Pilot-Data: An Abstraction for Distributed Data

Andre Luckow[1], Mark Santcroos[1,2], Ashley Zebrowski[1], Shantenu Jha[1*]

(1) *RADICAL, Rutgers University, Piscataway, NJ 08854, USA*
(2) *Bioinformatics Laboratory, AMC, University of Amsterdam, NL*
(*) *Contact Author:* shantenu.jha@rutgers.edu

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 addressing each of these challenges, 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 [1]. 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.

Working with large volumes of data involves many challenges beyond its storage and management. A specific challenge is that of data and compute (processing) in a distributed environment. 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 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 [2, 3]. 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 [4] 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, in the ESGF dataset; the scheduling of data analysis jobs needs to be responsive to these spatio-temporal data changes.

In order 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

Supercomputing'13 2013, Denver, CO, USA

access to heterogeneous distributed cyberinfrastructure, how can these abstractions also be used to provide the ability to reason about "when" to distribute as well as "how", viz., 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 show how the Pilot-Data abstraction addresses the first three questions, and equally importantly, outlines a research path to understanding other questions. We will examine the general challenges and issues in the specific context of BWA– a well known Next-Generation Sequencing (NGS) analysis application [5]. NGS analytics, in addition to being a big data problem (O(TB)), is also a computationally demanding and distributed computing problem. The computational demands arise from the often complex and intensive analysis that has to be performed on data, which in turn arise from algorithms that are designed to account for repetitions, errors and incomplete information. The distributed aspects arise at multiple levels: for example, the simple act of having to move data from source (generation) to the destination where computing (analysis) is a challenge due to the volumes involved. Trade-offs exists between the cost/challenges in distributing data versus IO saturation or memory bottlenecks. When coupled with the compute intensive nature of the problem, it soon emerges that a fundamental challenge is not only whether to distribute, but *where* to distribute, *what* to distribute (should the computing move to the data, or the data move to the compute), and *how* to distribute (what tools and infrastructure to use). In addition, NGS (and similar) applications on distributed infrastructure have dynamic aspects; for example, workload decomposition and distribution may be determined dynamically based upon data location, processing profile or network profile, when attempting to utilize resources optimally.

Overview and Outline

As the ESGF and NGS analytic examples establish, a performance bottleneck is distributed compute-data management, placement and scheduling. 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 [6], PRACE [7], XSEDE [8], and OSG [9], which will be agnostic to specific security and data-sharing policies.

In other words, our focus is on addressing the compute-data management and scheduling problem in the context of *production* DCI and not research prototypes, or "feature rich" but closed or specific data-cyberinfrastructure and back-end systems. These challenges are invariant to specific infrastructure and distributed application implementation issues. 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 replication, partitioning, 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.

Our starting point is the abstraction provided by Pilot-Jobs, which have a demonstrable record of effective distributed resource utilization and supporting a broad range of application types [10], [11]. We explore the generalization of the Pilot-Job concept via Pilot-Data, as a way to efficiently manage distributed data and its dynamic placement with respect to computation. Specifically, we introduce *Pilot-Data (PD)* as a novel abstraction for data-intensive applications that provides flexible placement and scheduling capabilities for data by separating the allocation of physical storage and application-level Data-Units. As an example of how this is achieved, 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 placement, in that it is not constrained to a specific scheduling algorithm or infrastructure.

In §2 we discuss some related work and briefly allude to earlier work on the P* Model – a minimal and complete model for Pilot-Jobs. Combined, §2 provides the reader with a better appreciation for the scope and context of our work with production distributed infrastructures in mind. §3 presents a detailed overview of Pilot-Data – the concept, its relation to Pilot-Job and its implementation in BigJob. §3 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. We design and conduct a series of experiments in §4 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. RELATED WORK

The landscape of solutions that have been devised over the years to address the challenges and requirements of distributed data is vast. Any discussion of relevant related work has to be focused by design and limited by necessity. Thus in this section we provide a brief discussion of relevant efforts that either support compute-data co-placement and/or support data management in production DCI, such as XSEDE, OSG and EGI.

2.1 Distributed Data Management

A myriad of storage solutions exist, ranging from local and parallel filesystems, e. g. Lustre [12], to distributed data stores, such as Amazon S3 [13], or federated file systems, such as the Global Federated Filesystem (GFFS) [14] which provide 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 [15]. Typically these

systems do not provide defined quality-of-service and applications are typically unaware of throughput and latencies to expect.

