AN EXTENSIBLE AND SCALABLE PILOT-MAPREDUCE FRAMEWORK FOR DATA INTENSIVE APPLICATIONS ON DISTRIBUTED CYBERINFRASTRUCTURE

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Systems Science

in

The Department of Computer Science

by
Pradeep Kumar Mantha
B.Tech., Jawaharalal Nehru Technological University, 2006
August 2012

Acknowledgments

First and foremost, I express my gratitude to my advisor, Dr. Shantenu Jha for giving me the opportunity to work under him. His continuous guidance, encouragement and patience helped me immensely in my research. I sincerely thank Dr. Andre Luckow upon whose previous work I continued to build on and for his support and involvement in this work. I thank the members of my committee, Dr. Gabrielle Allen and Dr. Randall Hall for their valuable time. I would like to thank Dr. Joohyun Kim for the excellent discussions over the time. I thank all the people in the SAGA-Devel team, who were always responsive and helpful. I thank LONI, XSEDE and FutureGrid for the HPC resources I used. I sincerely thank LBRN, LA-SIGMA for important funding for my Masters, and Cybertools project (PI Jha) for SAGA with additional support from the Louisiana Board of Regents. Lastly, I thank my parents for all the love, support and encouragement.

Table of Contents

List of Tables

List of Figures

Abstract

The volume and complexity of data that must be analyzed in scientific applications is increasing exponentially. Often, this data is distributed; thus, the ability to analyze data by localizing it will yield limited returns. Therefore, an efficient processing of large distributed datasets is required, whilst ideally not introducing fundamentally new programming models or methods. For example, extending MapReduce - a proven effective programming model for processing large datasets, to work more effectively on distributed data and on different infrastructure (such as non-Hadoop, general-purpose clusters) is desirable. We posit that this can be achieved with an effective and efficient runtime environment and without refactoring MapReduce itself. MapReduce on distributed data requires effective distributed coordination of computation (map and reduce) and data, as well as distributed data management (in particular the transfer of intermediate data units). To address these requirements, we design and implement Pilot-MapReduce (PMR) - a flexible, infrastructure-independent runtime environment for MapReduce. PMR is based on Pilot abstractions for both compute (Pilot-Jobs) and data (Pilot-Data): it utilizes Pilot-Jobs to couple the map phase computation to the nearby source data, and Pilot-Data to move intermediate data using parallel data transfers to the reduce computation phase. We analyze the effectiveness of PMR over applications with different characteristics (e. g. different volumes of intermediate and output data). Our experimental evaluations show that the Pilot abstraction for data movement across multiple clusters is promising, and can lower the execution time span of the entire MapReduce execution. We also investigate the performance of PMR with distributed data using a Word Count and a genome sequencing application over different MapReduce configurations. We find that PMR is a viable tool to support distributed NGS analytics by comparing and contrasting the PMR approach to similar capabilities of Seqal and Crossbow, two Next Generation Sequencing(NGS) Hadoop MapReduce based applications. Our experiments show that PMR provides the desired flexibility in the deployment and configuration of MapReduce runs to address specific application characteristics and achieve an optimal performance, both locally and over wide-area multiple clusters.

Chapter 1
Introduction

There are various challenges associated with processing of data at extreme scales: which has become a critical factor in many science disciplines, e. g. in the areas of fusion energy (ITER), bioinformatics (metagenomics), climate (Earth System Grid), and astronomy (LSST) [6, 13]. The volumes of data produced by these scientific applications is increasing rapidly, driven by advanced technologies (e. g. increasing compute capacity and higher resolution sensors) and decreasing costs for computation, data acquisition and storage [12]. The number of scientific applications that either currently utilize, or need to utilize large volumes of potentially distributed data is immense. Recent advances in high-throughput DNA sequencing technologies such as Next-Generation Sequencing (NGS) platforms have resulted in unprecedented challenges in the areas of bioinformatics and computational biology [29, 37, 39, 4, 27]. These challenges are to some extent novel because of the need of the cross-cutting and integrated solutions leveraging algorithmic advances, tools and services, and scalable cyberinfrastructure and middleware.The challenges faced by these applications are interoperability, efficiently managing compute tasks, and moving data to the scheduled compute location.

Processing large volumes of data is a challenging task. MapReduce is an effective programming model for addressing this challenge. MapReduce involves two major computation phases called map and reduce, separated by a shuffle phase, which involves movement of intermediate data. MapReduce starts with chunking of the input data with user configured chunk size and assign each chunk to a single user defined mapper function in map phase. Once the map phase is completed i.e., when all the mapper functions completed, the output of map phase is scattered equally to the reduce phase based on a partitioning function. The reduce phase involves gathering of input data relevant to a reduce and execute the user defined reducer function on the data. MapReduce [8] as originally developed by Google aims to address the big data problem by providing an easy-to-use abstraction for parallel data processing. The most prominent framework for doing MapReduce computations is

Reprinted by permission of "MAPREDUCE'12 Workshop"

Apache Hadoop [3]. However, there are limitations to the current MR implementations: (i) They lack a modular architecture, (ii) are tied to specific infrastructure, e. g. Hadoop relies on the Hadoop File System (HDFS), and (iii) do not provide efficient support for dynamic and processing distributed data, e. g. Hadoop is designed for cluster/local environment, but not for a high degree of distribution.

It is a challenging requirement for the distributed application to manage the coupling between tasks and the resources. It becomes more complex in case of distributed cyber-infrastructure(DCI) model since the computing resources varies in load and capability dynamically. The ability to utilize a dynamic resource pool is an important attribute of any application that needs to utilize distributed cyberinfrastructure (DCI) efficiently. Pilot abstractions enable the clean separation of resource management concerns and application/frameworks. In particular, Pilot-Jobs have been notable in their ability to manage large numbers of compute units across multiple high performance clusters, providing decoupling application-level scheduling and system-level resource management. But, there is also a need of an abstraction to liberate applications from the challenging task of compute-data placement and scheduling. The Pilot-API [23] aims to address this issue by providing a unified API for managing both compute and data pilots. *BigData (BD)* is an extension of the BigJob framework (BJ) [36] to data. Both BigJob and BigData provide a full implementation of the Pilot-API and enable the management of resources, compute & data units as well as the relationships between them. Specifically, the Pilot-API promotes affinities as a first class characteristic for describing such relationships between compute and data elements and to support dynamic decision making.

A critical aspect of MapReduce, is the management of data and compute localities as well as the management of data movements, e. g. between the map and the reduce phase. In this thesis, we demonstrate the efficient support of these capabilities via the Pilot abstractions. We design and implement Pilot-MapReduce – a novel Pilot-based MapReduce implementation which enables clean separation of resource management and MapReduce application. Our Pilot-MapReduce framework demonstrates how Pilot abstractions are used for managing the map and reduce tasks and intermediate shuffle data between them and the advantages of the Pilot-based architecture in terms of flexibility, extensibility, scalability and performance; for example, we discuss the usability of Pilot-abstractions in designing dynamic execution workflows which involves multiple MapReduce computations.

Before we proceed further, it is critical to emphasize that it is not the aim of this thesis to suggest PMR as a replacement to Hadoop. However, we posit that where MR-based applications need to be employed over distributed data, including but not limited to clusters connected over WAN, or production distributed cyberinfrastructure such as XSEDE, EGI, PMR provides a flexible, extensible implementation of MR that is also efficient.

At this point, I would like to clarify my contribution to this work. The Pilot-API was developed in [23]. I used this Pilot-API to develop the Pilot-MapReduce framework. I evaluated the performance and scalability of the different MapReduce configurations using the Word Count application on natural language and on random data as well as the genome sequencing application. The work was published and accepted at MapReduce 2012 [26] and ECMLS 2012 workshops [25]. I was also a part, in profiling Pilot abstractions on a variety of infrstrucures like XSEDE [2], FutureGrid [10] and OSG [41]. This work was done as part of a publication [24] submitted to SC 2012, which is currently under review. My contributions also involve extending BigJob capabilities on FutureGrid/XSEDE machines, which lack necessary software infrastructure support for scaling applications on distributed clusters. For example, due to the non-availability of Globus on FutureGrid, BigJob usage was limited only to a single cluster. This led to the development of *pbs-ssh* plugin, which extended BigJob capabilities to utilize multiple clusters of FutureGrid. Later the capabilities of *pbs-ssh* are further enhanced to support Kraken (CRAY XT5) XSEDE cluster, which enabled remote job submissions to Kraken. To enable application scaling completely across different infrastructures and scheduling systems, developed a *sge-ssh* plugin (similar to *pbs-ssh* plugin), to support remote job submissions to SGE XSEDE machines like LoneStar and Ranger. Both these plugins provide flexibility and extensibility of BigJob to support distributed workflows on XSEDE, FutureGrid and LONI. As a part of supporting users of BigJob on production infrastructures like LONI and XSEDE, I was involved in providing the templates and documentation to develop workflows using BigJob. The documentation involves the deployment and usage aspects of BigJob. The documentation and the developments were used by Extended Collaborative Support Service (ECSS) project for running Molecular Dynamic simulations on large number of cores. The work related to ECSS [35] project has been submitted to XSEDE 2012 and is currently under review.

This thesis is organized as follows: Chapter 2 describes the limitations of traditional compute and data management and provides an overview of Pilot abstractions as a solution to these problems and then discuss the design and implementation of the Pilot-MapReduce framework for distributed data analysis. In Chapter 3 we evaluate the performance and scalability of Pilot-MapReduce. Chapter 4 provides an overview of BigJob applications/workflows executed on XSEDE and LONI. The conclusion and future work are given in Chapter 5.

Chapter 2
Pilot Abstractions

In this section we describe some of the different components of Pilot Abstractions that are important for understanding this work, and their application and importance on distributed cyber infrastructure.

First, in Section 2.1 we will describe distributed cyber-infrastructure, In Section 2.2 we focus on the traditional job submission methodologies and their problems. In Section 2.3 we describe the Pilot Abstractions and their implementation for both compute and data . In Section 2.4 we describe how Pilot-Jobs and Pilot-Data provide effective management of distributed cyber-infrastructure.

