. Pobre, R. Schweitzer, The earth system grid federation: An open infrastructure for access to distributed geospatial data, in: E-Science (e-Science), 2012, IEEE 8th International Conference on, pp. 1–10.
[18] E. Corso, S. Cozzini, F. Donno, A. Ghiselli, L. Magnoni, M. Mazzucato, R. Murri, P. Ricci, H. Stockinger, A. Terpin, V. Vagnoni, R. Zappi, StoRM, an SRM implementation for LHC analysis farms, computing in high energy physics, in: Proceedings of the International CHEP 2006, Mumbai, India, February 2006.
[19] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, USENIX Association, Berkeley, CA, USA, 2004, pp. 137–150.
[20] A. Dorigo, P. Elmer, F. Furano, A. Hanushevsky, Xrootd-a highly scalable architecture for data access, 2005.
[21] Dpm: Disk pool manager. https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm (last accessed 2013).
[22] Eucalyptus Walrus. http://open.eucalyptus.com/wiki/EucalyptusStorage_v1.4.
[23] European Grid Infrastructure, 2013. http://www.egi.eu.
[24] G. Fedak, H. He, F. Cappello, Bitdew: a programmable environment for large-scale data management and distribution, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, ser. SC '08, IEEE Press, Piscataway, NJ, USA, 2008, pp. 45:1–45:12.
[25] M.J. Fischer, N.A. Lynch, M.S. Paterson, Impossibility of distributed consensus with one faulty process, J. ACM 32 (2) (1985) 374–382.
[26] I. Foster, Globus Online: Accelerating and democratizing science through cloud-based services, IEEE Internet Comput. 15 (2011) 70–73.
[27] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-G: A computation management agent for multi-institutional grids, Cluster Comput. 5 (3) (2002) 237–246.
[28] P. Fuhrmann, V. Gülzow, in: W. E. Nagel, W. V. Walter, and W. Lehner (Eds.), Euro-Par, pp. 1106–1113.
[29] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.
[30] G. Ganger, M.F. Kaashoek, Embedded inodes and explicit grouping: Exploiting disk bandwidth for small files, in: Proceedings of the 1997 USENIX Technical Conference, 1997, pp. 1–17.
[31] J. Gantz, D. Reinsel, The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east, 2012. http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf.
[32] GFFS—Global Federated File System. http://genesis2.virginia.edu/wiki/Main/GFFS.
[33] D. Ghoshal, L. Ramakrishnan, Frieda: Flexible robust intelligent elastic data management in cloud environments, in: High Performance Computing, Networking Storage and Analysis, SC Companion:, vol. 0, 2012, pp. 1096–1105.
[34] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, C. Smith, A Simple API for Grid Applications (SAGA), Open Grid Forum, OGF Recommendation Document, GFD.90, 2007. http://ogf.org/documents/GFD.90.pdf.
[35] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. von Laszewski, C. Lee, A. Merzky, H. Rajic, J. Shalf, SAGA: A simple API for grid applications, high-level application programming on the grid, Comput. Methods Sci. Technol. 12 (1) (2006) 7–20. [Online]. Available: http://saga.cct.lsu.edu/publications/saga_paper-a_simple_api_for_grid_applications_sc05.pdf.
[36] Google Cloud Storage. https://developers.google.com/storage/.
[37] J. Gray, D.T. Liu, M. Nieto-Santisteban, A. Szalay, D.J. DeWitt, G. Heber, Scientific data management in the coming decade, SIGMOD Rec. 34 (4) (2005) 34–41. [Online]. Available: http://doi.acm.org/10.1145/1107499.1107503.
[38] T. Hey, S. Tansley, K. Tolle (Eds.), The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft Research, Redmond, Washington, 2009.
[39] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, I. Stoica, Mesos: a platform for fine-grained resource sharing in the data center, in: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI'11, USENIX Association, Berkeley, CA, USA, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488.
[40] S. Jha, N.C. Hong, S. Dobson, D.S. Katz, A. Luckow, O. Rana, Y. Simmhan, J. Weissman, Introducing distributed dynamic data-intensive (D3) science: understanding applications and infrastructure, 2013, Technical Report, Rutgers University.
[41] S. Jha, J. Qiu, A. Luckow, P.K. Mantha, G.C. Fox, A tale of two data-intensive paradigms: Applications, abstractions, and architectures, CoRR, vol. abs/1403.1528, 2014.
[42] T. Kosar, M. Livny, Stork: Making data placement a first class citizen in the grid, in: Distributed Computing Systems, 2004, Proceedings, 24th International Conference on, 2004, pp. 342–349.
[43] M. Lamanna, The LHC computing grid project at CERN, Nucl. Instrum. Methods Phys. Res. A 534 (1–2) (2004) 1–6. Proceedings of the IXth International Workshop on Advanced Computing and Analysis Techniques in Physics Research. [Online]. Available: http://www.sciencedirect.com/science/article/B6TJM-4D01K33-B/2/83c424822f331a47206c44afcaf90869.
[44] H. Li, R. Durbin, Fast and accurate long-read alignment with burrows wheeler transform, Bioinformatics 26 (2010) 589–595. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp698.
[45] Llama—Low Latency Application MAster, 2013. http://cloudera.github.io/llama/.