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. Systems like SRM [16] and iRODS [17] combine storage services with services for metadata, replica, transfer management and scheduling. Also, different services covering singular aspects such as replica management (e. g. the Replica Location Service (RLS) [18] or the LCG File Catalogue (LFC) [19]) exist. Globus Online [20] is a hosted transfer service that is based on GridFTP.

Storage Resource Manager (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 data locality; this is left to other components, e. g. a combination of an information system and a replica manager (BDII [21] & LFC in the case of EGI).

iRODS is a comprehensive distributed data management solution designed to operate across geographically distributed, federated storage resources. 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 data set to a set of resources.

2.2 Data-Compute Scheduling

There are several projects that aim to provide integrated data and compute scheduling. For example, the Stork [22] 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. [23] 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.

The MapReduce framework Hadoop [24] provides a highly integrated environment for data-intensive applications. The new Hadoop resource scheduler, Yarn [25], tightly integrates the Hadoop filesystem with the compute resources of the cluster. While Hadoop provides integrated capabilities, such capabilities are currently missing from the production DCI; on the other hand, Hadoop was not designed for distributed scenarios, highlighting the challenges in providing combined integrated and distributed capabilities.

Various abstractions for optimizing access and management of distributed data have been proposed: Filecule [26] 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.[27], Ganger et al. [28] and BitDew [29]. Another example is DataCutter [30], a framework that enables exploration and querying of large datasets while minimizing the necessary data movements.

Finally, different research on when to (potentially dynamically) distribute and replicate data has been conducted: for example, Foster [31] and Bell [32] investigate different data replication management system 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.

2.3 Pilot-Jobs

Pilot-Jobs have been successful abstractions in distributed computing as evidenced by a plethora of PJ frameworks: Condor-G/Glide-in [33], DIANE [34], ToPoS [35], Nimrod/G [36], Falkon [37] and MyCluster [38], to name a few. However, there is insufficient and incomplete support for data movement and placement in many Pilot-Job implementations [10]. 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 three kinds of data management: (i) the ability to stage-in/stage-out files, (ii) the integration of a particular storage backend (e. g. SRM) and (iii) the provisioning of sophisticated data/compute scheduling mechanisms.

An example for (i) is Condor-G/Glide-in, which provides a basic mechanism for file staging. Another example is Swift [40], which provides a data management component called Collective Data Management (CDM). DIANE provides in-band data transfer functionality over its CORBA channel.

DIRAC [39], which is used in the LHC Computing Grid, is an example for a type (ii) system. It interfaces to SRM storage resources and enables the application to stage-in/out data to this system.

Also some examples for type (iii) systems exist: PanDA [41] provides support for the retrieval of input data and collecting of output data, with the ability to replicate popular input data to underutilized resources via PanDA Dynamic Data Placement [**?**] 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. Another example is Falkon, which was extended to support data-aware scheduling on top of a pool of dynamically acquired compute and data resources [**?**]. The so called data diffusion mechanism automatically caches data on Pilot-level enabling the efficient re-use of data.

3. PILOT-DATA: AN UNIFIED ABSTRACTION FOR COMPUTE AND DATA

A major limitation of many Pilot-Job frameworks, is the unsatisfactory and insufficient approach to distributed data management. 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 Data-Units from physical storage enabling efficient compute/data placement using various strategies (e. g. moving compute to data and vice versa, opportunistic replication, partitioning). Pilot-Data provides consistent semantics for data management in

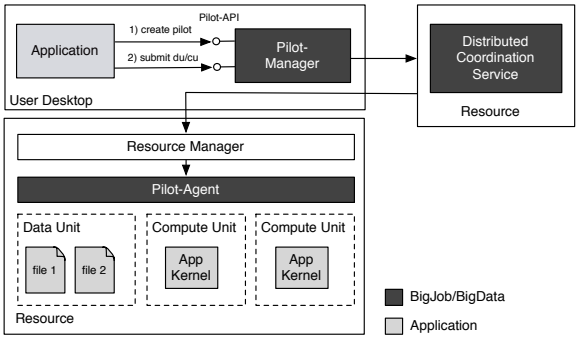

Figure 1: BigJob High-Level Architecture: The Pilot-Manager is the central coordinator of the framework, which orchestrates a set of Pilots. Each Pilot is represented by a decentral component referred to as the Pilot-Agent, which manages the set of resources assigned to it.

