VIRTUAL CLUSTER MANAGEMENT FOR ANALYSIS OF GEOGRAPHICALLY DISTRIBUTED AND IMMOVABLE DATA

Yuan Luo

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements for the degree

Doctor of Philosophy

in the School of Informatics and Computing,

Indiana University

August 2015

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Beth Plale, Ph.D.
(Chairperson)

Geoffrey Fox, Ph.D.

Judy Qiu, Ph.D.

Yuqing Wu, Ph.D.

Philip Papadopoulos, Ph.D.

August 7th, 2015

In loving memory of my grandfather.

To my parents, my cousin, and all family members.

Acknowledgements

It has been a great pleasure working with the faculty, staff, and students at Indiana University, during my seven-year PhD study. I would like to express my sincere gratitude to my advisor, Professor Beth Plale. This dissertation would never have been possible without her guidance, encouragement, challenges, and support throughout my research at Indiana University. I am grateful for the many opportunities that Beth has provided for me to research, collaborate, publish, teach, and travel independently.

I thank my other research committee members, Professor Geoffrey Fox, Professor Judy Qiu, Professor Yuqing Wu, and Dr. Philip Papadopoulos, for their invaluable discussion, ideas, and feedback.

I also would like to thank Professor Xiaohui Wei, Dr. Peter Arzberger, and Dr. Wilfred Li for their mentorship before I joined Indiana University. I would never have started my PhD without their influence.

I thank all current and past members in the Data to Insight Center at Indiana University for making the lab an enjoyable and stimulating place to work. I would like to name these individuals in particular: Peng Chen, Yiming Sun, Dr. You-Wei Cheah, Zong Peng, Jiaan Zeng, Guangchen Ruan, Quan Zhou, Dr. Scott Jensen, Dr. Devarshi Ghoshal, Felix Terkhorn, Jenny Olmes-Stevens, Jodi Stern, and Robert Ping. I also thank Tak-Lon Wu and Zhenhua Guo from Digital Science Center for multiple discussions.

I want to thank all mentors, colleagues and friends in the PRAGMA community for research discussions, project collaborations, and hospitalities: Shava Smallen, Dr. Kevin Kejun Dong, Professor Jose Fortes, Nadya Williams, Cindy Zheng, Dr. Yoshio tanaka, Professor Osamu Tatebe, Teri Simas, and many more.

I thank all faculty and staff at School of Informatics and Computing for the excellent teaching and service. I am grateful to the school for providing me academic fellowship at the time of enrollment.

I would like to thank Jenny Olmes-Stevens, Marisha Zimmerman, and Yi Qiao for all their help in thesis proofreading.

Finally, I thank my family for all of the love, support, encouragement, and prayers throughout my journey.

Yuan Luo

VIRTUAL CLUSTER MANAGEMENT FOR ANALYSIS OF GEOGRAPHICALLY DISTRIBUTED AND IMMOVABLE DATA

Scenarios exist in the era of Big Data where computational analysis needs to utilize widely distributed and remote compute clusters, especially when the data sources are sensitive or extremely large, and thus unable to move. A large dataset in Malaysia could be ecologically sensitive, for instance, and unable to be moved outside the country boundaries. Controlling an analysis experiment in this virtual cluster setting can be difficult on multiple levels: with setup and control, with managing behavior of the virtual cluster, and with interoperability issues across the compute clusters. Further, datasets can be distributed among clusters, or even across data centers, so that it becomes critical to utilize data locality information to optimize the performance of data-intensive jobs. Finally, datasets are increasingly sensitive and tied to certain administrative boundaries, though once the data has been processed, the aggregated or statistical result can be shared across the boundaries.

This dissertation addresses management and control of a widely distributed virtual cluster having sensitive or otherwise immovable data sets through a controller. The Virtual Cluster Controller (VCC) gives control back to the researcher. It creates virtual clusters across multiple cloud platforms. In recognition of sensitive data, it can establish a single network overlay over widely distributed clusters. We define a novel class of data, notably immovable data that we call “pinned data”, where the data is treated as a first-class citizen

instead of being moved to where needed. We draw from our earlier work with a hierarchical data processing model, Hierarchical MapReduce (HMR), to process geographically distributed data, some of which are pinned data. The applications implemented in HMR use extended MapReduce model where computations are expressed as three functions: Map, Reduce, and GlobalReduce. Further, by facilitating information sharing among resources, applications, and data, the overall performance is improved. Experimental results show that the overhead of VCC is minimum. The HMR outperforms traditional MapReduce model while processing a particular class of applications. The evaluations also show that information sharing between resources and application through the VCC shortens the hierarchical data processing time, as well satisfying the constraints on the pinned data.

Beth Plale, Ph.D.
(Chairperson)

Geoffrey Fox, Ph.D.

Judy Qiu, Ph.D.

Yuqing Wu, Ph.D.

Philip Papadopoulos, Ph.D.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As new sources of research data become available worldwide, either through their creation or through enhanced mechanisms for sharing, the scenario in which a researcher wishes to carry out data analysis on multiple data sets that are widely geographically located will become commonplace, and because of the sensitive nature of the data, it may have been repatriated for instance, the computation must occur on a cluster co-located with the data. Further, if the data is sensitive, additional measures of protection may be needed while the data travels in the network, requiring establishment of something like an overlay network. In a scenario like this, the researcher is faced with the difficult task of setting up and utilizing virtual clusters that are located in multiple locations around the world and doing so for purposes of carrying out a single data analysis investigation. Such a task can quickly become overwhelming.

Qiu et al. [73] reviews High Performance Computing Enhanced Apache Big Data Stack HPC-ABDS and summarizes the capabilities in 21 identified architecture layers in Fox et al. [64]. The summary of these 21 identified architecture layers covers message and data protocols, distributed coordination, monitoring, infrastructure management, DevOps, in-

teroperability, cluster and resource management, programming models, and many more. In the first part of this dissertation, described in Section 1.2.1, we primarily focus on the DevOps layer and interoperability layer in these 21 arhictecture layers. These DevOps tools, such as Chef [6], Puppet [17], OpenStack [107], CloudMesh [8] etc, define computer systems in a variety of languages and automate system deployment, to ease the system administration tasks and provide cloud infrastructure interoperabitily. Typical interoperability libaraies are Libcloud [14], jClouds [4], DeltaCloud [9], etc. One of the common standards for interoperability is OASIS TOSCA [25] which focuses on the deployment of cloud services while workflow standards such as BPEL [23] focuses on business execution.

The motivating application of our work is data sharing, user management of resources, and performance optimization in the multi-institutional academia cloud, PRAGMA Cloud [128]. The PRAGMA Cloud infrastructure is both heterogeneous and distributed, with resources from multiple institutes from the Pacific Rim including the United States. The PRAGMA Cloud is loosely-coupled with differing cloud software, cluster sizes, and hardware architectures. In order to share datasets internationally among multiple research institutions, the PRAGMA community firstly pioneered solutions using a grid computing approach for resource and data sharing. The PRAGMA experience with grid software can be summarized as "Real science can be performed on such multi-institution environments, but the activation barrier is high." [129] More specifically, resources coordination, interoperable software deployments, and resource/user policies are significantly complicated because applications may have different requirements for middleware and resource allocation. To reduce the human cost of setting up sophisticated environments, the PRAGMA community started with a demonstration of modifying copies of VM images to run under different hypervisors. The goal was, and continues to be, to allow researchers to author their own application virtual

machines (VMs) using their preferred VM platforms and then deploy these VMs as a virtual cluster across different sites - the PRAGMA Cloud. A platform to automatically set up virtual cluster on demand on multi-cloud environments becomes key to reduce human cost so that the data sharing barrier can be reduced.

Researchers utilize multiple virtual clusters for analysis, each of which might consist of a small number of nodes tied to a co-located database. In our scenario, the data is tied to a location because of protections on the data, and any computation on the data has to occur co-located with the data. The Hathi Trust Research Center (HTRC) [12] enables text mining and non-consumptive research of the HathiTrust [11] Library while preventing intellectual property misuse within the confines of current U.S. copyright law. At the time of this writing, HathiTrust has 11 million volumes of digital publications. Processing 1 million volumes/books (2 TB large) using n-grams algorithm (implemented in MapReduce) on 1,024 cores takes about 22 hours, including loading data from Lustre [34] clusters to the compute resource for processing. The processing requires raw data to be moved from protected sources to a centralized cluster. To prevent the leak of the raw data, HTRC uses a restricted configuration approach, data capsule [33], which limits the computation resource (virtual machines) configurability. The approach has pre-defined stages to follow to guarantee that raw data cannot be accessed directly by users. The current HTRC data protection approach is based on the assumption that users are trustworthy. To overcome the limitation is this assumption, a proper data model needs to be discovered.

MapReduce [52] is de facto the most popular programming paradigm for processing large datasets running on a large number of compute resources. Researchers have been exploring variant MapReduce extensions and optimization methods to accommodate their needs. Most MapReduce implementations have been focused on running on tightly-

coupled homogeneous clusters, where the input data is often pre-placed locally. However, the homogeneity assumptions does not hold when scaling to multi-cloud environments such as PRAGMA Cloud. The primary motivations to scale up a data processing model, e.g. MapReduce, onto multi-cloud environments are driven from the following application characteristics.

- *Compute-Intensive:* A researcher may have access to several research clusters in their lab or university. These clusters often consist of a limited number of nodes, and the nodes in one cluster may be very different from those in another cluster in terms of CPU frequency, number of cores, cache size, memory size, and storage capacity. Commonly, a data processing framework, e.g. MapReduce, is deployed in a single cluster to run jobs, but any such individual cluster does not provide enough resources to deliver significant performance gain. For example, researchers at Indiana University have access to resources such as FutureSystems [61] [10], XSEDE [22], and PlanetLab [49] resources, but each cluster imposes a limit on the maximum number of nodes that a user can use anytime. Aggregating these isolated virtual clusters into the appearance of a single virtual cluster gives a powerful platform for seamless data analysis.
- *Data-Intensive:* It is increasingly common that data analysis on a single dataset or multiple datasets are widely dispersed. There are large differences in I/O speeds from local disk storage to wide area networks. Feeding a large dataset repeatedly to remote computing resources becomes the bottleneck. Moving computation to data becomes key to tackling this highly-distributed data issue. Therefore, the computing environment is no longer tightly-coupled. MapReduce has been deployed over Clouds that utilize multiple data centers [2] [3] [21] [31] , Grids that federate multiple HPC clus-

ters [86] [122], and Hybrid of Grids and Clouds [41] [30].

- *Data-Sensitive:* Collaborative e-Science projects typically require data processing to be performed on distributed datasets. Owners of data sets also have a range of concerns that include proper attribution [103], limited distribution of raw data prior to analysis (Hathi Trust Research Center [12]), and legal requirements to keep data within administrative boundaries. However, once the data has been processed, the aggregated or statistical result can be shared across the boundaries. When the data is highly distributed among multiple institutes (administrative domains), proper data model and computing paradigms need to be identified to process such workloads.

Our early work to scale up MapReduce to multi-cloud environments is the Hierarchical MapReduce (HMR) [86] project. HMR gathers computation resources from different clusters and runs MapReduce jobs across them. The applications implemented in this framework adopt a Map-Reduce-GlobalReduce model where computations are expressed as three functions: Map, Reduce, and GlobalReduce. Both the Map and the Reduce are executed on local clusters. The GlobalReduce is executed on one of the local clusters, collecting the output from all the local clusters. The initial implementation of HMR was a joint work at Indiana University with Zhenhua Guo, and was focused on compute-intensive applications.

1.2 Challenges and Contributions

What are the specific challenges? Controllability for one. This is the researcher's ability to easily set up and control the multi-hosted environment in which they will run the analysis. There could be issues with interoperability as well, as the virtual clusters could be hosted in developing countries for instance. These issues need to be addressed in a way that preserves

good performance. Processing sensitive and pinned data that is geographically distributed raises another challenge to secure proper data processing models. Moreover, although the transparency of cloud computing enables on-demand virtual resource provisioning without worrying about the physical infrastructure, it lacks information sharing between resource providers and applications. Therefore, the actual data locality and job locality cannot be obtained in the cloud environment without information sharing, which leads to inaccurate data locality scheduling at the application level.

This dissertation addresses techniques to enhance cloud controllability and interoperability, geographically distributed processing of immovable data, specifically in section 1.2.1 and section 1.2.2.

1.2.1 Multi-Cloud Controllability and Interoperability

Shared compute resources largely at research institutions, data analytics on a single but geographically distributed dataset, or multiple datasets that are similar but not part of a single uniformly handled dataset, and the constraints of sensitive data that effectively pins a dataset to a physical location is a unique environment is in which research is carried out. It is a scenario that will likely grow in need as data becomes more available worldwide. The tools to carry out research in this environment are often created for other similar but distinctly different environments. In particular, the assumption of heterogeneous, "pinned" data leads us to create new tools.

Typically, there is a gap between how resource providers manage their resources and how these resources are actually utilized by applications, a gap that is often filled by manual activity on the part of system administrators. The gap can be filled with a researcher-oriented controller that provides control over the resources and applications. The controller

service must be stateful so that resource status and application deployment/execution are monitored and recorded to enable management, fault tolerance, and application performance tuning.

To enable efficient computing on multi-cloud hosted virtual clusters simultaneously, information known at the resource provider level needs to be shared with the application and vice versa. This includes for instance, virtual cluster topology and provenance information or database location, and could be used to minimize I/O overhead. A resource allocation mechanism tries to guarantee that the users' minimum requirements are met by the provider's infrastructure. Without expressing application level resource requirements during the resource allocation stage, it is possible, for instance, that no two nodes are allocated close enough to minimize I/O overhead in the application. Therefore, the information sharing needs to be tridirectional. The current Infrastructure-as-a-Service based cloud systems are usually unaware of the characteristics of the applications and therefore allocate resources independently of their application profiles and data characteristics, which can significantly impact performance and security for distributed data-intensive and data-sensitive applications.

Contribution 1:

The first contribution of this dissertation is a virtual cluster controller (VCC) to enhance cloud controllability and interoperability that can make research in this unique environment easier to do.

VCC creates virtual clusters across multiple cloud platforms, builds a network overlay, and manages the virtual cluster life-cycle in a fault-tolerant manner. VCC extends HTCondor and leverages VPN technology to enhance user controllability and cloud in-

teroperability. Virtual machine images composed in different countries can be transfered across trusted sites via secured channels so that they can be instantiated at multiple places to form a cross-institutional virtual cluster. A virtual machine instance spinning at one institution might be migrated to another institution for any workload or administrative reasons. We develop a tri-directional information sharing model to assist resource allocation and application scheduling. We identify what information to share and how the information is shared. During the resource allocation phase, application specification and data specification will be utilized as key factors to better allocate resources in a matchmaking process. During the application execution phase, resource information, such as resource topology and provenance, will be utilized by application schedulers to achieve better performance.

The VCC differs from the works in Chapter 2.1 because it enables tri-directional resource-application-data information sharing, providing richer information for potential performance optimization for data loading, virtual cluster allocation and application scheduling. The information sharing are based on data locality, resource and application specification, and virtual cluster topology. With that being said, data is treated as a first class citizen where it negotiates with resource and application during resource allocation and application scheduling phases.

1.2.2 Manage, Access, and Process of Pinned Data

Collaborative e-Science projects typically require data processing to be performed on distributed datasets. A shortcoming of the current proof-of-principle infrastructure is the lack of support for controlled sharing of data in its various forms (e.g. flat files, databases, sensor data, and others). The input datasets could be distributed among clusters or even across data centers so that it becomes critical to utilize data locality information to opti-

mize the performance of data-intensive jobs. Furthermore, some datasets are so sensitive that they cannot be processed out of certain administrative boundaries. Owners of data sets also have a range of concerns that include proper attribution [103], limited distribution of raw data prior to analysis [12], and legal requirements to keep data within administrative boundaries. However, once the data has been processed, the aggregated or statistical result can be shared across the boundaries. Taking MapReduce as an example, the conventional MapReduce performs poorly when processing a dataset due to the nature of global data shuffling when the data is highly distributed among multiple institutions (administrative domains). Moreover, the conventional MapReduce even fails to process the data with locality sensitive constraint due to the global data exchange during the shuffling phase. The challenge is to strike the right balance between ensuring that the data processing carried out does not violate non-consumptive use, while keeping the data management services as flexible as possible by not overly limiting the kinds of use.

Contribution 2:

The second contribution of this dissertation is a conceptual definition of pinned data and its instantiation in Hierarchical MapReduce as Data Processing as a Service.

We define a novel data model, "pinned data" that, describes non-consumptive constraint of the data that raw data cannot leave a political jurisdiction. The "pinned data" design is inspired by Object Oriented Programming that data is encapsulated and accessed by method calls. A package of accessible pinned data is called a suitcase which contains certain data properties and operations that the suitcase can perform. Encapsulation combines data and behavior in one suitcase and hides the implementation of the data from the user of the suitcase. The data in a suitcase is called its content, and the functions and procedures that

operate on the data are called its methods. A suitcase does not expose data for public access directly. All communications are via method calls. Whenever a method is applied to a suitcase, provenance information is recorded.

The pinned data processing model is instantiated in Hierarchical MapReduce. Both the Map and the Reduce are executed on separate clouds where pinned data constraint meets. The GlobalReduce is executed on one of the clouds, collecting the aggregated results from all participant clouds. We enhance the original HMR framework so that different Map/Reduce functions can be encapsulated in different pinned data suitcases at different clouds before the GlobalReduce function aggregates data to produce final results. Therefore, a pinned data suitcase provides "Data-Processing-as-a-Service" capability.

Our solution differs from other solutions in Chapter 2.2 such as data capsules [33] in that data processing logic in our solution is regulated, and is part of a pinned data suitcase packaged by data providers which asserts the satisfaction of non-consumptive constraint. Our solution also differs from a Cloud Terminal [91] solution in that the output of application logics in our solution can be retrieved from the data source for further analysis.

1.3 Definition of Terminologies

- VCC: Virtual Cluster Controller.
- HMR: Hierarchical MapReduce.
- PND: Pinned data, or immovable data.
- PDS: Pinned data suitcase.
- HMR+: Enhanced Hierarchical MapReduce for processing pinned data.

1.4 Dissertation Outline

The remainder of this dissertation is organized as follows. Chapter 2 presents the related work. Chapter 3 presents Hierarchical MapReduce (HMR) framework and its application. It also describes the connection between HMR and PND where the local MapReduce tasks are part of the PND suitcases. Chapter 4 presents the Virtual Cluster Controlller (VCC) and addresses the cloud controllability and interoperability challenge by introducing virtual cluster controller. The chapter defines information sharing schema among resources, applications, and data. The schemas are used for resource allocation, application job scheduling, and pinned data definition. Chapter 5 defines pinned data (PND) and an enhanced HMR framework to process pinned data, where the local MapReduce tasks are part of the PND suitcases. Chapter 6 concludes the dissertation and lays out future work.

Chapter 2

Related Work

2.1 Multi-Cloud Controllability and Interoperability

Resource Controllability

The solutions to resource management are fragmented. Open source or commercial cloud platforms, such as Openstack, Eucalyptus [99], OpenNebula [93], Nimbus [75], Microsoft Azure and Amazon EC2 all provide dedicated VM resources but leave the bulk of the multi-cloud resource management burden to the programmers. Mesos [70] and YARN [120] provide resource allocation abstractions so that resources can be shared among different schedulers and applications but expect the scheduler to handle the resource management burden. On the other hand, conventional batch systems, e.g. PBS [69], SGE [63], LSF [130], provide inflexible static and pre-configured environments (queues) using policies designed to serve different types of parallel processing (e. g. MPI). Jobs submitted by different users in those system are typically queued to be executed and the user has limited control on where the jobs will be executed.

MapReduce [52] assumes resource allocation has already been performed before starting and also lacks the notion of user-level resource control and guarantee. Although Za-

haria et al. [124] provides a pool (set of slots) for each user to run MapReduce jobs within a MapReduce cluster, there is still no significant difference compared to conventional batch systems of user controllability.

Cloud Interoperability

Buyya et al. [40] introduces InterCloud, a federated cloud environment that allows the sharing of resource from multiple cloud providers. The InterCloud uses a market oriented approach [38] that facilitates pricing aware application brokering to call cloud coordinators on each cloud data center for resource allocation. Open Cirrus [27] is a heterogeneous federated cloud system that provides access to both virtualized resource and physical resource. OPTIMIS [57] is aimed at optimizing the whole service life cycle, including service construction, deployment, and operation. It can be used in multi-cloud scenarios where resources from more than one providers are offered via a cloud broker approach. RESERVOIR [105] also makes use of broker architecture in order to overcome cloud interoperability issue.

A cloud broker negotiates, monitors, and manages virtual clusters across multiple cloud providers using a network overlay. A cloud broker can manage resources that span multiple clouds, bursting resource from a private cloud to a public cloud in a hybrid cloud environment, enabling users to select resources from a federated cloud. Cloud brokers include ManageIQ [15], CloudForms [7], Rightscale [18], and Scalr [19]. These brokers typically integrate a set of tools to provide automation of lifecycle management of resources, and assist users in development of their customized cloud solutions. Open source toolkits, such as Libcloud [14], jClouds [4], DeltaCloud [9] , JumpGate [13], and PRAGMA Bootstrap [16], are capable of connecting an external cloud provider to a broker with the appropriate API.

Apache Whirr [5] is a set of libraries for running cloud services that build on jClouds [4], a java based abstraction that provides a common interface to a set of cloud providers such as Amazon EC2, Rackspace, VMWare vCloud, Openstack and CloudStack, etc. MapReduce applications can be directly deployed using Whirr.

Celesti et al. [43] develop a match-making strategy to select resources on federated clouds. Keahey et al. [76] introduces Sky Computing that provides end users virtual clusters interconnected with ViNe [118] across wide-area networks. Cloudmesh [8] provides Cloud Testbeds as a Service and focuses on the user being able to run repeatable experiments on different types of infrastructures. Cloudmesh, which emerged from the FutureSystems project [10], provides a convenient interface for managing user experiments and tracking usage on the different platforms. VCC differs from CloudMesh in that its focus is on providing information sharing and matchmaking among resources, applications and data.

Nimbus Phantom [77] is a multi-cloud, auto-scaling system that manages VM instances and aggregates monitoring information across different cloud providers. It allows the user to instrument VMs with a lightweight agent that retrieves information from a broker and reconfigures the VMs to perform different contextualization tasks, for example, to setup Torque/PBS schedulers. To the best of our knowledge, it does not provide virtual network configurability over allocated resources from multiple clouds. Commercial systems such as Amazon Virtual Private Cloud (Amazon VPC) lets a customer launch AWS resources in a customer-defined virtual private network (VPN), allowing the option to network in that customer's local machines into their Amazon VPC. However, VPC does not provide resource locality information to customers.

Liu et al. [85] enables cloud operators that use a declarative language to specify opti-

mization policy goals and constraints given provider objectives and customer SLAs. HTCondor's classad matchmaking framework uses a semi-structured data model that combines schema, data, and query in a simple specification language, and can be utilized in distributed environment with decentralized resources management.

Resources-Application-Data Cooperative Optimization

Lee et al. [80] built a prototype for topology-aware resource allocation (TARA) that adopts predication engine to estimate the performance of a given resource allocation and a genetic algorithm to find an optimized solution in the search space. The predict engine takes an objective function, an application description, and available resource information to compare and rank different candidates. However, TARA only supports Hadoop-based MapReduce framework. When considering application description, the prediction engine only takes Hadoop configuration file, job-specific resource requirements such as selectivity (input/output ratio), CPU cycles and overhead, without considering in advance application job data locality.

Palanisamy et al. [100] proposed Purlieus, a model for provisioning virtual MapReduce clusters in a locality-aware manner to improve runtime performance of individual jobs and reduce network traffic. Purlieus uses heuristics specifically developed for either Map-input heavy jobs, Reduce-input heavy jobs, or both to couple data and VM placement. However, their optimization strategy does not consider the time cost of loading data from sources.

Similar to Purlieus, Li et al. [82] introduces a cloud resource manager, CAM, that specially designed to maximize the locality for MapReduce in the cloud. CAM employs a min-cost flow based approach for data and VM placement to achieve VM closeness and hotspot avoidance. CAM uses network topology and storage topology information to assist

data and VM placement and MapReduce job scheduling. The goal of its resource allocation is to maximize the locality so that MapReduce job execution time can be minimized, without considering the time cost of data loading. Although CAM couples the resource allocation and job scheduling process, it only works in the homogeneous environment. The data placement strategy will be completely changed when considering data loading time cost in a multi-cloud environment where the system is heterogeneous.

Bu et al. [36] proposed CoTuner, a coordinated auto-configuration framework that co-configure VM and application, to optimize the system performance using Simplex-based optimization and reinforcement learning methods. However, they limit their scope within CPU and memory configuration, without considering network metrics and data locality.

Our work differs from those works by enabling generalized tri-directional resource-application-data information sharing, making optimization plans for data loading, virtual cluster allocation and application scheduling, based on data locality, resource and application specification, and virtual cluster topology.

2.2 Manage, Access, and Use of Pinned Data

Non-consumptive Data

Data capsules [33] is a mechanism for containing sensitive data within a virtualized environment with the goal of minimizing the available channels to leak that data. The Capsule system has two primary modes of operation: normal mode and secure mode. In normal mode, the computer behaves the same as it would without the Capsule system. In secure mode, the primary OS is blocked from sending output to the external network or to devices that can store data. The data capsules design is well-suited to non-expressive use, in that

it is able to maintain strong security guarantees on the sensitive data while being highly permissive to users, who may need to install custom software or perform other actions that would require the granting of many privileges in a normal environment. Zeng et al. [127] introduced a cloud framework that enforces non-consumptive constraint by extending the data capsules.

The basic assumption that these efforts based on, was that the users are trustworthy. However, users could intentionally deploy algorithms that leak copyrighted data through the VNC channel. An algorithm could intentionally obscures a final result by encoding copyrighted text in it. The data capsules solution limits users to running their analysis on a single VM, which needs extending to accommodate analysis workload that typically uses distributed data processing paradigm such as MapReduce. Our solution differs from those data capsules solution that algorithms are regulated, and are part of pinned data suitcase that packaged by data providers.

Martignoni et al. [91] presented Cloud Terminal, in which the only software running on the user side is a lightweight secure thin terminal, and most application logic is in remote servers. A cloud rendering engine on a remote server renders output of the application logic and send back bitmaps to the thin terminal on the user side for visualization. The Cloud Terminal simply supplies a secure display and input path to remote software, and therefore, secures data from malicious access. Although the Cloud Terminal is able to prevents data from leaking to user hosts, it does not provide a programmatic solution to use the output data. The bitmaps visualization only works for human-computer interactions. Our solution differs from the Cloud Terminal that the output of application logics can be retrieved from the data source for further analysis.

