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.
Copyright 2012 ACM 978-1-4503-1343-8/12/06 ...$10.00.

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 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 [15] aims to address this issue by providing a unified API for managing both compute and data pilots. In this paper, we present *BigData (BD)*, an extension of the BigJob framework (BJ) [20] 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 paper, 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. We show how Pilot abstractions are used for managing the map and reduce tasks and intermediate shuffle data between them. In addition, we show 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 paper 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.

This paper is organized as follows: Section 2 presents related works. Section 3 gives an overview of Pilot abstractions and the BigData framework. In section 4 we discuss the design and implementation of the Pilot-MapReduce framework. Section 5 gives experiment setup and result analysis. The conclusion and future work are given in Section 6.

2. RELATED WORK

The MapReduce programming model [5] and the distributed file system (Google File System (GFS) [9]) were originally pioneered by Google. Apache Hadoop [1] is an open source implementation of MapReduce. Hadoop also includes an implementation of a distributed file system – the Hadoop File System (HDFS) [3]. In addition the Hadoop ecosystems includes several other projects, such as HBase (a system also inspired by Google's BigTable), Hive, Pig and Zookeeper. The main limitation of Hadoop MapReduce is that it forces applications into a very rigid model. Hadoop e. g. is well suited for running a single MapReduce application, but very limited in terms of extensibility, e. g. it cannot efficiently run an ensemble of MapReduce simulations or support a pipeline of multiple iterative MapReduce tasks. Also, the deployment of Hadoop on HPC resources remains a challenge: the deployment in user-space is complex and error-prone. Also, HPC resources often have shared distributed file system and only a small amount of local storage, which leads to sub-optimal Hadoop performance. Running Hadoop on multiple resources is in principal possible, but firewall regulations often prohibit such deployment on many infrastructures in practice.

Sector/Sphere [10] is a parallel data processing framework consisting of a distributed file system (Sector) and a data processing engine (Sphere). In contrast to Hadoop, which operates on file chunks, Sphere can execute arbitrary user-defined functions on a stream of data.

Twister [6] is a MapReduce addressing particularly the requirement for supporting iterative MapReduce jobs. Twister allows the flexible composition of applications by specifying map and reduce tasks and the data flow between these tasks. Dryad [12] provides distributed execution of coarse-grain data-parallel applications. The entire execution is represented in the form of data flow graphs, where the vertices represent the computational tasks that can be paralleled on a set of computational resources and the links between vertices tell Dryad what other vertices need to complete before a particular vertex can start.

Several proposal for deploying MapReduce and Hadoop on distributed data and resources exist: Weissman et al. [4] explore different Hadoop configurations to accommodate different distributed resource configurations. Qiu et al. [16] also propose a hierarchical MapReduce configuration that implements the Map-Reduce-Global Reduce pattern on top of distributed resources.

SAGA-MapReduce [21] is a SAGA-based MapReduce implementation that utilizes the SAGA-API for accessing system-level features, such as resource & file management and coordination. The SAGA-based approach enabled the decoupling of infrastructure and application concerns enabling the support of a wide-set of distributed infrastructure (e. g, grids and clouds). The utilization of Pilot abstractions has several advantages compared to the SAGA-only approach: (i) compute and data pilots allow an efficient decoupling of resource allocation and usage, i. e. the MapReduce master can efficiently schedule compute units containing mapper and reduce tasks; (ii) the co-location of data and compute units can be descriptively defined, and are automatically handled by Pilot framework; this enables the applications to easily trade-off data transfers and available compute capacities.

Pilot-MapReduce utilizes Pilot abstractions for de-coupling the MapReduce runtime, application-level scheduling and resource management in order to provide a high degree of flexibility and extensibility.

3. PILOT ABSTRACTIONS

The P* model [15] aims to provide a unified model for describing and analyzing Pilot-Job implementations. Figure 1 shows the elements of the P* model. The P* model defines common elements of both compute and data pilot implementations. Pilot abstractions for both compute and data are the foundation of the Pilot-MapReduce. In this section we give an overview of the Pilot-API, which exposes the elements of the P* model to the applications as well as the implementation of the BigJob and BigData framework.