2.1 Distributed Cyber-Infrastructure

Distributed cyber-infrastructure(DCI), in contrast to a static resource utilization model utilizes computing resources, which varies in load and capability. Domain Scientists understand scientific applications related to their field by experimenting on DCI. Some of the requirements and characteristics of these applications require broad usage of DCI which are significantly different from regular HPC applications in several fundamental ways. Often, distributed applications are designed to support peak utilization of resources by a number of tasks. On distributed dynamic resource pool, it is an important attribute of any application to utilize the infrastructure efficiently. Production Grid Infrastructures (PGIs) as well as the Programming Systems and Tools (PST) used to develop distributed applications need to address these and other fundamental distributed application characteristics [22].

2.2 Traditional compute and data management

Existing PST support number of applications to utilize DCI. Even though several distributed applications use distributed infrastructures successfully, either those applications failed to use distributed infrastructures effectively or have had to implement new capabilities at one or more levels, which includes application, programming system, middleware and/or infrastructure level. The urge to utilize

distributed infrastructures effectively made the design and development of distributed applications more complex task [23]. For example, many programming systems and tools for distributed applications are either incomplete and/or often out-of-phase with requirements or inflexible with respect to application needs, e.g. tools that support the master-worker paradigm often only address failures of workers and not of the master. Additionally, tools and development systems often don't support the specific usage modes that maybe required for a certain application scenario, with the level of robustness and scalability required, i.e., solutions work well in small or controlled environments, but not at-scale. These and other concerns have motivated developers to "roll out their own" capabilities, in turn further adding to an existing large range of tools, programming systems and environments and adding to challenges of providing interoperability. Thus to the extent possible, extensibility and interoperability must be built as fundamental design objective of PST for distributed applications and infrastructure. Although it will not be possible to support all of the following properties, PST should address some of these aspects: (i) new application domains and usage-modes, (ii) extending the functionality supported, (iii) extension to new infrastructures, (iv) extend across scales of operation, (v) uptake by communities other than the developer (community usage) and, (vi) reuse and support patterns and abstractions for distributed computing. The extend to which the above design objectives will succeed depends not only on the resulting programming system, but also on the availability of usable and extendable abstractions and their suitability for given production infrastructures. Interestingly, the Pilot-Job abstraction has been widely used across several different PGIs. However, the existing Pilot-Job frameworks are all heavily customized and often tightly coupled to a specific infrastructure, and not extensible or usable across different systems, e.g. there is no such "unifying" and "extensible" Pilot-Jobs that supports a range of application types and characteristics. [22, 23]

Many scientific applications have immense data requirements, which are projected to increase dramatically in the near future [23]. The management of data in distributed systems remains a challenge due to various reasons: (i) the placement of data is often decoupled from the placement of Compute Units i. e. the application must often manually stage in and out its data using simple scripts; (ii) heterogeneity, e. g. with respect to storage, filesystem types and paths, often prohibits or at least

complicates late bind- ing decisions; (iii) higher-level abstraction that allow applications to specify their data dependencies on an abstract, logical level (rather than on file basis) are not available; (iv) due to lack of a common treatment for compute and data, optimizations of data/compute placements are often not possible. In addition, applications must cope with various other challenging, data-related issues, e.g. varying data sources (such as sensors and/or other application components), fluctuating data rates, transfer failures, optimizations for different queries, data-compute co-location etc. While these issues can be in principal handled in an application-specific way, the usage of higher-level abstractions, such as a common Pilot-based abstraction for compute and data is preferable.

2.3 Pilot abstractions for compute and data

Pilot-abstractions provide effective management of compute and data units and the relationships between them(affinities). They liberate the applications from the challenging requirement of assigning/scheduling the compute or data unit onto a particular resource.

2.3.1 BigJob - SAGA Pilot Job

Workload management and resource scheduling can lead to significant dynamic fluctuations in workloads and resources, reducing the overall efficiency and speed of the desired calculations. A common approach for decoupling these competing allocation problems is the use of *pilot-jobs (PJ)*. The PJ abstraction is also a promising route to address additional requirements of distributed scientific applications [16, 20], such as application-level scheduling.

A SAGA-based PilotJob, BigJob (BJ) [36, 22], is a general-purpose pilot-job framework. BigJob has been used to support various execution patterns and execution workflows [38]. For example, SAGA-BigJob was used to execute scientific applications categorized as embarrassingly parallel applications and loosely coupled applications on scalable distributed resources [14, 15]

Figure 2.1 illustrates the architecture of BJ. BJ utilizes a Master-Worker coordination model. The BigJob-Manager is responsible for the orchestration of pilots, for the binding of sub-tasks. For submission of the pilots, SAGA relies on the SAGA Job API, and thus can be used in conjunction

with different SAGA adaptors, e. g. the Globus, the PBS, the Condor and the Amazon Web Service adaptor. Each pilot initializes a so called BJ-agent. The agent is responsible for gathering local information and for executing tasks on its local resource. The SAGA Advert Service API is used for communication between manager and agent. The Advert Service (AS) exposes a shared data space that can be accessed by manager and agent, which use the AS to realize a push/pull communication pattern. The manager pushes a sub-job to the AS while the agents periodically pull for new sub-jobs. Results and state updates are similarly pushed back from the agent to the manager. Furthermore, BJ provides a pluggable communication & coordination layer and also supports alternative c&c systems, e. g. Redis [34] and ZeroMQ [42].

In many scenarios it is beneficial to utilize multiple resources, e. g. to accelerate the time-to-completion or to provide resilience to resource failures and/or unexpected delays. BJ supports a wide range of application types, and is usable over a broad range of infrastructures, i. e. it is general-purpose and extensible (Figure 2.1). In addition there are specific BJ flavors for cloud resources such as Amazon EC2 and Microsoft Azure that are capable of managing set of VMs, as well as a BJ with a Condor-G based backend.

BJ supports dynamic resource additions/removals as well as late binding. The support of this feature depends on the backend used. To support this feature on top of various BigJob implementations that are by default restricted to single resource use (e. g. BJ), the concept of a BigJob pool is introduced. A BigJob pool consists of multiple BJs (each BigJob managing one particular resource). An extensible scheduler is used for dispatching compute units to one of the BJs of the pool (late binding). By default a FIFO scheduler is provided.

2.3.2 BigData - SAGA Pilot Data

Analogous to Pilot-Jobs, *Pilot-Data* (PD) abstraction provides late-binding capabilities for data by separating the storage allocation and application-level Data Unit [23]. For this purpose, the API defines the *Pilot-Data (PD)* and *Data Unit (DU)* entity: A PD function as a placeholder object that reserves storage spaces for a set of DUs.

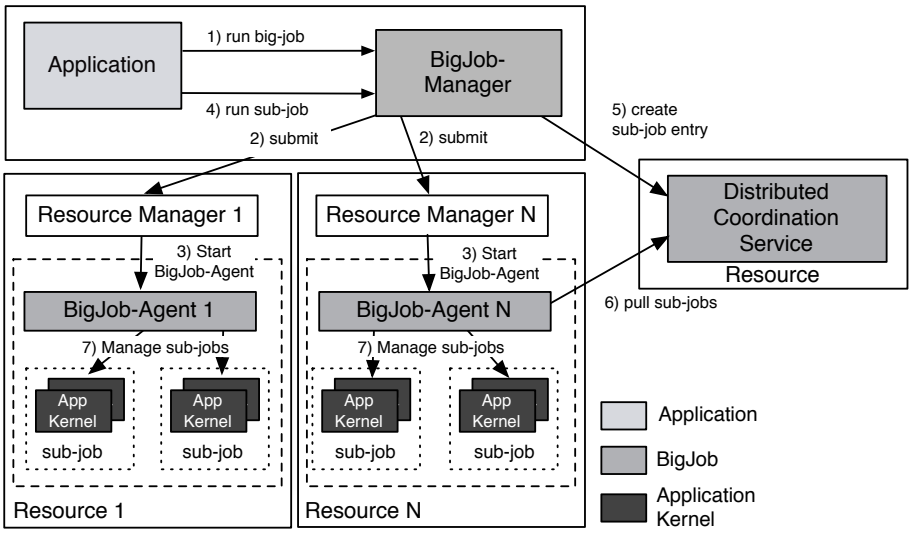

FIGURE 2.1: BigJob Architecture: The core of the framework, the BigJob-Manager orchestrates a set of pilots. Pilots are started using the SAGA Job API. The application submits WUs, the so-called sub-jobs via the BigJob-Manager. Communication between the BJ-Manager and BJ-Agent is done via a shared data space, the Advert Service. The BJ-Agent is responsible for managing and monitoring sub-jobs. From Ref. [22]

BigData (BD) is an implementation of the Pilot-Data abstraction. BigData is designed as an extension of BigJob [36] – a SAGA-based Pilot-Job implementation. Figure 2.2 provides an overview of the architecture of BigData. Similar to BigJob, it is comprised of two components: the BD-Manager and the BD-Agents, which are deployed on the physical resources. The coordination scheme used is Master-Worker (MW), with some decentralized intelligence located at the BD-Agent. Analogous to BJ, the SAGA Advert Service [28] provides a distributed communication mechanism in a push/pull mode.

The BD-Manager is responsible for (i) meta-data management, i. e. it keeps track of all PD and associated DUs, (ii) for scheduling of data movements and replications (taking into account the application requirements defined via affinities), and (iii) for managing data movements activities. BigData supports plug-able storage adaptors – currently an adaptor for SSH, WebHDFS [40] and Globus Online [9] is provided.