Geographically Distributed Data Processing in Clouds

Clouds give users a notion of virtually unlimited, on-demand resources for computation and storage. Attributed to its ease of executing large scale data-driven loosely coupled parallel applications, MapReduce has become a dominant programming model for running applications in a cloud. Therefore, we focus on MapReduce in this section.

MapReduce can be run on a single domain or multiple domains. A domain typically refers to a single cluster or multiple clusters that exist within one administrative boundary. There are several ways to deploy MapReduce over multiple domains, 1) deploy independent MapReduce clusters at each domain and a global layer to manage MapReduce jobs [86] [30]; 2) deploy MapReduce onto a virtual cluster aggregated from multiple domains [76] [92] [79]; or 3) deploy MapReduce directly onto multiple domains without adding virtual network layer [122] [108] [89].

Resources can be allocated for either shared or dedicated use. A group of dedicated resources can be a set of physical machines, a set of virtual machines provisioned in Clouds [31], or a set of nodes allocated in batch systems [86]. A group of shared resources can be a set of Volunteer Computing machines [83] [50] [115], or computing cycles from batch systems [122] [89].

Elteir et al. [56] classifies MapReduce jobs into recursively reducible jobs and non-recursively reducible jobs. Recursively reducible MapReduce has no inherent synchronization requirement between the Map and Reduce phases. Such jobs can be processed in a hierarchical reduction manner. The Hierarchical MapReduce (HMR) model [86] [88] was inspired by the combiner function from the original MapRedcue model and recursively reducible jobs discussed in Elteir et al. [56].

2.3 Background Related Work

MapReduce

MapReduce [52] is de facto the most popular programming paradigm for processing large datasets running on a large number of compute resources. MapReduce is applied to running on various types of resources. The Google MapReduce [52] and Hadoop [123] are optimized for running on single homogeneous clusters. Ranger et al. [104] describes Phoenix, an MapReduce implementation for shared-memory systems. Catanzaro et al. [42] and He et al. [68] implement MapReduce frameworks for graphics processors. Misco [54] and Hyrax [90] apply MapReduce model on Mobile Computing environments. Lin et al. [83], Costa et al. [50], Tang et al. [115] implements MapReduce on Volunteer Computing environments. Bicer et al. [30] run MapReduce jobs using a hybrid resources of a local cluster and Clouds.

Job scheduling is key to maximum system utilization and job turnaround time. Zaharia et al. [126] point out that heterogeneous environments seriously degrade the MapReduce job performance. The objective of Map Reduce job scheduling on heterogeneous environments mainly focuses on either performance [126] [26] [102] or cost [32].

Beside MapReduce itself, several variations of the MapReduce model have been created to handle different application scenarios, e.g., Hierarchical MapReduce [86], Iterative MapReduce [55], etc. Based on the scope of our thesis, which focuses on heterogeneous environments, we discuss of the original MapReduce and Hierarchical MapReduce below. The Iterative MapReduce, which involves multiple rounds of data combining and redistribution, is out of our scope.

Resource Allocation and Job Scheduling

Researchers have put significant effort into submission and optimal scheduling of massive parallel jobs in clusters, grids [60], and clouds. Conventional job schedulers, such as Condor [84], SGE [63], PBS [69], LSF [130], etc., provide highly optimized resource allocation, job scheduling, and load balancing within a single cluster environment. Grid brokers and metaschedulers on the other hand, e.g., Condor-G [62], CSF [53], Nimrod/G [39], GridWay [71], provide an entry point to multi-cluster grid environments. They enable transparent job submission to various distributed resource management systems via protocols such as GRAM [51] in Globus [59], hiding issues such as the locality of execution and availability of resources there.

In grid computing, advance reservations and co-allocation [58] on independently controlled and administered resources are needed to guarantee QoS criteria such as reliability, availability, cost and performance. Clouds ease the advance reservation and co-allocation process by fully control the resources and provision virtual machines before running applications. Typically, resource allocation in clouds are policy based [113] [98]. Resource schedulers are often used to apply these policies to the clouds. For instance, Haizea [111] [112] is a resource lease manager for OpenNebula [93], offering advance reservation policy of VM placement.

The goals of resource allocation solutions in clouds are different. Some of the solutions try to optimize energy consumption in clouds [28] [29] [46], or look into virtual network optimizations that aim to maximize the revenue of the service provider [47] [48]. Some other work focus on optimizing the application performance by meeting the job deadline at an acceptable cost [121] [30] [35] [78].

Park et al. [101] dynamically reconfigure VMs that increases or decrease the computing

capability of each node to enhance locality-aware job scheduling. It increases the computing resource of a VM where the next task has its data, and remove a idel virtual CPU from another VM in the same virtual cluster, keeping a constant cost for the user. Chen et al. [44] looks into MapReduce processing and build a resource-time cost model. The cost model is used to assist finding optimal amount of resources that can minimize the monetary cost with job finish time constraint.

From application scheduling view point with respect to data locality, Mohamed et al. [94] proposed a close-to-files (CF) job-placement algorithm to place job components on clusters with enough idle processors close to input files. Guo et al. [66] investigated data locality in MapReduce and propose an algorithm that schedules multiple tasks simultaneously rather than one by one to give optimal data locality. Zaharia et al. [125] proposes delay scheduling for MapReduce that increases system utilization and data locality. Li et al. [81] proposes a deadline-enabled delay (DLD) scheduling algorithm that optimizes job delay decisions according to real-time resource availability and resource competition, while still meeting job deadline constraints. Bu et al. [37] presents an interference and locality-aware task scheduler for MapReduce in virtual clusters and designs a task performance prediction model for interference-aware scheduling.

Existing Technologies for Our Work

HTCondor: HTCondor [117] is a high-throughput distributed batch system that provides job management, scheduling, resource monitoring, resource management, and fault tolerance. HTCondor seamlessly combines the computational power of distributed resources including VMs into one resource for users to leverage for their work. One of HTCondor's key features is that it provides a flexible and expressive framework for matching resource

requests with resources through its ClassAd mechanism. HTCondor integrates grid and cloud resources using a simple ASCII-based protocol, Grid Ascii Helper Protocol (GAHP) that handles both synchronous and asynchronous calls. A GAHP server receives ASCII commands via a socket from HTCondor and then directly interfaces with the individual grid or cloud services.

VPN: Virtual private networks (VPN) have been widely used for enabling wide-area access to resources in private organizational networks. Due to the simplicity of configuration and management we have integrated IPOP (IP-over-P2P) into VCC. IPOP is an open-source user-centric software virtual network allowing end users to define and create their own virtual private networks (VPNs). IPOP creates end-to-end peer-to-peer IP tunnels without requiring physical nor virtual routing infrastructure, using peers themselves as virtual routers and leveraging online social networks to establish VPN overlays among trusted endpoints.

Drools Rule Engine:

Drools implements and extends the Rete algorithm; Leaps used to be provided but was retired as it became unmaintained. The Drools Rete implementation is called ReteOO, signifying that Drools has an enhanced and optimized implementation of the Rete algorithm for object oriented systems. The Rules are stored in the Production Memory and the facts that the Inference Engine matches against are kept in the Working Memory. The Agenda manages the execution order of these conflicting rules using a Conflict Resolution strategy. The rule engine provides declarative programming capability to ease expression of solutions to difficult problems and consequently have those solutions verified. A rule engine also provides logic and data separation so that the logic can be much easier to maintain as there are changes in the future, as the logic is all laid out in rules. It is also very efficient to

match rule patterns to domain object data when datasets are changed in small portions as the rule engine can remember past matches.

Provenance: Provenance [109] capture is through Karma [110], a standalone tool that can be added to existing cyberinfrastructure for purposes of collection and representation of Open Provenance Model (OPM) [95] provenance data. Karma utilizes a modular architecture that permits support for multiple instrumentation plugins that make it usable in different architectural settings. We implement a improved Karma architecture by asynchronous publishing of provenance through a publish-subscribe system (messaging bus). The messaging bus acts as a buffer to the Karma server that offers less response time to applications for provenance ingestions.

Chapter 3

Background Work

In this chapter, we present our early work, Hierarchical MapReduce (HMR) [86], as background work to this dissertation. HMR gathers computation resources from different clusters and runs MapReduce jobs across them. The applications implemented in this framework adopt a Map-Reduce-GlobalReduce model where computations are expressed as three functions: Map, Reduce, and GlobalReduce. Both the Map and the Reduce are executed on local clusters. The GlobalReduce is executed on one of the local clusters, collecting the output from all the local clusters.

The initial implementation and evaluation of HMR, a joint work with Zhenhua Guo shown in Section 3.1, Section 3.3.1, and Section 3.4.1, focused on compute-intensive applications. Subsequent research and development was done solely by Yuan Luo on data-intensive applications, and the evaluation is presented throughout the rest of the sections in this chapter.

3.1 Hierarchical MapReduce Framework

Hierarchical MapReduce consists of two layers: a top layer has a global controller that accepts user submitted MapReduce jobs and distributes them across different local cluster

domains. Upon receiving a user job, the global controller divides the job into sub-jobs according to the capability of each local cluster. If the input data has not been deployed onto the cluster already, the global controller also partitions input data proportionally to the sub-jobs, and sends them to these clusters. After the jobs are all finished on all clusters, the global controller collects the outputs to perform a final reduction using the GlobalReduce which is also supplied by the user. The bottom layer consists of multiple local clusters. Each receive sub-jobs and input data partitions from the global controller, performs local MapReduce computation and sends results back to the global controller.

A HMR programmer need only supply two Reduces - one "local" $Reduce$, and one "global" $Reduce$ - instead of just one for the regular MapReduce. The only requirement is that the programmer must be sure that the formats of the local $Reduce$ output keys/value pairs match those of the GlobalReduce input key/value pairs. However, if the job is map-only, the programmer does not need to supply any $Reduce$ function, and the global controller simply collects the map results from all clusters and places them under a common directory.

3.1.1 Architecture

A high-level architecture diagram of the HMR is shown in Figure 3.1, consists of a global controller at the top layer and local MapReduce clusters at the bottom layer. The top layer consists of a job scheduler, a data manager, and a workload collector. The bottom layer consists of multiple clusters for running the distributed local MapReduce jobs, where each cluster has a HMR daemon with a workload reporter and a job manager. The compute nodes inside each of the cluster are not accessible from the outside.

When a user submits a MapReduce job to the global controller, the job scheduler splits

the job into a number of sub-jobs and assigns each to a local cluster based on several factors, including but not limit to the input dataset distribution, the current workload reported by the workload reporter from each local cluster, as well as the capability of individual nodes making up each cluster. This is done to achieve load-balance by ensuring that all clusters will finish their portion of the job in approximately the same time. The global controller also partitions the input data in proportion to the sub-job sizes if the input data have not been deployed before-hand. The data manager transfer the user supplied MapReduce jar and job configuration files with the input data partitions to the clusters. As soon as the data transfer finishes for a particular cluster, the HMR daemon of that cluster to start the local MapReduce job. Since data transfer is very expensive, we recommend that users only use the global controller to transfer data when the size of input data is small and the time spent for transferring the data is insignificant compared to the computation time. For large data sets, it would be more efficient and effective to deploy them before-hand, so that the jobs get the full benefit of parallelization and the overall time does not get dominated by data transfer. After the local sub-jobs are finished on a local cluster, if the application requires, the clusters will transfer the output to one of the clusters for global reduction. Upon receiving all the output data from all local clusters, a GlobalReduce will be invoked to perform the final reduction task, unless the original job is map-only.

3.1.2 Programming Model

The programming model of the HMR is the Map-$Reduce$-$GlobalReduce$ model where computations are expressed as three functions: Map, $Reduce$, and $GlobalReduce$. We use the term "$GlobalReduce$" to distinguish it from the "local" $Reduce$, but conceptually as well as syntactically, a $GlobalReduce$ is just another conventional $Reduce$. The Map,

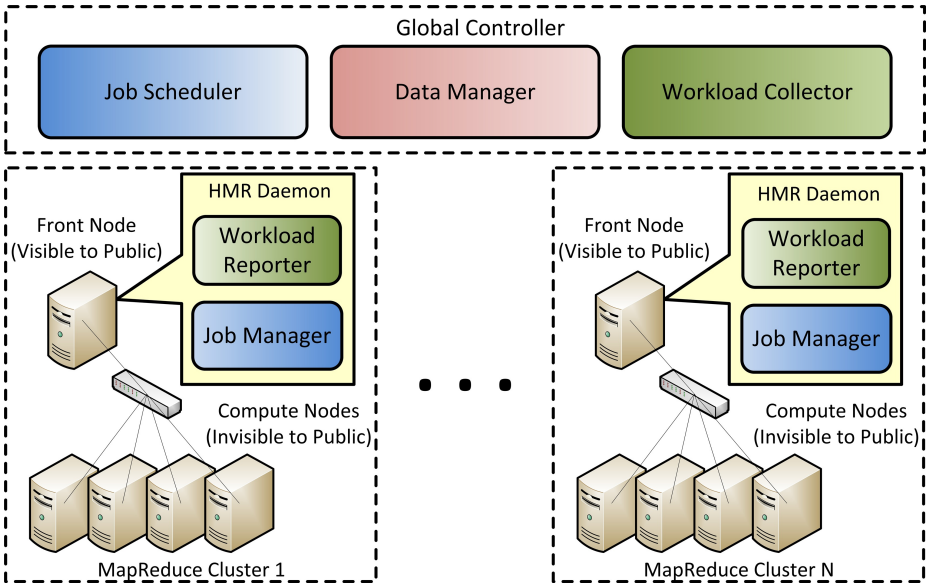

Figure 3.1: Hierarchical MapReduce Architecture.

just as with a conventional Map, takes an input pair and produces a set of intermediate key/value pairs; likewise, the $Reduce$, just as with a conventional $Reduce$, takes an intermediate input key and a set of corresponding values produced by the Map task, and outputs a different set of key/value pairs. Both the Map and the $Reduce$ are executed on local clusters. The $GlobalReduce$ is executed on one of the local cluster which is marked as global controller, using the output from the local clusters. Table 3.1 lists these 3 functions and also the input and output data types. An extended HMR model is introduced where different local Map/$Reduce$ functions can be executed, before the $GlobalReduce$ function aggregates data to produce final results. Note that the formats of the local $Reduce$s output keys/value pairs must match those of the $GlobalReduce$ input key/value pairs.

Figure 3.2 uses a tree-like structure to show the data flow across Map, $Reduce$, and $GlobalReduce$ functions. The leaf rectangle nodes with dotted line represent MapReduce clusters that execute the Map and $Reduce$ functions, and the root rectangle node with

Table 3.1: Input and output types of Map, Reduce, and GlobalReduce functions.

Function Name	Input	Output
Map	(k^i, v^i)	$[(k^m, v^m)]$
Reduce	$(k^m, [v_1^m, ..., v_n^m])$	$[(k^r, v^r)]$
GlobalReduce	$(k^r, [v_1^r, ..., v_n^r])$	v^o

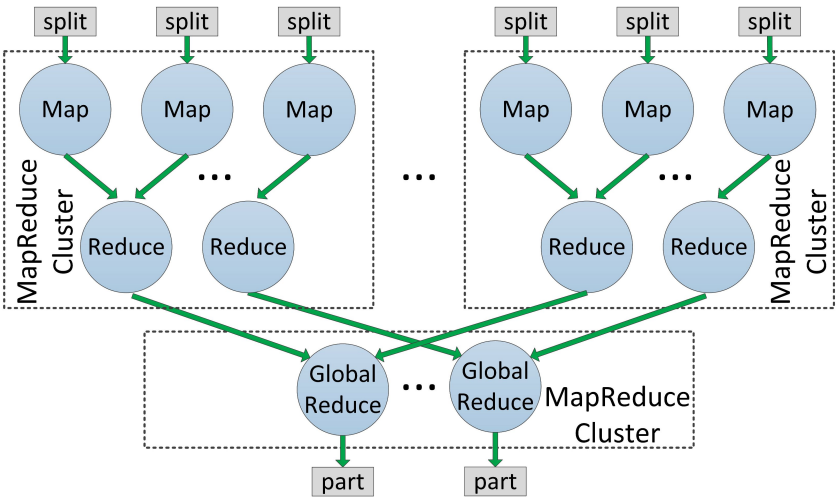

Figure 3.2: HMR Data Flow.

dotted line is a MapReduce cluster on which the $GlobalReduce$ takes place. The arrows indicate the direction in which the data sets (key/value pairs) flow. When a job is submitted into the system, the input dataset is partitioned into splits. The splits then are passed to the leaf nodes where Map tasks are launched. Each Map task consumes an input key/value pair and produces a set of intermediate key/value pairs. The set of intermediate pairs then are passed to the $Reduce$ tasks, which are also launched at the same cluster as the Map tasks. Each $Reduce$ task consumes an intermediate key with a set of corresponding values, and produces a different set of key/value pairs as output. All the local $Reduce$ results are sent to one local cluster to perform $GlobalReduce$ task. Each $GlobalReduce$ takes in a key and a set of corresponding values that were originally produced from the local $Reduce$ tasks, and computes and produces the final output.

Theoretically, the model can be extended to more than just two hierarchical layers, i.e. the tree structure in Figure 3.2 can have more depth by adding intermediate $Reduce$ steps that partially reduce the output from previous $Reduce$ functions.

3.2 HMR Job Scheduling

Application job scheduling is highly customizable and is a layer above the VCC. However, I illustrate through a Hierarchical MapReduce application, how resource topology information is used to assist job scheduling in the application layer.

The input dataset for a particular HMR job may be either submitted by the user to the global controller before execution, or pre-deployed on the local clusters (or sub-VCs in VCC terminology) and is exposed via application specification/metadata to the user who runs the MapReduce job. The early work [86] introduces Compute Capacity Aware Scheduling (CCAS) algorithm, described in section 3.2.1, that optimizes compute-intensive

jobs, by only considering compute power of each cluster without taking runtime topology information from resources. The subsequent work [87] introduces Data Location Aware Scheduling (DLAS) algorithm, described in section 3.2.1, identifies a candidate cluster for processing a data partition requires the physical residence of the data partition replica, but did not consider network metrics. In section 3.2.3, we introduces Topology and Data Location Aware Scheduling (TDLAS) algorithm, which takes virtual cluster topology information and data location information to schedule jobs.

3.2.1 Compute Capacity Aware Scheduling

The CCAS aims to optimize compute-intensive jobs. Assumptions are made where the input data of each Map tasks are equal in size and the Map tasks take approximately the same amount of time to run.

Consider running a MapReduce job on n clusters. Let $MapSlots_i$ be the maximum number of Map tasks that can be run concurrently on $Cluster_i$; $MapOccupied_i$ be the number of Map tasks currently running on $Cluster_i$; γ_i be the number of available slot to run Map tasks on $Cluster_i$; $NumCore_i$ be the total number of CPU Cores on $Cluster_i$, where $i \in \{1, \ldots, n\}$. And ρ_i defines the maximum number of Map tasks per core, so that,

$$MapSlots_i = \rho_i \times NumCore_i \tag{3.1}$$

and,

$$\gamma_i = MapSlots_i - MapOccupied_i \tag{3.2}$$

The weight of each sub-job W_i can be calculated from equation 3.3 where the factor θ_i is the computing power of each cluster, e.g., the CPU speed, memory size, storage capacity,

etc. The actual θ_i varies depending on the characteristics of the jobs,

$$W_i = \frac{\gamma_i \times \theta_i}{\sum_{j=1}^{n} \gamma_j \times \theta_j} \tag{3.3}$$

Let $JobMap$ be the total number of Map tasks for a particular job, which can be calculated from the number of keys in the input to the Map tasks, and $JobMap_i$ be the number of Map tasks to be scheduled to $Cluster_i$, so that

$$JobMap_i = W_i \times JobMap \tag{3.4}$$

Data-wise, let $Dsize$ be the total size of dataset for and SZ_i be the size of data to be scheduled to $Cluster_i$, so that

$$SZ_i = W_i \times Dsize \tag{3.5}$$

After splitting a MapReduce job into sub-MapReduce jobs based on equation 3.4, the dataset is partitioned and staged to the destination clusters accordingly. See Algorithm 1.

Algorithm 1 Compute Capacity Aware Scheduling

Require: $Dsize$ Dataset size, n Clusters

1: **function** CCAS($\gamma, \theta, n, Dsize$)
2: **for** i in n **do**
3: Calculate W_i using on equation 3.3.
4: $SZ_i \leftarrow W_i \times Dsize$
5: **end for**
6: **return** SZ ▷ Return the list of size of data to be processed on each cluster
7: **end function**

3.2.2 Data Location Aware Scheduling

The DLAS requires of a candidate cluster the physical residence of the data partition. Typically, with the assistance of a global file system, data partitions can be replicated among clusters. If more than one candidate cluster is found, the scheduling algorithm maps that data partition to one of the candidate clusters in a way that all selected clusters have similar ratio of data over cluster compute capacity, or called, balanced processing time.

Consider a dataset $DS = \{D_1, \ldots, D_m\}$ which has been partitioned to m partitions, residing on n clusters. Each partition has been replicated among N_j clusters, with the data size defined as SZ_{D_j}, where $1 \leq N_j \leq n$ and $j \in \{1, \ldots, m\}$. A scheduling plan k contains a list of subset of dataset DS. Let SDS_i^k be a subset of DS for $Cluster_i$, and

$$\exists k((\cup_{i=1}^{n} SDS_i^k = DS) \wedge (\cap_{i=1}^{n} SDS_i^k = \emptyset)) \tag{3.6}$$

The compute capacity of $Cluster_i$ is defined as W_i , where $i \in \{1, \ldots, n\}$, so that in $Cluster_i$, the ratio of data partitions collection over compute capacity in scheduling plan k is defined as R_i^k, and

$$R_i^k = \frac{\sum_{x \in SDS_i^k} SZ_x}{W_i} \tag{3.7}$$

And the total data processing time under a scheduling plan k is defined as,

$$T_k = \omega(max_{i \in \{1,\ldots,n\}} R_i^k) \tag{3.8}$$

in which ω is a constant value. The following equation finds minimized total processing time in all K scheduling plans.

$$T_{min} = min_{k \in K} T_k \tag{3.9}$$

If, $T_{min} = T_k$, then plan k is the optimal plan. To make the workload balanced among these

Algorithm 2 Data Location Aware Scheduling Algorithm

Require: m Data Partitions, n Clusters

1: **function** PREPAREDATASET(DS) ▷ Dataset $DS = \{D_1, \ldots, D_m\}$
2: Sort DS in decreasing order of partition size ▷ $SZ_{D_i} \geq SZ_{D_{i+1}}$
3: **for** $M[i][j]$ in M **do** ▷ M is a $m \times n$ matrix
4: **if** Data Partition D_i exists on $Cluster_j$ **then** $M[i][j] \leftarrow SZ_{D_i}$
5: **else** $M[i][j] \leftarrow 0$
6: **end if**
7: **end for**
8: **return** M
9: **end function**
10: **function** DLAS(DS, W)
11: $M \leftarrow$ PREPAREDATASET(DS)
12: **for** j in $1 \ldots n$ **do** ▷ Loop through each cluster
13: Calculate E_j based on equation 3.11.
14: **for** i in $1 \ldots m$ **do** ▷ Sum of size of un-planned data partitions in $Cluster_j$
15: $P_j \leftarrow P_j + M[i][j]$
16: **end for**
17: $CL_j \leftarrow 0$ ▷ CL_j is the current workload of $Cluster_j$
18: $Plan[j] \leftarrow null$ ▷ $Plan[j]$ is the name of the cluster to process data partition D_j
19: **end for**

20: **for** i in $1 \ldots m$ **do** ▷ Loop through each data partition
21: Sort $Clusters_{\{1 \ldots x\}}$ in increasing order of P_x/W_x values.
22: **for** j in $1 \ldots n$ and $M[i][j] \neq 0$ and $Plan[i] == null$ **do**
23: **if** $CL_j + SZ_{D_j} \leq E_j$ **then** $Plan[i] \leftarrow Cluster_j$; $CL_j \leftarrow CL_j + SZ_{D_i}$
24: **end if**
25: **end for**
26: **if** $Plan[i] == null$ **then**
27: $Clist \leftarrow$ List of $Cluster_x$ having smallest CL_x/W_x value
28: $k \leftarrow x$ of $Cluster_x$ in $Clist$ having smallest P value
29: $Plan[i] \leftarrow Cluster_k$; $CL_k \leftarrow CL_k + SZ_{D_i}$
30: **end if**
31: **for** j in $1 \ldots n$ **do**
32: $P_j \leftarrow P_j - SZ_{D_i}$ ▷ Update the size of un-planned data partitions in $Cluster_j$
33: **end for**
34: **end for**
35: **return** $Plan$ ▷ Return a scheduling plan
36: **end function**

clusters, R_i^k values in plan k should be as close as possible. Ideally,

$$R_1^k = R_2^k = \cdots = R_n^k \tag{3.10}$$

so that the expected size of the data to be run on $Cluster_i$ are calculated as E_i.

$$E_i = \frac{W_i}{\sum_{k=1}^{n} W_k} \times \sum_{j=1}^{n} SZ_{D_j} \tag{3.11}$$

The DLAS algorithm is an greedy algorithm towards finding an optimal plan. In the current version of DLAS, a few assumptions are made, 1) the data partitions are relatively small in comparison to the whole dataset so that no further partitions to be made, because smaller granularity of data partition leads to better chance of load balance; 2) data replicas are randomly replaced among clusters; 3) no data transfer activities are allowed.

In Algorithm 2, the data partitions are sorted in decreasing order of size, and an $m \times n$ matrix M is constructed in which $M[i][j]$ denotes the size of data partition D_i if exists on $Cluster_j$. The expected data sizes on each cluster are calculated as E_i. For each data partition D_j, Sort candidate clusters (which contains replica of D_j) in increasing order of the total un-planned data size, and assign data partition D_j to first candidate cluster, if current load on that cluster plus SZ_{D_j} is less than E value; otherwise, assign data partition D_j to the first cluster with least current load. The returned schedule plan contains a list of data partition and scheduled cluster mappings.

3.2.3 Topology and Data Location Aware Scheduling

The DLAS algorithm does not take into consideration transferring data among clusters, so it does not perform well under conditions of an imbalanced replica distribution scenario. Besides, if data partitions are relatively large in proportion to the whole dataset, DLAS

fails to balance the workload without further partitioning the existing data partitions. A Topology and Data Location Aware Scheduling (TDLAS) algorithm is briefly introduced, which takes virtual cluster topology information and data location information to schedule jobs.