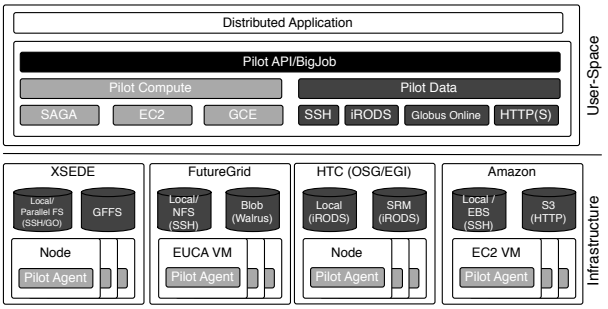

Figure 2: BigJob Pilot Abstractions and Supported Resource Types: BigJob provide a unified abstraction to a heterogeneous set of distributed compute and data resources. Resources are either accessed via SAGA [46, 45] or via a custom adaptor.

conjunction with Pilot-Jobs. 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. 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 parts of a distributed workflow. Pilot-Data facilitates and utilizes late binding of data and physical resources for optimal coupling and management.

The P* model defines the fundamental elements and characteristics of Pilot-Job implementations. 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. In the context of compute, the entity that is submitted to a resource to allocate computational resources (e. g. cores) is referred to as Pilot-Compute (PC). The application workload consisting of Compute-Units (CUs) is then submitted to a PC; the PC has the ability to schedule the CUs independently of the resource's built-in scheduler due to having already allocated the required computational resources. Pilot-Data can be thought of as symmetrical to Pilot-Jobs, i. e., Pilot-Data is conceptually similar as a logical container for dynamic data placement and scheduling as Pilot-Jobs is to computational tasks.

3.1 BigJob: A Pilot-Compute and Data Implementation

Architecture and Design: Consistent with our aims of providing complete Pilot-Job capabilities, we implement Pilot-Data as an extension of BigJob (BJ) [11, 43], which is a SAGA-based Pilot-Job implementation. BigJob provides an 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. Figure 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 [44]).

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 Figure 2 BJ supports various types of HPC/HTC resources via SAGA-Python [45] (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 protocols and services, e. g. SSH, GSISSH, GridFTP [15] and Globus Online [20]. Other storages 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 3.2), 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.

Figure 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 3.2). After submission, DUs and CUs are put into a 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 & communication service is to

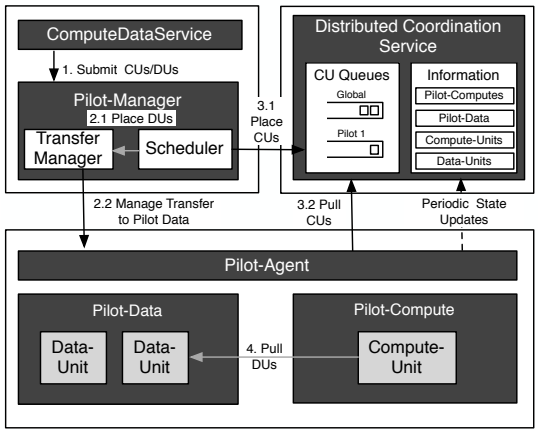

Figure 3: BigJob Application Workload Management: The figure illustrates the typical steps involved for placing and managing the application workload, i. e. the DUs and CUs.

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 [47], for this purpose. 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 is pushed to the Redis server and used by the Pilot-Manager to conduct e. g. placement decisions; (ii) CU are 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 transfer 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 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 [48]. 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 reconnect 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.

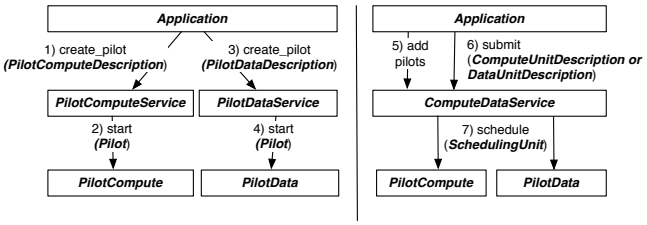

Figure 4: The Pilot-API exposes two primary functionalities: The PilotComputeService and PilotDataService are used for the management of Pilot-Computes and Pilot-Data. The application workload is submitted via the Compute-Data Service.

Nevertheless, in most cases the ability to quickly restart the Redis server (if necessary on another resource) is sufficient. Another error source are 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.

3.2 Pilot-API: Managing Distributed Data & Compute

The Pilot-API [42] provides a well-defined control and programming interface for PJ frameworks, which is built upon the entities defined by the P* model [10]. It is an interoperable and extensible API which exposes the core functionalities of a Pilot framework and can be used across multiple distinct production cyberinfrastructures. The API separates the concerns (i) Pilot/resource and (ii) application workload management (see Figure 4).