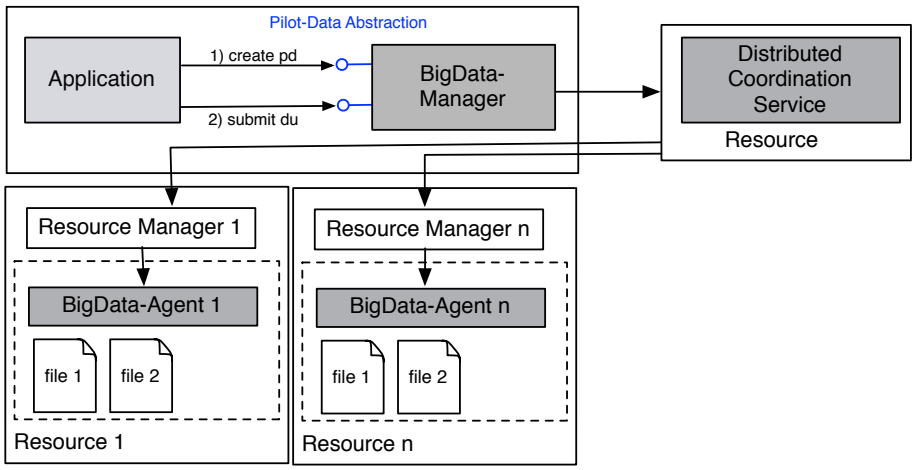

FIGURE 2.2: BigData Architecture and Interactions

2.3.3 Pilot API and Affinities

A critical requirement for data-intensive application, is the management of compute and data dependencies, also referred to as *affinities*. The Pilot-API promotes affinities as a first class characteristic for describing relationships between data and/or compute supporting dynamic decision making. Unfortunately, most production infrastructure lack system-level support for affinities, e. g. resource localities cannot be introspected. Data storage in particular in distributed settings, such as in the XSEDE or the EGI environment, is often a black box for the application with unknown quality of services, i.e., the application usually does not know what bandwidths and latencies it can expect. To address these deficiencies the Pilot-API introduces affinities at the application-level: applications can associate compute and data units with affinity labels. The BigJob/BigData runtime ensures that CUs and DUs are placed with respect to the affinity requirements.

The PMR framework assigns each file output from a map task to a reduce partition. For each reduce partition, a DU containing the respective files is created. Then, PMR submits the reduce CUs and DUs using the Pilot-API. The affinity-aware scheduler assigns CUs and DUs to appropriate resources taking into account data localities and minimizing the amount of necessary data movements, i. e. if possible a CU is always moved to a DU. The Pilot-API and BigJob/BigData provide an effective

way to manage both compute and data units and the relationships between them liberating the applications from the challenging task of assigning/scheduling/managing Compute and Data Unit.

2.4 Scalability and Usability of Pilot Abstractions

Pilot abstractions proved to provide effective scaling at various levels [38, 23, 22] and they can be defined as

- scale-up: Refers to the ability (performance) of using many cores efficiently
- scale-out: Measures the number of tasks that can be concurrently executed & managed
- scale-across: Measures the number of distinct compute homogenous or heterogenous resources that an application can utilize.

We demonstrate the usability of Pilot-abstractions to design a flexible, infrastructure-independent runtime environment for MapReduce application. Pilot-MapReduce heavily relies on Pilot abstractions for de-coupling the MapReduce runtime, application-level scheduling and resource management providing a high degree of flexibility and extensibility.

2.5 Pilot-MapReduce – A Pilot-based MapReduce Implementation

Pilot-MapReduce (PMR) is a Pilot-based implementation of the MapReduce programming model. By decoupling job scheduling and monitoring from the resource management using Pilot-based abstraction, PMR can efficiently re-use the resource management and late-binding capabilities of BigJob and BigData. PMR exposes an easy-to-use interface, which provides the complete functionality needed by any MapReduce algorithm, while hiding the more complex functionality, such as chunking of the input, sorting the intermediate results, managing and coordinating the map & reduce tasks, etc., which are implemented by the framework.

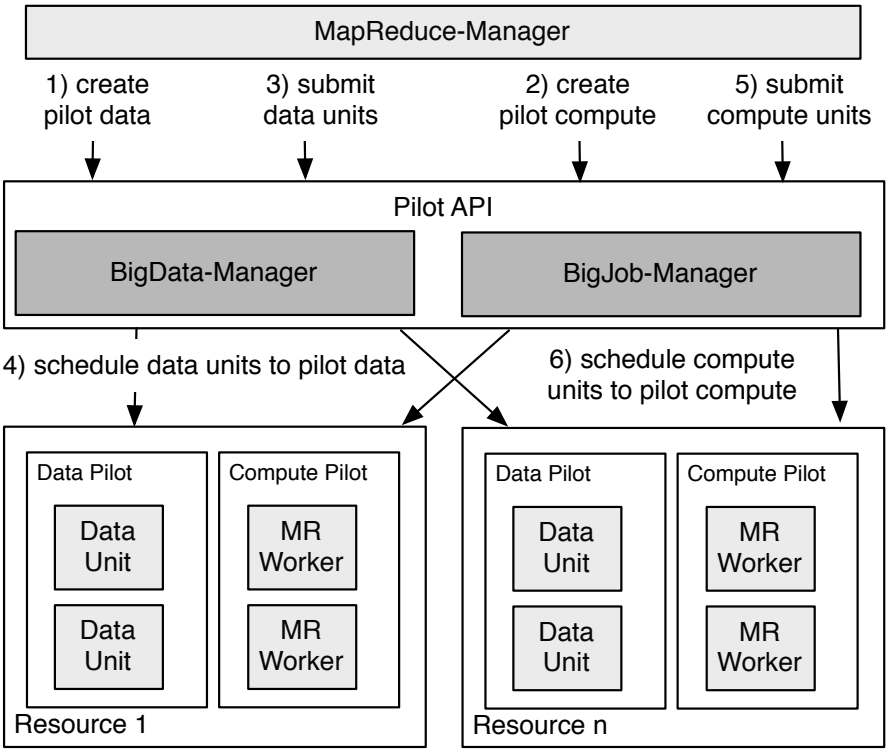

FIGURE 2.3: Pilot-based MapReduce: Each pilot (both compute and data pilot) can be associated with an affinity label. The BigData and BigJob Manager will ensure that CUs and DUs are placed with respect to these requirements.

2.5.1 Architecture of Pilot-MapReduce

Pilot-MapReduce introduces a clean separation of concerns between management of compute and data on the one hand, with their scheduling in a distributed context. The pilot abstractions enable the easy acquisition of both compute and storage resources.

Figure 2.3 shows the architecture of the Pilot-MapReduce framework. PMR relies on BigJob to launch MapReduce workers through a set of Pilots. The MR Workers are responsible for running chunk, map and/or reduce tasks. MR-Manager packages data chunks into DUs and associates them with Pilot-Data objects, which are placed close to Pilot-Computes by BigData. The MR-Manager can focus on orchestrating this resource pool.

The flow of a typical MapReduce application involves the chunking of the data, the execution of the map compute tasks, shuffling and moving the intermediate data to the reduce task and finally the execution of the reduce tasks. Pilot-MapReduce utilizes a set of compute and data pilots for this application workflow:

A. Initially, the MR-Manager allocates a set of compute and data resources by starting one (or most often a set of) compute and data pilots on different resources. In general, on each resource one compute and one data pilot is co-located. The data pilot is either created with reference to local input data or the input data is moved to the data pilot after its creation.

B. **Chunking:** The MR-Manager executes a CU on each resource, which splits the input data on the respective resource with respect to the defined chunk size. Each chunk is stored in a new DU. BigJob and BigData – in particular the `ComputeDataService` – are used as the common abstraction for managing the Compute Units and Data Unit.

C. **Mapping:** The MR-Manager assigns a *map* CU to each chunk created in step B. Again, BJ is used for managing the CUs. BJ and BD ensures that each CUs is co-located with an appropriate DU taking into account data localities and minimizing the amount of data movements.

D. **Shuffling:** After the map phase is completed the output data is sorted and partitioned. For each partition a DU is created. Each partition is then processed by a reduce task. For this purpose, the MR-Manager assigns each reduce CU to a DU. Each DU comprises of a group of sorted, partitioned map output files. CUs and DUs are then submitted through the `ComputeDataService` of BJ and BD. The affinity-aware scheduler ensure that CUs are assigned to local DUs minimizing the amount of data transfers. For each reduce task a Data Unit containing the necessary input files is created and submitted.

E. **Reducing:** The *reduce* tasks are prepared and executed on the DUs representing the intermediate data. The management of the data transfers is done by BJ/BD taking into account the specified affinities.

F. The Pilots are terminated.

The PMR relies on the master/worker coordination model, i. e. a central MR-Manager orchestrates a set of MapReduce workers, which in turn are responsible for executing map and reduce tasks. The MR-Manager utilizes BigJob and BigData, and in particular the central `ComputeDataService` for executing mapper and reduce tasks. This architecture can also efficiently support workloads that currently not supported well enough by Hadoop, e. g. iterative applications.

2.5.2 Compute and Data Management

The Pilot-API provides a well-defined interface for supporting the late-binding of compute and data units decoupling resource assignment from resource usage. Using BJ and BD, PMR can allocate both storage and compute resources, which can then be flexibly utilized for executing map and reduce tasks as well as for storing both intermediate and output data.

The API also allows the expression and management of relationships between data units and/or compute units. BigJob and BigData provide an implementation of the Pilot-API. These frameworks ensure that the data and compute affinity requirements of the MapReduce applications are met for each step of the MapReduce workflow. For example, in the shuffle phase for each reduce task a DU and CU is generated. These are then submitted to BigJob and BigData framework, which handles the scheduling, transfer of the DU and execution of the CU. PMR assigns a resource affinity to each DU and CU. BJ and BD then ensure that each CU is co-located to the right DU.

The efficiency of PMR on multiple resources depends on the management of the the intermediate data. BigData not only provides flexibility to manage the relationship between data and compute units, but also allows *parallel* data transfers between machines and between data units. BigData is used for moving the intermediate output files of the mapper tasks to the resource where the reduce compute units are executed.