In TDLAS, PR_{ab} denotes the processing rate for data on an outside location a to be processed on VC_b that

$$PR_{ab} = \frac{S}{T_{ab} + T_b} \tag{3.12}$$

where T_{ab} is the time cost of transferring data from a to b, T_b is the time cost of compute data on b, and S is the data size. Since $T_{ab} = \frac{S}{B_{ab}} + L_{ab}$ where B_{ij} and L_{ij} are bandwidth and latency value from a to b, and $T_b = \frac{S}{CR_b}$ where CR_b is the rate to compute data on b, we can compute PR_{ab}

$$PR_{ab} = \frac{S}{\frac{S}{B_{ab}} + L_{ab} + \frac{S}{CR_b}} \tag{3.13}$$

To simplify the PR_{ab} calculation process, we use $T'_{ab} = \frac{S}{B_{ab}}$ by removing the L_{ab}, which will be added back in Equation 3.15. Therefore, we compute PR'_{ab} instead of PR_{ab}

$$PR'_{ab} = \frac{S}{T'_{ab} + T_b} = \frac{B_{ab} \times CR_b}{B_{ab} + CR_b} \tag{3.14}$$

Note that the CR_b is derived from application history data and its value varies from case to case. Let PT_j be data processing time on resource j. We compute

$$PT_j = \sum_{i=1}^{m} PR'_{ij} \times S_{ij} + \max_{i \in M} L_{ij} \tag{3.15}$$

where S_{ij} is the size of the data on place i that assigned to be processed on resource j. Therefore, the makespan of the job, T, is the maximum value of the data processing time on each sub-VC:

$$PT = \max_{j \in N} PT_j \tag{3.16}$$

subject to

$$\max_{j \in N} PT_j - \min_{j \in N} PT_j < \theta \tag{3.17}$$

where N is the full set of sub-VCs in the provisioned virtual cluster, and θ is a threshold to balance the workload among in the virtual cluster.

The TDLAS algorithm is briefly described as follows:

1. Calculate PR' value for each subset of data against all sub-VCs.
2. Assign each subset of data to a sub-VC with maximum PR' value.
3. If Inequality 3.17 is not satisfied, re-partition data on a sub-VC, VC_j, where $PT_j = \max_{i \in N} PT_i$, distributed partial data to other sub-virtual-clusters with less workload but smaller PR' value. Repeat step 3 until inequality 3.17 is satisfied.

3.3 Hierarchical MapReduce Applications

3.3.1 Compute-Intensive Applications

AutoDock [97] is a suite of automated docking tools for predicting the bound conformations of flexible ligands to macromolecular targets. It is designed to predict how small molecules of substrates or drug candidates bind to a receptor of known 3D structure. Running AutoDock requires several pre-docking steps, e.g., ligand and receptor preparation, and grid map calculations (AutoGrid), before the actual docking process can take place. There are desktop GUI tools for processing the individual AutoDock steps, such as AutoDockTools (ADT) [97] and BDT [119], but they do not have the capability to efficiently process thousands to millions of docking processes. Ultimately, the goal of a docking experiment is to illustrate the docked result in the context of macromolecule, explaining

Table 3.2: AutoDock HMR input fields and descriptions.

Field	Description
ligand_name	Name of the ligand
autodock_exe	Path to AutoDock executable
input_files	Input files of AutoDock
output_dir	Output directory of AutoDock
autodock_parameters	AutoDock parameters
summarize_exe	Path to summarize script
summarize_parameters	Summarize script parameters

the docking in terms of the overall energy landscape. Each AutoDock calculation results in a docking log file containing information about the best docked ligand conformation found from each of the docking runs specified in the docking parameter file (dpf). The results can then be summarized interactively using the desktop tools such as AutoDockTools or with a python script. A typical AutoDock based virtual screening consists of a large number of docking processes from multiple targeted ligands that takes a large amount of time to finish. However, the docking processes are data independent, so if several CPU cores are available, these processes can be carried out in parallel to shorten the overall makespan of multiple AutoDock runs.

We apply the Hierarchical MapReduce programming model to running multiple AutoDock instances to support the feasibility of our approach. The key/value pairs of the input of the Map tasks are ligand names and the location of ligand files. We designed a simple input file format for AutoDock MapReduce jobs. Each input record, which contains the 7 fields shown in Table 3.2, corresponds to a Map task.

The Map, $Reduce$, and $GlobalReduce$ functions are implemented as follows:

1) Map: The Map task takes a ligand to run the AutoDock binary executable against a shared receptor, and then runs a Python script $summarize\_result4.py$ to output the lowest energy result using a constant intermediate key.

2) $Reduce$: The $Reduce$ task takes all the values corresponding to the constant intermediate key and sorts the values by the energy from low to high, and outputs the sorted results to a file using a local $Reduce$ intermediate key.

3) $GlobalReduce$: The $GlobalReduce$ finally takes all the values of the local reducer intermediate key, sorts and combines them into a single file by the energy from low to high.

3.3.2 Data-Intensive Applications

A HMR version of $grep$ is written as a data-intensive application. The Map, $Reduce$, and $GlobalReduce$ functions are implemented as follows:

1) Map: The Map function captures the matching lines of a regular expression input.

2) $Reduce$: The $Reduce$ function outputs all the matching lines from local Hadoop execution.

3) $GlobalReduce$: The GlobalReduce function collects all the output from local Hadoop execution and combines them into a single output file.

The total execution time varies with respect to different data distribution and different regular expression input.

For the compute-intensive applications such as AutoDock, the compute capacity aware scheduling algorithm (CCAS) works well. CCAS is further discussed in Chapter 3.2.1. It

will not necessarily be the case when an application has larger data sets that data movement becomes significant. As an alternative to transferring data explicitly from site to site, we explore using a shared file system to share datasets among MapReduce clusters. In this section, we discuss the implementation of using a distributed file system called Gfarm [116] on top of the Hadoop clusters to support data location aware scheduling algorithm (DLAS). DLAS is further discussed in Chapter 3.2.2.

Gfarm is a global distributed file system that federates local file systems on compute nodes to maximize distributed file I/O bandwidth, and allows the storage of multiple file replicas in different locations to enhance parallel I/O, to avoid read access concentration of hot files, and for fault tolerance.

Since data transfer can be expensive, if using architecture in Figure 3.1, we recommend that users only use the global controller to transfer data when the size of input data is small and the time spent for transferring the data is insignificant compared to the computation time. For large data sets, it would be more efficient to deploy them before-hand, so that the jobs get the full benefit of parallelization and the overall time does not become dominated by data transfer. More efficiently, when using a shared file system, the large date sets do not even need to move across clusters. Instead of transferring data between the user site and MapReduce clusters upon making scheduling decisions, the new modification has Gfarm file system deployed at head nodes of MapReduce clusters. In this particular setting, a Gfarm metadata server sits on a clusterś head node, and Gfarm I/O servers and Gfarm client sit on every head node of clusters. The system will take advantage of the data locality when making scheduling decisions.

Figure 3.3 describes the steps of a HMR execution with Gfarm settings. The blue box steps are executed on the Global Controller and the yellow box steps are executed on local

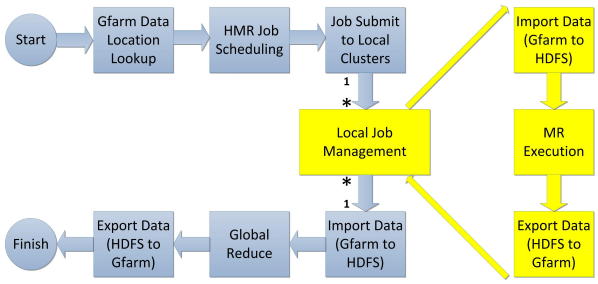

Figure 3.3: HMR Execution Steps.

Hadoop clusters. When a HMR job is launched, the framework first looks up the location of the input dateset in Gfarm. A DLAS scheduling decision is made based on the input data locality afterwards. The sub-MapReduce jobs are then submitted to local clusters. On local clusters, input dataset is imported from Gfarm to HDFS before MapReduce execution, so that the data set can be distributed over the compute nodes of a cluster. When local MapReduce execution is complete, the output dataset is exported from HDFS to Gfarm file system. The Global Controller then imports the local output dataset from Gfarm to HDFS on the Global Controller node before performing GlobalReduce execution. The output of GlobalReduce then is exported from HDFS to Gfarm and before the end of HMR execution.

3.4 HMR Framework Evaluation

3.4.1 Evaluation of AutoDock HMR

The HMR is evaluated using the AutoDock application. HMR is written in Java and Shell scripts and fully uses the Hadoop system. ssh and scp scripts are used to manage the data stage-in and stage-out. On the local clusters side, the workload reporter is a component that exposes Hadoop cluster load information accessed by global job scheduler.

Our evaluation was carried out on the IU Quarry cluster and two clusters, Hotel and

Table 3.3: Cluster Node Specifications.

Cluster	CPU	CPU/Node	Cache size	Memory	OS	Nodes Alloc
Hotel	Intel Xeon 2.93GHz	8	8192KB	24GB	Linux 2.6.18 SMP	21
Alamo	Intel Xeon 2.67GHz	8	8192KB	12GB	Linux 2.6.18 SMP	21
Quarry	Intel Xeon 2.0GHz	8	6144KB	16GB	Linux 2.6.18 SMP	21

Alamo, in FutureGrid, see Table 3.3 for specifications. IU Quarry is a classic HPC cluster with several login nodes that are publicly accessible from outside. After a user logs in, he/she can do various job-related tasks, including job submission, job status query and job cancellation. The computation nodes however, cannot be accessed from outside. Two distributed file systems (e.g., Lustre [34], GPFS [106]) are mounted to each computation node for storing input data accessed by the jobs. FutureGrid partitions the physical cluster into several parts, each of which provides a different testbed such as Eucalyptus, Nimbus, and HPC.

To deploy Hadoop to traditional HPC clusters, the built-in job scheduler (PBS) is used to allocate nodes. Although PBS has no knowledge of data locality when allocating nodes, it is still suitable for compute-intensive jobs in which data transfer cost is insignificant. To balance maintainability and performance, the Hadoop program is installed in a shared directory while storing data in a local directory, because the Hadoop program (Java jar files, etc.) is loaded only once by Hadoop daemons whereas the HDFS data is accessed multiple times. 21 nodes for each cluster are allocated, within which one node is a dedicated master node (HDFS namenode and MapReduce jobtracker) and other nodes are data nodes and task trackers.

Considering that AutoDock is a CPU-intensive application, ρ_i is set to 1 per Section 3.3 so that the maximum number of Map tasks on each node is equal to the number of cores

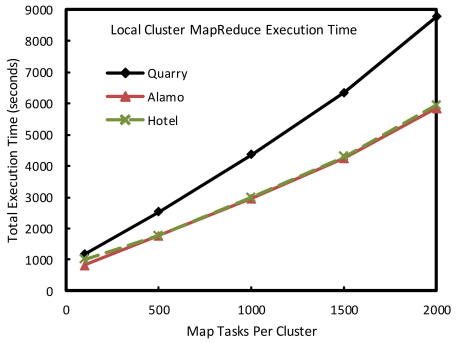

Figure 3.4: Local cluster MapReduce execution time based on different number of Map tasks.

on the node. The version of AutoDock is 4.2 which is the latest stable version. The global controller does not care about low-level execution details because our local job managers hide the complexity.

During the experiments, 6,000 ligands and 1 receptor are used. One of the most important configuration parameters is ga_num_evals - number of evaluations. The larger its value is, the higher the probability that better results may be obtained. Based on prior experiences, the ga_num_evals is typically set from 2,500,000 to 5,000,000. We configure it to 2,500,000.

Test Case 1

The first test case is a base test case not involving the Global Controller. It is used to determine how each local Hadoop cluster performs under different numbers of Map tasks. The AutoDock application is executed in the Hadoop to process 100, 500, 1000, 1500 and 2000 ligand/receptor pairs in each of the three clusters.

As is shown in Figure 3.4, the total execution time vs. the number of Map tasks on each

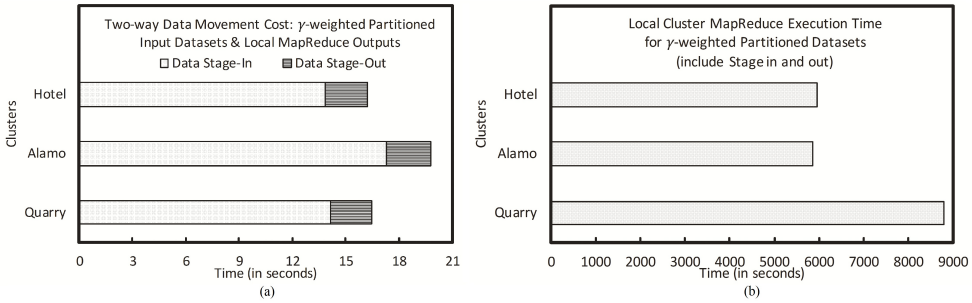

Figure 3.5: (a) Two-way data movement cost of γ-weighted partitioned datasets: local MapReduce inputs and outputs; (b) Local MapReduce turnaround time of γ-weighted datasets, including data movement cost.

cluster is close to linear, regardless of the startup overhead of the MapReduce jobs. The total execution time of the jobs running on the Quarry cluster is approximately 50% slower than running on Alamo and Hotel. The main reason is that nodes of the Quarry cluster have slower CPUs compared with that of Alamo and Hotel.

Test Case 2

The second test case shows the performance of executing MapReduce jobs with γ-weighted partitioned datasets on different clusters, which is based on the following parameters setup. For equation (4) from section 3.3, we set $\theta_i = C$, where C is a constant, and $i \in \{1, \ldots, n\}$ for our three clusters. The calculation shows $\rho_1 = \rho_2 = \rho_3 = 160$, given no MapReduce jobs are running beforehand. Therefore, the weight of Map tasks distribution on each cluster is $W_i = \frac{1}{3}$. We then equally partition the dataset (apart from shared dataset) into 3 pieces, stage the data together with the jar executable and job configuration file to local clusters for execution in parallel. After the local MapReduce execution, the output files will be staged back to the global controller for the final reduce.

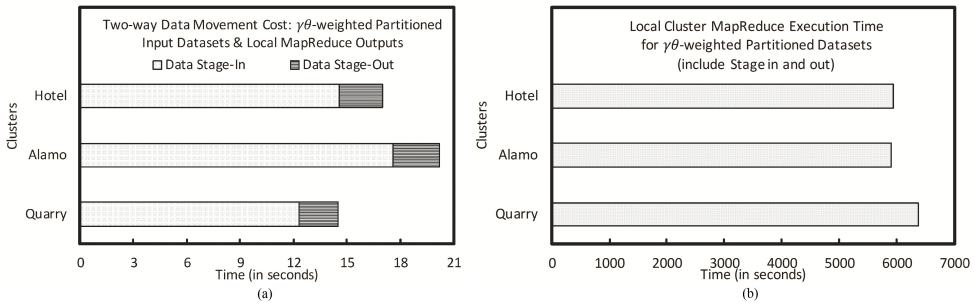

Figure 3.6: (a) Two-way data movement cost of $\gamma\theta$-weighted partitioned datasets: local MapReduce inputs and outputs; (b) Local MapReduce turnaround time of $\gamma\theta$-weighted datasets, including data movement cost.

Figure 3.5(a) shows the data movement cost during stage-in and stage-out. The input dataset of AutoDock contains 1 receptor and 6000 ligands. The receptor is described as a set of approximately 20 gridmap files totaling 35MB in size, and the 6000 ligands are stored in 6000 separate directories, each of which is approximately 5-6 KB large. In addition, the executable jar and job configuration file together has a total of 300KB in size. For each cluster, the global controller creates a 14MB tarball containing 1 receptor file set, 2000 ligands directories, the executable jar, and job configuration files, all compressed, and transfers it to the destination cluster, where the tarball is decompressed. This is called global-to-local procedure "data stage-in". Similarly, when the local MapReduce jobs finish, the output files together with control files (typically 300-500KB in size) are compressed into a tarball and transferred back to the global controller. This is called local-to-global procedure "data stage-out". As can be seen from Figure 4, the data stage-in procedure takes 13.88 to 17.3 seconds to finish, while the data stage-out procedure takes 2.28 to 2.52 seconds to finish. The Alamo cluster takes a little longer to transfer the data but the difference is insignificant compare to the relatively long duration of local MapReduce

executions.

The time it takes to run 2000 Map tasks on each of the local MapReduce clusters varies due to the different specification of the clusters. The local MapReduce execution makespan, including data movement costs (both data stage-in and stage-out) is shown in Figure 3.5(b). The Hotel and Alamo clusters take similar amount of time to finish their jobs, but the Quarry cluster takes approximately 3,000 additional seconds to finish, about 50% more than Hotel and Alamo. The $GlobalReduce$ task is only invoked after all the local results are ready in the global controller, and it takes only 16 seconds to finish. Thus, the relatively poor performance on Quarry becomes the bottleneck on the current job distribution.

Test Case 3

The third test case evaluates the performance of executing MapReduce jobs with $\gamma\theta$-weighted partitioned datasets on different clusters, which is based on the following setup. From test cases 1 and 2, we observe that although all clusters are assigned the same number of compute nodes and cores to process the same amount of data, they take significantly different amount of time to finish. Among the three clusters, Quarry is much slower than Alamo and Hotel. The specifications of the cores on Quarry, Alamo and Hotel are Intel Xeon E5335 2GHz, Intel Xeon X5550 2.67GHz, and IntelXeon X5570 2.93GHz, respectively. The inverse ratio of CPU frequency and that of processing time match roughly. The hypothesis is that the difference in processing time is mainly due to the different core frequencies, therefore, it is not enough to merely factor in the number of cores for load balancing, and the computation capabilities of each core are also important. Next, the scheduling policy is refined to add CPU frequency as a factor to set θ_i . Here we set $\theta_1 = 2.93$ for Hotel, $\theta_2 = 2.67$ for Alamo, and $\theta_3 = 2$ for Quarry. As is for test case 2, ρ values are calculated

where $\rho_1 = \rho_2 = \rho_3 = 160$, given no MapReduce jobs are running beforehand. Thus, the weights are $Weight_1 = 0.3860$, $Weight_2 = 0.3505$, and $Weight_3 = 0.2635$ for Hotel, Alamo, and Quarry respectively. The dataset is also partitioned according to the new weight, which is 2316 Map tasks on Hotel, 2103 on Alamo, and 1581 on Quarry.

Figure 3.6(a) shows the data movement cost in the weighted partition scenario. The variation in the size of the tarball for different number of ligand sets is quite small, smaller than 2MB. As one can see from the graph, the data stage-in procedure takes 12.34 to 17.64 seconds to finish, while the data stage-out procedure takes 2.2 to 2.6 seconds to finish. Alamo takes a little bit longer to transfer the data but the difference is also insignificant given the relatively long duration of local MapReduce executions as in the previous test case.

With weighted partition, the local MapReduce execution makespan, including data movement costs (both data stage-in and stage-out) are shown in Figure 3.6(b). All three clusters take similar amount of time to finish the local MapReduce jobs. We can see that our refined scheduler configuration improves performance by balancing workload among clusters. In the final stage, the global reduction combines partial results from local reduces and sorts the results. The average GlobalReduce time taken after processing 6000 Map tasks (ligand/receptor docking) is 16 seconds.

3.4.2 Gfarm Over Hadoop Clusters

Preliminary Tests

We demonstrate the execution on PRAGMA [128] testbed using ”grep” application and give preliminary experimental results. Two virtual clusters (pragma-f0 and pragma-f1)

were provisioned, each of which contains a virtual front node and 3 virtual compute nodes. All the virtual nodes are provisioned with 1 core of CPU, 1GB memory, and 80GB storage. Both of the virtual clusters are deployed as local Hadoop clusters and one of the virtual clusters (pragma-f0) is deployed as Global Controller. Gfarm Metadata server is deployed also on pragma-f0 cluster. Both pragma-f0 and pragma-f1 are installed with Gfarm I/O server (Data Nodes), and Gfarm client.

The total execution time varies with respect to different file distribution and different regular expression input. In our preliminary test, the input data set contains 10 files. We generate each of the file with the size of 20MB, 40MB, 60MB, 80MB, 100MB large. Those 10 files were evenly distributed between the two clusters. The total execution time increases linearly from 210 seconds to 243 seconds when the size of input increases.

Chapter 4

Virtual Cluster Controller

In this chapter, we introduce a virtual cluster management solution, virtual cluster controller (VCC), a building block for geographically distributed data processing. VCC creates virtual clusters across multiple cloud platforms, builds a network overlay, and manages the virtual cluster life-cycle in a fault-tolerant manner. VCC extends HTCondor and leverages VPN technology to enhance user controllability and cloud interoperability. A standalone matchmaker is in place to matchmake resources, applications, and data. Virtual machine images composed in different countries can be transfered across trusted sites via secured channels so that they can be instantiated at multiple places to form a cross-institutional virtual cluster. A virtual machine instance spinning at one institution might be migrated to another institution for whatever workload or administrative reasons.

The virtual cluster controller (VCC) is positioned as the middle layer of the cloud stack. This chapter introduces the design and implementation of VCC and its application in PRAGMA.

4.1 Architecture

VCC is architected to layer on top of cloud providers and beneath the application. The layers are defined as follows.

Application: a program or a group of programs that utilize cloud resources to perform specialized tasks. Applications typically differ from each other in terms of resource requirements, communication patterns, and data localities.

Controller: VCC enables allocation of cloud resources from multiple providers with minimum system administrator assistance. This includes transferring VM images to the remote resources, booting the VMs, building the network overlay, and monitoring and managing the the resources with minimal effort from the user. A resource allocation algorithm in VCC takes into account resource and application specifications as input data and generates a resource allocation plan aimed at optimizing execution of a user's applications. The resource allocation model is further discussed in Section 4.3.

Cloud provider: Cloud provider is a self-contained system that provisions VM instances on physical resources on behalf of a user. A user can often have access to multiple cloud providers. A network overlay can then be used to integrate VMs on different physical machines and geographic locations so that they appear like they are on the same private network. This eases the management burden to the user as their VMs then appear as nodes within the same cluster. The network overlay details are discussed in Section 4.1.1.

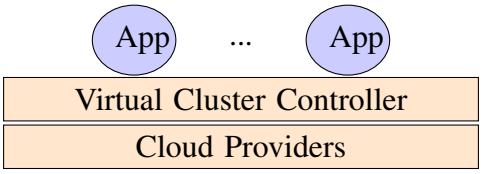

Figure 4.1: Resource management system for hybrid cloud infrastructures.

4.1.1 VCC Design

VCC has four main components: Resource Registry, Resource Allocator, and Allocation Manager, and Runtime Information Manager. Each component is further described below.

- Resource Registry (RR): A resource registry is a cache that contains resource specifications (RS). The RS in RR are frequently updated with the latest status from cloud providers. More formally, the resource availability ($RS.avail$) values in RR are eventually consistent with the actual resource availability, i.e., an out-of-date $RS.avail$ value will cause a VCC provision request to fail, forcing a refresh of status data and corresponding update to $RS.avail$.
- Resource Allocator (RA): A resource allocator provisions resources utilizing the resource allocation module described in the previous section.
- Allocation Manager (AM): An allocation manager starts, stops, monitors, and handles fault tolerance of the allocated resources.
- Runtime Information Manager (RIM): An runtime information manager collects virtual cluster topology information and provenance information. The information is accessible by Resource Allocator and Applications.

4.1.2 VCC Implementation

The implementation of VCC leverages existing tools as described further. In summary, VCC extends the resource management and fault tolerant capabilities of HTCondor [117] via implementing a pluggable interface. VCC, shown in Fig. 4.2, through existing tools, manages and configures various resources such as PRAGMA Bootstrap [16] and IPOP [74]. Access is through an easy-to-use Web interface.

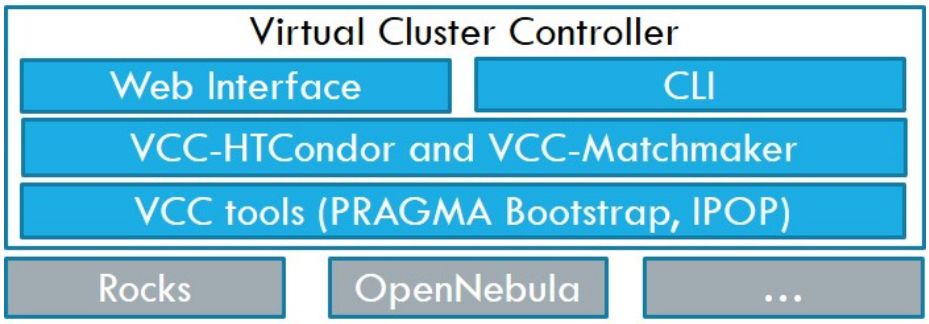

Figure 4.2: Virtual Cluster Controller Architecture

PRAGMA Bootstrap

PRAGMA Bootstrap is a set of scripts used to instantiate virtual clusters in PRAGMA Cloud. The PRAGMA Bootstrap scripts 1) prepare both a front end and a compute node image selected by the user; 2) verify resources availabilities such as public IP, private IPs, MAC addresses, and computing resources; and 3) transfer and boot VMs. The PRAGMA Bootstrap currently supports cloud platforms such as Rocks [20] and OpenNebula [96]. Virtual cluster images can be located in a local physical machine, http repository, or Cloud-Front [1] repository.

HTCondor Extension

VCC's three main components are implemented as follows:

- Resource Registry (RR): The resource registry is implemented using customized HTCondor ClassAds and cron job scripts.
- Resource Allocator (RA): The resource allocator is implemented using a stand-alone matchmaker.

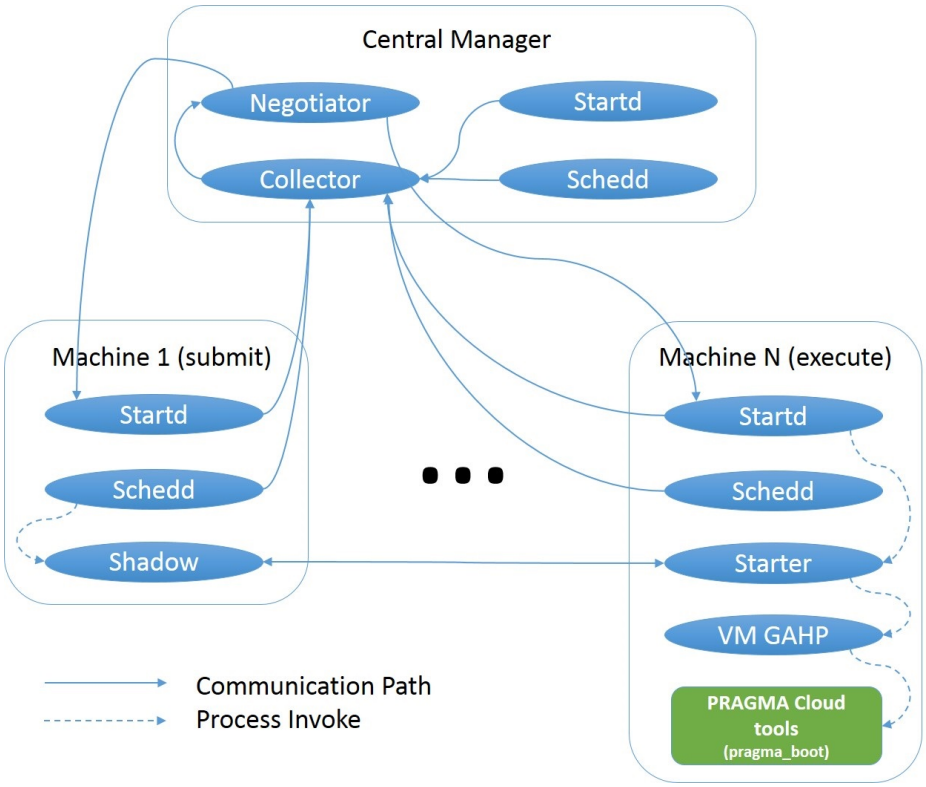

Figure 4.3: Architecture of VCC-HTCondor Pool

- Allocation Manager (AM): The allocation manager is HTCondor with a VCC extension called VCC-HTCondor.