Pilot Management: The Pilot-Compute Service is responsible for controlling the lifecycle of Pilot-Computes; the Pilot-Data Service manages Pilot-Data. Using Pilot-Abstractions different types of distributed compute resources, storage infrastructures and transport protocols etc. can be marshaled providing a seamless, unifying environment to the application for managing resources. Applications can simply acquire resources in form of Pilots and then assign their workload to these resources.

Application Workload Management: The application or tool that uses the Pilot-API utilizes Data-Units (DUs) and Compute-Units (CUs) as abstraction for expressing its workload. 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 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 abstracting low-level details, such as the persistent backend. 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. 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 appli-

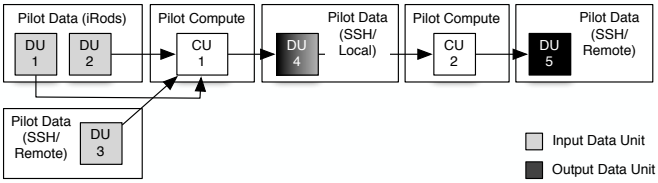

Figure 5: DU and CU Interactions and Data Flow: Each CU can specify a set of input DU. The framework will ensure that the DU is transferred to the CU. Having terminated the job, the specified output is moved to the output DU.

cations. 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.

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 the workload management service – the Compute-Data Service– provided by BigJob. The Pilot-Compute, Pilot-Data and Compute-Data Service classes provide an interface for submitting and managing application workloads. CUs and DUs are described by the use of a Compute-Unit-Description (CUDs) and a 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. Having submitted the description object, a Compute-Unit/Data-Unit instance is returned. This instance can then be used for state queries and lifecycle management (e. g. canceling a CU). In case (ii), the runtime system of the Compute-Data Service is responsible for placing CUs and DUs on a Pilot. For this purpose, it relies on different information and heuristics e. g. on the localities of the DUs, to facilitate scheduling and other types of decision making (see section 3.3).

Figure 5 shows an example of a data flow between multiple phases of compute. As described, applications are required to organize their data in 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. The input data is made available in the working directory of the CU.

Typically, scientific applications involve multiple steps of data generation and processing. The Pilot-API and BigJob also supports more complex application usage modes, e. g. ensembles, coupled ensembles, more complex pipelines possible comprising of different kinds of raw and derived data [3], MapReduce-based applications and workflows. We have successfully shown that the Pilot-API efficiently supports dynamic workflows [49] as well as MapReduce-based applications [43]. In these scenarios, the Pilot-API is used to efficiently manage data flow between a set of dependent CUs on heterogeneous infrastructure. Often, it is e. g. necessary to pull input data from different types of Pilot-Data, e. g. from an iRODS distributed data management system and a filesystem accessible via GridFTP. Intermediate and output data, however, is often best kept locally to facilitate efficient further processing (e. g. for additional filtering and/or aggregation) before moving it to another remote resource. Further, the Pilot-API enables users to share datasets contained in Data-Units. An application can e. g. pull in multiple public dataset, fuse it with a private dataset and produce another private dataset.

3.3 Affinities and Compute-Data Scheduling

Managing data locality in a distributed environment is a challenge due to the heterogeneous landscape of data/-compute infrastructures coupled with the large amount of dynamism associated with distributed environments.

An aim of Pilot-Data is to enable the efficient management and co-placement of data and compute units. Different investigations (e. g. [50, 31]) 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. We propose affinities as one way of modeling these correlations and relationships.

Affinities: Affinities are an abstraction for supporting efficient data/compute placements. They are an useful tool for modeling different resource topologies, application characteristics and allow for effective application-driven reasoning about resource topologies and data/compute dependencies.

The concept of affinity is used to describe the relationship between the different entities of a PJ-Framework, e. g. between CUs, DUs, Pilots and resources. This model enables the framework to reason about different trade-offs, e. g. to make decisions on whether to move data versus move the compute based on resource and bandwidth availabilities or to reason about different infrastructural properties (such as data access speeds). The framework relies on two kinds of affinities: (i) Pilot/resource affinities describe the relationship between multiple Pilots and the resources they are running on; (ii) compute/data affinities describe the relationships between CUs and DUs.

Resource affinities describe the relationship between a set

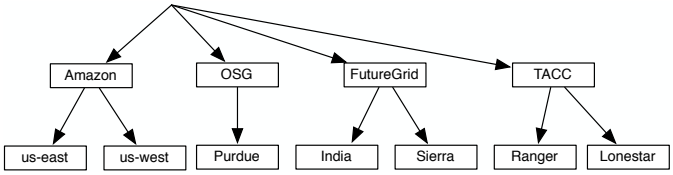

Figure 6: Affinities between Distributed Resources: Pilot-Data assigns each resource an affinity based on a simple hierarchical model. The smaller the distance between two resources, the larger the affinity.