Interestingly, Hadoop also utilizes a job and task tracker: the job tracker is the central manager that dispatches map and reduce tasks to the nodes of the Hadoop cluster. On each node the task tracker is responsible for executing the respective tasks. The main limitation of this architecture is the fact that it intermixes both cluster resource management and application-level task managements. Thus, it is not easily possible to integrate Hadoop with another resource management tool, e. g. PBS or Torque. Also, the job tracker represents a single point of failure and scalability bottleneck.

```
pds = PilotDataService()
pd_desc=
{"service_url":"ssh://india.futuregrid.org/pilotdata",
"size":100,
"affinity_datacenter_label":'india',
"affinity_machine_label":'india'}
pd=pds.create_pilot( pilot_data_description=pd_desc)
cds.add_pilot_data_service(pds)
```

Listing 2.1: Pilot Data Creation: Instantiation of a Pilot Data using Pilot Data Description

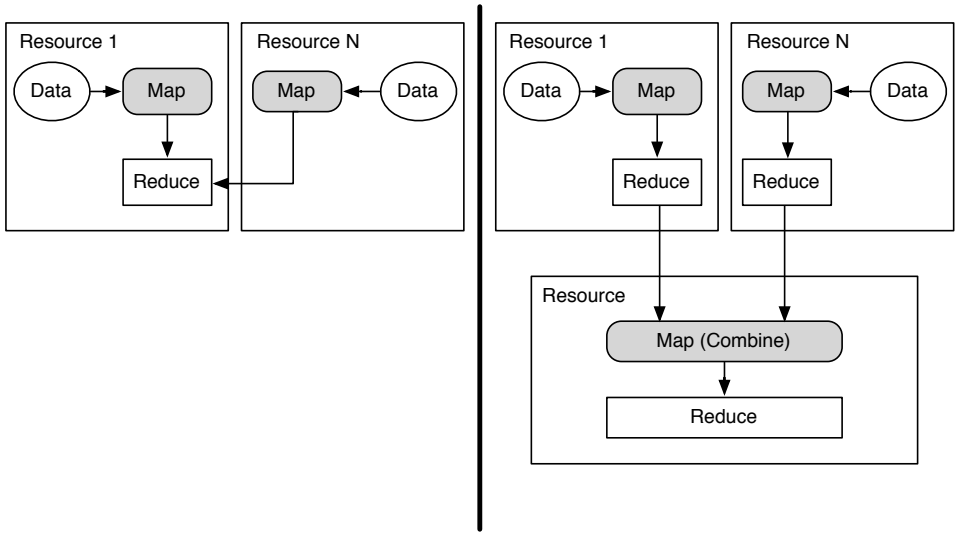

FIGURE 2.4: Pilot-MapReduce Deployment Scenarios: In the distributed scenario (left), the mapping tasks are run close to the data; reduced tasks are then run on a single resource. In the hierarchical scenario (right) two full MapReduce runs are conducted.

2.5.3 Distributed and Hierarchical MapReduce

An increasing amount of data that scientific applications need to operate on is distributed. Often data generation and processing are far apart: For example, the Earth Science Grid federates data of various climate simulations [5]. Meta-genomic workflows need to process and analyze data generated by various sequencing machines [13]; the localization onto a single resource is often not a possibility.

Several options for running Hadoop on distributed data have been proposed [7]: (i) in a global MapReduce setup one central JobTracker and HDFS NameNode is used for managing a distributed set of resources; (ii) in a hierarchical MapReduce setup multiple MapReduce clusters are used: a MapReduce cluster close to the data source for pre-processing data and a central cluster for aggregating the different de-central data sources. The volume of the pre-processed data is generally lower and thus, can be easily moved to another processing resource.

Ref [7] shows that a hierarchical Hadoop configuration leads to a better performance than a global Hadoop cluster for some applications. A drawback of this approach is the increased complexity: Hadoop is not designed with respect to a federation of multiple MapReduce clusters. Setting up such a system typically requires a lot of manual effort.

Pilot-MapReduce supports different distributed MapReduce topologies: (i) *local*, (ii) *distributed* and (iii) *hierarchical*. A local PMR performs all map and reduce computations on a single resource.

Figure 2.4 shows options (ii) and (iii): A distributed PMR utilizes multiple resources often to run map tasks close to the data to avoid costly data transfers; the intermediate data is then moved to another resource for running the reduce tasks. BigJob and BigData are used for managing CUs and DUs and the necessary data movements. In contrast, in a hierarchical PMR the outputs of the first complete MapReduce run are moved to a central aggregation resource. A complete MapReduce run is then executed on this resource to combine the results.

Pilot-MapReduce uses the Pilot-API as an abstraction for compute and data resources, as well as managing both Compute Units (i. e. map and reduce tasks) and Data Unit. Using these abstractions, PMR can efficiently manage data and compute localities and operate on a dynamic and distributed pool of storage and compute resources. Using descriptive affinities label the data flow between CUs, i. e. the transfer of the intermediate data, can be efficiently managed. Using this capability PMR can be easily scaled out to multiple resources to support scenarios (ii) and (iii).

Chapter 3
Evaluation of Pilot-MapReduce

In this chapter we analyze the performance and scalability of Pilot-MapReduce and compare it to Hadoop MapReduce using different applications. For this purpose we run several experiments on FutureGrid [10]. We run the experiment on the following FutureGrid resources: India, Sierra and Hotel. Each experiment is repeated at least three times. For our Hadoop experiments, we use Hadoop 0.20.2. At the begin of each run a Hadoop cluster is started via the Torque resource management system on a specified number of nodes. The first assigned node is used as master node running the Hadoop JobTracker and the NameNode. The HDFS replication factor is set to 2 and number of reduces to 8.

3.1 MapReduce-Based Applications

MapReduce has been utilized in various science applications. A key performance factor is the amount of data that must be moved through the MapReduce system. The degree of data aggregation of the map tasks is thus, an important characteristic of a MapReduce application [7].

MapReduce application can be classified with respect to different criteria: (i) the volume of the intermediate data (i. e. the size of the output of the map tasks), and (ii) the volume of the output data, (i. e. the size of reduce phase output), and the relative proportion of these data volume. In the following we investigate two application scenarios: Word Count and a Genome Sequencing application.

3.1.1 Word Count

The Word Count application is the basis for many machine learning use cases, used e. g. for the classification of documents or clustering. Word Count generates a large volume of intermediate data (∼200%). The volume of the output data depends on the type of input data, e. g. the size of the output data is larger for a random input than for an input in a natural language.

3.1.2 Genome Sequencing (GS)

High-throughput genome sequencing techniques provided by Next Generation Sequencing (NGS) platforms are changing biological sciences and biomedical research. The data volumes generated by sequencing machines is increasing rapidly. The distributed processing of this data requires a sophisticated infrastructure. For this purpose, we utilize MapReduce to model an important part of the sequencing workflow i.e, the read alignment and the duplicate removal.

3.1.3 Short Read Alignment

Short reads alignment and the de-novo assembly are the required first steps in every pipeline software tool that aims to analyze sequencing data from NGS platforms. De-novo assembly still remains a challenge, because of complications arising from the short length of sequencing reads from NGS machines. In most of situations, read alignment (or mapping process) is the first task of NGS workflows, and two Hadoop-based tools, Seqal and Crossbow provided two mapping tools, BWA and Bowtie, respectively.

In general, for RNA-Seq data analysis, in particular with eukaryote genomes, the spliced aligner such as TopHat [31] is used. In our work, we consider an alternative strategy, to use a non-spliced aligner and later splicing events are detected separately, justifying the use of non-spliced aligners such as BWA and Bowtie for the RNA-Seq data. These non-spliced aligner tools mapped reads onto human reference genome hg19.

3.1.4 Post-Alignment

Duplicate read removal step might be required after short read alignment, because sample preparation processes before sequencing might contain artifacts stemming from high-throughput read amplification; many duplicates introduced are not relevant to true biological conditions.

Seqal is a Hadoop MapReduce application which implements the alignment in map phase using BWA aligner and a duplicate removal step using the same criteria as the Picard MarkDuplicates [33, 32] in reduce phase. We use two implementations of the workflow: the Hadoop-based Seqal [33]

application and a custom implementation of this workflow GS/PMR. Both application implement the read alignment in the mapping phase of the application using BWA aligner [18]. In the Seqal case the duplicate removal in the reduce phase is implemented using Picard's rmdup [30]. The GS/PMR reduce phase is not an exact implementation of Seqal's Picard rmdup implementation.We developed a custom script in python which is based on duplicate removal description provided in [33]. The GS/PMR reducer removes duplicate reads based on the key fields-chromosome, position, strand of GS/PMR mapper output.

Crossbow [17] is a scalable software automatic pipeline, combines Alignment and SNP finding tools for DNA sequencing analysis. Crossbow contains 4 steps - preprocessing, Alignment, SNP finding and post processing. Each step is a Hadoop streaming-based MapReduce application and the output of each step is stored in HDFS and read from HDFS by the next step. In our experiments we focused on Crossbow alignment which uses Bowtie aligner in map phase and has a dummy reducer.

3.2 Characterizing Word Count

In the first experiment, we benchmark the performance of Pilot-MapReduce and Hadoop using a simple Word Count application on a single resource. For both frameworks, 8 nodes on India machine are used. In all scenarios the input data is pre-staged on the respective resources, i. e. for Hadoop the data is located in HDFS, for PMR the data is stored on a shared file system. We set the total number of reduces to 8 for both Hadoop and Pilot-MapReduce; further, the default chunk size of 128 MB is used. A HDFS replication factor of 2 is used.

The runtime of PMR includes the time to chunk input data, running the mapping CUs, shuffling (which again comprises of sorting and the intermediate data transfer, and finally running the reduce CUs. Figure 3.1 shows the results. The runtime of Hadoop MapReduce includes the time to load input source data into HDFS and MapReduce runtime.

The time to solution increased linearly as data size increased; the performance of both Hadoop and PMR is comparable up to 8 GB. However, for the largest volumes of input data we examined, PMR shows a better performance than Hadoop. In particular, the setup, map and shuffle phase in the Hadoop case are longer. Both the map and shuffle phase are the most data-intensive phases –

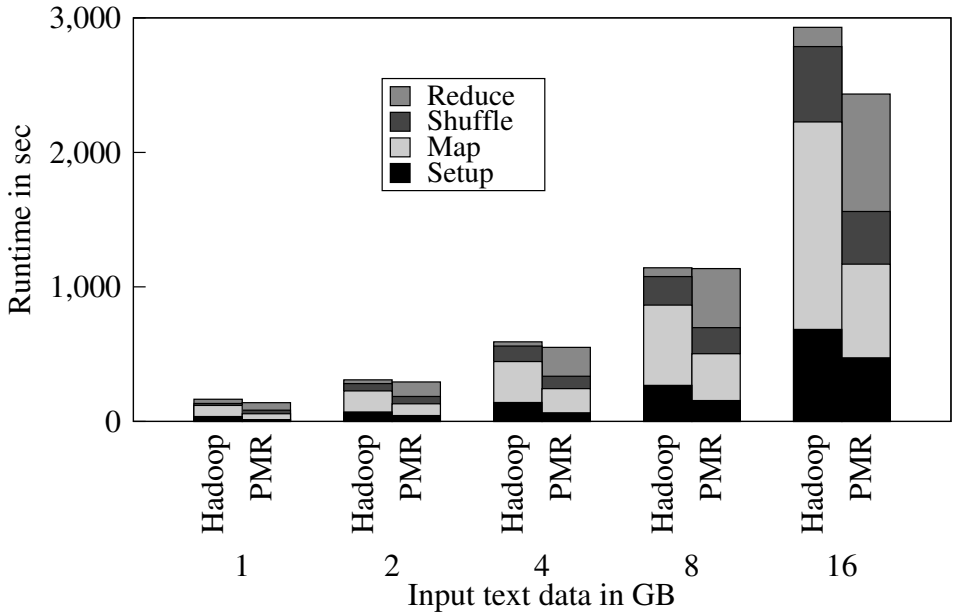

FIGURE 3.1: Word Count PMR vs. Hadoop: The performance of Hadoop and PMR is comparable. The runtime increase with the input data size. Hadoop tasks have a notable higher startup time.