HTCondor can start and control VMs from its virtual machine universe. The stock VM GAHP code, $condor\_vm\_gahp$, uses Xen/KVM/VMWare to start and control VMs under HTCondor's Startd. The VCC extension to HTCondor is a new VM GAHP server based on modified HTCondor source code for $condor\_schedd$, $condor\_startd$, and $condor\_starter$ daemons, and $condor\_vm\_gahp$ that allows for the invocation of wrapper scripts. Wrapper scripts were then written to invoke external tools such as PRAGMA Bootstrap and IPOP.

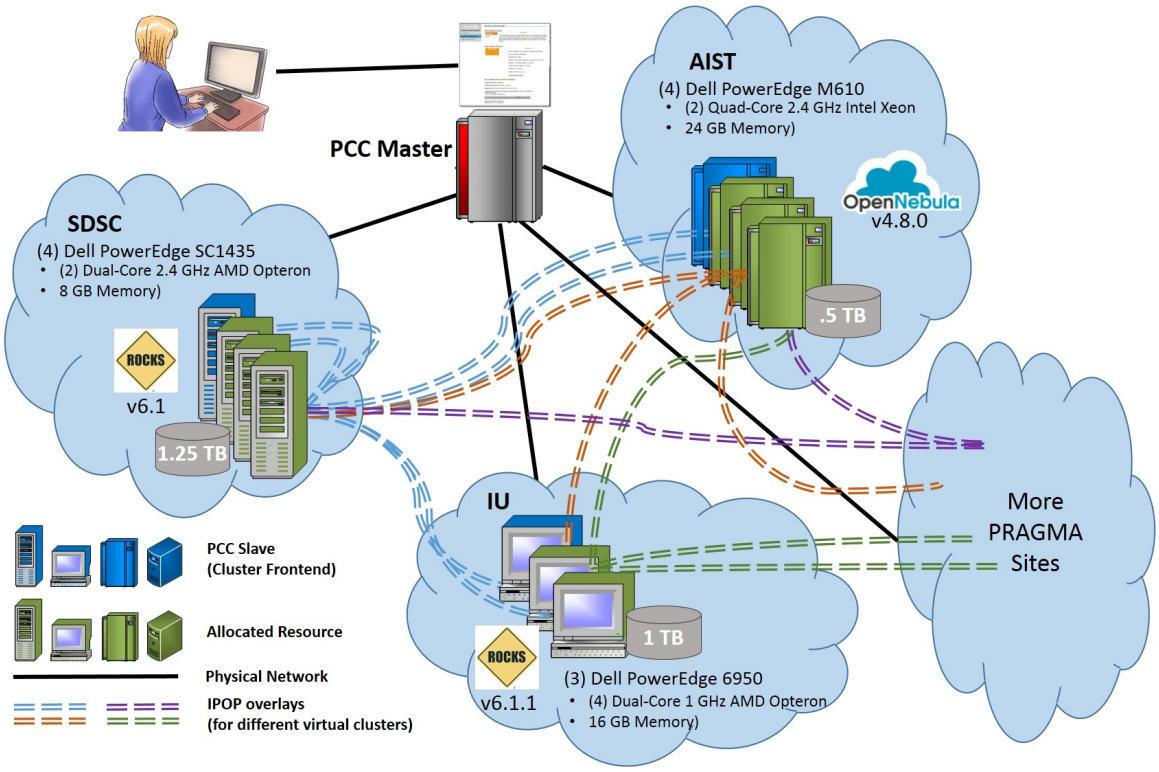

Figure 4.4: VCC Enabled PRAGMA Cloud

Figure 4.3 shows the architecture of VCC-HTCondor and the modified HTCondor daemon layout when a job submitted from Machine 1 is running. The blue daemons have been modified slightly but function just like the stock HTCondor daemons. The green module, VCC tools, refers to the wrapper script that then invokes PRAGMA Bootstrap and IPOP commands to build a virtual cluster.

The procedure of configuring IPOP to build the network overlay is part of the VCC tools that is invoked during the HTCondor job execution. Before PRAGMA Boostrap starts up a virtual cluster, the front end and compute VM images are mounted to the physical machine and the network is configured on the fly. We enhanced the PRAGMA Bootstrap code so

that the IPOP code, configuration files, and startup script are also installed then. Then when PRAGMA Bootstrap transfers and boots the images, IPOP will be started to create the network overlay among all VM instances.

Runtime Information Manager

The fourth component of VCC, runtime information manager (RIM), is implemented as a persistent service. The RIM builds and updates virtual cluster topology information, collects lineage of virtual machines that belongs to a virtual cluster, and makes this information accessible via web services interface.

4.1.3 PRAGMA Cloud

Fig. 4.4 gives a comprehensive picture of the PRAGMA Cloud powered by a VCC. The VCC controls VMs running on physical clusters from multiple organizations. The frontend nodes are marked in blue and are running a cloud management tool, e.g., Rocks, OpenNebula, and have a VCC slave and PRAGMA Bootstrap installed. The unclaimed resources and allocated resources are marked in gray and green respectively. The allocated resource are VMs that have been composed together as virtual clusters. All VMs in a virtual cluster are linked with a IPOP virtual network. Historical logs for the resource allocation procedures and runtime resource information for application scheduling are hosted at VCC master node.

4.2 Specifications of Cloud Resources, Applications and Data

The introduction posited that resource-application-data information sharing needed to be tridirectional to be most effective. The sharing depends on the kind of information known

to the virtual cluster controller about resources, applications, and data. Before we jump into pinned data, cloud resource-application-data cooperative optimization in the following chapters, we identify what information to share and how the information is shared.

4.2.1 Deployment vs Execution

Cloud computing becomes more valuable when the semi-automatic creation and management of application layer services is ported across different cloud implementation environments to achieve cloud interoperablity.

TOSCA [25] specification provides a language to describe service topology and orchestration processes. TOSCA includes topology and orchestration specification using node type and relationship type, and plan as workflow. Types define reusable entities and their properties, and templates form the cloud service's topology using these types, which are then instantiated as instances of the described cloud service. A topology template consists of a set of node templates and relationship templates that together define the topology model of a service as a directed graph. Plans defined in a service template describe the creation and termination of service instances.The specification relies on existing languages such as BPMN [24] or BPEL [23]. BPEL is a markup language for composing a set of discrete services into an end-to-end workflow. BPEL defines a model and a grammar for describing the behavior of a business process based on interactions among the process and its peers. To this end, TOSCA focuses on the deployment of cloud services and BPEL focuses on business execution. The union of TOSCA and BPEL covers the entire service deployment and execution space.

VCC vs BPEL:

VCC mainly focuses on virtual cluster deployment. The primary goal of VCC is to

deploy virtual clusters, and start/stop/suspend them. The service template (TOSCA plan) is a fixed template that embedded in the VCC Allocation Manager. Instead of using existing workflow specification language, such as BPEL, to describe the virtual cluster creation and termination, VCC coded the virtual cluster management process using C++ and python/perl scripts. Since the virtual cluster creation and termination workflows steps does not change frequently (if not at all), the coding/scripting solution is acceptable.

VCC vs TOSCA:

Inspired by TOSCAs node type specification, we add interface and properties as the attributes of the specifications. Interface in one node is defined so that another node type can access; properties are defined so that VCC can matchmaking resource, application, and data. To this end, our specifications become TOSCA-like specification.

4.2.2 Resource, Application and Data Specifications

Specification of the kinds of information collected for the resources, applications, and data are important considerations. Inspired by TOSCAs node type specification, we add "interface" and "properties" as attributes of specifications. "Interface" in one node is defined so that another node type can access; "properties" are defined so that VCC can matchmaking resource, application, and data. To this end, our specifications become subset of TOSCA specification, where TOSCA has extra "capability" and "requirements" associated with its node type, and relationship type. (The relationship in our framework is defined by applications.)

Resource Specification(RS). The resource specification, maintained and periodically updated by VCC, is described in Table 4.1.

Application Specification(AS). The application specification, provided by each appli-

cation, is described in Table 4.2.

Data Specification(DS). The data specification is described in Table 4.3.

Table 4.1: Resource Specification

Fields		Description
Interface		API of accessing resource (for VCC)
Properties	Cluster configuration	($RS.conf$) 1) number of nodes per cluster; 2) CPU speed; 3) number of cores per node; 4) memory size; 5) disk size; 6) operating system; 7) hypervisor; 8) cloud platform; and 9) inner-cloud network bandwidth.
	Inter-cluster network metrics	($RS.network$) contains end-to-end network bandwidth and latency values between the frontend nodes of every two clusters.
	Availability of each cluster	($RS.avail$) the remaining capacity of the cluster for resource allocation.

4.2.3 Runtime Information for Provisioned Resource

Provided to the application at runtime is information about the provisioned resource, including its topology and virtual machine provenance traces. In this section we only cover resource topology information as shown in Fig. 4.5 and including physical placement of each VM and network metrics.

Table 4.2: Application Specification

Fields		Description
Interface		API of accessing applications (for users/agents, not for VCC)
Properties	Resource requirements	($AS.req$) 1) number of nodes; 2) CPU speed; 3) number of cores per node; 4) memory size; 5) disk size; 6) hypervisor; 7) minimum network bandwidth.
	Application characteristics	The application characteristics ($AS.\alpha$) is instantiated by a vector, $\alpha = (\alpha_1, \alpha_2, \alpha_3, ...)$. The value of α is given by the application user, based on historical application execution experiments, where $\alpha_1(0 \leq \alpha_1 \leq 1)$ describes Compute Intensive; $\alpha_2(0 \leq \alpha_2 \leq 1)$ describes Data Intensive; $\alpha_3(0 \leq \alpha_3 \leq 1)$ describes I/O Intensive. A larger value in α indicates a higher intensity of that corresponding application characteristic and the most dominant characteristic of an application is the element with the largest value. The α vector can be further extended if more application characteristics are required.

Table 4.3: Data Specification

Fields		Description
Interface		API to access data (for applications, not for VCC)
Properties	Metadata (excludes locality)	Size, ownership, etc.
	Data Source	Data source ($DS.source$) is described by an array of of handlers of protocol and access point.
	Access constraints	Data access constraints ($DS.constraints$) are primarily used to ensure proper handling of licensing and/or security of sensitive data (e.g., locations of a rare species of plant) on restricted resources. The constraints are instantiated by allow/deny lists.

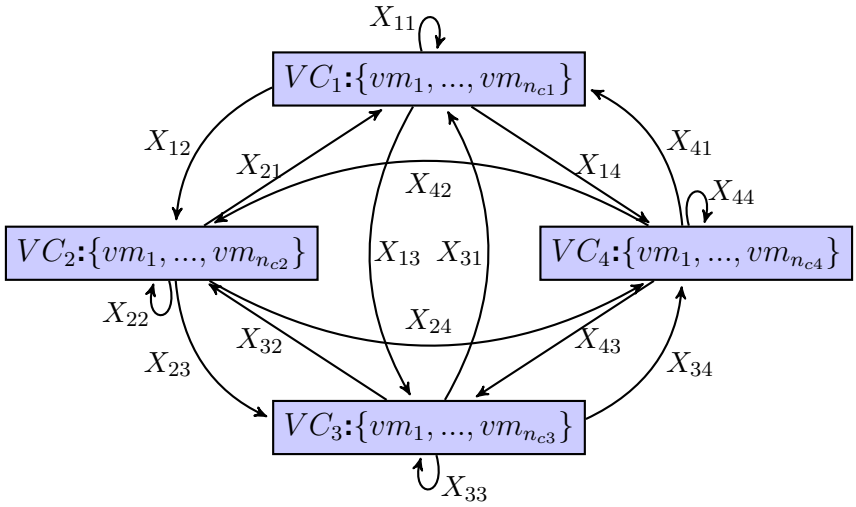

Figure 4.5: Runtime topology information of a virtual cluster. VC_k is a subset of a virtual cluster on the physical cluster k.X_{ij} represents network bandwidth (B_{ij}) and latency (L_{ij}) measured from VC_i to VC_j.

For the physical placement of each VM, the provisioned resource, known as a virtual cluster VC, consists of multiple virtual machines. These virtual machines are grouped as sub-VCs (VC_k) by the physical placement on different physical clusters, where k is the physical cluster ID. Each virtual machine in a virtual cluster is identified by its unique IP from VPN setup.

For the network metrics, $X = \{B, L\}$, where X_{ij} represents network bandwidth (B_{ij}) and latency (L_{ij}) measured from VC_i to VC_j.

4.3 VCC Resource Allocation: A Matchmaking Process

4.3.1 Application Aware Resource Allocation

To facilitate optimal resource allocation, the VCC needs to consider resource, application and data. A resource allocation module in VCC takes resource, application and data specifications as input and generates a resource allocation plan, shown in Fig. 4.6.

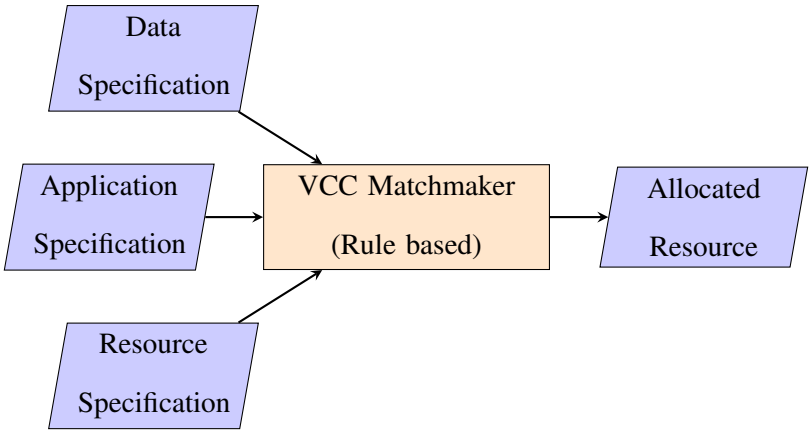

Figure 4.6: Resource Allocation Model

The default application-aware resource allocation (AARA) algorithm in VCC, given in

Algorithm 3, shows how resource, application and data specifications are utilized in the resource allocation process. For data-intensive or I/O intensive workloads, only applications that are sensitive to data locality and network bandwidth are considered. Network latency sensitive applications are not considered in this algorithm. First, $GetCandidateList$ loops through the set of candidate resources (RS) based on the resource requirements and application constrains (AS). The candidate resource list is then sorted based on the α value. If element values in α are equal, no list sorting is performed. However, if the dominant characteristic is compute intensive, the list will be sorted in descending order based on the computing power of each physical cluster using the $SortByCP$ procedure. Otherwise, if the dominant characteristic is data intensive, the resources listed in $DS.source$ are selected first, followed by the $AppendList$ procedure that adds resources with higher bandwidth to any of the resources in $DS.source$. If the dominant characteristic is I/O intensive, the resource is sorted with higher inbound or outbound network bandwidth. The allocation plan will then be executed using the $AllocationPlan$ procedure, which provisions resources from the weighted candidates according to the data distribution which is provided by $DS.source$.

4.3.2 VCC Matchmaker

VCC Matchmaker is a stand-alone service that continuously reads all the preferences and constraints and candidate entities, matches and sorts candidate entities that satisfy the preferences and constraints.

VCC Matchmaker is used in the VCC to dynamically determine the resource allocation plan for applications and data services. The matchmaker takes as input the preferences and constraints of resource specifications, application specifications and data specifications, as

Algorithm 3 Application-Aware Resource Allocation Algorithm

Require: Resource Specification RS, Application Specification AS, Data Specification DS

1: **procedure** RESOURCE-ALLOCATION(RS, AS, DS)
2: $CandidateList$=GETCANDIDATELIST(RS, AS, DS);
3: **if** $Max(\alpha) = \alpha_1$ **then**
4: $CandidateList$=SORTBYCP($CandidateList$);
5: **else if** $Max(\alpha) = \alpha_2$ **then**
6: $CandidateList$=$DS.source$;
7: $CandidateList$=APPENDLIST($CandidateList$, $RS.network$);
8: **else if** $Max(\alpha) = \alpha_3$ **then**
9: $CandidateList$=SORTBYBW($CandidateList$, $RS.network$);
10: **end if**
11: ALLOCATIONPLAN($CandidateList$,$DS.source$);
12: **end procedure**

well as rules to find the optimum resource allocation plan.

Architecture

The matchmaker is designed in client/server model, which communicates over a messaging bus or a web service. The Matchmaker server has two main components, see Figure 4.7.

- Matchmaker engine: The main functionality is to take JSON input, and output possible matches. The rule files and JSON schemas are flexible. A rule jar file consists of a rule file, schema related code that are automatically generated, and help functions.
- Matchmaker driver: It is configurable to either info mode, or broker mode. The driver takes input from clients, invokes matchmaker engine, and reformat output from matchmaker engine, and send results back to clients based on communication patterns.

These two components are independent of each other. New communication patterns can be implemented by only change the matchmaker driver.

Matchmaker Engine

The matchmaker engine leverages Drools rule engine to make matchmaking decisions. We also implemented a plugin mechanism that allows new rules and optionally associated helper java classes to be added. VCC Matchmaker has no hardcoded keywords of any kind, nor does it restrict to use a particular JSON schema. It generates on-the-fly POJO classes source code based on user input JSON files, compile to customized jar file along with user defined rules and associated helper java classes, and instantiate POJOs (based on JSON files) without a pre-defined schema. However, if a JSON schema is pre-defined, the matchmaker can generate POJO code and compile jar files offline, and only instantiate

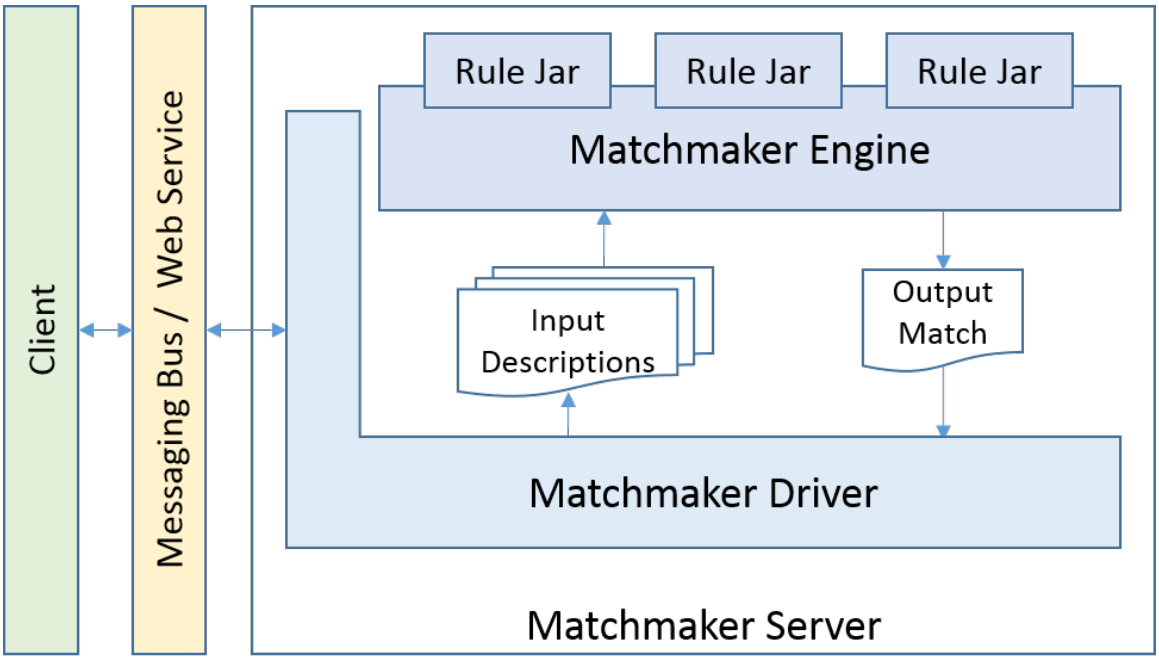

Figure 4.7: Matchmaker Architecture

POJO objects (based on JSON files) at runtime, making matchmaking process much faster. See matchmaker engine flowcharts in Figure 4.8.

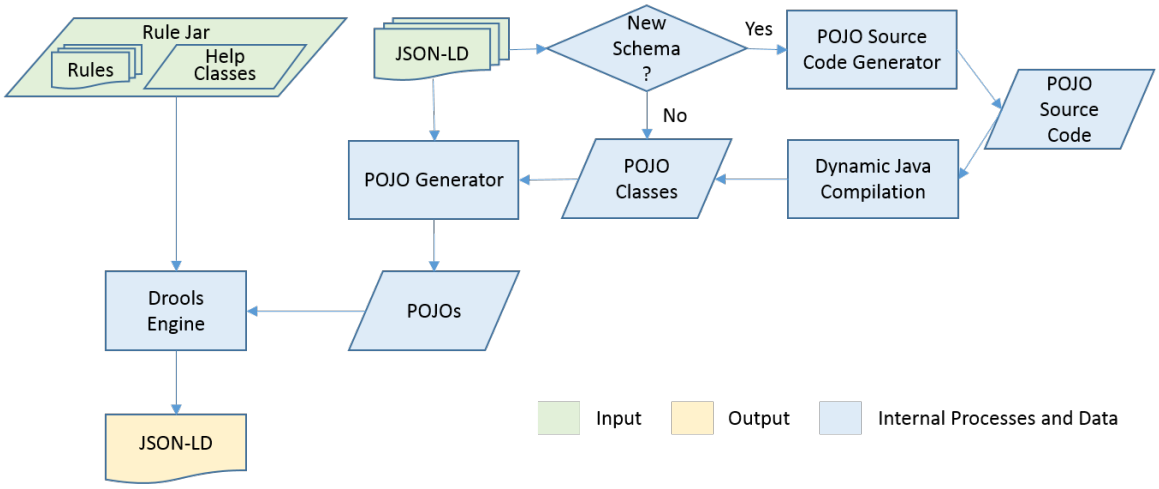

Figure 4.8: Matchmaker Engine flowchart

Matchmaker Driver

The matchmaker driver takes input from users and invokes the matchmaker engine, and reformat output from matchmaker engine, and send results to users based on different communication patterns.

Rule Jar

A rule jar consists of user defined rules and optionally associated helper methods. A matchmaker rule is essentially a Drools rule which is as simple as shown in Table 4.4

LHS, operates on JavaBean (POJO) objects, is the conditional parts of the rule, which follows a certain syntax. RHS is basically a block that allows dialect specific semantic code to be executed, including java code.

Table 4.4: A basic Drools rule

```
rule ”name”
        when
                Left Hand Side(LHS)
        then
                Right Hand Side(RHS)
end
```

The Drools rules adopt “when-then” logic. In matchmaker engine, each rule invokes one or more of the Java methods, shown in Figure 4.5, in the “then” statement to update the candidate list. The logic behind this rule invocation process is that the initial candidate list is always a full list. By applying rules, the candidate list will be updated to a subset of the full candidate list. Therefore, the order of rules will have no impact to the final result so that it ease the burden of rule creation/verification. New rules can be added independently, without looking back to the existing rules.

The helper methods in a rule jar file can optionally contain external queries that a JSON document may refer. The implementation of the queries vary based on different scenarios and they can be added later.

4.4 VCC Framework Evaluation

We evaluate 1) overhead of VCC as captured by overhead during the resource provisioning phase; 2) overhead of using VPN when running applications; and 3) performance comparison of four combinations of resource allocation algorithms and application scheduling algorithms, showing the benefit of resource-application-data information sharing. Experimental results are provided using the first release of VCC and a data-intensive HMR appli-

Table 4.5: RHS Java Methods that operate on candidate list

Method	Method Description
restrict	Restrict candidate list to a given list.
notAllowed	Remove selected candidates from the candidate list.
preferred	Tag ”preferred” to a list of candidates.
setWeight	Set weight to a candidate.
addWeight	Add weight to a candidate.
reduceWeight	Reduce weight to a candidate.

cation. In this evaluation we are not focusing on conveying the usefulness of multi-cloud nor delivering the comparison of virtual cluster deployment time consumption between single-cloud and multi-cloud.

4.4.1 Testbed

Table 4.6: Clusters Node Specifications.

Cluster	Nodes	CPU	Cores	Memory	Ethernet	OS	VMM	Cloud Platform
SDSC	4	AMD Opteron 2216 2.4GHz	4	8GB	BCM95721 1000Base-T	2.6.32-431.11.2.el6	KVM	Rocks 6.1.1
IU	4	AMD Opteron 8216 2.4GHz	8	16GB	BCM5708 1000Base-T	2.6.32-431.11.2.el6	KVM	Rocks 6.1.1

VCC is deployed to the PRAGMA Cloud, described in Section 4.1.3, and select two clusters: one at Indiana University (IU) and the other at the San Diego Supercomputer Center (SDSC). Both clusters run Rocks [20] version 6.1.1 with the KVM roll installed and PRAGMA Bootstrap to provision the VMs. The detailed specifications of the clusters are given in Table 4.6.

4.4.2 VCC Overhead Evaluation

Without VCC, VMs can be created directly with the individual cloud infrastructures. In addition to the VM image transfers and VM booting performed by PRAGMA Bootstrap, VCC has the additional steps of submitting jobs and calculating resource allocation plans in VCC-HTCondor, and setting up the IPOP network in PRAGMA Bootstrap. The time cost of performing these extra steps is considered the VCC overhead and is defined in equation 4.1, where T_α is the total execution time of creating a virtual cluster using VCC, while T_β is the total execution time of transferring and instantiating multiple VMs without using VCC.

$$Overhead(VCC) = T_\alpha - T_\beta \tag{4.1}$$

The $Overhead(VCC)$ is break down into two pieces, shown in equation 4.2, where $T_{VCC-HTCondor}$ is the time between submission of the virtual cluster job and the invocation of PRAGMA bootstrap and T_{IPOP} is the execution time of deploying and configure IPOP using PRAGMA Bootstrap. It also includes the Matchmaker overhead. Other steps in PRAGMA Bootstrap, including VM transferring and booting are also performed in non-VCC scenarios and therefore are not part of VCC overhead.

$$Overhead(VCC) = T_{VCC-HTCondor} + T_{IPOP} \tag{4.2}$$

A number of virtual clusters are created, ranging between 2 to 24 VMs from the two different physical clusters and measured the VCC overhead. The default resource allocation strategy is, without breaking the application constraints, to allocate as many VMs as possible from a single cluster, before allocating from another one. As shown in Fig. 4.9 and Fig. 4.10, there is stair-step effect when allocating more than 14 VMs. When creating virtual clusters with less than 14 VMs, the VMs are all allocated from the SDSC cluster which

is local to the VCC master node. When allocating more than 14 VMs, the dominant factor of the VCC-HTCondor overhead is invoking HTCondor jobs from the frontend node of the IU cluster.