The Pilot-API is an *interoperable* and *extensible* API which exposes the core functionalities of a Pilot framework via a unified interface providing a common API that can be used across multiple distinct production cyber infrastructures. The API provides five core classes: the `PilotComputeService` for the management of Pilot-Jobs, `PilotDataService` for the management of Pilot-Data and the `ComputeDataSer-`

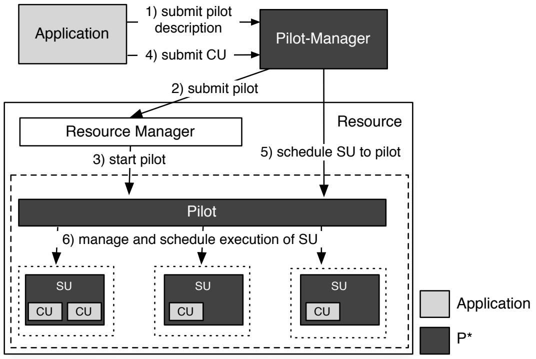

Figure 1: P* Model: Elements and Interactions: The pilot manager (PM) is the central entity of a Pilot framework. It has two functions: it manages 1) Pilots (step 1-3) and 2) the execution of CUs.

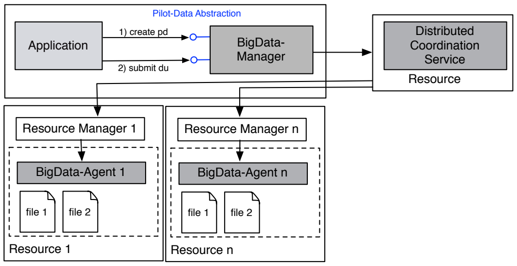

Figure 2: BigData Architecture and Interactions

`vice` for the management of `ComputeUnits` (CUs) and `DataUnits` (DUs). A CU represents a primary self-containing piece of work, while a DU represents a logical set for data [15].

3.1 BigJob: A Pilot-Compute Implementation

The abstraction of a *Pilot-Job* (PJ) generalizes the reoccurring concept of utilizing a placeholder job as a container for a set of compute tasks; instances of that placeholder job are commonly referred to as Pilot-Jobs or pilots. The PJ provides application- (user-) level control and management of the set of allocated resources.

BigJob (BJ) is a SAGA-based PJ framework that implements the Pilot-API. BJ has been designed to be general-purpose and extensible. While BJ has been originally built for HPC infrastructures, such as XSEDE and FutureGrid, it is generally also usable in other environments, such as OSG. This extensibility mainly arises from the usage of SAGA as a common API for accessing distributed resources.

3.2 BigData: A Pilot-Data Implementation

Analogous to Pilot-Jobs, *Pilot-Data* (PD) abstraction provides late-binding capabilities for data by separating the storage allocation and application-level Data Units [15]. 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.

BigData (BD) is an implementation of the Pilot-Data abstraction. BigData is designed as an extension of BigJob [20] – a SAGA-based Pilot-Job implementation. Figure 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 [17] 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 [22] and Globus Online [7] is provided.

3.3 Scheduling 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 Units.

4. 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.

4.1 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 pi-

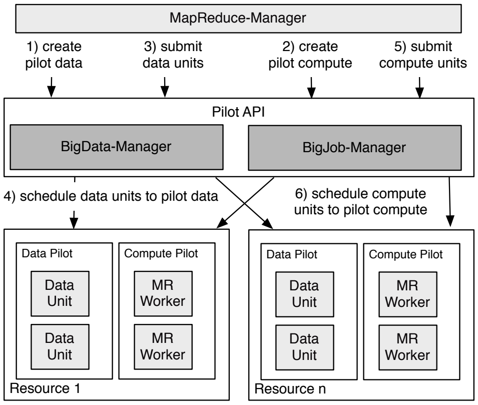

Figure 3: Pilot-based MapReduce: Each pilot (both compute and data pilot) can be associated with an affinity label. The BigData and BigJob Manager will ensure that CUs and DUs are placed with respect to these requirements.

lot abstractions enable the easy acquisition of both compute and storage resources.