Word Count needs to read all input files and generates intermediate data with the size of about 200 % of the input data. The worse performance of Hadoop indicates a potential issue with HDFS. PMR relies mostly on the shared file system for handling the intermediate data.

3.3 Characterizing Genome Sequencing

In this section, we compare and contrast GS/PMR and Seqal. For both applications, we utilize the same set of input data comprising of different sizes of read files and the reference genome. Seqal, however, expects the input data in a different format (prq instead of fastq); thus, the data was previously converted to meet the Seqal requirements. For PMR, the fastq files from sequencing machines are directly used; further, a custom chunk script is used to chunk the fastq files based on the number of reads. We make sure that the chunk size for both Seqal and PMR is equal. For both frameworks, a total of 4 nodes on FutureGrid Sierra machine, 8 reduces, 2 workers/node, default chunk size of 128 MB is used. For Hadoop based Seqal, the replication factor of two is used. Since Seqal and GS/PMR utilize different duplicate removal tools in the reduce phase, we focus our investigation on the map phase.

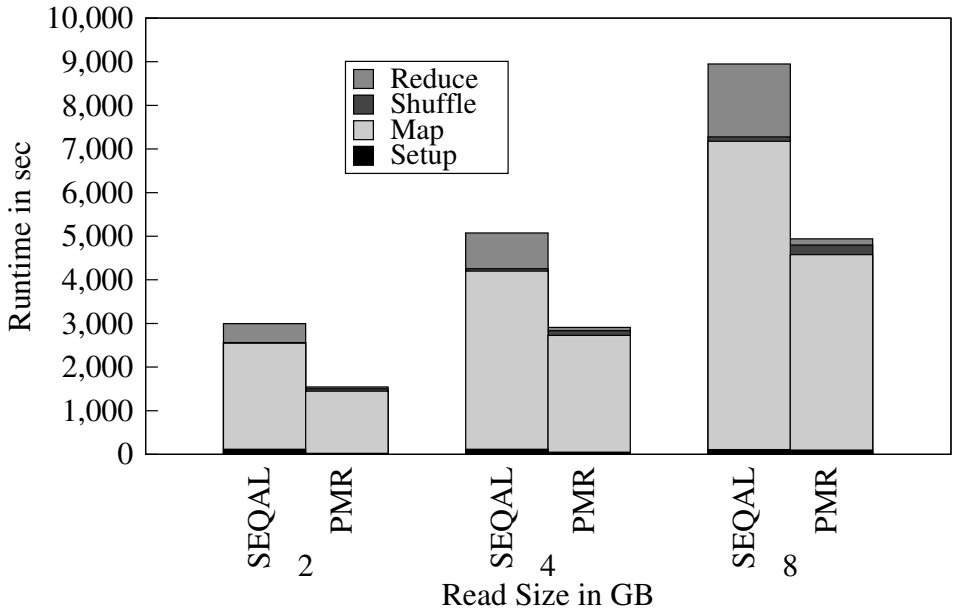

FIGURE 3.2: Seqal and GS/PMR: GS/PMR provides a marginal better performance that Seqal. The overhead of Seqal is mainly attributed to the used HDFS configuration using a shared file system.

Figure 3.2 shows the results of both applications. In the setup time of Seqal, Hadoop copies the reference genome archive to all the nodes and extracts it so it is available locally. In comparison to Word Count both GS applications are more compute intensive, i. e. the ratio between computation in the map phase and the size of the input data is significant larger. Notably, Seqal requires a longer time-to-completion than GS/PMR. Both the map and reduce phase of Seqal are longer. While the map phase of Seqal relies on the same BWA implementation as GS/PMR, the reduce phase uses Picard's rmdump [30] for duplicate removal, which has a significant longer runtime than the duplicate removal process in the reduce phase of GS/PMR.

This is mainly caused by a non-optimal configuration of Hadoop: The local disks available on FutureGrid is too small for the used input data; thus, HDFS had to be configured to utilize a shared, distributed file system, which leads to a non-optimal performance during the I/O intensive map phase. The difference in the reduce phase mainly originate from the different implementations of the duplicate removal process in Seqal and GS/PMR.

3.4 Distributed and Hierarchical MapReduce

In this section, we evaluate the performance and scalability of the (i) distributed and (ii) hierarchical PMR configuration (see section 2.5.3) using the Word Count application on natural language and on random data as well as the genome sequencing application. In the distributed scenario (i) the map CUs are distributed across two machines, in the hierarchical scenario (ii) two resources are used each executing an independent MR run. The MapReduce run for combining and aggregating the output of the first round is executed on one of these machines. The performance of each application depends on the amount of generated intermediate and output data. Table 3.1 summarizes the characteristics of the used applications.

Application	**Input**	**Intermediate**	**Output**
GS/PMR	80 GB	71 GB	17 GB
Word Count (English)	16 GB	26 GB	20 MB
Word Count (random)	16 GB	30 GB	30 GB

TABLE 3.1: Data Volumes for different Applications

3.4.1 Word Count

For Word Count we compare a distributed and hierarchical PMR configuration with the performance of two Hadoop configurations: a single resource Hadoop configuration (half of the data is initially moved to that cluster) and a hierarchical Hadoop setup with two resources. We utilize two machines, Sierra and Hotel. The initial input data of 16 GB is equally distributed on these two machines. As mentioned, for the single resource Hadoop configuration half of the input data needs to be moved from Sierra to Hotel prior to running the actual MapReduce job. Unfortunately, the FutureGrid firewall rules prohibited the usage of a distributed Hadoop setup. For all configurations, we use 8 nodes.

Figure 3.3 shows the results. For natural language input, both Hadoop and PMR show a comparable performance. A major performance factor for Hadoop in the case of distributed data is the necessity to move parts of the data (half of the input data) to the central Hadoop cluster. The performance of PMR is determined by the runtime of the map and reduce phase, which are slightly longer than for

Hadoop mainly due to the resource heterogeneity and the resulting scheduling overhead: the slowest node determines the overall runtime of both the map and reduce phase.

Both the hierarchical Hadoop and PMR perform better than the distributed PMR and single resource Hadoop configuration. The performance is mainly influenced by the data that needs to be moved. In the distributed case half of the intermediate data needs to be moved to the other resource; in the hierarchical case half of the output data requires movement. Since the output data in the hierarchical case is a magnitude smaller than the intermediate data in the distributed case (cmp. table 3.1) – 20 MB in comparison to 30 GB – the performance in the hierarchical case is significant better.

For random data, the distributed PMR and single resource Hadoop perform better than the hierarchical PMR and Hadoop configuration. In this case the output data is about equal to the intermediate data (30 GB), i. e. the advantage of a reduced transfer volume does not exit. In this case the additional MapReduce run represents an overhead. In the Hadoop case, the moved data needs to get loaded into HDFS, which represents another overhead.

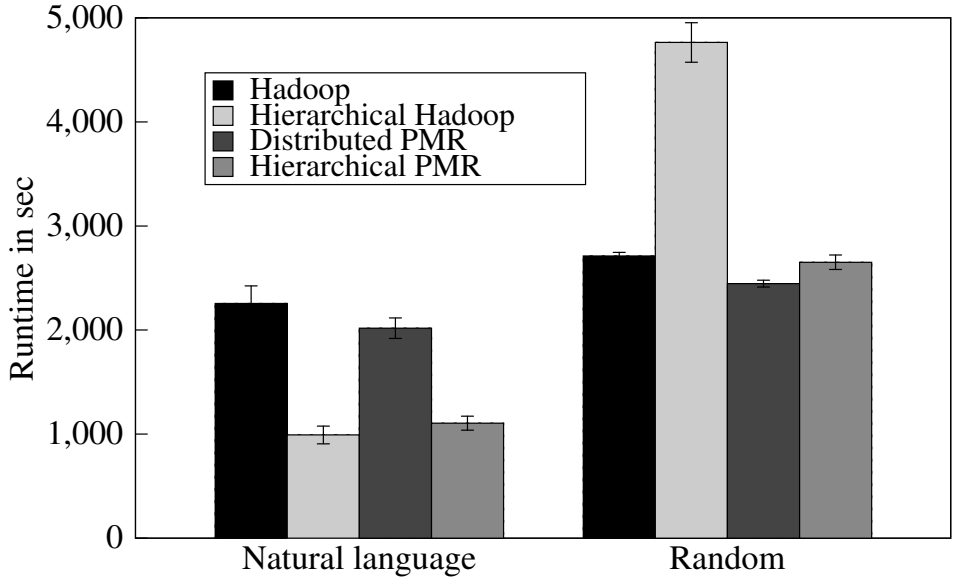

FIGURE 3.3: Word Count on 16 GB Data Using Hadoop, Hierarchical Hadoop, Distributed PMR and Hierarchical PMR