There are the following two solutions to enable IPOP VMs:

- IPOP is installed and configured in an ad-hoc fashion. VCC downloads and copies over the IPOP package to the VM image and configures it. The IPOP ingestion and configuration overhead is shown in show in Fig. 4.10.
- IPOP is pre-installed inside the VM so only configuration is performed when the image is mounted to the local file systems. The IPOP configuration overhead is shown in Fig. 4.9.

The two IPOP enabling solutions have significant overhead differences since copying the IPOP package (1 MB) over the network is time-consuming. However, the total overhead introduced by VCC is less than 46 seconds in either IPOP enabling cases. This overhead is negligible in comparison to the VM image transfers performed by PRAGMA bootstrap, which can take tens of minutes to hours depending on the image size.

4.4.3 Network Overhead Evaluation

The network overhead refers to the bandwidth and latency overhead when IPOP's VPN is enabled. The next subsections discuss the tri-directional network bandwidth and round-trip time measurements between IU and SDSC without and with IPOP.

TCP/UDP Test without IPOP

The TCP/UDP bandwidth tests are performed using the Bandwidth Test Controller (BWCTL) [67] tool and the tests results are shown in Fig. 4.11.

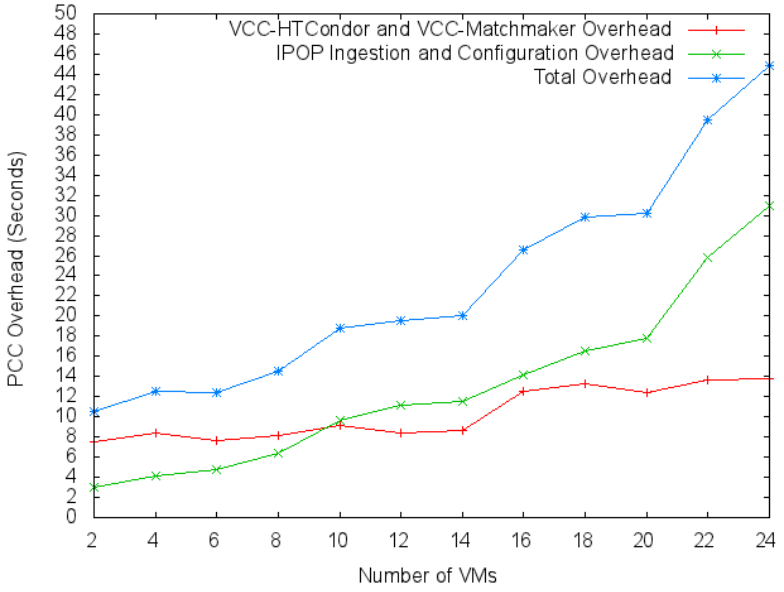

Figure 4.9: VCC overhead with ad-hoc IPOP installation

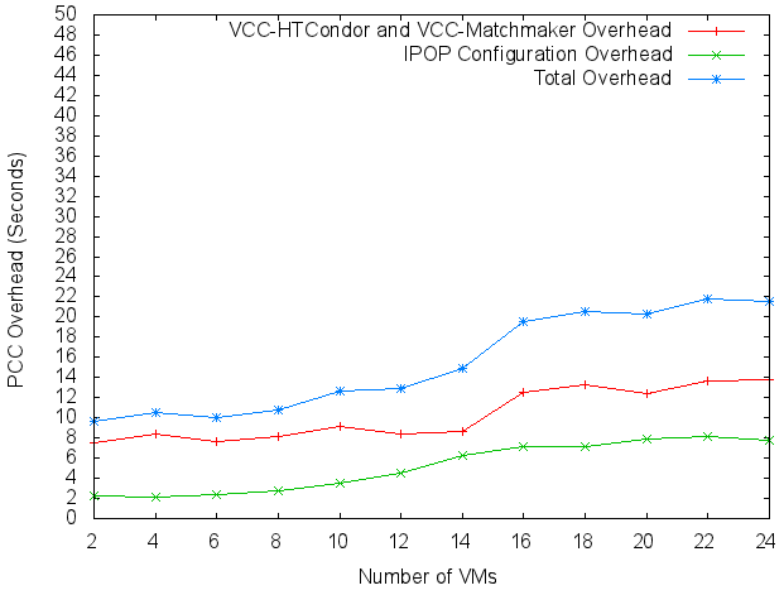

Figure 4.10: VCC overhead with IPOP pre-installed

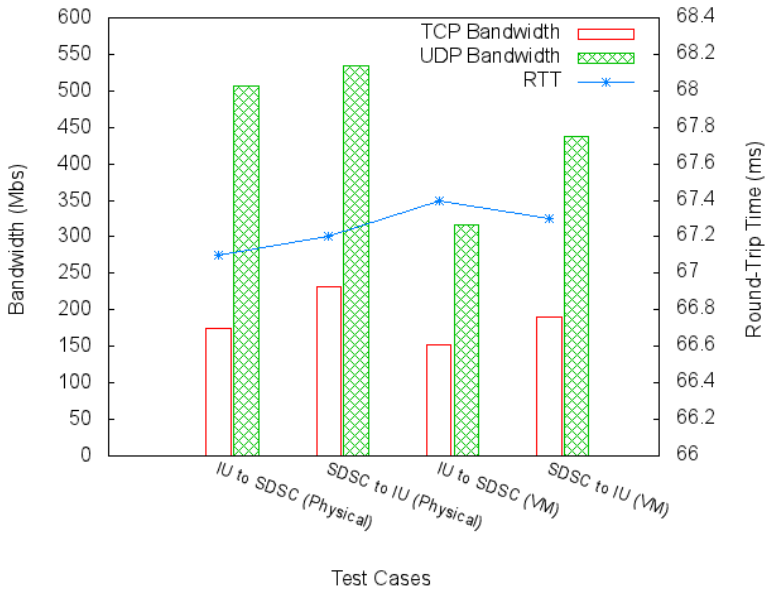

Figure 4.11: TCP/UDP Bandwidth and RTT Tests

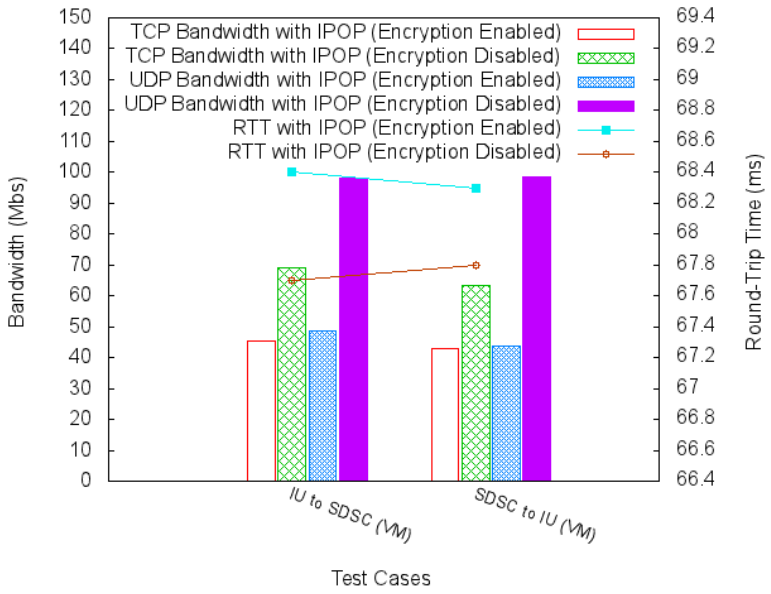

Figure 4.12: TCP/UDP Bandwidth and RTT Tests over IPOP

The TCP bandwidth from IU to SDSC on physical machines measured on average at 174Mb/s and 232Mb/s from SDSC to IU. The TCP bandwidth from IU to SDSC measured on VMs was on average at153Mb/s and 190Mb/s from SDSC to IU.

The UDP bandwidth from IU to SDSC on physical machines measured on average at 507Mb/s and 534Mb/s from SDSC to IU. The UDP bandwidth from IU to SDSC measured on VMs was on average at 317Mb/s, and 438Mb/s vice versa.

TCP/UDP Test with IPOP

IPOP encapsulates TCP packets over UDP. As shown in Fig. 4.12 when IPOP is enabled, the TCP bandwidth measured on VMs from IU to SDSC decreased to an average of 45.4Mb/s and 42.9Mb/s from SDSC to IU. The UDP bandwidth measured on VMs from IU to SDSC decreased to an average of 48.8Mb/s and 43.8Mb/s from SDSC to IU. This decrease is caused by 1) per-packet encapsulation overhead (14% overhead has been measured by [114]), 2) kernel/user and user/kernel crossing (memory copies, system call handling, and O/S scheduling policies), and 3) user-level processing. By default, IPOP encrypts messages. When disabling this encryption feature in IPOP, the TCP bandwidth measured from IU to SDSC increased to 69.1Mb/s and 63.4Mb/s from SDSC to IU. The UDP bandwidth measured on VMs from IU to SDSC increased to 98.1Mb/s and 98.3Mb/s from SDSC to IU.

RTT Test with/without IPOP

The round-trip times between IU and SDSC as shown in Fig. 4.11 and Fig. 4.12 were relatively stable: 67.1-67.4ms without IPOP, 68.3-68.4ms with encryption enabled IPOP, and 67.7-67.8ms with encryption enabled IPOP.

We consider the IPOP integration as proof-of-concept for building wide-area private networks over VCC-enabled environments. Although there is overhead of using IPOP, we still benefit from the minimum amount of configuration effort that IPOP introduces while sustaining these virtual private networks.

4.5 Information Sharing Evaluation

The effectiveness of resource-application-data information sharing can be shown through evaluating 4 combinations of one of resource allocation algorithms and one application scheduling algorithms listed as: 1) First Available Resource Allocation (FARA): allocate all virtual machines from the first available cloud providers; In this experiment, the SDSC resource is considered to be available first. 2) Application Aware Resource Allocation (AARA); 3) Compute Capacity Aware Scheduling (CCAS) [86] that was briefly introduced in Chapter 3; 4) Topology and Data Location Aware Scheduling (TDLAS): described in Chapter 3. Experiments are carried out for each of the combinations, where the resource-application-data information sharing characteristics are listed in Table 4.7.

Table 4.7: Resource-Application-Data Information Sharing Characteristics

	CCAS	TDLAS
FARA	No Sharing	Unidirectional Sharing (Resource-to-Application)
AARA	Unidirectional Sharing (Application-to-Resource)	Tridirectional Sharing

A HMR version of $grep$ is written as a data-intensive application. The Map function captures the matching lines of a regular expression input. The $Reduce$ function outputs all

Table 4.8: Compute Rate (CR) of a sub-virtual-cluster under different number of virtual machines

	1 VM	2 VMs	3 VMs	4 VMs
IU	3.6MB/s	5.3MB/s	6.7MB/s	7.9MB/s
SDSC	3.1MB/s	4.8MB/s	6.0MB/s	7.1MB/s

Table 4.9: Data Distribution Scenarios & Resource Allocation Results

Scenario	Data Distribution		VM Distribution (# of VMs)			
			FARA		AARA	
	IU	SDSC	IU	SDSC	IU	SDSC
1	0%	100%	0	4	0	4
2	25%	75%	0	4	1	3
3	50%	50%	0	4	2	2
4	75%	25%	0	4	3	1
5	100%	0%	0	4	4	0

the matching lines from each sub-VC MapReduce execution. The GlobalReduce function collects all the output from all sub-VC MapReduce execution and combines them into a single output file. The total execution time varies with respect to different data distribution and different regular expression input. Delay code is added to the Map function so that the Map tasks run significantly longer than loading data and initiating the MapReduce job. Therefore a negative correlation can be guaranteed between the number of nodes and the job execution time.

In order to get the compute rate CR_i of each sub-VC, VC_i, I allocated one, two, and three virtual machines on each of the physical cluster respectively and ran the HMR $grep$ application against 1-5 GB dataset multiple times throughout a week-long period. Each virtual machine was configured with 1 core of CPU, 1 GB memory, and 80 GB of disk space. For this specific application, the CR_i values listed in Table 4.8 are affected by two factors:1) loading data to HDFS, and 2) MapReduce execution.

In the experiments, the input data set contains 100 text files, each of which is 100MB large. The dataset is placed to IU and SDSC physical clusters using different data distribution scenarios, and allocate virtual machines from IU and SDSC clusters using FARA and AARA algorithms, shown in Table 4.9. To take advantage of the CR values, the same virtual machine configuration is applied. Data transfer is carried out through $rsync$. Transferring data within the same institute (disk-to-disk data transfer over LAN) has average speed of 35MB/s, which is significantly faster than transferring data between nodes that belong to different institutes (disk-to-disk transfer over WAN).

Leaving the LAN data transfer speed as it is, but manipulating the WAN data transfer speed, Fig. 4.13, 4.14 and 4.15 show the job execution time under WAN data transfer speed of 3MB/s, 5MB/s and 7MB/s respectively. The tests in each figure use 5 data distri-

bution scenarios described in Table 4.9. The tri-directional information sharing approach, implemented in AARA+TDLAS combination, achieved shortest job execution time. The runner-up combination, AARA+CCAS, which uses application aware resource allocation but without data locality consideration at runtime, performance slightly slower than the tri-directional information sharing approach. Other two solutions that did not consider application specification (data locality) during resource allocation phase (early phase), perform much worse than the ones that optimize resource allocation.

While data distributions could affect the performance of these algorithms combinations, AARA+TDLAS shows relatively stable performance in the tests. The tri-directional information sharing approach performs much better in comparison to other algorithm combinations, using different data transfer speed ratio of LAN over WAN. Fig. 4.16 shows the average speedup of tri-directional information sharing approach (AARA+TDLAS) over other approaches. The WAN data transfer speed is set to 7MB/s, 5MB/s and 3MB/s respectively so that the data transfer speed ratio (R) of LAN over WAN, as indicated in x-axis, are valued 5, 7, and 11.7. As R (network heterogeneity) increases, the speedup values remain above 1.

4.6 Summary

VCC creates virtual clusters by allocating virtual machines from multiple cloud platforms, builds a network overlay, and manages the virtual cluster life-cycle in a fault-tolerant manner. The resource allocation strategy is optimized by taking into consideration resource specifications and application requirements and characteristics; an application scheduling algorithm is developed that takes into consideration the resource topology and data location. VCC extends HTCondor and integrates IPOP. Evaluation of the overhead of VCC dur-

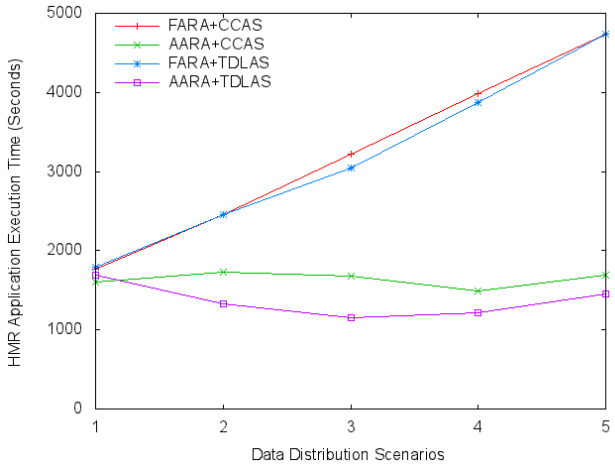

Figure 4.13: Data-Intensive HMR application execution time using 4 algorithms combinations under different data distribution. The data transfer speed between IU and SDSC is 3MB/s.

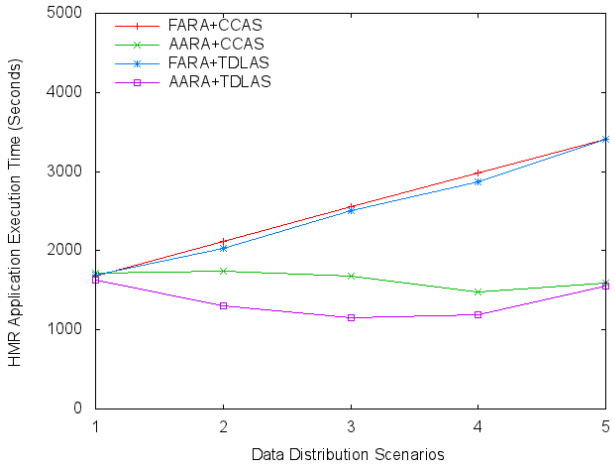

Figure 4.14: Data-Intensive HMR application execution time using 4 algorithms combinations under different data distribution. The data transfer speed between IU and SDSC is 5MB/s.

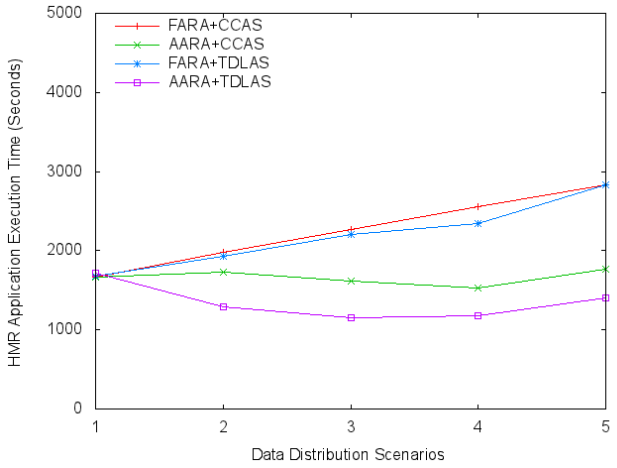

Figure 4.15: Data-Intensive HMR application execution time using 4 algorithms combinations under different data distribution. The data transfer speed between IU and SDSC is 7MB/s.

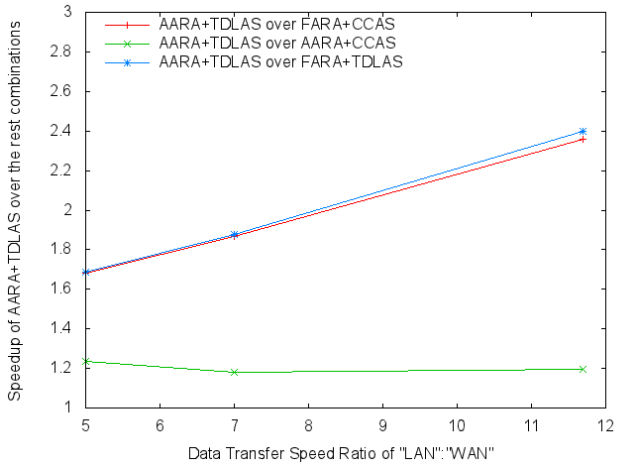

Figure 4.16: Speedup of AARA+TDLAS algorithms combination over other algorithms combinations. The x-axis indicates the data transfer speed ratio of LAN over WAN.

ing cluster startup and overhead of IPOP when running applications shows reasonable latencies.The evaluations also show that tridirectional information sharing among resources, applications, and data shortens the application execution time significantly.

The VCC controller strengths are in interoperability, scalability, and applicability.With respect to *interoperability*, VCC enhances cloud interoperability by federating cloud resources through a unified layer, PRAGMA Bootstrap. The VPN setup also adds value to enhance the interoperability among virtual machines. With respect to *scalability*, to each cloud provider, only one HTCondor compute node (VCC slave) needs to be deployed. The resource capacity of a VCC-enabled Cloud is not limited by VCC, but all the underlying cloud providers that provide resources. However, the virtual machine states are managed by VCC through HTCondor job management. HTCondor is reported to handle pools of tens of thousands of execution slots and job queues of hundreds of thousands of jobs. Finally, with respect to *applicability*, although the implementation of VCC is specialized for PRAGMA community that heavily uses Rocks and OpenNebula, the design principle can be adopted to tools that are used by larger community. The PRAGMA Bootstrap can be replaced by tools such as Libcloud, jClouds, DeltaCloud, JumpGate, and ManageIQ, with extensions to set up network overlays. VCC-HTCondor can also be replaced by other open source CMPs as a platform to develop the information sharing approach.

Chapter 5

Manage, Access, and Use of Pinned Data

In this chapter, we first define a novel data model, "pinned data" that, describes non-consumptive constraint of the data that raw data cannot leave a political jurisdiction. The "pinned data" design is inspired by Object Oriented Programming that data is encapsulated and accessed by method calls. A package of accessible pinned data is called a suitcase, with certain data properties and operations that the suitcase can perform. Encapsulation combines data and behavior in one suitcase and hides the implementation of the data from the user of the suitcase. The data in a suitcase are called its content fields, and the functions and procedures that operate on the data are called its methods. A suitcase does not expose data for public access directly. All communications are via method calls. With that being said, data is treated as a first class citizen where it negotiates with resource and application during resource allocation and application scheduling phases.

We enhance HMR framework to HMR+ to support PND processing. In HMR+, different Map/Reduce functions can be encapsulated in different pinned data suitcases at different clouds, before the GlobalReduce function aggregates data to produce final results. Therefore, a pinned data suitcase provides "Data-Processing-as-a-Service" capability.

We evaluated 1) PDS creation overhead for both internal/external data model; 2) the

system performance under increasing workloads for PDS method access; 3) HMR+ performance using two test applications, AutoDock HMR and Grep, under different percentages of pinned data.

5.1 Introduction

A shortcoming of the current proof-of-principle infrastructure is the lack of support for controlled sharing of data in its variety of forms (e.g. flat files, databases, sensor data, and others). Owners of data sets also have a range of concerns that include proper attribution [103], limited distribution of raw data prior to analysis [12], and legal requirements to keep data within administrative boundaries. The challenge is to strike the right balance between ensuring that the data processing carried out does not violate non-consumptive use while keeping the data management services as flexible as possible by not overly limiting the kinds of use. Therefore, we define a novel data model, notibly "pinned data" (defined in Definition 5.1) that describes the non-consumptive constraint of the data that raw data cannot leave a political jurisdiction. Data and its processing are given equal importance in the pinned data model.

5.2 Pinned Data Model

5.2.1 Pinned Data (PND)

Definition 5.1. *Pinned data (PND) is a data model describing non-consumptive constraint that data for social/political/legal reasons cannot leave a political jurisdiction. Data itself exists in varies forms, e.g., files systems, RDBMs, noSQL, etc, depending on different application scenarios.*

We define three (3) states, shown in Definition 5.2, where data can be. In state Rest, non-consumptive constraint is not violated. In state Movement, the non-consumptive constraint could be violdated depending on the destination of the data movement. In state Computatoin, the non-consumptive contraint could be preserved if the output data does not violate any non-consumptive constraint so that it can be moved freely.

Definition 5.2. *Data States:*

- *Rest: data is stored without being moved and computed.*
- *Movement: Data is in the process of being copied or moved.*
- *Computation: Data is being consumed by a process.*

The handling of "pinned data" is inspired by Object Oriented Programming that data is encapsulated and accessed by method calls. Modeling pinned data by using object-oriented techniques offers to use the power of data encapsulation. We use an object oriented approach to represent different pinned data types and encapsulate the details of data processing and representation.

5.2.2 Pinned Data Suitcase (PDS)

Definition 5.3. *A pinned data suitcase is a data capsule that consists of the combination of data (or data access point), data properties, and data processing logic. The data in a suitcase is called its content, and the functions and procedures that operate on the data are called its methods.*

A package of accessible pinned data is called a suitcase, defined in Definition 5.3, with certain data properties and operations that the suitcase can perform. Encapsulation combines data and behavior in one suitcase and hides the implementation of the data from the

user of the suitcase. The data in a suitcase is called its content, and the functions and procedures that operate on the data are called its methods. Access the pinned data and the pinned data processing logic is provided by well-defined interfaces. A suitcase does not expose data for public access directly. All communications are via method calls. Whenever a method is applied to a suitcase, provenance information is recorded. With that being said, data is treated as a first class citizen where it negotiates with resource and application during resource allocation and application scheduling phases.

Pinned Data Suitcase Specification

For each pinned data suitcase (PDS), there exists a specification to describe metadata, processing logic, and PDS restrictions, with respect to non-consumptive. A pinned data suitcase specification, shown in Table 5.1, inherits the general data specification described in Chapter 4.2. A PDS specification consists of multiple methods and properties. There are multiple properties associated with the pinned data itself, such as size, ownership, data source, and access contraints. Each method has a name, description, ACL, and a handler. The reason to have separate ACLs for different methods is that not all methods satisfy non-consumptive contraints.

Pinned Data Suitcase Management

A PDS is packaged into a virtual machine or a virtual cluster. At minimum, a PDS has a specification, and methods that are implemented into executables, eg. jar files if using Java. The methods are accessible by users/applications outside the PND site. When we introduce data source in the specification, we did not mention if the data was actually co-located with the methods. We do not mandate a PDS to have actual data. A handler can point data to

Table 5.1: Pinned Data Suitcase Specification

Fields		Description
Interface (Methods)*	Name	Name of Method i.
	Description	Description of the method.
	ACL	Access control: allow IP list, and deny IP list.
	Handler	Access point of this method.
Properties	Metadata	Size, ownership, etc.
	Data Source	Data source is described by a handler that consists of protocol and access point.
	Access constraints	Data access constraints are primarily used to ensure proper handling of licensing and/or security of sensitive data (e.g., locations of a rare species of plant) on restricted resources. The constraints are instantiated by allow/deny lists.

* The interface field contains multiple methods, each of which has attributes of name, description, ACL, and handler.

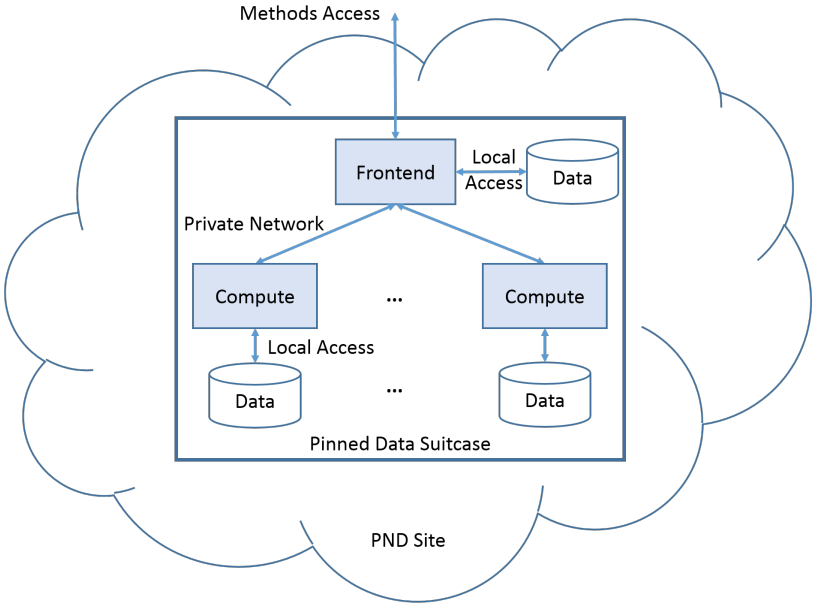

Figure 5.1: Pinned Data Suitcase with Local Data

a external location, provided the link between the PDS and the data source is secure. This leads to two (2) configurations of a PDS, introduced below:

- PDS with data: Data is bundled into a PDS. PDS is running at a PND site so that no data is been transferred. See Figure 5.1.
- PDS without data: Data is remotely accessible by PDS methods. The link between the PDS and the data source is restricted within the PND site. See Figure 5.2.