Figure 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 Units.
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.

4.2 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.

4.3 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 [2]. Meta-genomic workflows need to process and analyze data

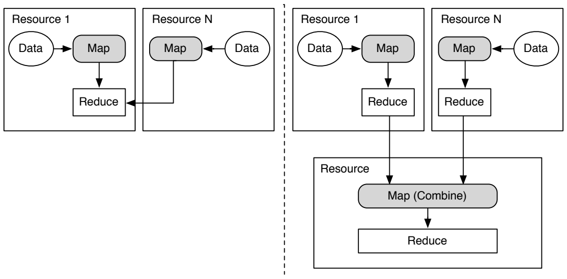

Figure 4: Pilot-MapReduce Deployment Scenarios: In the distributed scenario (left), the mapping tasks are run close to the data; reduced tasks are then run on a single resource. In the hierarchical scenario (right) two full MapReduce runs are conducted.

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 [4]: (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 preprocessing data and a central cluster for aggregating the different de-central data sources. The volume of the preprocessed data is generally lower and thus, can be easily moved to another processing resource.

Ref [4] 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 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 Units. 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).

5. EXPERIMENTS AND RESULTS

In this section 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 (FG) [8]. We run the experiment on the following FG 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 begining 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.

5.1 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 [4].

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.

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.

Genome Sequencing (GS): High-throughput genome sequencing as provided by Next Generation Sequencing (NGS) platforms is 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. We utilize MapReduce to model an important part of the sequencing workflow: the read alignment and the duplicate removal. We use two implementations: the Hadoop-based SEQAL [19] application and a PMR-based implementation GS/PMR. Both applications implement the read alignment in the mapping phase of the application using BWA aligner [14]. In the SEQAL case the duplicate removal in the reduce phase is implemented using Picard's `rmdup` [18]. The GS/PMR reducer removes duplicate reads based on the key fields-chromosome, position and strand of the mapper output.

5.2 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 in-turn comprises of sorting and the intermediate data transfer), and finally running the reduce CUs. Figure 5 shows the results. The runtime of Hadoop MapReduce includes the time to load input source data into HDFS and MapReduce runtime.

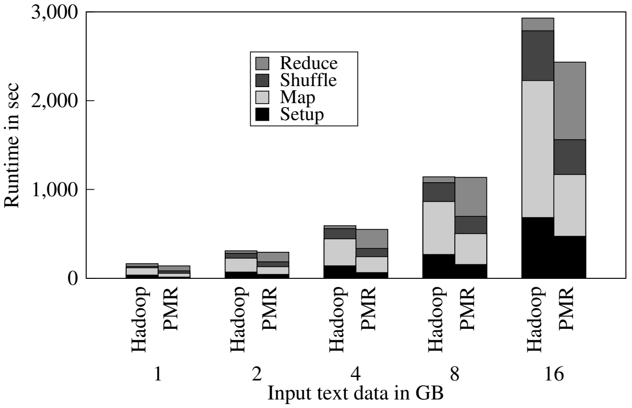

Figure 5: PMR vs. Hadoop (Word Count): The performance of Hadoop and PMR is comparable. The runtime increases with the input data size. Hadoop tasks have a notable higher startup time.

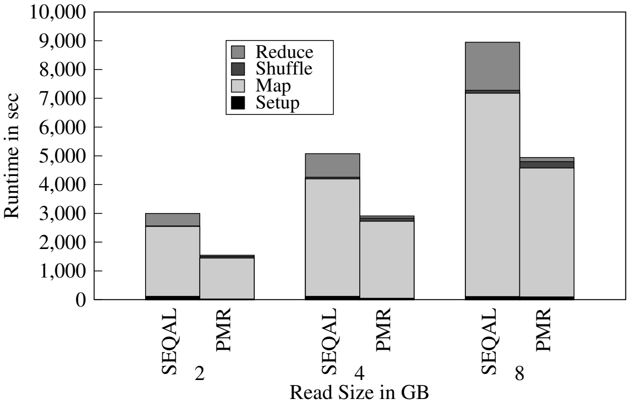

Figure 6: SEQAL and GS/PMR: GS/PMR provides a marginal better performance that SEQAL. The overhead of SEQAL is mainly attributed to the usage of shared file system for HDFS.