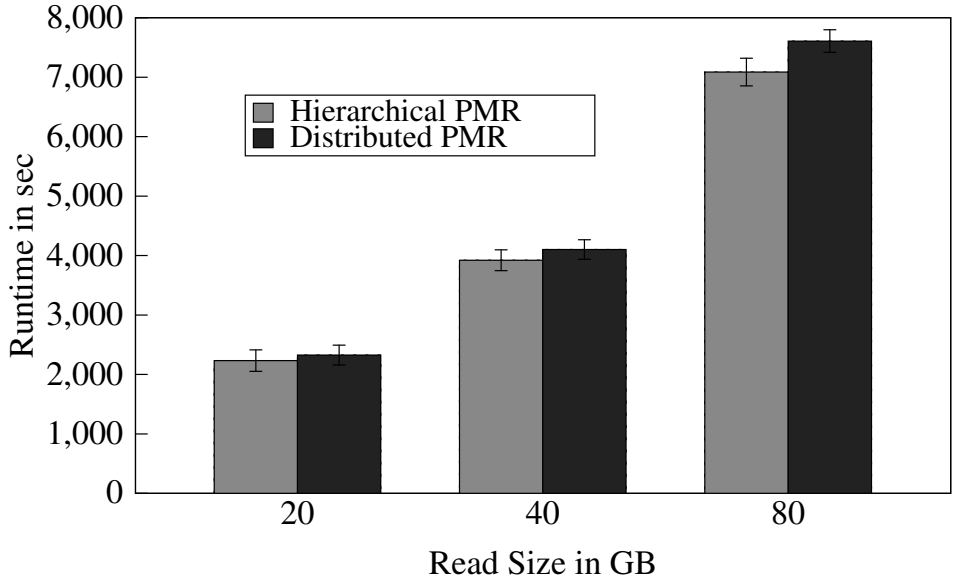

FIGURE 3.4: GS/PMR Using Hierarchical and Distributed PMR

3.4.2 Genome Sequencing

For the genome sequencing application, we utilize India and Hotel, a total of 32 nodes and different input data size between 20 and 80 GB. Figure 3.4 shows the results. In both scenarios the runtime increases with the input data size. For the distributed PMR, a significant part of the performance is determined by the movement of the intermediate data – 71 GB for the 80 GB problem set (see table 3.1). In the hierarchical PMR scenario, the main overhead arises from the additional MapReduce run. For GS/PMR the hierarchical configuration shows a slight advantage over the distributed setup since the amount of data that needs to be transferred is significant less: half of the output, i. e. 8.5 GB, respectively, of the intermediate data, i. e. 36 GB. However, a great amount of the time saving is absorbed by the overhead of the additional MapReduce run in the hierarchical case.

Running MapReduce on distributed data is not a trivial task – the overall performance is determined by many factors, e. g. the application's characteristics, current machine and network loads, etc. Different MapReduce configurations, such as the distributed and hierarchical configuration, can address certain application characteristics. For example, depending on the volume of the intermediate and output data, a distributed or hierarchical configuration may show a better performance. In applications with a smaller volume of output than intermediate data, such as GS and Word Count on natural languages, a hierarchical MapReduce is a good choice since it involves less data movement.

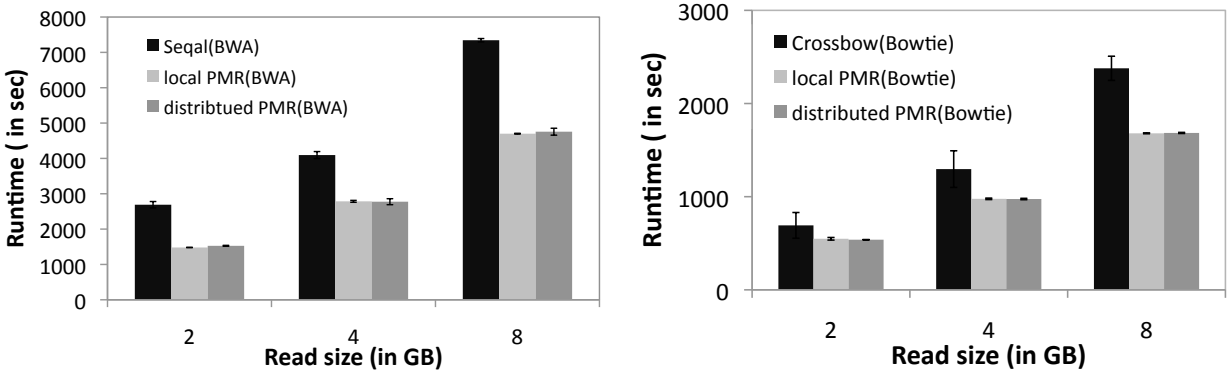

FIGURE 3.5: Comparison of runtimes for the map phase. The map phase of Seqal, local-PMR(BWA), distributed-PMR(BWA), local-PMR(Bowtie), distributed-PMR(Bowtie), and Crossbow(Bowtie) are compared. The aligner used for each case is indicated in a parenthesis. For this experiment, the number of nodes, N_{node} is 4, the number of Workers, N_W is 8, and the number of reads in each chunk is 292,763. For the distributed-PMR, two machines of FutureGrid, Sierra and Hotel were used, whereas Sierra was used for other cases [25].

PMR provides the flexibility to deploy MapReduce workloads in different configurations optimizing the performance with respect to the characteristics of different applications. Hadoop, in contrast, is very inflexible in supporting different kind of MapReduce configurations. In our case e. g. we were not able to run Hadoop across more than two machines on FutureGrid due to firewall issues.

3.4.3 Extensibility and Parallelism

The extensibility of PMR is demonstrated with two aligners – BWA and Bowtie. One of the important reasons why multiple aligners are needed is because of the difficulty of validation of an aligner used[19]. It is well studied that each aligner implements different strategies to deal with the requirement of computational loads, memory usage, and sensitivity associated with decision on algorithms and computational implementations of indexing, search, and match tasks.

Indeed, the decision of which aligner affects not only alignment results but also investigate downstream analysis that aim to study genome variation, transcriptome analysis, and DAN-protein interactions. Therefore, it is not an overstatement to emphasize the importance of supporting multiple tools as well as providing an effective means for implementing such tools within a reasonably short development period for infrastructure of NGS data. Fig. 3.5, evaluates the performance of read alignment in the map phase of both Hadoop and PMR based applications for Bowtie and BWA aligners. Hadoop implementations - Crossbow uses Bowtie aligner and Seqal uses BWA aligner. Custom python

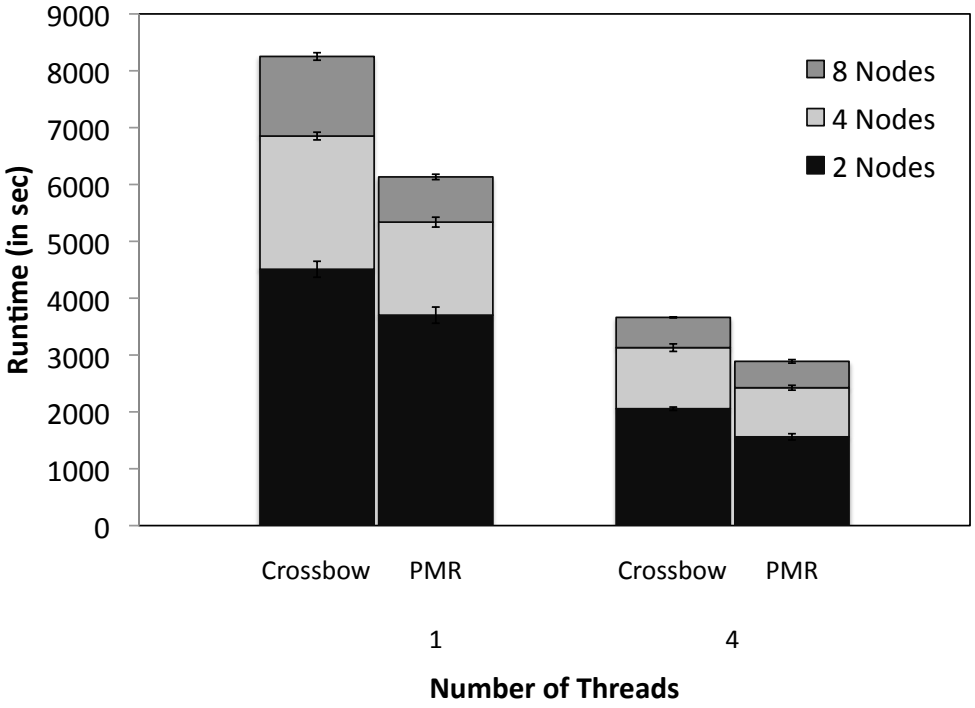

FIGURE 3.6: The map phase runtimes of PMR(Bowtie) and Crossbow(Bowtie) are compared, by varying number of threads for each map task. Number of workers/Node = 2 and input data size is 8 GB. The maximum number of cores assigned to a worker is 4, so we used 4 threads to achieve maximum fine-grain parallelism [25]

wrappers to Bowtie and BWA aligner are developed to execute alignment in the map phase of PMR. In the evaluation, both Hadoop based implementations face the problem of non-optimal configuration of Hadoop, i.e usage of shared file system for HDFS, where as both local and distributed PMR perform better than Hadoop map phase for both aligners. The PMR is extensible and can support multiple NGS analytic tools.

Extending PMR to support new NGS analytic tools involve development of simple map and reduce wrapper scripts to the tools. The wrapper scripts could be developed in any language. To some extent, Hadoop streaming supports this types of extensibility but still requires complexity of managing computational resources to maintain Hadoop cluster. PMR liberates the user from the complex task of maintaining and acquiring computational resources and executing map and reduce tasks on them.

PMR supports multiple levels of parallelisms – thread, task and multiple-cores, and enables the flexible configurations of codes. For example, BWA and Bowtie can be invoked to use varying number of threads (fine-grained parallelism). In Fig. 3.6, we showed how such options could affect the performance. Even though it is feasible for other tools such as Seqal or Crossbow to handle such

options, the PMR approach of separating the runtime environment (Pilot) from the code invocation in the map and reduce phases, provides the capability of utilizing the fine-grained parallelism along with the coarse grain parallelism provided by MapReduce. The fine grain parallelism provided by Pilot-Job framework is demonstrated in replica exchange implementation [21].

One of the advantages of PMR is it doesn't impose any restriction on number of compute nodes that can be assigned to a particular map or reduce task. This leads to a natural and native support for MPI-based NGS tools. For example, NovoalignMPI [1] is a message passing version of Novoalign, with claims of a more accurate aligner, allows a single alignment process to use multiple compute nodes. The MPI versions of Novoalign are more beneficial when large computing infrastructures are available. Hadoop doesn't provide flexibility to assign multiple compute nodes to a single compute task, thus leading to an impedance mismatch between Hadoop MR and MPI based NGS analytic tools [25].