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009

[46] A. Luckow, L. Lacinski, S. Jha, SAGA BigJob: an extensible and interoperable pilot-job abstraction for distributed applications and systems, in: The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2010, pp. 135–144.
[47] A. Luckow, M. Santcroos, A. Merzky, O. Weidner, P. Mantha, S. Jha, P*: A Model of Pilot-Abstractions, in: 8th IEEE International Conference on e-Science 2012, 2012.
[48] J. Ma, W. Liu, T. Glatard, A classification of file placement and replication methods on grids, Future Gener. Comput. Syst. 29 (6) (2013) 1395–1406. Including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000344.
[49] T. Maeno, PanDA: distributed production and distributed analysis system for ATLAS, J. Phys. Conf. Ser. 119 (6) (2008) 062036. [Online]. Available: http://stacks.iop.org/1742-6596/119/i=6/a=062036.
[50] T. Maeno, K. De, S. Panitkin, PD2P: PanDA dynamic data placement for ATLAS, J. Phys. Conf. Ser. 396 (2012) 032070. [Online]. Available: http://iopscience.iop.org/1742-6596/396/3/032070.
[51] T. Maeno, K. De, T. Wenaus, P. Nilsson, G.A. Stewart, R. Walker, A. Stradling, J. Caballero, M. Potekhin, D. Smith, for The Atlas Collaboration, Overview of atlas panda workload management, J. Phys. Conf. Ser. 331 (7) (2011) 072024. [Online]. Available: http://stacks.iop.org/1742-6596/331/i=7/a=072024.
[52] P.K. Mantha, A. Luckow, S. Jha, Pilot-MapReduce: an extensible and flexible MapReduce implementation for distributed data, in: Proceedings of third International Workshop on MapReduce and its Applications, ser. MapReduce'12, ACM, New York, NY, USA, 2012, pp. 17–24.
[53] C. Miceli, M. Miceli, B. Rodriguez-Milla, S. Jha, Understanding performance of distributed data-intensive applications, R. Soc. Lond. Philos. Trans. Ser. A 368 (2010) 4089–4102.
[54] NSF, Cyberinfrastructure for 21st century science and engineering: Advanced computing infrastructure vision and strategic plan, 2012. http://www.nsf.gov/pubs/2012/nsf12051/nsf12051.pdf.
[55] OpenStack Swift. http://swift.openstack.org.
[56] Oracle, Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System. White Paper, 2007.
[57] Pilot-API, 2013. http://saga-project.github.io/BigJob/sphinxdoc/.
[58] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, R. Quick, The open science grid, J. Phys. Conf. Ser. 78 (1) (2007) 012057.
[59] PRACE research infrastructure. http://www.prace-project.eu/.
[60] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde, Falkon: A fast and light-weight tasK executiON framework, in: SC'07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, ACM, New York, NY, USA, 2007, pp. 1–12.
[61] I. Raicu, Y. Zhao, I.T. Foster, A. Szalay, Accelerating large-scale data exploration through data diffusion, in: Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing, ser. DADC'08, ACM, New York, NY, USA, 2008, pp. 9–18. [Online]. Available: http://doi.acm.org/10.1145/1383519.1383521.
[62] A. Rajasekar, R. Moore, C.-Y. Hou, C.A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert, P. Tooby, B. Zhu, iRODS Primer: Integrated Rule-Oriented Data System, Morgan and Claypool Publishers, 2010.
[63] K. Ranganathan, I. Foster, Decoupling computation and data scheduling in distributed data-intensive applications, in: High Performance Distributed Computing, 2002, HPDC-11 2002, Proceedings, 11th IEEE International Symposium on, 2002, pp. 352–358.
[64] Redis, 2012. http://redis.io/.
[65] A. Romosan, D. Rotem, A. Shoshani, D. Wright, Co-scheduling of computation and data on computer clusters, in: Proceedings of 17th International Conference on Scientific and Statistical Databases Management, SSDBM, 2005.
[66] SAGA-Python. http://saga-project.github.io/saga-python/.
[67] M. Santcroos, B.D. van Schaik, S. Shahand, S.D. Olabarriaga, A. Luckow, S. Jha, Exploring flexible and dynamic enactment of scientific workflows using pilot abstractions, in: Proceedings of the 13th IEEE/ACM International Symposium of Cluster, Cloud and Grid Computing, Delft, Netherlands, 2013.
[68] F. Schmuck, R. Haskin, Gpfs: A shared-disk file system for large computing clusters, in: Proceedings of the 1st USENIX Conference on File and Storage Technologies, ser. FAST'02, USENIX Association, Berkeley, CA, USA, 2002, [Online]. Available: http://dl.acm.org/citation.cfm?id=1083323.1083349.
[69] I. Sfiligoi, D. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, F. Wurthwein, The pilot way to grid resources using glideinwms, in: Computer Science and Information Engineering, 2009, pp. 428–432.
[70] A. Sim, et al., GFD.154: The storage resource manager interface specification V2.2, Tech. Rep., oGF, 2008.
[71] The grid information system, 2013. https://tomtools.cern.ch/confluence/display/IS/Home.
[72] A. Tsaregorodtsev, N. Brook, A. Casajus Ramo, P. Charpentier, J. Closier, et al., DIRAC3: The new generation of the LHCb grid software, J. Phys. Conf. Ser. 219 (2010) 062029.
[73] M. Wilde, M. Hategan, J.M. Wozniak, B. Clifford, D.S. Katz, I. Foster, Swift: A language for distributed parallel scripting, Parallel Comput. 37 (9) (2011) 633–652.
[74] Windows Azure Blob Storage. https://www.windowsazure.com/en-us/develop/net/fundamentals/cloud-storage/.
[75] XSEDE: Extreme Science and Engineering Discovery Environment. https://www.xsede.org/.