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 – Word Count needs to read all input files and generates intermediate data with the size of about 200 % of the input data. Hadoop shows the worst shuffle performance. A reason is that we were unfortunately not able to run HDFS in an optimal configuration due to a lack of local storage on the FG machine. Thus, HDFS was configured to utilize a shared file system.

5.3 Characterizing Genome Sequencing

In this section, we compare and contrast GS/PMR and SEQAL. For both applications, we utilize the same 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`); data was converted to meet the SEQAL requirements. For GS/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. The chunk size for both SEQAL and PMR are equal. For both GS/PMR and SEQAL, a total of 4 nodes on FG 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.

Figure 6 shows the comparative results of both SEQAL and GS/PMR applications. The time required to copy and extract the reference genome to HDFS is included in the SEQAL set-up time. In comparison to Word Count GS applications are more compute intensive, i. e. the ratio between computation in the map phase and the size of the input data is significantly larger. Furthermore, SEQAL has a larger 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` [18] for duplicate removal, which has a significant longer runtime than the duplicate removal process in the reduce phase of GS/PMR. A reason for the slower runtime of SEQAL in the map phase is the non-optimal Hadoop configuration: As described, the local disks available on FG are too small; 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.

5.4 Characterizing Distributed and Hierarchical MapReduce

In this section, we evaluate the performance and scalability of the (i) distributed and (ii) hierarchical PMR configuration using the Word Count application on natural language and on random data as well as the genome sequencing application. In the distributed PMR scenario, the CUs are distributed across two machines; in the hierarchical PMR 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 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 1: Data Volumes for different Applications

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 and a hierarchical Hadoop setup with two resources. We utilize two machines, Sierra and Hotel. For all configurations, we use 8 nodes. The initial input data of 16 GB is equally distributed on these two machines. 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 FG firewall rules prohibited the usage of a distributed Hadoop setup.

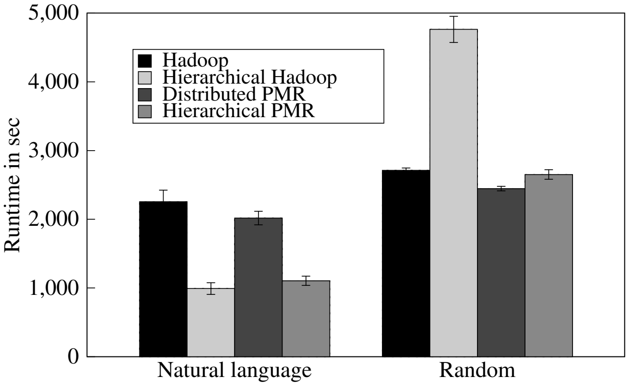

Figure 7: Word Count on 16 GB Data Using Hadoop, Hierarchical Hadoop, Distributed PMR and Hierarchical PMR

Figure 7 shows the results. For natural language input, both Hadoop and PMR show comparable performance. A major determinant of performance 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 hierarchical Hadoop and PMR perform better than the distributed PMR and single resource Hadoop configuration. The performance is mainly influenced by data movement costs. In the distributed PMR scenario, 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 1) – 20 MB in cmp. to 30 GB – the performance in the hierarchical case is significantly better.

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

Genome Sequencing

For the genome sequencing application, we investigate hierarchical and distributed PMR scenarios utilizing a total of 32 nodes on India and Hotel. We vary the input data size between 20 and 80 GB. Figure 8 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 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 saved in the transfer is offset by the overhead of the additional MapReduce run.