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 (similar to the tree spawned by an DNS topology). The further the distance between two resources, the smaller their affinity. Figure 6 shows e. g. how a distributed system consisting of different types of cloud and grid resources can be modeled. Using such an 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 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. The mapping between resource and 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 e. g. 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 [50] that suggests that considering both data and compute during placement decisions leads to a better performance. As shown in Figure 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 sec and recheck whether Pilot has a free slot.
4. If no Pilot is found, the CU is placed in global queue and pulled by 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.

4. 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 suprisingly, 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 4.2 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 4.3 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 4.4 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.

4.1 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 con-

sidered:

- T_Q defined as the queue waiting time at a given resource. $T_{Q_{Pilot}}$ is the queue time of the Pilot-Job. $T_{Q_{Task}}$ is defined as the Pilot-internal queueing time.
- T_C defined as the compute time.
- T_X defined as the time to transfer data from one resource to another.
- T_S is the staging time, which is defined as T_X plus the time to register the data into the system.
- $T_R(R)$ 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 considerations, as well as the capability of the tools and middleware in use.

When both data and compute can be scheduled/placed, the decision about which entity to place/schedule 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 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 provides one way of providing this information to the "scheduling engine" of BigJob.

4.2 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 machine 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. Figure 7 illustrates T_S for different backends, i. e. the time necessary to populate a Pilot-Data on

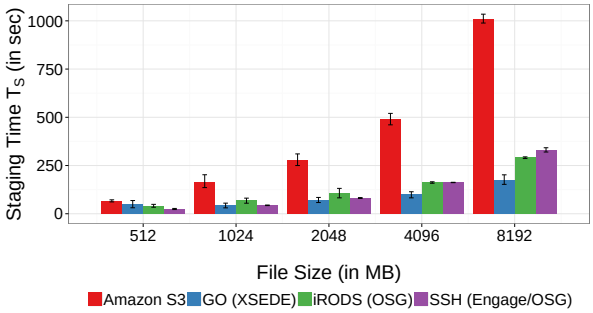

Figure 7: Pilot-Data on Different Infrastructures: Time to instantiate a Pilot-Data with a dataset of given size. For iRODS, we measure only the staging time and not the time required to replicate the data. SSH/iRODS perform well for small data sizes; Globus Online for larger data volumes. S3 performance is limited by Internet connectivity.

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 a directory on Lonestar accessed via Globus Online and in scenario 4 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. While for smaller data sizes SSH performs best, Globus Online particularly performs well for larger data volumes. 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 it is bound to the available Internet bandwidth. 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: (i) iRODS/OSG with group-based replication (osgGridFTPGroup), (ii) iRODS/OSG with sequential replication in which one replica is created after the other, and (iii) EGI with sequential replication using SRM/LFC. For these scenarios the data is replicated to resource sets of different types and sizes: on EGI to 11 SRM resources of the Dutch e-Science Grid, in the OSG sequential scenario to 6 iRODS resources and in the osgGridFTP-

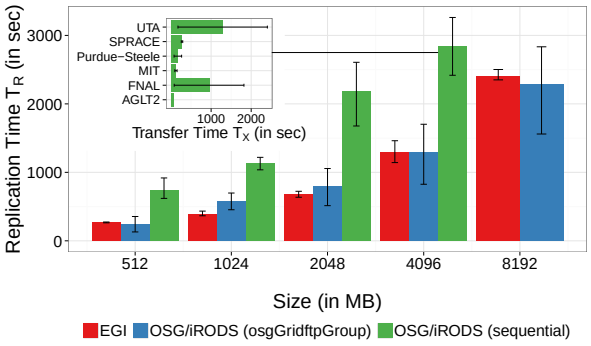

Figure 8: Replication on EGI and OSG: T_R on different infrastructures and resource sets: on EGI to 11 SRM resources of the Dutch e-Science Grid, in the OSG scenario to 6 iRODS resources and in the OSG (osgGridFTPGroup) scenario to 9 iRODS resources that are members of this group. The inset shows the distribution of T_R with respect to the different hosts for the 4 GB & OSG/iRODS scenario.

Group scenario to 9 iRODS resources that are members of this group.

Figure 8 illustrates the results: On OSG the T_R for the group-based replication is significantly better than for the sequential replication. In both cases, 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 (ii), the individual T_X are depicted in the inset of Figure 8.