PDS has two cluster types: transient and persistent. Each can be useful, depending on the task and system configuration.

- Transient PDS: Transient PDS are clusters that shut down when the job or the steps (series of jobs) are complete. In contrast, a persistent PDS continues to run after data processing is complete. If a PDS cluster will be idle for the majority of the time, it is best to use transient clusters. For example, if you have a batch-processing job that

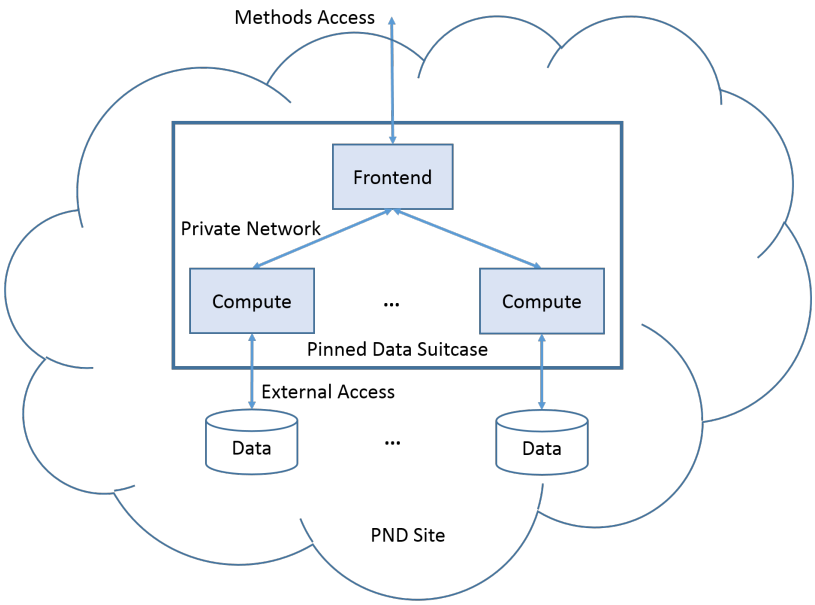

Figure 5.2: Pinned Data Suitcase with External Data

pulls data from pinned data storage and processes the data once a day, it is more cost effective to use transient clusters to process data and shut down the nodes upon job completion. In summary, consider transient clusters in one or more of the following situations:

1. The total number of hours to PDS creation and data processing per day is less than 24 so that users benefit from shutting down PDS cluster when it is not being used.
2. Data is not bundled into a PDS.

- Persistent PDS: As the name implies, a persistent PDS continues to run after the data processing job is complete. Similar to transient PDS, a persistent PDS has its own cost and benefits. Consider persistent PDS for one or more of the following situations:

  1. Pinned data is frequently processed at a PDS where it is beneficial to keep the

cluster running after the previous job.

2. Pinned data processing tasks have an input-output dependency on one another. Although it is possible to share data between two independent PDS, it may be beneficial to store the result of the previous job on a PDS's internal storage for the next task to process.

3. In some cases output data needs to be preserved for a while before sending it out to clients.

Both transient and persistent PDSs share a foundamental root that a template or an instance of PDS never leaves its PND site, and VCC has no direct access to them. However, the control level of virtual cluster specifications differs.

- VCC controlled PDS provisioning: Per VCC request, PND creates a virtual cluster as a PDS that is deployed at PND site. The virtual cluster templated is created by the PND site. VCC controls the resource requirement.
- PND site controlled PDS provisioning: Pinned data as a service. A virtual cluster is created at a PND site that serves as a data service cluster. The virtual cluster template is created by the PND site. Virtual cluster scales up and down independently of VCC.

PDS is managed by a PND service running at a PND site. The procedure to create and run a PDS, shown in Figure 5.3, is described as follows:

1. Create a virtual cluster (VC) based on resource requirements provided by either the VCC or a PND site.

2. If PDS is configured with data bundled in the virtual cluster, import data from external storage to local file systems.

3. Deploy methods. Methods are typically built into jar files.

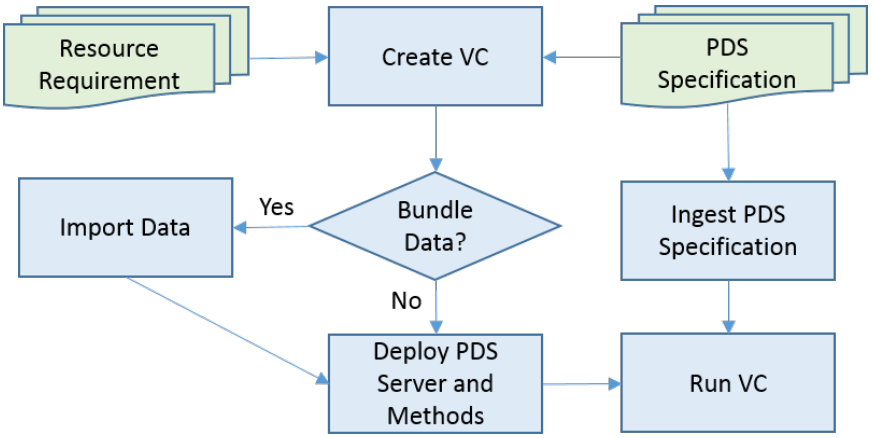

Figure 5.3: Create a Pinned Data Suitcase (PDS) and run it.

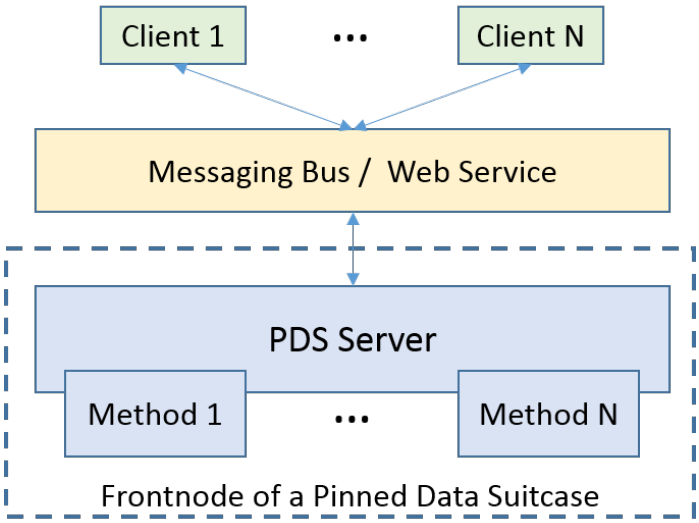

Figure 5.4: Access Pinned Data Suitcase (PDS) via Messaging Bus and Web Service.

4. Ingest PDS specifications into the virtual cluster.

5. Run the VC, which automatically starts a server listening for data access requests.

5.2.3 Access Pinned Data

Access pinned data and the pinned data processing logic is provided by well-defined interfaces. A suitcase does not expose data for public access directly. All communications are via method calls. Therefore, we create services that takes remote data access request and invoke PDS methods. The methods are written in a way that the output of each method does not violate non-consumptive constraints. The methods are authored and examed by personnel at the PND site. Figure 5.4 shows the communication methods between clients and a pinned data suitcase. In the frontnode of a PDS cluster, there exists a PDS server that takes remote call requests via either a messaging bus or a web service interface and invokes corresponding methods. A PDS method could be a MapReduce application, which will be further described in the next section. A PDS server handles all in-and-out PDS traffic.

5.3 Pinned Data Processing with HMR+

5.3.1 HMR+ Architecture

When the input dataset is pinned, or partially pinned, the local Map and $Reduce$ functions can only be executed on the pinned site with respect to the pinned portion of data, provided additional restriction that local $Reduce$ output does not violate the non-consumptive constraints. This is achieved by encapsulate local Map and $Reduce$ functions as PDS methods. A slightly modification to the original HMR framework is needed to allow data to feed from PDS methods (local Map / $Reduce$) output to $GlobalReduce$. We discussed in Section 5.2.3 that PDS methods are exposed by remote calls.

An enhanced HMR (HMR+) architecture diagram is shown in Figure 5.5. In comparison with the regular HMR architecture shown in Figure 3.1, HMR+ add a new PDS daemon

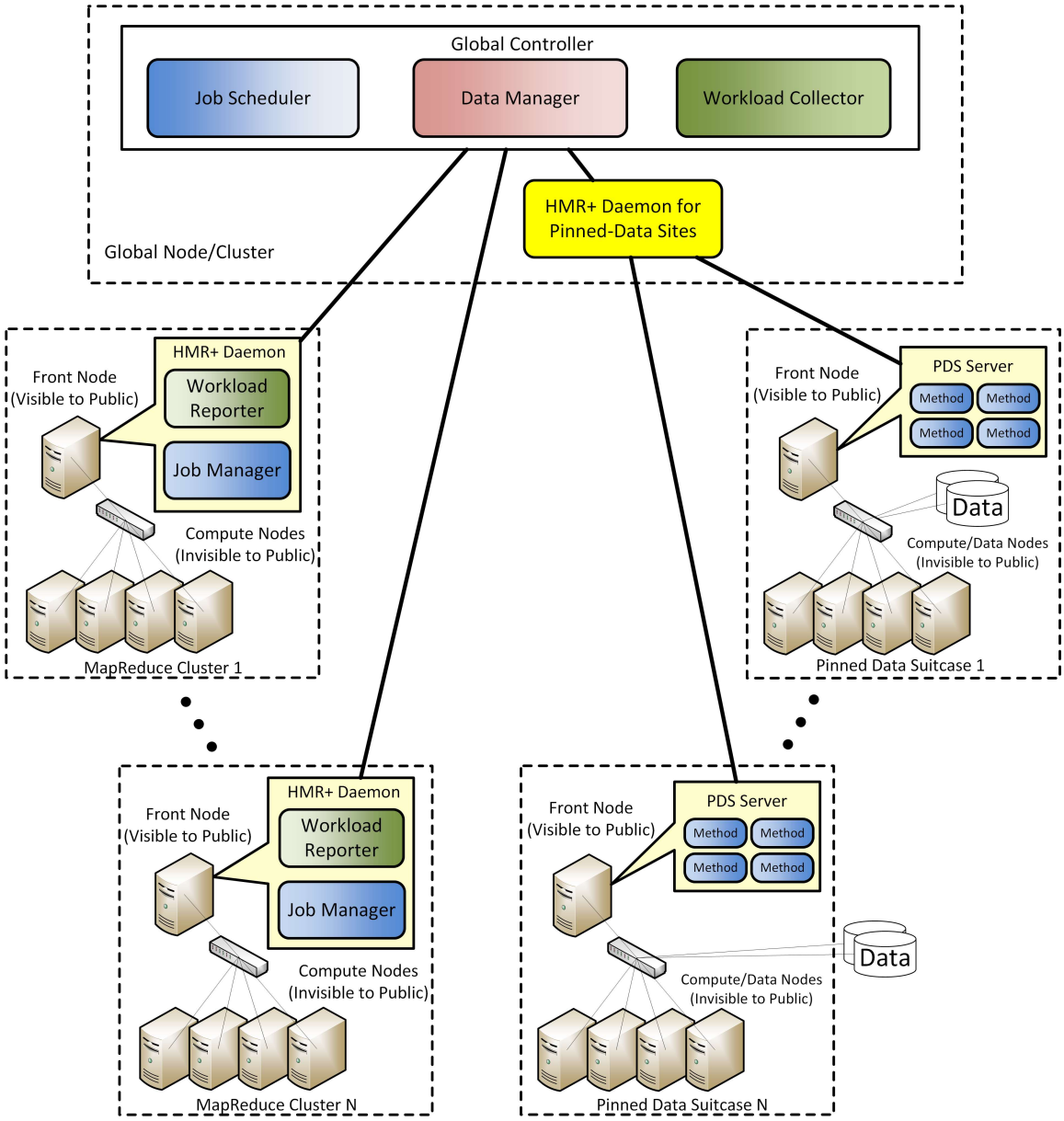

Figure 5.5: Enhanced Hierarchical MapReduce Architecture with Pinned Data Support

to communicate with pinned data suitcases (PDSs). PDSs bundled in MapReduce clusters that run local Map and $Reduce$ tasks but only to expose local MapReduce output via PDS service API. The PDS daemon is co-located with global controller. The reason to keep PDS daemons away from PDSs is that PDSs can only be accessible via remote calls that solely serve for PDS methods. The PDS daemon invokes PDS methods via remote calls and the output is collected by data manager in HMR global controller.

The input dataset consists of two data types: pinned data and unpinned data. When a user submits a MapReduce job to the global controller, the job scheduler firstly dispatches pinned data processing tasks to PDSs, where each PDS provides estimated processing time based on historical data. The job scheduler then splits the unpinned data processing job into a number of sub-jobs and assigns each to a local cluster based on several factors, including but not limit to the current workload reported by the workload reporter from each local cluster, the capability of individual nodes making up each cluster. This is done to achieve load-balance by ensuring that all clusters will finish their portion of the job to process unpinned data in approximately the same time, if not less than the maximum execution time of pinned data processing on each PDS. The global controller also partitions the movable data if the input data have not been deployed before-hand. The data manager transfer the user supplied MapReduce jar and job configuration files with the input data partitions to the MapReduce clusters for processing unpinned data. As soon as the data transfer finishes for a particular cluster, the HMR daemon of that cluster to start the local MapReduce job. The jobs on PDSs are controlled by a special HMR daemon that deployed at global controller node. After the local sub-jobs are finished on a local cluster, if the application requires, the clusters will transfer the output to one of the clusters for global reduction. Upon receiving all the output data from all local clusters, a GlobalReduce will be invoked to perform the

final reduction task, unless the original job is map-only.

5.3.2 Pinned-Data Applications

The distinction between pinned-data applications and regular compute-intensive or data-intensive applications are not in applications themselves, but the datasets characteristics that applications use. For instance, HathiTrust has 11 million volumes of digital publications. Processing 1 million volumes/books (2 TB large) using n-grams algorithm (implemented in MapReduce) on 1024 cores takes about 22 hours, including loading data from Lustre clusters to the compute resource for processing. The processing requires raw data to be moved from protected sources to a centralized cluster and pinned to that centralized cluster. With that being said, the compute-intensive AutoDock application and data-intensive Grep application can be categorized as pinned-data application, given that partial or entire input dataset is immovable. We reuse these two applications and pack them into a sample pinned-data suitcase.

5.4 Pinned Data Evaluation

To understand the performance of the PDS, we investigated the following three aspects.

- PDS creation overhead for both internal/external data model discussed in Figure 5.1 and Figure 5.2. The PDS creation is an onetime operation and therefore its overhead negligible if PDS is configured as "persistent". However, if the PDS is configured as "transient", the overhead of PDS creation is considerably important.
- PDS runtime performance under increasing workloads for PDS method access. The PDS framework is independent of its application, where PDS methods can be deployed by pinned data owners. We capture the overhead of running PDS server over

direct data processing methods.

- HMR+ performance using two test applications: AutoDock HMR, and Grep, under different percentage of pinned data. We test these two application only to demonstrate the pinned data processing over HMR+ in general. Other applications such as HTRC has similar characteristics, of which the actual performance behaviors can be learnt from these sample applications.

5.4.1 Experimental Setup

We wrote all our code in Java, using J2SE version 7. Both experiment use the PDS verson 1.0. For PDS server, the default value for the maximum number of threads allowed to invoke PDS methods is set to 20. We use HDFS for data storage both in internal PDS storage and external data source.

We selected three physical clusters in the PRAGMA Cloud for all the experiments in this Chapter: one at Indiana University (IU), one at the San Diego Supercomputer Center (SDSC), and another one at Chinese Academy of sciences' Computer Network Information Center (CNIC),. The detailed specifications of the clusters are given in Table 5.2.

Table 5.2: PDS Testbed: Clusters Node Specifications.

Cluster	Nodes	CPU	Cores	Memory	Ethernet	OS	VMM	Cloud Platform
SDSC	4	AMD Opteron 2216 2.4GHz	4	8GB	BCM95721 1000Base-T	2.6.32-431.11.2.el6	KVM	Rocks 6.1.1
IU	4	AMD Opteron 8216 2.4GHz	8	16GB	BCM5708 1000Base-T	2.6.32-431.11.2.el6	KVM	Rocks 6.1.1
CNIC	1	Intel Xeon E5620 2.4GHz	16	96GB	Broadcom NetXtremeII 5709	2.6.18-238.19.1.el5	XEN	Rocks 5.4.3

5.4.2 PDS Creation Evaluation

PDS is a specialized virtual cluster in which the frontnode is a PDS Server that responsible for accepting incoming data processing requests and invoke corresponding PDS methods. In addition to conventional virtual cluster provisioning, PDS creation has extra steps of 1) setting up a PDS Server; 2) deploying PDS methods; and 3) optionally ingest dataset. The time cost of performing these extra steps is considered as PDS creation overhead and is defined in equation 5.1, where T_α is the total execution time of creating a PDS instance, while T_β is the total time of creating a conventional virtual cluster that forms the PDS. In this experiment, we assume that the conventional virtual cluster is a MapReduce cluster which has Hadoop framework pre-installed.

$$Overhead_{PDS-Creation} = T_\alpha - T_\beta \tag{5.1}$$

The $Overhead(PDS\_Creation)$ breaks down into two pieces, if data is not bundled, shown in equation 5.2, where T_{PDS_Server} is the execution time of deploying and configure PDS server, and T_{PDS_Server} is the execution time of deploying and configure PDS methods.

$$Overhead_{PDS-Creation} = T_{Deploy-PDS-Server} + T_{Deploy-Methods} \tag{5.2}$$

The $Overhead(PDS\_Creation)$ can also break down into three pieces, if data is to be bundled, shown in equation 5.3, where T_{PDS_Server} is the execution time of deploying and configure PDS server, T_{PDS_Server} is the execution time of deploying and configure PDS methods, and $T_{Bundle-Data}$ is the execution time of transferring data from external source into the file system of PDS.

$$Overhead_{PDS-Creation} = T_{Deploy-PDS-Server} + T_{Deploy-Methods} + T_{Bundle-Data} \tag{5.3}$$

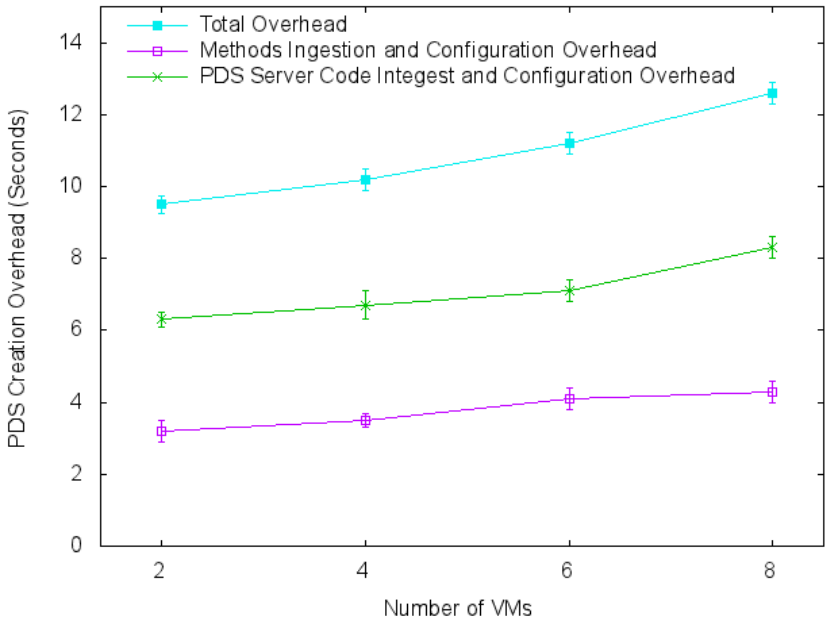

Figure 5.6: PDS Creation Overhead (without data bundle).

The PDS server code is 2.3MB large. The PDS method tested in this experiment is an MapReduce application, grep, which is 1.6MB large. Figure 5.6 shows the overhead of creating PDS cluster when the number of nodes increases from 2 to 8. The total overhead of PDS server and PDS method ingest and configuration is approximately 9.6 to 12.6 seconds.

Then we keep the virtual cluster size at 4, but transfer data from external source to the HDFS in the virtual cluster. We tested the data size from 0GB to 10GB with 1GB intervals. Figure 5.7 shows that when the data size increases, the bundle time also increases. The bundle time becomes the dominate factor in the total overhead, as is described by the percentage in the total overhead.

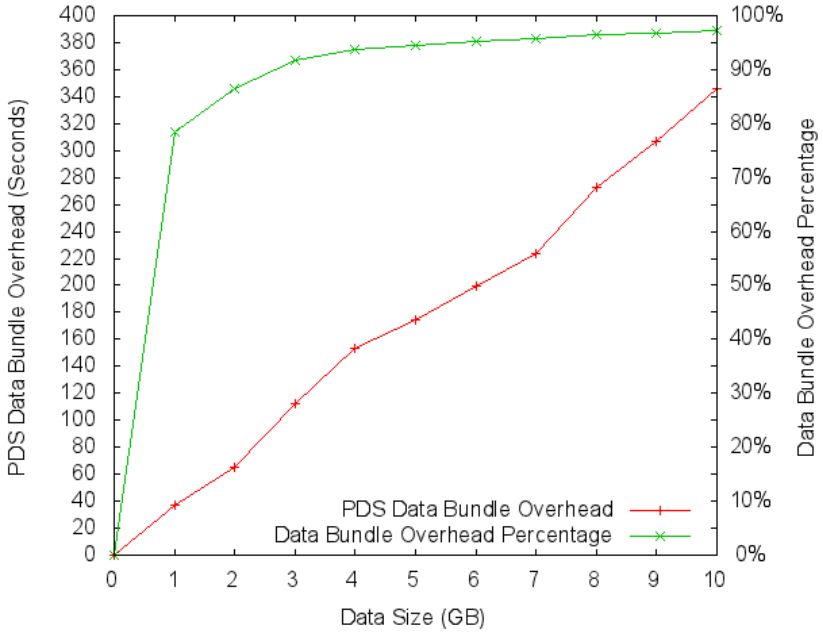

Figure 5.7: PDS Data Bundle Overhead.

5.4.3 PDS Server Evaluation

To evaluate PDS server in Figure 5.4, we designed two experiments (Experiment-1 and Experiment-2) that test PDS server with two communication approaches that PDS supports: 1) a publish-subscribe messaging system; and 2) a web service based system.

In Experiment-1, we use RabbitMQ server version RabbitMQ 3.5.1 and Erlang version R16B03-1 as the messaging setup. In Experiment-2, we use Apache Tomcat version 8.0.24 and Apache Axis2 version 1.6.3 as web service container. In Experiment-1, we run RabbitMQ server on a virtual machine at IU, which is configured with 2 core of CPU, 4 GB RAM, and 80 GB of disk space. In either Experiment-1 and Experiment-2, the PDS virtual cluster frontnode, where PDS server is located, is configured with 2 core of CPU, 4 GB RAM, and 80 GB of disk space.

We conduct this evaluation when the PDS methods call is communicating via RabbitMQ. We place the client at IU, and PDS cluster at IU, SDSC and CNIC respectively. We measure the overhead of PDS runtime and show this under increasing workloads.

We investigate how well the PDS server performs when increasing the data processing request rate. While the request rate can be arbitrarily manipulated, the requests will be queued in the messaging bus, waiting for the PDS server to pick up. Due to the resource capacity of a PDS cluster, the PDS server will not pick up request from the queue instantly. The PDS server will wait for PDS to free resources that has been used for previous data processing jobs. Therefore, we exclude the time that a jobs spends in the queue, as the queue time has nothing to do with the PDS overhead but the size of the cluster. The overhead of PDS runtime is formalized in Equation 5.4.

$$Overhead_{PDS-Runtime} = T_r - T_q \tag{5.4}$$

where T_r is round-trip time between a client and PDS, which excludes the job execution time; and T_q is the time that a job spends in the queue. We created a PDS method that has small execution time (around 500 millisecond), and configure PDS server thread to 40. The results is shown in Figure 5.8. As the job request rate increases, the time of roundtrip between a client and PDS server only increases slightly when the request rate is below 100 request per second, due to the multi-threading feature and short job execution feature in PDS; when the rate exceeds 100, the roundtrip time increase dramatically because the PDS server consumes requests from the job queue at a significant low rate due to the limit of computation resource in a PDS cluster. The time a job spends in the queue also increases because the size of the queue increases as the request rate increases. Therefore, the PDS overhead is relatively small, which is under still under 2 seconds when the request rate hits

200 requests per second. The PDS overhead is small even under the condition of large average job execution time, because PDS server will reach a stable low dequeue rate to keep the PDS cluster from overloading.