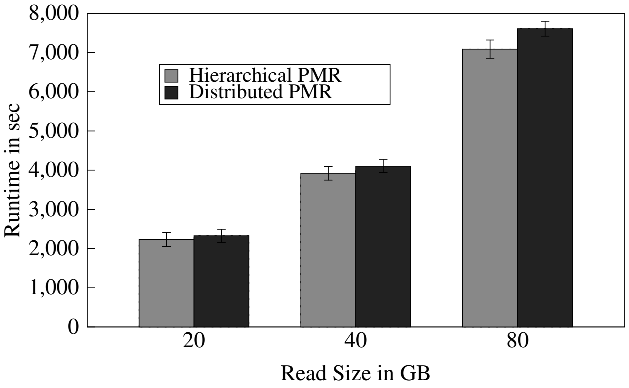

Figure 8: Time to completion of GS/PMR for 20, 40 and 80GB using Hierarchical and Distributed PMR. These measurements were performed utilizing a total of 32 nodes on India and Hotel.

In summary, optimizing MapReduce for distributed data is not a trivial task: Depending on the volumes of the intermediate and output data, a distributed or hierarchical configuration may have better performance. In applications with a less output volume compared to intermediate data, such as GS and Word Count on natural languages, a hierarchical MapReduce is a good choice since it involves less data movement. 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, due to deployment challenges (e. g. we were not able to run Hadoop across more than two machines on FG due to firewall issues) as well as runtime limitations.

6. DISCUSSION AND FUTURE WORK

Pilot-MapReduce provides a flexible runtime environment for MapReduce applications on general-purpose distributed infrastructures, such as XSEDE and FutureGrid. 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. Pilot-MapReduce 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 Pilot abstraction, and specific the BigJob and BigData implementation have proven to be a powerful tool for developing PMR. Using finer-grain affinity specifications for compute/data units and resources, the runtime is able to optimize compute and data placement as well as transfers. These capabilities are essential for PMR, especially when dealing with large amounts of distributed data; in order to achieve an optimal performance in this case the application must be able to reason and trade-off properties, such as data/compute localities and data transfers to achieve an optimal performance.

Moving forward, we will 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, we will 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.

References

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

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

[3] D. Borthakur. The hadoop distributed file system: Architecture and design. Technical report, Apache Software Foundation, 2007.

[4] 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.

[5] 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.

[6] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM.

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

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

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37:29–43, 2003.

[10] Y. Gu and R. L. Grossman. Sector and sphere: the design and implementation of a high-performance data cloud. *Philosophical Transactions of the Royal Society - Series A: Mathematical, Physical and Engineering Sciences*, 367(1897):2429–2445, 2009.

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

[12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41:59–72, March 2007.

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

[15] A. Luckow, M. Santcroos, O. Weider, A. Merzky, S. Maddineni, and S. Jha. Towards a common model for pilot-jobs. In *Proceedings of The International ACM Symposium on High-Performance Parallel and Distributed Computing*, 2012.

[16] Y. Luo, Z. Guo, Y. Sun, B. Plale, J. Qiu, and W. W. Li. A hierarchical framework for cross-domain mapreduce execution. In *Proceedings of the 2nd international workshop on emerging computational methods for the life sciences*, pages 15–22, NY, NY, USA, 2011. ACM.

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

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

[19] 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.

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

[21] S. Sehgal, M. Erdelyi, A. Merzky, and S. Jha. Understanding application-level interoperability: Scaling-out mapreduce over high-performance grids and clouds. *Future Gener. Comput. Syst.*, 27:590–599, May 2011.

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

Acknowledgments

This work is funded by NSF CHE-1125332 (Cyber-enabled Discovery and Innovation), HPCOPS NSF-OCI 0710874 award, NSF-ExTENCI (OCI-1007115) and NIH Grant Number P20RR016456 from the NIH National Center For Research Resources. Important funding for SAGA has been provided by the UK EPSRC grant number GR/D0766171/1 (via OMII-UK) and the Cybertools project (PI Jha) NS-F/LEQSF (2007-10)-CyberRII-01. SJ acknowledges the e-Science Institute, Edinburgh for supporting the research themes: Distributed Programming Abstractions & 3DPAS. SJ acknowledges useful related discussions with Jon Weissman (Minnesota) and Dan Katz (Chicago). We thank J Kim (CCT) for assistance with genome sequencing application. This work has been made possible thanks to resources provided by TeraGrid TRAC award TG-MCB090174 (Jha). This document was developed with support from the US NSF under Grant No. 0910812 to Indiana University for FutureGrid.