Nevertheless, sequential replication is well suited for creating a small number of replicas. If the compute resources of a subset of sites are sufficient and available, it is e. g. not necessary to replicate all data across OSG. As can be seen, different sites have very different performance characteristics. Thus, the ability for applications to optimize data/-compute 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.

4.3 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 mounted to all compute nodes. On OSG this is not simply 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 [5]. 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 concurrent BWA instances. 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 [45] and GlideinWMS [51], a workload management system built on top of the Pilot capabilities of Condor-G/Glide-in [33].

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 data set 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.

Figure 9 shows the runtime of 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 queueing time mainly depend 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 8.3 GB data remotely leads to 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 ∼1,418 sec, $T_{D_{SSH}}$ is only ∼338 sec for scenario 4/5, but does not have a replication component (see inset of Figure 9). Thus, $T_{S_{CU}}$ is significantly higher for SSH than for iRODS. However, even after including T_D, the performance of iRODS is 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 re-

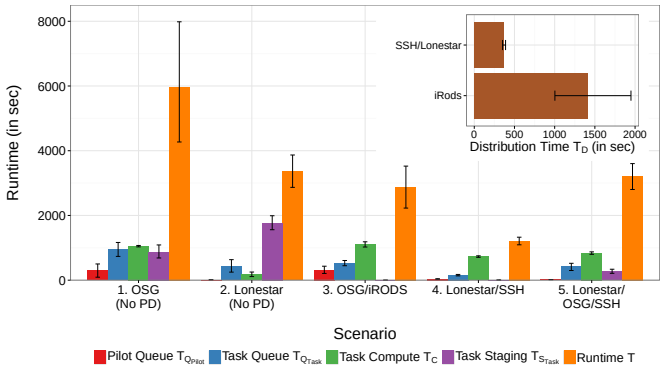

Figure 9: Pilot-based Runtimes for running BWA on 2 GB of sequence read files using 8 tasks for five different infrastructure configurations on XSEDE and OSG. The inset shows the distribution time (T_D) for the 8.3 GB dataset. T_D for iRODS includes the replication time. The usage of PD (scenarios 3-5) leads to an performance improvement compared to a naive data management (scenario 1-2).

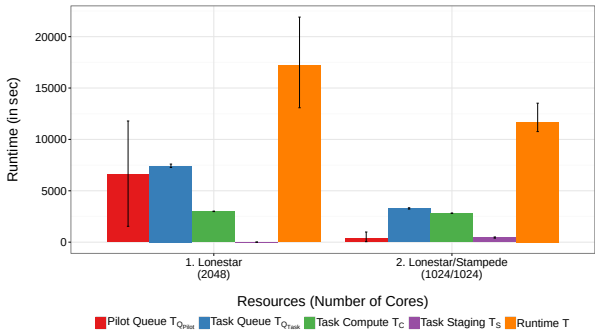

Figure 10: Running 1024 tasks each consuming 9 GB data on multiple XSEDE machines: The BJ-internal overhead observable in $T_{Q_{Task}}$ increased significantly compared to previous experiments with fewer tasks. Using multiple resources improved T despite the overhead introduced for data movements, in particular due to shorter queuing times.

duced. 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.

4.4 Understanding Scalability & Distribution

Offloading tasks to distributed/remote resources is a viable strategy to minimize bottlenecks, such as high queueing times, insufficient compute or IO resources for a certain workload, data placed on distributed resources, etc. During an initial assessment, we conducted an IO benchmark on two production XSEDE resources: Stampede and Lonestar by concurrently reading/writing 2 GB data/node to a shared, parallel Lustre filesystem for up to 256 nodes; i. e. we read/write up to 512 GB of data per experiment run. While the overall throughput numbers achieved on both machines are impressive, the observed numbers indicate that disk scaling may be constraining for data-intensive applications at large scale. Further, many resources provide significantly less IO capacity than these flagship machines. Also, actual IO 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.

The main barrier for distributing large-scale data-intensive applications across multiple resources resources is the necessity to move data. We evaluate two scenario for distribution compute and data onto multiple resources. 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 9,200 GB of data. For each tasks two cores are requested.