Andre Luckow is a consultant in the Radical Group at Rutgers University, New Jersey, USA. He studied Computer Science at the Potsdam University where he obtained his doctorate degree in 2009. His main research interests are: distributed systems, fault tolerance, computational sciences and programming languages.

Mark Santcroos is currently pursuing a Ph.D. under the supervision of Dr. Shantenu Jha through a Leeds fellowship. His research interests are distributed systems in general, and more specifically abstractions that enable the effective use of these systems for scientific research. These abstractions include Pilots (our work on P*) and everything workflow related. He holds a guest position at the Bioinformatics Laboratory at the Academic Medical Center in Amsterdam and he is an EGI Champion.

Ashley Zebrowski is a Ph.D. Candidate working with Dr. Shantenu Jha and the RADICAL group at Rutgers University in New Jersey. Her current research interests are focused on distributed computing, especially with regard to scheduling distributed jobs and data in a flexible, scalable manner. She is currently engaged in developing software and algorithms for the SAGA and BigJob projects, and has had experience with the Cactus Framework for Numerical Relativity in the past. Her ongoing work is directed in an effort to increase the scalability, performance, and reach of distributed computation via use of appropriate abstractions and algorithms.

Shantenu Jha is an Assistant Professor at Rutgers University and leads the Research in Advanced DIstributed Cyberinfrastructure and Applications Laboratory (RADICAL) http://radical.rutgers.edu. His research interests lie at the triple point of Applied Computing, Cyberinfrastructure R&D and Computational Science. Shantenu is also part of the SAGA project (http://saga-project.org), which is a community standard and is used on most major production distributed infrastructure—such as US NSF's XSEDE and the European Grid Infrastructure.

Please cite this article in press as: A. Luckow, et al., Pilot-Data: An abstraction for distributed data, J. Parallel Distrib. Comput. (2014), http://dx.doi.org/10.1016/j.jpdc.2014.09.009