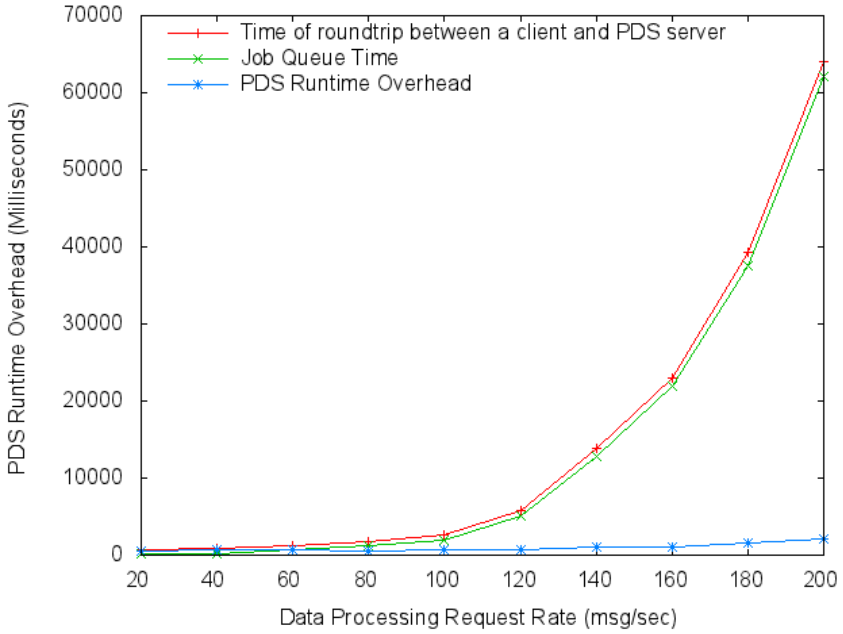

Figure 5.8: PDS Runtime Overhead

We evaluate how small a PDS overhead is in comparison to its methods encapsulated, such as a MapReduce job. Figure 5.9 shows the execution time of 4683 MapReduce experiments that we recorded in the past few years in temporal order. These MapReduce jobs were executed on machines at Indiana University, San Diego Supercomputer Center, Jilin University in China, Computer Network Information Center at Chinese Academy of sciences. Resource allocation for each job varies, from single-node machine up to 20-node cluster. The applications also vary. We tested applications bundled in the Hadoop package, eg., Word Count, Grep, Terasort; we also tested applications such as AutoDock and

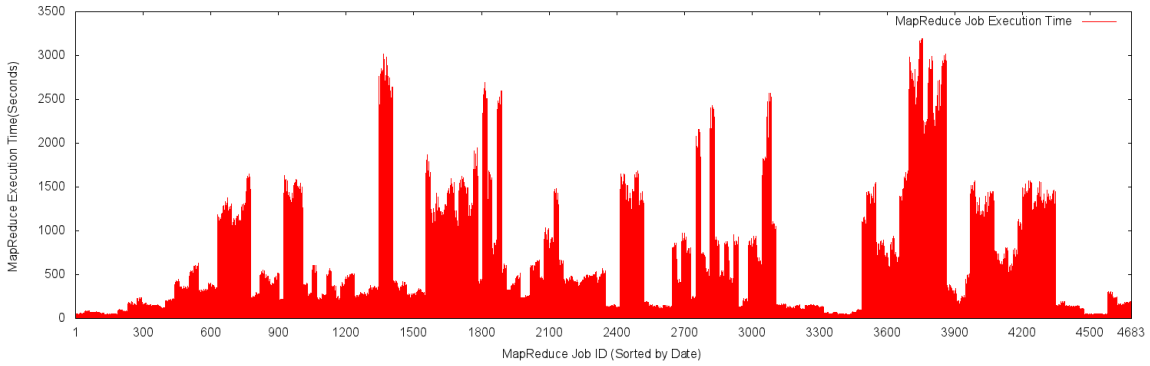

Figure 5.9: 12-month MapReduce Job Execution Time (sorted by date)

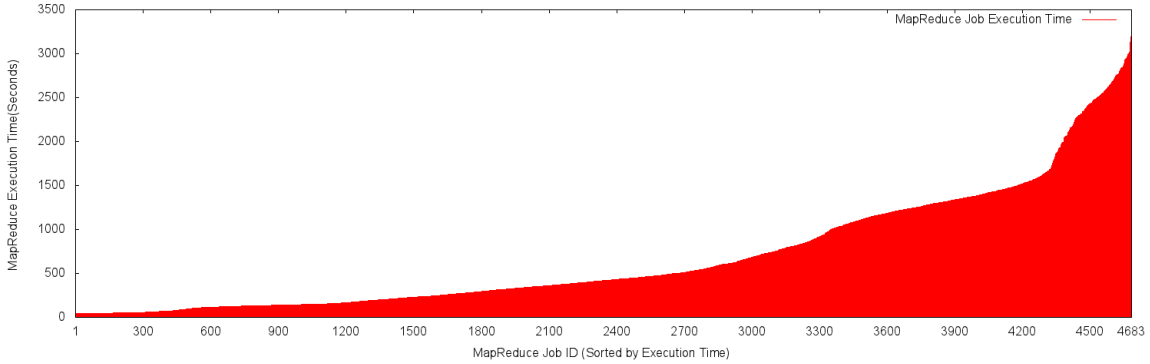

Figure 5.10: 12-month MapReduce Job Execution Time (sorted by execution time)

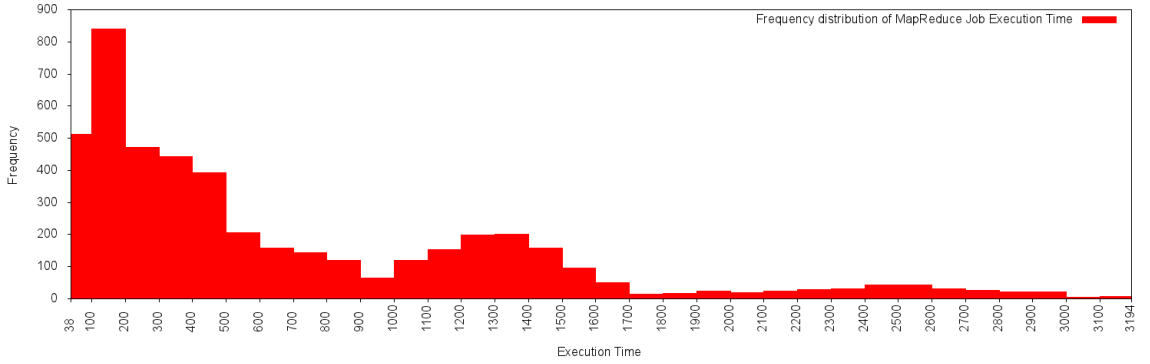

Figure 5.11: 12-month MapReduce Job Execution Time Distribution

BLAST. The size of input dataset also varies, from 1MB to 50GB. In these 4683 jobs, the shortest job finished in 38 seconds, and the longest job finished in 3194 seconds. Figure 5.10 sorts the job ID by job execution time, and Figure 5.11 gives frequency distribution of job execution time. The workload suggests that the average job execution time is 687 seconds, which produces a number of 1/687 job completion per second.

Experiments show that PDS server can receive tens to hundreds of requests per second for PDS method calls. Data processing methods such as MapReduce applications typically are long-run applications and has less than 1/687 job completion per second. Once the requests are received and methods are invoked, the bottleneck shifts to the methods. Therefore, the performance of PDS server will service the pinned data processing well.

5.4.4 Pinned Data on HMR+ Evaluation

We use the same applications that we used for original HMR evaluation. The trick is to control the size percentage of pinned data over the entire dataset. We create a PDS cluster at IU, with and without data; and a MapReduce cluster each at SDSC and CNIC. We allocated at each physical cluster, a 4-node virtual cluster for the PDS cluster and MapReduce clusters. Each node in each cluster has the same specification of 2-core CPU, 4GB RAM, and 80 GB of disk.

AutoDock in HMR+

The details of the original AutoDock HMR was described in Chapter 3.3.1. Considering that AutoDock is a CPU-intensive application, ρ_i is set to 1 per Chapter 3.2.1 so that the maximum number of Map tasks on each node is equal to the number of cores on the node. The version of AutoDock is 4.2 which is the latest stable version. The global controller

does not care about low-level execution details because our local job managers hide the complexity.

During the experiments, 1 shared receptor and 500 ligands are used, which means the job can generate 500 Map tasks. One of the most important configuration parameters is ga_num_evals - number of evaluations. The larger its value is, the higher the probability that better results may be obtained. Based on Chapter 3.3.1, the ga_num_evals is typically set from 2,500,000 to 5,000,000. We configure it to 2,500,000.

We distribute the receptor to each virtual cluster. We then set the pinned data (ligands) percentage to 0%, 20%, 40%, 60%, 80%, and 100% respectively, and distribute unpinned data processing tasks to SDSC and CNIC virtual clusters, according to task distribution algorithm used in Test Case 3 of Chapter 3.4.1. When the PDS cluster at IU is created with dataset bundled, it cannot process data outside of the PDS. However, if the PDS is created without dataset, it accepts both pinned and unpinned data from external sources. Figure 5.12 shows the AutoDock execution on HMR+ where the PDS cluster is bundled with dataset. Figure 5.13 shows the AutoDock execution on HMR+ where the PDS cluster can process external dataset. The result indicates that the higher percentage of pinned data results in less flexibility of job scheduling in both data source scenarios.

Grep in HMR+

The input data consists of 100 text files, each of which is 100MB large, all located at IU. We uses two PDS models in this evaluation: 1) PDS with dataset bundled in; and 2) PDS without dataset. We set the pinned data percentage to 0%, 20%, 40%, 60%, 80%, and 100% of the 10GB dataset respectively on the following three test cases:

Test Case 1: PDS with dataset bundled. The PDS can only process pinned data that is

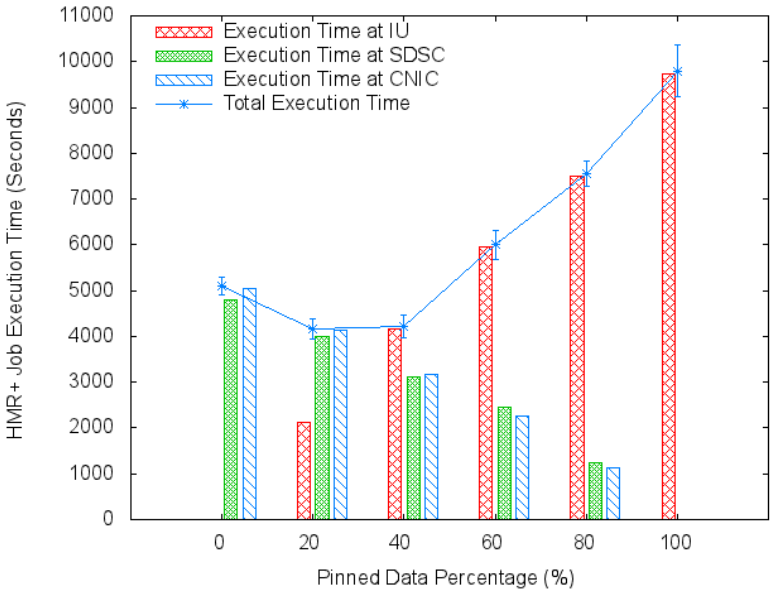

Figure 5.12: Evaluation of AutoDock with Pinned Data on HMR+: PDS with Bundled Dataset

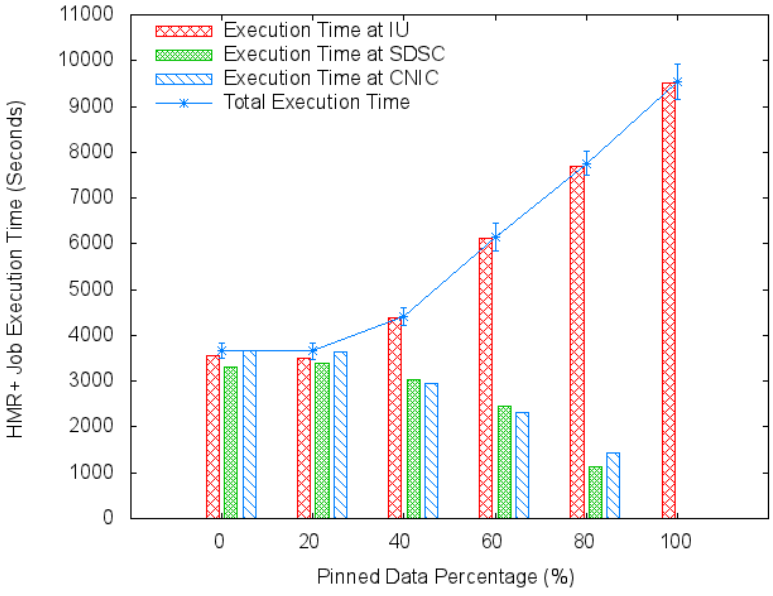

Figure 5.13: Evaluation of AutoDock with Pinned Data on HMR+: PDS with External Data Sources

encapsulated in the suitcase located at IU. The unpinned portion of the dataset needs to be processed on SDSC and CNIC clusters. Figure 5.14 shows the results of this test case. The IU cluster only processes pinned data. There is no data loading from pinned data source to HDFS, because the pinned data is already in the HDFS on the PDS cluster. The rest of the unpinned data sits at IU but outside the PDS. The unpinned data needs to be loaded onto HDFS of the SDSC cluster and CNIC cluster, remotely from IU before processing using MapReduce. When the pinned data percentage increases, IU's PDS takes for workload, therefore, reduces the workload of the clusters at SDSC and CNIC. The total job execution time is summed based on two variables: 1) the maximum sum of data transfer (in and out) and processing time at each cluster, and 2) the $GlobalReduce$ execution time. As we can see in the graph, the optimal value of the total execution time is around 404 seconds at 80% of pinned data. The percentage is higher than the previous AutoDock case, which is around 20% to 40%. The reason for getting this high percentage is because the all the data is located at IU, where the locality is more importance in this data-intensive application.

Test Case 2: PDS without dataset bundled. PDS can only process pinned data that is co-located at IU. The unpinned portion of the dataset also needs to be processed on SDSC and CNIC clusters. The difference between this test case and the previous one is that, it takes PDS fairly large amount of time to load the pinned data from external data sources onto HDFS, in comparison to no pinned data loading for test case 1. Figure 5.15 shows the results of this test case, which has two takeaways: 1) the minimum value of total job execution time increases from Test Case 1's 404 seconds to 542 seconds due to the extra cost of loading pinned data into PDS; and 2) the percentage of pinned data on which the optimal execution time was based, shift from Test Case 1's 80%, to 60%, also due to the extra cost of loading data at IU, which put the workloads at IU still compelling but less

favorable than that of in Test Case 1.

Test Case 3: PDS without dataset bundled. PDS is able to 2) load pinned data from external IU sources, and 2) load unpinned data anywhere accessible. When the percentage of pinned data is low, the PDS can load unpinned data and process it. When the percentage of pinned data hits 40%, the workload at IU reaches an even value that all three clusters can roughly finish at the same time. When the percentage of pinned data goes beyond 40%, the workload at IU exceeds the workloads at SDSC and CNIC, therefore increases the total execution time of the HMR+ job.

Pinned Data on HMR+ Evaluation Summary

From both AutoDock and Grep applications, we see the more data is pinned, the less flexible jobs the HMR+ can schedule. The job execution time will reach an optimal number when the percentage of pinned data increases, and becomes less optimal once the percentage goes beyond a certain number. To receive performance gain when pinned data percentage is high, the PDS virtual cluster will need to scale up to accommodate the increasing workload.

5.5 Summary

In this chapter, we first define a novel data model, "pinned data" that, describes non-consumptive constraint of the data that raw data cannot leave a political jurisdiction. The "pinned data" design is inspired by Object Oriented Programming that data is encapsulated and accessed by method calls. A package of accessible pinned data is called a suitcase, with certain data properties and operations that the suitcase can perform. Encapsulation combines data and behavior in one suitcase and hides the implementation of the data from

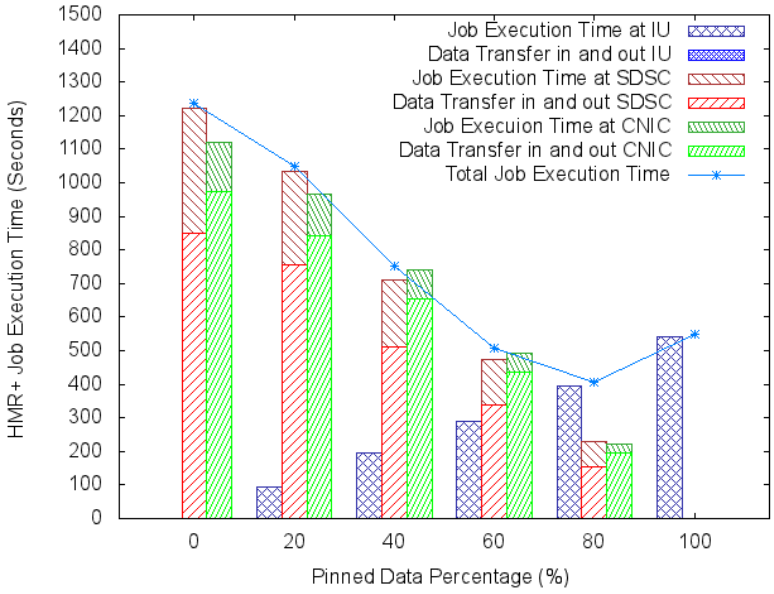

Figure 5.14: Evaluation of grep with pinned data on HMR+: PDS with bundled dataset

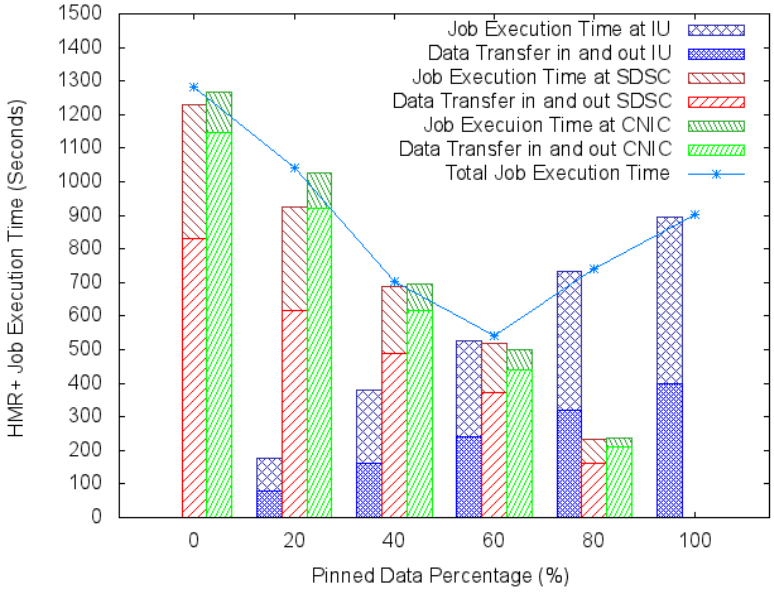

Figure 5.15: Evaluation of grep with pinned data on HMR+: PDS without bundled dataset, PDS is limited to process external pinned dataset only.

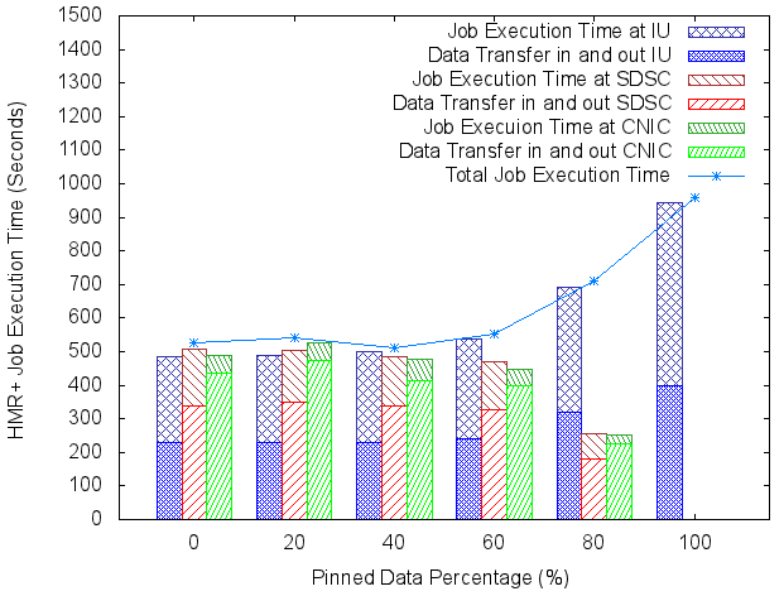

Figure 5.16: Evaluation of grep with pinned data on HMR+: PDS without bundled dataset, PDS is able to process any external dataset.

the user of the suitcase.

We enhance HMR framework to HMR+ to support PND processing. In HMR+, different Map/Reduce functions can be encapsulated in different pinned data suitcases in different clouds before the GlobalReduce function aggregates data to produce final results. Therefore, a pinned data suitecase provides "Data-Processing-as-a-Service" capability.

We evaluated 1) PDS creation overhead for both internal/external data model; 2) the system performance under increasing workloads for PDS method access; 3) HMR+ performance using two test applications, AutoDock HMR and Grep, under different percentages of pinned data.

The pinned data model and its suitcase framework are demonstrated to process geographically distributed and immovable data with minimum runtime framework overhead

while complying to all constrants these data and applications provide.

Chapter 6

Conclusion and Future Work

6.1 Summary

This dissertation addresses techniques to enhance cloud controllability and interoperability, and geographically distributed processing of immovable data, each summarized below.

This dissertation presents a Virtual Cluster Controller (VCC) to enhance the cloud controllability and interoperability. It creates virtual clusters across multiple cloud platforms. In recognition of sensitive data, VCC can establish a single network overlay over widely distributed clusters. Further, by facilitating information sharing among resources, applications, and data, the overall performance is improved. Experimental results show that the overhead of VCC is minimum. The HMR outperforms the traditional MapReduce model while processing a particular class of applications. The evaluations also show that information sharing between resources and application through the VCC shortens the hierarchical data processing time.

This dissertation defines a novel class of data, notably immovable data that we call "pinned data", where the data is treated as a first-class citizen instead of being moved to where needed. We draw from our earlier work with a hierarchical data processing model, Hierarchical MapReduce (HMR), to process geographically distributed data, some

of which is pinned data. The applications implemented in HMR use extended the MapReduce model where computations are expressed as three functions: Map, Reduce, and GlobalReduce. We enhance HMR framework to HMR+ to support PND processing. In HMR+, different Map/Reduce functions can be encapsulate in different pinned data suitcase at different clouds, before the GlobalReduce function aggregates data to produce final results. Therefore, a pinned data suitcase provides "Data-Processing-as-a-Service" capability. Evaluation shows that the overhead of creating PDS mainly contributes to virtual cluster provisioning cost and optional data packing cost; the overhead of running a PDS server is negligible in comparison to data processing costs.

6.2 Conclusion

The PRAGMA community experience shows that there are significant large barriers to set up and control the multi-hosted environment in which data can be shared and analyzed. Although researchers can do real science on such multi-institution environments, the human cost becomes a disincentive to share data in the first place. We have learned from the use of VCC that the barrier can be significantly reduced by automating the virtual cluster management process. Researchers will be largely encouraged to share and use their data worldwide.

The pinned data model provides a programmatic solution of using geographically distributed and immovable data. We consider the pinned data model as an early step to standardize computational accesses to copyright data, of which users are untrusted. The pinned data suitcase instantiated in Hierarchical MapReduce provides a framework for developers to develop and deploy new data processing applications to suit their needs.

6.3 Future Work

The future work of this dissertation includes:

- Enhance Virtual Cluster Controller by adding a provenance collection mechanism. The provenance will help record how a virtual cluster can be instantiated. In a loosely coupled architecture, a standalone provenance service receives notifications (e.g., "data sent") about activity within VCC. Events originate in VCC and can be exposed through notifications generated directly by an adapter deployed at VCC. This is different from our previous solutions [72] [65] where events are exposed through statements written to application log files which can be parsed by a specialized adaptor to generate provenance notifications which are sent to the standalone provenance service asynchronously. The provenance service correlates, organizes, and stores the notifications into provenance sequences. Provenance graphs are formed and visualized [45] when queries are issued to the service.
- Enhance the pinned data suitcase by adding provenance collection mechanism. The pinned data suitcase is a great start point of recording data revisions. Since access of pinned data is solely done through method calls in PDS, the PDS can be instrumented for provenance collection. This will help tracking the use of pinned data.
- Broaden the use of pinned data applications. Applications with immovable data are everywhere, from health data to K-12 student records, especially for large corporation that operates globally. The pinned data model can be used to prevent an employee from accessing sensitive data that he or she has no clearance to, but still obtains non-consumptive results from the computation. The potential of applying the pinned data model to everyday applications are endless.

Bibliography

[1] Amazon CloudFront. http://aws.amazon.com/cloudfront.

[2] Amazon EC2. http://aws.amazon.com/ec2/.

[3] Amazon Elastic MapReduce. http://aws.amazon.com/elasticmapreduce/.

[4] Apache jclouds. http://jclouds.apache.org.

[5] Apache Whirr. http://whirr.apache.org.

[6] Chef. https://www.chef.io/.

[7] CloudForms. http://www.redhat.com/en/technologies/cloud-computing/cloudforms.

[8] CloudMesh. http://cloudmesh.github.io.

[9] Deltacloud. http://deltacloud.apache.org/.

[10] FutureSystems. http://www.futuresystems.org.

[11] HathiTrust. http://hathitrust.org.

[12] Hathitrust Research Center. http://hathitrust.org/htrc.

[13] Jumpgate. http://softlayer.github.io/jumpgate/.

[14] LibCloud. http://libcloud.apache.org/.

[15] ManageIQ. http://manageiq.org/.

[16] PRAGMA Bootstrap. https://github.com/pragmagrid.

[17] Puppet. http://www.puppetlabs.com/.

[18] Rightscale. http://rightscale.com/.

[19] Scalr. http://scalr.com/.

[20] The Rocks Cluster. https://rockscluters.org/.

[21] Windows Azure. http://www.windowsazure.com/.

[22] XSEDE. https://xsede.org/.

[23] Web Services Business Process Execution Language Version 2.0, 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[24] Business Process Model And Notation (BPMN) Version 2.0, 2011. http://www.bpmn.org/.

[25] Topology and Orchestration Specification for Cloud Applications, 2013. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

[26] Faraz Ahmad, Srimat T. Chakradhar, Anand Raghunathan, and T. N. Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. *SIGARCH Comput. Archit. News*, 40(1):61–74, March 2012.

[27] A.I. Avetisyan, R. Campbell, I. Gupta, M.T. Heath, S.Y. Ko, G.R. Ganger, M.A. Kozuch, D. O'Hallaron, M. Kunze, T.T. Kwan, K. Lai, M. Lyons, D.S. Milojicic,

Hing Yan Lee, Yeng Chai Soh, Ng Kwang Ming, J-Y. Luke, and Han Namgoong. Open cirrus: A global cloud computing testbed. *Computer*, 43(4):35–43, April 2010.

[28] A. Beloglazov and R. Buyya. Energy efficient resource management in virtualized cloud data centers. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 826–831, May 2010.

[29] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.*, 28(5):755–768, May 2012.

[30] Tekin Bicer, David Chiu, and Gagan Agrawal. A framework for data-intensive computing with cloud bursting. *Cluster Computing, IEEE Int. Conf. on*, 0:169–177, 2011.

[31] Tekin Bicer, David Chiu, and Gagan Agrawal. Mate-ec2: a middleware for processing data with aws. In *Proc. of the 2011 ACM Int. workshop on Many task computing on grids and supercomputers*, MTAGS '11, pages 59–68, New York, NY, USA, 2011. ACM.

[32] Tekin Bicer, David Chiu, and Gagan Agrawal. Time and cost sensitive data-intensive computing on hybrid clouds. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, CCGRID '12, pages 636–643, Washington, DC, USA, 2012. IEEE Computer Society.

[33] Kevin Borders, Eric Vander Weele, Billy Lau, and Atul Prakash. Protecting confidential data on personal computers with storage capsules. In *Proceedings of the 18th*

Conference on USENIX Security Symposium, SSYM'09, pages 367–382, Berkeley, CA, USA, 2009. USENIX Association.

[34] P. J. Braam. Lustre: A Scalable, High Performance File System, 2002. http://www.lustre.org.