The results in Figure 10 show that adding additional resources can lead to a better time to completion despite the overhead introduced for data movement. As shown, a request for 2048 cores on a single resource (Lonestar in this case) leads to a high mean and variance in the queueing times. 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. Despite the necessity to move data for some of the tasks – moving the 9 GB input data from Lonestar to Stampede required on average 450 sec per task that is run on Stampede – T improves significantly. This improvements can be mainly attributed to the shorter Lonestar queueing times for the 1024 core Pilot (compared to the 2048 core Pilot in scenario 1). Further, the queuing time difference between both resources primarily determines the number of tasks that are executed on the remote resources. As the queue time on the original machine, i. e. Lonestar, is small relative to the time to transfer (only about 345 sec), the number of tasks moved to the remote machine is small (on average about 60 tasks). After the data-local Pilot becomes active, tasks are mostly placed on this local Pilot. Although in scenario 2, the improvements (in total about 30 %) can be attributed to shorter queuing times, it is often the case that data-intensive applications saturate the IO sub-system of clusters much sooner [52]; in such cases the performance advantages will not be limited to queue-time arbitraging.

In summary, Pilot abstractions enable large-scale applications to make interoperable use of different infrastructures and to use varied 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, particularly with larger ensembles, fault tolerance becomes a challenge. During the runs we observed failures due to walltimes or 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 trans-

fers). Also, more advanced strategies, such as data replication and more fine granular partitioning, will improve the distributability of large-scale applications.

5. DISCUSSION AND FUTURE WORK

At a macroscopic level, there are three primary compute-data placement paradigms for scalable distributed cyberinfrastructure: (i) when the data is essentially localized, either from being "poured" into a cloud 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 European Grid Initiative/Open Science Grid for particle physics and the discovery of the Higgs, 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.

At the most basic level Pilot-Data enhances the utility and usability of Pilot-Jobs by extending the use of Pilot-abstractions to data and thus by providing a missing critical component. 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 facing distributed applications and systems.

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 an unifying abstraction for distributed data. 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 provide 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.

Our discussion of affinities suggested they are a good abstraction for capturing relationships between computational tasks and associated data as well as helping map these dependencies to Pilots. Affinities can also be used to map Pilots and resources; in general a multi-level affinity model can be used to enhance multi-level scheduling capabilities, a feature often associated with Pilot-Jobs. In future work, we will explore the use of affinity-based scheduling abilities of Pilot-Data by supporting heterogeneous workloads (ensembles, data-intensive tasks, workflows, etc.). For this it is necessary to consider the entire application scenario and optimize Pilot-Data internal parameters (e. g. the number of threads for dispatching, downloading data etc.) accordingly. Further, we will work on supporting patterns, such as data partitioning and opportunistic replication to facilitate a more efficient data/compute distribution.

Acknowledgements

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-FG02-12ER-26115). 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 via for supporting the e-Science Research themes "Distributed Programming Abstractions" & 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.

6. REFERENCES

[1] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, Washington: Microsoft Research, 2009.

[2] NSF, "Cyberinfrastructure for 21st century science and engineering: Advanced computing infrastructure vision and strategic plan," http://www.nsf.gov/pubs/2012/nsf12051/nsf12051.pdf, 2012.

[3] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Heber, "Scientific data management in the coming decade," *SIGMOD Rec.*, vol. 34, no. 4, pp. 34–41, Dec. 2005. [Online]. Available: http://doi.acm.org/10.1145/1107499.1107503

[4] 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, and 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*, 2012, pp. 1–10.

[5] H. Li and R. Durbin, "Fast and accurate long-read alignment with burrows wheeler transform," *Bioinformatics*, vol. 26, pp. 589–595, March 2010. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp698

[6] "European grid infrastructure," http://www.egi.eu, 2013.

[7] "PRACE research infrastructure," http://www.prace-project.eu/.

[8] XSEDE: Extreme Science and Engineering Discovery Environment, https://www.xsede.org/.

[9] R. P. *et al*, "The open science grid," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012057, 2007.

[10] A. Luckow, M. Santcroos, A. Merzky, O. Weidner, P. Mantha, and S. Jha, "P*: A Model of Pilot-Abstractions," in *8th IEEE International Conference on e-Science 2012*, 2012.

[11] A. Luckow, L. Lacinski, and 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.

[12] Oracle, "Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System (White Paper)," 2007.

[13] "Amazon S3 Web Service," http://s3.amazonaws.com.

[14] GFFS – Global Federated File System, http://genesis2.virginia.edu/wiki/Main/GFFS.

[15] 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.

[16] A. Sim et al, "GFD.154: The Storage Resource Manager Interface Specification V2.2," Tech. Rep., 2008, oGF.

[17] 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, and B. Zhu, *iRODS Primer: integrated Rule-Oriented Data System.* Morgan and Claypool Publishers, 2010.