Chapter 4

Workflow Management Using BigJob on XSEDE and LONI

In this chapter we discuss about how BigJob has been used to support Extended Collaborative Support Service (ECSS) project on XSEDE and simple workflows on LONI to calculate molecular distance of a f2 molecule. We also discuss the developments and testing involved as a part of project support.

4.1 Extended Collaborative Support Service (ECSS) on XSEDE

The ECSS of XSEDE is a means of providing support for advance user requirements that cannot and should not be supported via a regular ticketing system. Recently two ECSS projects were awarded by XSEDE management to support the high-throughput of high-performance (HTHP) molecular dynamics (MD) simulations; both of these ECSS projects are using SAGA-based Pilot-Jobs approach as the technology required to support the HTHP scenarios. More significantly, these projects were envisioned as three-way collaborations: between the application stakeholders, advanced/research software development team and the resource providers.

4.1.1 Developments and Testing

Job submission is another interesting issue on XSEDE. Lonestar and ranger use SGE, Kraken and Trestles use PBS job scheduling systems. While SAGA retains the ability to submit jobs through its globus [11] job adaptor, it is an un-necessary burden on users. Furthermore, when globus submitted jobs fail, they generate a very lengthy error report without much useful information. Both projects needed an immediate, clear and fail safe mechanism to submit jobs and this lead to the development of the *pbs-ssh* and *sge-ssh* plugins to support both the PBS and SGE scheduling systems. The plugins enable local/remote launch of BigJob agents using traditional PBS/SGE script over SAGA ssh job adaptors.

4.2 Workflow Execution on LONI

The workflow execution demonstrates the flexibility and scalability of SAGA BigJob by conducting experiments to execute computationally intensive ensembles in various configurations. The configuration involves BigJob size which is number of cores requested, the wall time which is the expected time to complete all the jobs associated with that BigJob and the number of machines used to run the jobs(scale out). The ensembles are computationally intensive, which require 16 cores to execute and are configured to run 20 monte carlo passes with given molecular distance of f2 molecule.

BigJob size(cores)	%Cluster resources in use	# of Generations	Average waiting time(min)	Average execution time(min)	Walltime requested(min)
128	68.5	1	2	36	40
128	90	1	39	36	40
64	68.5	2	1	79	80
32	90	4	1	156	160

TABLE 4.1: BigJob configurations used to execute 8 subjobs(16 cores each) on LONI Eric machine at different resource available situations.

Table 4.1 shows the trade-off between the number of cores requested,waiting time and the total execution time depending upon the system resource availablity. The user based on the cluster resources available can configure BigJob size and wall time of the Job. If the system is busy, it is hard to get large number of resources and the queue wait time increases, so keeping BigJob size low yield resources quickly, but the number of jobs per generation decreases and thus increases the number of generations. The system queue wait time is unpredictable and could be serious bottleneck for execution of jobs. BigJob proves to be advantegous in this kind of situation, and utlizes resources effectively yeilding less runtime to solution.

# of machines	%BigJob size /machine(cores)	Average waiting time(min)	Average execution time(min)
2	64	2	37
3	44	2	36
4	32	2	38

TABLE 4.2: BigJob configurations used to execute 8 ensembles(16 cores each) on LONI Eric, Poseidon, Oliver, Louie machines when 90% of cluster resources are in use.

Table 4.2 shows how BigJob takes advantage of scaling-out ensembles to multiple resources. If more number of cluster resources are in use, the queue waiting time of request with more number of

resources increases on a single machine. BigJob provides interoperability, flexibility to utilize resources on multiple machines of same/different infrastructure in a uniform manner.

Chapter 5
Conclusions and Future Work

Scientists in many science disciplines, where enormous amounts of data is generated, e. g. in the areas of fusion energy, bioinformatics, climate and astronomy, utilize distributed cyber-infrastructure to conduct experiments and improve their understanding about the scientific applications. Domain scientists face various challenges associated with processing of data at extreme scales on distributed cyber-infrastructures. MapReduce is an effective programming model for processing huge amounts of data. Hadoop is an open-source implementation of MapReduce programming model but is designed for shared-nothing environments and its performance is affected on a distributed file system. On DCI like FutureGrid, we were not able to run Hadoop on multiple clusters. Pilot-MapReduce provides a flexible runtime environment for MapReduce applications on general-purpose distributed infrastructures, such as XSEDE and FutureGrid.

Pilot-MapReduce is a novel Pilot-based MapReduce implementation which enables clean separation of resource management and MapReduce application. It brings the advantages of the Pilot abstraction to MapReduce, and enables utilization of federated and heterogeneous compute and data resources. In contrast to Hadoop, no previous cluster setup, which includes running several Hadoop/HDFS daemons, is required.The experiment results prove Pilot-MapReduce shows good performance on distributed cyberinfrastructures and can be a good alternative to Hadoop. PMR provides a extensible runtime environment, which allows the flexible usage of sorting in the shuffle, more fine-grained control of data localities and transfer, as well as support for different MapReduce topologies. Using these capabilities, applications with different characteristics, e. g. compute/IO and data aggregation ratios, can be efficiently supported.

The effectiveness of MapReduce topology depends on the application's work load aggregation. Distributed PMR performs better than hierarchical PMR for applications whose output data is

greater than or equal to input data. Similarly, hierarchical PMR performs better than distributed PMR in case of applications where the output data is less than input data.

Implemenation of Pilot abstractions, BigJob and BigData proved to be effective tools for developing PMR. The flexibility to provide affinities between compute/data units and resources, enabled optimization of runtime by efficient intermediate data transfers and effective placement of compute and data units. Pilot abstractions are proved to be an effective abstractions to scale out applications onto multiple cross-domain infrastructures. Since PMR, built on pilot abstractions, the scalability of PMR depends directly on the scalability of pilot abstractions.

Recent advances in high-throughput DNA sequencing technologies such as Next-Generation Sequencing (NGS) platforms resulted in unprecedented challenges in the areas of bioinformatics and computational biology. Dealing with unprecedented data and required data analytics and downstream analyses of such high-throughput deep sequencing techniques, MapReduce-based approaches were added to an arsenal of computational biologist and PMR provides a viable solution for scale-across and extensible NGS analytics. In fact, PMR not only supports scale-across, it provides some unique features, viz., support for distributed data analysis and multiple tools that can each exploit multiple levels of parallelism. PMR provides an extensible runtime environment with which minimally modified, yet standalone target tools are executed and the overall workflow can be dynamically optimized by exploiting multiple levels of parallelism. Furthermore, as indicated by results for BWA and Bowtie for alignment, PMR allows further extensions of existing implementation with other complementary tools or a flexible pipeline development.

Future work in this research may extend the capabilities of PMR and BigData to support use cases, such as data streaming, data caching as well as different data/compute scheduling heuristics. Further, explore scenarios and applications with dynamic data and execution. An obvious and trivial extension will be to implement Iterative MapReduce using PMR. A clear advantage will be to obviate the need to distinguish between static and dynamic data, for PMR will be able to treat both symmetrically. Our future goal also involves to develop an integrative pipeline service for RNA-Seq data, and the development presented in this thesis is indicative of preliminary progresses toward such a goal.

Bibliography

[1] http://www.novocraft.com/.

[2] Extreme science and engineering discovery environment (xsede). https://www.xsede.org/.

[3] Apache Hadoop. http://hadoop.apache.org/, 2012.

[4] A. Bateman and J. Quackenbush. Editorial -Bioinformatics for Next Generation Sequencing. *Bioinformatics*, 25:429, 2009.

[5] D. Bernholdt, S. Bharathi, and D. B. et al. The earth system grid. *Proc. of the IEEE*, 93(3):485–495, 2005.

[6] G. B. Berriman and S. L. Groom. How will astronomy archives survive the data tsunami? *Queue*, 9:21:20–21:27, October 2011.

[7] M. Cardosa, C. Wang, A. Nangia, A. Chandra, and J. Weissman. Exploring mapreduce efficiency with highly-distributed data. In *Proceedings of 2nd international workshop on MapReduce and its applications*, pages 27–34, New York, NY, USA, 2011. ACM.

[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

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

[10] FutureGrid: An Experimental, High-Performance Grid Test-bed. https://portal.futuregrid.org/, 2012.

[11] Globus. http://globus.org/.

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

[13] S. Jha, J. D. Blower, N. C. Hong, S. Dobson, D. S. Katz, A. Luckow, O. Rana, Y. Simmhan, and J. Weissman. 3dpas: Distributed dynamic data-intensive programming abstractions and systems. 2011.

[14] J. Kim, W. Huang, S. Maddineni, F. Aboul-ela, and S. Jha. Exploring the rna folding energy landscape using scalable distributed cyberinfrastructure. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 477–488, New York, NY, USA, 2010. ACM.

[15] J. Kim, S. Maddineni, and S. Jha. Characterizing deep sequencing analytics using bfast: towards a scalable distributed architecture for next-generation sequencing data. In *Proceedings of the second international workshop on Emerging computational methods for the life sciences*, ECMLS '11, pages 23–32, New York, NY, USA, 2011. ACM.

[16] S.-H. Ko, N. Kim, J. Kim, A. Thota, and S. Jha. Efficient runtime environment for coupled multi-physics simulations: Dynamic resource allocation and load-balancing. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 349–358, Washington, DC, USA, 2010. IEEE Computer Society.

[17] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg. Searching for SNPs with cloud computing. *Genome Biol.*, 19(11):R134, 2009.

[18] H. Li and R. Durbin. Fast and accurate long-read alignment with burrows wheeler transform. *Bioinformatics*, 26:589–595, March 2010.

[19] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11(5):473–483, 2010.

[20] A. Luckow and S. Jha. Abstractions for loosely-coupled and ensemble-based simulations on azure. *Cloud Computing Technology and Science, IEEE International Conference on*, pages 550–556, 2010.

[21] A. Luckow, S. Jha, J. Kim, A. Merzky, and B. Schnor. Adaptive Replica-Exchange Simulations. *Royal Society Philosophical Transactions A*, pages 2595–2606, June 2009.