[18] A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and 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. Washington, DC, USA: IEEE Computer Society, 2004, pp. 182–191. [Online]. Available: http://dx.doi.org/10.1109/HPDC.2004.27

[19] J.-P. Baud, J. Casey, S. Lemaitre, and 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.

[20] I. Foster, "Globus online: Accelerating and democratizing science through cloud-based services," *IEEE Internet Computing*, vol. 15, pp. 70–73, 2011.

[21] "The grid information system," https://tomtools.cern.ch/confluence/display/IS/Home, 2013.

[22] T. Kosar and 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.

[23] A. Romosan, D. Rotem, A. Shoshani, and 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.

[24] "Apache Hadoop," http://hadoop.apache.org/, 2013.

[25] "Apache Hadoop NextGen MapReduce (YARN)," http://hadoop.apache.org/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/YARN.html, 2013.

[26] A. Aamnitchi, S. Doraimani, and 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.

[27] A. Amer, D. Long, and R. Burns, "Group-based management of distributed file caches," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, 2002, pp. 525 – 534.

[28] G. Ganger and M. F. Kaashoek, "Embedded inodes and explicit grouping: Exploiting disk bandwidth for small files," in *In Proceedings of the 1997 USENIX Technical Conference*, 1997, pp. 1–17.

[29] G. Fedak, H. He, and 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. Piscataway, NJ, USA: IEEE Press, 2008, pp. 45:1–45:12.

[30] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz, "Distributed processing of very large datasets with datacutter," *Parallel Comput.*, vol. 27, no. 11, pp. 1457–1478, Oct. 2001. [Online]. Available: http://dx.doi.org/10.1016/S0167-8191(01)00099-0

[31] K. Ranganathan and 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.

[32] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, and F. Zini, "Optorsim: A grid simulator for studying dynamic data replication strategies," *International Journal of High Performance Computing Applications*, vol. 17, no. 4, pp. 403–416, 2003.

[33] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, July 2002.

[34] J. Moscicki, "Diane - distributed analysis environment for grid-enabled simulation and analysis of physics data," in *Nuclear Science Symposium Conference Record, 2003 IEEE*, vol. 3, 2003, pp. 1617 – 1620.

[35] "Topos - a token pool server for pilot jobs," https://grid.sara.nl/wiki/index.php/Using_the_Grid/ToPoS, 2011.

[36] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid," *International Conference on High-Performance Computing in the Asia-Pacific Region*, vol. 1, pp. 283–289, 2000.

[37] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon: A Fast and Light-Weight TasK ExecutiON Framework," in *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing.* New York, NY, USA: ACM, 2007, pp. 1–12.

[38] E. Walker, J. Gardner, V. Litvin, and E. Turner, "Creating personal adaptive clusters for managing scientific jobs in a distributed computing environment," in *Challenges of Large Applications in Distributed Environments, 2006 IEEE*, 0-0 2006, pp. 95–103.

[39] 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.*, vol. 219, p. 062029, 2010.

[40] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[41] T. Maeno, K. De, T. Wenaus, P. Nilsson, G. A. Stewart, R. Walker, A. Stradling, J. Caballero, M. Potekhin, D. Smith, and T. A. Collaboration, "Overview of atlas panda workload management," *Journal of Physics: Conference Series*, vol. 331, no. 7, p. 072024, 2011. [Online]. Available: http://stacks.iop.org/1742-6596/331/i=7/a=072024

[42] Pilot API, http://saga-project.github.com/BigJob/apidoc/.

[43] P. K. Mantha, A. Luckow, and 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. New York, NY, USA: ACM, 2012, pp. 17–24.

[44] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, Nov. 1994.

[45] SAGA-Python, http://saga-project.github.io/saga-python/.

[46] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and 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.

[47] Redis, http://redis.io/, 2012.

[48] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[49] M. Santcroos, B. D. van Schaik, S. Shahand, S. D. Olabarriaga, A. Luckow, and 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.

[50] C. Miceli, M. Miceli, B. Rodriguez-Milla, and S. Jha, "Understanding Performance of Distributed Data-Intensive Applications," *Royal Society of London Philosophical Transactions Series A*, vol. 368, pp. 4089–4102, Aug. 2010.

[51] I. Sfiligoi, D. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, and F. Wurthwein, "The pilot way to grid resources using glideinwms," in *Computer Science and Information Engineering*, 2009, pp. 428 –432.

[52] Advancing Next-Generation Sequencing Data Analytics with Scalable Distributed Infrastructure, Joohyun Kim, Sharath Maddineni, Shantenu Jha, Computing and Concurrency: Practise and Experience (in press) 2013, (DOI: 10.1002/cpe.3013).