[22] 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*, pages 135–144, 2010.

[23] A. Luckow, M. Santcroos, O. Weider, A. Merzky, S. Maddineni, and S. Jha. P*: A model of pilot-abstractions. In *Proceedings of The International ACM Symposium on High-Performance Parallel and Distributed Computing*, 2012.

[24] A. Luckow, M. Santcroos, O. Weider, A. Merzky, P. Mantha, and S. Jha. P*: A model of pilot-abstractions. 2012.

[25] P. Mantha, N. Kim, J. Kim, A. Luckow, and S. Jha. Understanding MapReduce-based Next-Generation Sequencing Alignment on Distributed Cyberinfrastructure, 2012. Accepted for ECMLS workshop 2012 (HPDC12).

[26] P. Mantha, A. Luckow, and S. Jha. Pilot-MapReduce: An Extensible and Flexible MapReduce Implementation for Distributed Data, 2012. Accepted for MapReduce workshop 2012 (HPDC12).

[27] J. D. McPherson. Next-Generation Gap. *Nature Methods*, 6:s2–s5, 2009.

[28] A. Merzky. SAGA API Extension: Advert API. OGF Document Series 177, http://www.gridforum.org/documents/GFD.177.pdf, 2011.

[29] M. L. Metzker. Sequencing technologies - the next generation. *Nat. Rev. Genet.*, 11(1):31–46, 2010.

[30] Picard. http://picard.sourceforge.net, 2012.

[31] S. Pepke, B. Wold, and A. Mortazavi. Computation for ChIP-seq and RNA-seq studies. *Nature Methods*, 6:S22–S32, 2009.

[32] L. Pireddu, S. Leo, and G. Zanetti. Mapreducing a genomic sequencing workflow. In *Proceedings of the second international workshop on MapReduce and its applications*, MapReduce '11, pages 67–74, New York, NY, USA, 2011. ACM.

[33] L. Pireddu, S. Leo, and G. Zanetti. SEAL: a distributed short read mapping and duplicate removal tool. *Bioinformatics (Oxford, England)*, 27(15):2159–2160, 2011.

[34] Redis. http://redis.io/, 2011.

[35] M. Romanus, P. Mantha, M. McKenzie, T. Bishop, A. Merzky, Y. E. Khamra, and S. Jha. The Anatomy of Successful ECSS Projects: Lessons of Supporting High-Throughput High-Performance Ensembles on XSEDE, 2012. Submitted for XSEDE 2012 (HPDC12).

[36] SAGA BigJob. https://github.com/saga-project/BigJob/wiki, 2012.

[37] The 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467:1061–1073, 2010.

[38] A. Thota, A. Luckow, and S. Jha. Efficient large-scale replica-exchange simulations on production infrastructure. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1949):3318–3335, 2011.

[39] Z. Wang, M. Gerstein, and M. Snyder. RNA-seq : a revolutionary tool for transcriptomics. *Nat. Rev. Genet.*, 10(1):57–63, 2009.

[40] WebHDFS REST API. http://hadoop.apache.org/common/docs/r1.0.0/webhdfs.html, 2012.

[41] OSG: Open Science Grid. https://www.opensciencegrid.org/, 2012.

[42] ZeroMQ. http://www.zeromq.org/, 2011.

APPENDIX (Permission to Reprint Publications)

Pilot-MapReduce: An Extensible and Flexible MapReduce Implementation for Distributed Data

Pradeep Kumar Mantha
Center for Computation and Technology
Louisiana State University
216 Johnston
Baton Rouge, LA
pmanth2@cct.lsu.edu

Andre Luckow
Center for Computation and Technology
Louisiana State University
216 Johnston
Baton Rouge, LA
aluckow@cct.lsu.edu

Shantenu Jha
Center for Autonomic Computing
Rutgers University
94 Brett Road
Piscataway, NJ
shantenu.jha@rutgers.edu

ABSTRACT

The volume and complexity of data that must be analyzed in scientific applications is increasing exponentially. Often, this data is distributed, thus efficient processing of large distributed datasets is required, whilst ideally not introducing fundamentally new programming models or methods. For example, extending MapReduce – a proven and effective programming model for processing large datasets – to work more effectively on distributed data and on different infrastructure is desirable. MapReduce on distributed data requires effective distributed coordination of computation (map and reduce) and data, as well as distributed data management (in particular the transfer of intermediate data). We posit that this can be achieved with an effective and efficient runtime environment and without refactoring MapReduce itself. To address these requirements, we design and implement Pilot-MapReduce (PMR) – a flexible, infrastructure-independent runtime environment for MapReduce. PMR is based on Pilot abstractions for both compute (Pilot-Jobs) and data (Pilot-Data): it utilizes Pilot-Jobs to couple the map phase computation to the nearby source data, and Pilot-Data to move intermediate data using parallel data transfers to the reduce phase. We analyze the effectiveness of PMR on applications with different characteristics (e. g. different volumes of intermediate and output data). We investigate the performance of PMR with distributed data using a Word Count and a genome sequencing application over different MapReduce configurations. Our experimental evaluations show that the Pilot abstractions are powerful abstractions for distributed data: PMR can lower the execution time on distributed clusters and that it provides the desired flexibility in the deployment and configuration of MapReduce runs to address specific application characteristics.

MapReduce'12, June 18-19, 2012, Delft, The Netherlands.

Categories and Subject Descriptors

D.1.3 [**Software**]: Concurrent Programming-Distributed programming/parallel programming

General Terms

Design, Experimentation, Performance

Keywords

MapReduce, Distributed Computing, Pilot Job and Data, Simple API for Grid Applications (SAGA), Genome Sequence Alignment, BWA

1. INTRODUCTION

There are various challenges associated with processing of data at extreme scales: which has become a critical factor in many science disciplines, e. g. in the areas of fusion energy (ITER), bioinformatics (metagenomics), climate (Earth System Grid), and astronomy (LSST) [13]. The volumes of data produced by these scientific applications is increasing rapidly, driven by advanced technologies (e. g. increasing compute capacity and higher resolution sensors) and decreasing costs for computation, data acquisition and storage [11]. The number of applications that either currently utilize, or need to utilize large volumes of potentially distributed data is immense. The challenges faced by these applications are interoperability, efficiently managing compute tasks, and moving data to the scheduled compute location.

Processing large volumes of data is a challenging task. MapReduce is an effective programming model for addressing this challenge. MapReduce [5] as originally developed by Google aims to address the big data problem by providing an easy-to-use abstraction for parallel data processing. The most prominent framework for doing MapReduce computations is Apache Hadoop [1]. However, there are limitations to the current MR implementations: (i) They lack a modular architecture, (ii) are tied to specific infrastructure, e. g. Hadoop relies on the Hadoop File System (HDFS), and (iii) do not provide efficient support for dynamic and processing distributed data, e. g. Hadoop is designed for cluster/local environment, but not for a high degree of distribution.

Pilot abstractions enable the clean separation of resource management concerns and application/frameworks. In particular, Pilot-Jobs have been notable in their ability to manage large numbers of compute units across multiple high performance clusters, providing decoupling application-level

pradeep kumar Mantha <pradeepm66@gmail.com>

Permission to reuse the MapReduce2012 publication.

Simon Delamare <simon.delamare@inria.fr> Mon, Jul 9, 2012 at 3:18 AM
Reply-To: simon.delamare@ens-lyon.fr
To: pradeep kumar Mantha <pmanth2@cct.lsu.edu>
Cc: Shantenu Jha <sjha@cct.lsu.edu>

Le mercredi 04 juillet 2012 à 01:08 -0600, pradeep kumar Mantha a
écrit :
> Hi!
>
> I am Pradeep Mantha the primary author of publication "Pradeep Mantha,
> Andre Luckow and Shantenu Jha. Pilot-MapReduce: An Extensible and
> Flexible MapReduce Implementation for Distributed Data" submitted to
> MapReduce 2012 workshop.
>
> My master's thesis is completely based on the above publication, and
> re-used most of the components from paper in my thesis document. As a
> part of my thesis authorization, I need permission from the
> publisher's to reuse the publication. The thesis, once authorized will
> be posted on a electronic digital library, which is publicly
> accessible.
>
> But my grad school is concerned about the part of copyright statement
> of publishers - "To copy otherwise, to republish, to post on servers
> or to redistribute to lists, requires prior specific permission and/or
> a fee".
>
> Could you suggest a way or give permission to reuse the publication
> for my thesis.
>
> thanks
> pradeep

Hello Pradeep,

No problem from our side if you reuse your publication in your thesis.

Regards,
Simon

Vita

Pradeep Kumar Mantha is a computer science graduate student in the Louisiana State University. His interests lie in the high-performance grid, distributed and data intensive computing, Next Generation Sequencing analytics fields. He previously received a bachelor's degree from Jawaharlal Nehru and Technological University, Hyderabad, India.

Publications

1. Pradeep Mantha, Andre Luckow, and Shantenu Jha. Pilot- MapReduce: An extensible and flexible MapReduce implementation for distributed data, 2012. Accepted for MapReduce workshop 2012.

2. Pradeep Mantha, Nayong Kim, Joohyun Kim, Andre Luckow, Shantenu Jha, Understanding MapReduce-based Next-Generation Sequencing Alignment on Distributed Cyber-infrastructure. Accepted for ECMLS workshop 2012.

3. Andre Luckow, Mark Santcroos, Ole Weidner, Andre Merzky, Pradeep Mantha, Shantenu Jha. P*: A Model of Pilot-Abstractions - Under Review submitted for SC 2012.

4. Melissa Romanus, Pradeep Mantha, Matt McKenzie, Tom Bishop, Andre Merzky, Yaakoub El Khamra, Shantenu Jha. The Anatomy of Successful ECSS Projects: Lessons of Supporting High-Throughput High-Performance Ensembles on XSEDE - Under Review submitted for XSEDE 2012.

5. Andre Luckow, Andre Merzky, Pradeep Mantha, Melissa Romanus, Ole Weidner, Yaakoub El Khamra, Shantenu Jha. Introduction to BigJob – A SAGA-based Interoperable, Extensible and Scalable Pilot-Job for XSEDE. Tutorial Accepted for XSEDE 2012

Other Achievements

Invited for Open Science Grid 2012 summer user school and XSEDE 2012 conferences.