[35] Tracy D. Braun, Howard Jay Siegel, Anthony A. Maciejewski, and Ye Hong. Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions. *J. Parallel Distrib. Comput.*, 68(11):1504–1516, November 2008.

[36] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. Cotuner: A framework for coordinated auto-configuration of virtualized resources and appliances. In *Proc. of the 7th Int. Conf. on Autonomic Computing*, ICAC '10, pages 75–76, New York, NY, USA, 2010. ACM.

[37] Xiangping Bu, Jia Rao, and Cheng-zhong Xu. Interference and locality-aware task scheduling for mapreduce applications in virtual clusters. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, pages 227–238, New York, NY, USA, 2013. ACM.

[38] R. Buyya, Chee Shin Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 5–13, Sept 2008.

[39] Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In

Proc. of the HPC ASIA'2000, China, pages 283–289. IEEE Computer Society Press, 2000.

[40] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.

[41] Michael Cardosa, Chenyu Wang, Anshuman Nangia, Abhishek Chandra, and Jon Weissman. Exploring mapreduce efficiency with highly-distributed data. In *Proceedings of the second international workshop on MapReduce and its applications*, MapReduce '11, pages 27–34, New York, NY, USA, 2011. ACM.

[42] Bryan Catanzaro, Narayanan Sundaram, and Kurt Keutzer. A map reduce framework for programming graphics processors. In *In Workshop on Software Tools for MultiCore Systems*, 2008.

[43] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. How to enhance cloud architectures to enable cross-federation. In *IEEE CLOUD*, pages 337–345, 2010.

[44] Keke Chen, J. Powers, Shumin Guo, and Fengguang Tian. Cresp: Towards optimal resource provisioning for mapreduce computing in public clouds. *Parallel and Distributed Systems, IEEE Transactions on*, 25(6):1403–1412, June 2014.

[45] Peng Chen, B. Plale, Y. Cheah, D. Ghoshal, S. Jensen, and Yuan Luo. Visualization of network data provenance. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–9, Dec 2012.

[46] Yuan Chen, D. Gmach, C. Hyser, Zhikui Wang, C. Bash, C. Hoover, and S. Singhal. Integrated management of application performance, power and cooling in data centers. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 615–622, April 2010.

[47] N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba. Network virtualization: State of the art and research challenges. *Comm. Mag.*, 47(7):20–26, July 2009.

[48] N.M.M.K. Chowdhury, M.R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791, April 2009.

[49] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, July 2003.

[50] Fernando Costa, Luis Silva, and Michael Dahlin. Volunteer cloud computing: Mapreduce over the internet. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, pages 1855–1862, Washington, DC, USA, 2011. IEEE Computer Society.

[51] Karl Czajkowski, Ian T. Foster, Nicholas T. Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *Proceedings of the Workshop on Job Scheduling Strate-*

gies for Parallel Processing, IPPS/SPDP '98, pages 62–82, London, UK, UK, 1998. Springer-Verlag.

[52] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[53] Zhaohui Ding, Xiaohui Wei, Yuan Luo, Da ma, Peter Arzberger, and Wilfred Li. Customized plug-in modules in metascheduler csf4 for life sciences applications. *New Generation Computing*, 25:373–394, 2007. 10.1007/s00354-007-0024-6.

[54] Adam Dou, Vana Kalogeraki, Dimitrios Gunopulos, Taneli Mielikainen, and Ville H. Tuulos. Misco: a mapreduce framework for mobile systems. In *Proc. of the 3rd Int. Conf. on PErvasive Technologies Related to Assistive Environments*, PETRA '10, pages 32:1–32:8, New York, NY, USA, 2010. ACM.

[55] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proc. of the 19th ACM Int. Symp. on High Performance Distributed Computing*, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM.

[56] Marwa Elteir, Heshan Lin, and Wu-chun Feng. Enhancing mapreduce via asynchronous data processing. In *Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems*, ICPADS '10, pages 397–405, Washington, DC, USA, 2010. IEEE Computer Society.

[57] Ana Juan Ferrer, Francisco Hernndez, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, Ral Sirvent, Jordi Guitart, Rosa M. Badia, Karim Djemame, Wolfgang Ziegler, Theo Dimitrakos, Srijith K. Nair, George Kousiouris, Kleopatra

Konstanteli, Theodora Varvarigou, Benoit Hudzia, Alexander Kipp, Stefan Wesner, Marcelo Corrales, Nikolaus Forg, Tabassum Sharif, and Craig Sheridan. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66 – 77, 2012.

[58] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Quality of Service, 1999. IWQoS '99. 1999 Seventh International Workshop on*, pages 27–36, 1999.

[59] Ian Foster and Carl Kesselman. Globus: a metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115–128, 1997.

[60] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[61] G. C. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo, S. Smallen, W. Smith, and A. Grimshaw. *FutureGrid: A Reconfigurable Testbed for Cloud, HPC and Grid Computing*. Chapman & Hall, 2012.

[62] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002. 10.1023/A:1015617019423.

[63] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Proc. of the 1st Int. Symp. on Cluster Computing and the Grid*, CCGRID '01, pages 35–, Washington, DC, USA, 2001. IEEE Computer Society.

[64] Shantenu Jha Supun Kamburugamuve Geoffrey Fox, Judy Qiu and Andre Luckow. Hpc-abds high performance computing enhanced apache big data stack. In *2nd International Workshop on Scalable Computing For Real-Time Big Data Applications (SCRAMBL'15)*, Shenzhen, Guangdong, China, 2015.

[65] Devarshi Ghoshal and Beth Plale. Provenance from log files: A bigdata problem. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 290–297, New York, NY, USA, 2013. ACM.

[66] Zhenhua Guo, Geoffrey Fox, and Mo Zhou. Investigation of data locality in mapreduce. In *Proc. of the 2012 12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 419–426, Washington, DC, USA, 2012. IEEE Computer Society.

[67] Andreas Hanemann and et al. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *Proc. of the Third Int. Conf. on Service-Oriented Computing*, ICSOC'05, pages 241–254, Berlin, Heidelberg, 2005. Springer-Verlag.

[68] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju, and Tuyong Wang. Mars: a mapreduce framework on graphics processors. In *Proc. of the 17th Int. Conf. on Parallel architectures and compilation techniques*, PACT '08, pages 260–269, New York, NY, USA, 2008. ACM.

[69] Robert Henderson. Job scheduling under the portable batch system. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 279–294. Springer Berlin / Heidelberg, 1995.

[70] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proc. of the 8th USENIX Conf. on Networked Systems Design and Implementation*, NSDI'11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

[71] Eduardo Huedo, Ruben S. Montero, and Ignacio M. Llorente. A framework for adaptive execution in grids. *Softw. Pract. Exper.*, 34:631–651, June 2004.

[72] S. Jensen, B. Plale, M.S. Aktas, Yuan Luo, Peng Chen, and H. Conover. Provenance capture and use in a satellite data processing pipeline. *Geoscience and Remote Sensing, IEEE Transactions on*, 51(11):5090–5097, Nov 2013.

[73] Andre Luckow Judy Qiu, Shantenu Jha and Geoffrey C.Fox. Towards hpc-abds: An initial high-performance big data stack, March 2014.

[74] Pierre St. Juste, David Wolinsky, P. Oscar Boykin, Michael J. Covington, and Renato J. Figueiredo. Socialvpn: Enabling wide-area collaboration with integrated social and overlay networks. *Comput. Netw.*, 54(12):1926–1938, August 2010.

[75] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Sci. Program.*, 13(4):265–275, October 2005.

[76] Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga, and Jose Fortes. Sky computing. *IEEE Internet Computing*, 13:43–51, September 2009.

[77] Kate Keahey and et al. Infrastructure outsourcing in multi-cloud environment. In *Proc. of the 2012 Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit*, FederatedClouds '12, pages 33–38, New York, NY, USA, 2012. ACM.

[78] Jong-Kook Kim, Sameer Shivle, Howard Jay Siegel, Anthony A. Maciejewski, Tracy D. Braun, Myron Schneider, Sonja Tideman, Ramakrishna Chitta, Raheleh B. Dilmaghani, Rohit Joshi, Aditya Kaul, Ashish Sharma, Siddhartha Sripada, Praveen Vangari, and Siva Sankar Yellampalli. Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. *J. Parallel Distrib. Comput.*, 67(2):154–169, February 2007.

[79] Praveenkumar Kondikoppa, Chui-Hui Chiu, Cheng Cui, Lin Xue, and Seung-Jong Park. Network-aware scheduling of mapreduce framework ondistributed clusters over high speed networks. In *Proc. of the 2012 workshop on Cloud services, federation, and the 8th open cirrus summit*, FederatedClouds '12, pages 39–44, New York, NY, USA, 2012. ACM.

[80] Gunho Lee, Niraj Tolia, Parthasarathy Ranganathan, and Randy H. Katz. Topology-aware resource allocation for data-intensive workloads. *SIGCOMM Comput. Commun. Rev.*, 41(1):120–124, January 2011.

[81] Hongliang Li, Xiaohui Wei, Qingwu Fu, and Yuan Luo. Mapreduce delay scheduling with deadline constraint. *Concurrency and Computation: Practice and Experience*, 26(3):766–778, 2014.

[82] Min Li, Dinesh Subhraveti, Ali R. Butt, Aleksandr Khasymski, and Prasenjit Sarkar. Cam: A topology aware minimum cost flow based resource manager for mapreduce

applications in the cloud. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 211–222, New York, NY, USA, 2012. ACM.

[83] Heshan Lin, Xiaosong Ma, Jeremy Archuleta, Wu-chun Feng, Mark Gardner, and Zhe Zhang. Moon: Mapreduce on opportunistic environments. In *Proc. of the 19th ACM Int. Symp. on High Performance Distributed Computing*, HPDC '10, pages 95–106, New York, NY, USA, 2010. ACM.

[84] Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor - a hunter of idle workstations. In *ICDCS*, pages 104–111, 1988.

[85] Changbin Liu, Boon Thau Loo, and Yun Mao. Declarative automated cloud resource orchestration. In *Proc. of the 2Nd ACM Symp. on Cloud Computing*, SOCC '11, pages 26:1–26:8, New York, NY, USA, 2011. ACM.

[86] Yuan Luo, Zhenhua Guo, Yiming Sun, Beth Plale, Judy Qiu, and Wilfred W. Li. A hierarchical framework for cross-domain mapreduce execution. In *Proc. of the second Int. workshop on Emerging computational methods for the life sciences*, ECMLS '11, pages 15–22, New York, NY, USA, 2011. ACM.

[87] Yuan Luo and Beth Plale. Hierarchical mapreduce programming model and scheduling algorithms. In *Proc. of the 2012 12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 769–774, Washington, DC, USA, 2012. IEEE Computer Society.

[88] Yuan Luo, Beth Plale, Zhenhua Guo, Wilfred W. Li, Judy Qiu, and Yiming Sun. Hierarchical mapreduce: towards simplified cross-domain dataprocessing. *Concurrency and Computation: Practice and Experience*, pages 878–893, 2012.

[89] Pradeep Kumar Mantha, Andre Luckow, and Shantenu Jha. Pilot-mapreduce: an extensible and flexible mapreduce implementation for distributed data. In *Proceedings of third international workshop on MapReduce and its Applications Date*, MapReduce '12, pages 17–24, New York, NY, USA, 2012. ACM.

[90] Eugene E. Marinelli. Hyrax: Cloud computing on mobile devices using mapreduce. (CMU-CS-09-164), September 2009.

[91] Lorenzo Martignoni, Pongsin Poosankam, Matei Zaharia, Jun Han, Stephen McCamant, Dawn Song, Vern Paxson, Adrian Perrig, Scott Shenker, and Ion Stoica. Cloud terminal: Secure access to sensitive applications from untrusted systems. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 165–182, Boston, MA, 2012. USENIX.

[92] Andréa Matsunaga, Maurício Tsugawa, and José Fortes. Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ESCIENCE '08, pages 222–229, Washington, DC, USA, 2008. IEEE Computer Society.

[93] Dejan Milojicic, Ignacio M. Llorente, and Ruben S. Montero. Opennebula: A cloud management tool. *IEEE Internet Computing*, 15(2):11–14, 2011.

[94] H.H. Mohamed and D.H.J. Epema. An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In *Cluster Computing, 2004 IEEE Int. Conf. on*, pages 287–298, Sept 2004.

[95] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The open provenance model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743 – 756, 2011.

[96] Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, 45(12):65–72, 2012.

[97] Garrett M. Morris, Ruth Huey, William Lindstrom, Michel F. Sanner, Richard K. Belew, David S. Goodsell, and Arthur J. Olson. Autodock4 and autodocktools4: Automated docking with selective receptor flexibility. *Journal of Computational Chemistry*, 30(16):2785–2791, 2009.

[98] Amit Nathani, Sanjay Chaudhary, and Gaurav Somani. Policy based resource allocation in iaas cloud. *Future Generation Computer Systems*, 28(1):94 – 103, 2012.

[99] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proc. of the 2009 9th IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, CCGRID '09, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[100] Balaji Palanisamy, Aameek Singh, Ling Liu, and Bhushan Jain. Purlieus: Locality-aware resource allocation for mapreduce in a cloud. In *Proc. of 2011 Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 58:1–58:11, New York, NY, USA, 2011. ACM.

[101] Jongse Park, Daewoo Lee, Bokyeong Kim, Jaehyuk Huh, and Seungryoul Maeng. Locality-aware dynamic vm reconfiguration on mapreduce clouds. In *Proc. of the 21st Int. Symp. on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 27–36, New York, NY, USA, 2012. ACM.

[102] Jorda Polo, David Carrera, Yolanda Becerra, Vicenc Beltran, Jordi Torres, and Eduard Ayguade. Performance management of accelerated mapreduce workloads in heterogeneous clusters. In *Proceedings of the 2010 39th International Conference on Parallel Processing*, ICPP '10, pages 653–662, Washington, DC, USA, 2010. IEEE Computer Society.

[103] John Porter, Peter Arzberger, Hans-Werner Braun, Pablo Bryant, Stuart Gage, Todd Hansen, Paul Hanson, Chau-Chin Lin, Fang-Pang Lin, Timothy Kratz, William Michener, Sedra Shapiro, and Thomas Williams. Wireless sensor networks for ecology. *BioScience*, 55(7):561–572, 2005.

[104] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, HPCA '07, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.

[105] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The reservoir model and architecture for open federated cloud computing. *IBM J. Res. Dev.*, 53(4):535–545, July 2009.

[106] Frank Schmuck and Roger Haskin. Gpfs: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[107] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: Toward an open-source solution for cloud computing. *Int. Journal of Computer Applications*, 55(3):38–42, October 2012. Published by Foundation of Computer Science, New York, USA.

[108] Saurabh Sehgal, Miklos Erdelyi, Andre Merzky, and Shantenu Jha. Understanding application-level interoperability: Scaling-out mapreduce over high-performance grids and clouds. *Future Generation Computer Systems*, 27(5):590 – 599, 2011.

[109] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005.

[110] Yogesh L. Simmhan, Beth Plale, Dennis Gannon, and Suresh Marru. Performance evaluation of the karma provenance framework for scientific workflows. In *Proceedings of the 2006 International Conference on Provenance and Annotation of Data*, IPAW'06, pages 222–236, Berlin, Heidelberg, 2006. Springer-Verlag.

[111] B. Sotomayor, R.S. Montero, I. Martin Llorente, and I. Foster. Resource leasing and the art of suspending virtual machines. In *High Performance Computing and*

Communications, 2009. HPCC '09. 11th IEEE International Conference on, pages 59–68, June 2009.

[112] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Capacity Leasing in Cloud Systems using the OpenNebula Engine. Cloud Computing and Applications 2008 (CCA08), 2009.

[113] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22, September 2009.

[114] P St Juste, K Jeong, H Eom, C Baker, and RJ Figueiredo. Tincan: User-defined p2p virtual network overlays for ad-hoc collaboration. *ICST Trans. on Collaborative Computing*, 14, 2014.

[115] Bing Tang, Mircea Moca, Stephane Chevalier, Haiwu He, and Gilles Fedak. Towards mapreduce for desktop grid computing. In *Proceedings of the 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 3PGCIC '10, pages 193–200, Washington, DC, USA, 2010. IEEE Computer Society.

[116] Osamu Tatebe, Kohei Hiraga, and Noriyuki Soda. Gfarm grid file system. *New Generation Computing*, 28:257–275, 2010. 10.1007/s00354-009-0089-5.

[117] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The condor experience: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(2-4):323–356, February 2005.

[118] Maurício Tsugawa and José A. B. Fortes. A virtual network (vine) architecture for grid computing. In *Proc. of the 20th Int. Conf. on Parallel and distributed processing*, IPDPS'06, pages 148–148, Washington, DC, USA, 2006. IEEE Computer Society.

[119] Montserrat Vaqué, Anna Arola, Carles Aliagas, and Gerard Pujadas. Bdt: an easy-to-use front-end application for automation of massive docking tasks and complex docking strategies with autodock. *Bioinformatics*, 22(14):1803–1804, 2006.

[120] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, and et al. Apache hadoop yarn: Yet another resource negotiator. In *Proc. of the 4th Annu. Symp. on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM.

[121] Christian Vecchiola, Rodrigo N. Calheiros, Dileban Karunamoorthy, and Rajkumar Buyya. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka. *Future Generation Computer Systems*, 28(1):58 – 65, 2012.

[122] Lizhe Wang, Jie Tao, Rajiv Ranjan, Holger Marten, Achim Streit, Jingying Chen, and Dan Chen. G-hadoop: Mapreduce across distributed data centers for data-intensive computing. *Future Gener. Comput. Syst.*, 29(3):739–750, March 2013.

[123] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[124] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Job scheduling for multi-user mapreduce clusters. Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley, Apr 2009.

[125] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: A simple technique for achieving lo-

cality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 265–278, New York, NY, USA, 2010. ACM.

[126] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

[127] Jiaan Zeng, Guangchen Ruan, Alexander Crowell, Atul Prakash, and Beth Plale. Cloud computing data capsules for non-consumptiveuse of texts. In *Proceedings of the 5th ACM Workshop on Scientific Cloud Computing*, ScienceCloud '14, pages 9–16, New York, NY, USA, 2014. ACM.

[128] Cindy Zheng, David Abramson, Peter Arzberger, and et al. The pragma testbed - building a multi-application international grid. In *Proc. of the Sixth IEEE Int. Symp. on Cluster Computing and the Grid*, CCGRID '06, pages 57–, Washington, DC, USA, 2006. IEEE Computer Society.

[129] Cindy Zheng, Mason J. Katz, Phil M. Papadopoulos, David Abramson, Shahaan Ayyub, Colin Enticott, Slavisa Garic, Wojtek Goscinski, Peter Arzberger, Bu Sung Lee, Sugree Phatanapherom, Somsak Sriprayoonsakul, Putchong Uthayopas, Yoshio Tanaka, and Yusuke Tanimur. Lessons learned through driving science applications in the pragma grid. *Int. J. Web Grid Serv.*, 3(3):287–312, August 2007.

[130] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Softw. Pract.*

Exper., 23:1305–1336, December 1993.

Curriculum Vitae

Yuan Luo

EDUCATION

Doctor of Philosophy, Computer Science

Indiana University, Bloomington, IN — August 2015

Master of Science, Computer Science

Indiana University, Bloomington, IN — September 2010

Master of Science, Computer Science

Jilin University, China — July 2008

Bachelor of Science, Computer Science

Jilin University, China — July 2005

PROFESSIONAL EXPERIENCE

Indiana University — August 2009 - August 2015

Research Assistant, Data To Insight Center — June 2010 - August 2015

- Creator and architect of Hierarchical MapReduce (HMR), an extension to MapReduce framework that runs MapReduce jobs across aggregated resources from multi-cloud environment.
- Architect and developer of Virtual Cluster Controller, that creates virtual clusters across multiple cloud platforms, builds a network overlay, and manages the virtual cluster life-cycle in a fault-tolerant manner.
- Technical Lead and principal architect of PRAGMA Cloud at Indiana University. PRAGMA Cloud currently has 13 VM hosting clusters, spanning 11 institutes and 7 different countries/regions over 4 different VM management platforms.

- Architect and developer of Matchmaker, a standalone service that dynamically loads user requirements in JSON, generate java classes and instantiate POJOs without predefined schema, and leverages Drools rule engine to make matchmaking decisions.
- Lead developer of Karma Provenance Collection Tool, a standalone tool that can be added to existing cyber-infrastructure for collection and representation of provenance data.
- Technical Lead of InstantKarma project, to collect provenance within the NASA Earth Science community.
- Lead developer of Hybrid Workflow. Qualitative and quantitative study of the trade-offs of a hybrid workflow solution that utilizes multiple workflow systems and solutions to execute a single workflow.
- Developer of Linked Environments for Atmospheric Discovery (LEAD) project. Built BPEL-based workflows.

Associate Instructor, School of Informatics and Computing August 2009 - May 2010, and Fall 2014

- Graduate Course: Management, Access, and Use of Big and Complex Data (Fall 2014), with Prof. Beth Plale. Designed projects that emphasis on (Twitter) data analytics and performance evaluation of data storage (MongoDB) and queries.
- Graduate Course: Distributed Systems (Spring 2010), with Prof. Beth Plale. Course project: Parallel ray tracing application using Amazon Elastic MapReduce and Web Services.
- Graduate Course: Algorithms Design and Analysis (Fall 2009), with Prof. Paul Purdom. Course project: Write a fast routine that multiplies large non-negative integers, code must be correct up to a million bits.

- Undergraduate Course: Introduction to Computers and Computing (Fall 2009), with Charles E. Pope. Basic principles of computers and software. Social and lifestyle effects of information technology.

IBM T.J. Watson Research Center July 2012 - October 2012

Research Intern, Hawthorne, NY

- Developed a combined infrastructure for running both transactional and analytics workloads. Mapping and Maintenance between a NoSQL store (HBase) and a distributed transactional cache (IBM Websphere eXtreme Scale).

University of California, San Diego 2006, and 2009

Research Intern, Center for Research in Biological Systems (CRBS) June 2009 - September 2009

- Co-developer of Opal Toolkit, wrapper for scientific applications as web services on Grid and Cloud resources. Integrated Opal with CSF4 meta-scheduler so that Opal jobs can be scheduled onto heterogeneous HPC clusters managed by batch schedulers.

Visiting Scholar, National Biomedical Computation Resource July 2006 - September 2009

- Developer of My Worksphere project, which served as one of the TeraGrid science gateway. It was an end to end prototype environment that allows existing applications to run transparently on the Grid.

PROFESSIONAL SERVICES AND ACTIVITIES

1. Technical Program Committee Member, The 3rd and 4th IEEE International Conference on Cloud Networking (CloudNet 2014 and 2015)

2. Workshop Co-chair, The 1st - 4th PRAGMA Students Workshop, 10/2012, 03/2013, 10/2013 and 04/2014

3. Session Chair and Program Committee Member, The 23rd - 27th PRAGMA Workshop at Seoul, Korea; Bangkok, Thailand; Beijing, China;

4. Tainan, Taiwan; Bloomington, Indiana, USA, 10/2012, 03/2013, 10/2013, 04/2014 and 10/2014

5. Reviewer, Journals: Concurrency and Computation: Practice and Experience(2010 -); Scalable Computing(2013); IEEE Systems Journal(2013);

6. Journal of Parallel and Distributed Computing(2014); Conference: IEEE/ACM CCGrid(2012); IEEE CloudNet (2014 -); IEEE eScience (2014)

7. Co-founder of PRAGMA Students Steering Committee, PRAGMA Students, 2012-2014

8. Volunteer, The 2nd IEEE International Conference on Cloud Computing Technology and Science, Indianapolis, Indiana, USA, 12/2010

HONORS

1. K. Jon Barwise Fellowship, Indiana University School of Informatics and Computing 2008-2009

2. Visiting Scholar, University of California, San Diego, 2006

3. Excellent undergraduate thesis (design) 2005, Jilin University, China; Excellent Student Leadership 2001-2002, Undergraduate Student Award 2003-2005, Graduate Stu-

dent Award 2005-2006, Jilin University, China; Graduate Student Assistantship with full tuition waiver and monthly stipend 2005-2008, Ministry of Education, China

PRESENTATIONS AND INVITED TALKS

1. Cross-Institute Virtual Cluster Management in PRAGMA, PRAGMA 27 Workshop, Bloomington, Indiana, USA, Oct 15-17, 2014

2. User-level controllability of virtual clusters using HTCondor, PRAGMA 26 Workshop, Tainan, Taiwan, April 9-11, 2014

3. Hierarchical MapReduce: Towards Simplified Cross-Domain Data Processing, Invited talk at Jilin University, China, October 22, 2013

4. A Hierarchical MapReduce Framework, Invited talk at IBM Student Workshop for Frontiers of Cloud Computing 2012, IBM's Thomas J. Watson Research Center in Hawthorne, New York, July 30-31, 2012

5. Hierarchical MapReduce: Towards Simplified Cross-Domain Data Processing, Invited talk at Cloud Computing Lecture, Indiana University, Oct 12, 2011.

6. Opal-Sigiri: Software as a Service on PRAGMA Testbed, PRAGMA 20 Workshop, Hong Kong, China, March 2-4. 2011.

7. Metascheduling using the Community Scheduler Framework (CSF4), NBCR Summer Institute 2009, UCSD, 2009.

8. My WorkSphere: Integrated and transparent access to Gfarm computational data drid through GridSphere portal with meta-scheduler CSF4, Invited talk at NBCR Special Seminar, UCSD, Aug 28th 2006.

9. Cluster and Grid Computing: Transparent access and workflow m anagement, NBCR Summer Institute 2006, UCSD, 2006.

PUBLICATIONS

Journal Papers

1. Luo, Y., Plale, B., Guo, Z., Li, W. W., Qiu, J. and Sun, Y. (2014), Hierarchical MapReduce: towards simplified cross-domain data processing. Concurrency Computat.: Pract. Exper., 26: 878893.

2. Jensen, S.; Plale, B.; Aktas, M.S.; Yuan Luo; Peng Chen; Conover, H., "Provenance Capture and Use in a Satellite Data Processing Pipeline," Geoscience and Remote Sensing, IEEE Transactions on , vol.51, no.11, pp.5090,5097, Nov. 2013.

3. Hongliang Li, Xiaohui Wei, Qingwu Fu, Yuan Luo. (2013) MapReduce Delay Scheduling with Deadline Constraint, Concurrency and Computation: Practice and Experience.

4. Ding, Z., Wei, X., Luo, Y., Ma D, Li, W. W., Arzberger, P. W. Customized Plug-in Modules in Meta-scheduler CSF4 for Life Sciences Applications, New Generation Computing, Vol.25 No.4 2007.

5. Ding, Z., Wei, X., Luo, Y., et al. A Virtual Job Model to Support Cross-Domain Synchronized Resource Allocation (In Chinese with English Abstract), Journal of Jilin University (Science Edition), Vol. 46, No.2, Mar 26, 2008.

Conference and Workshop Papers

1. Peng Chen, Beth Plale, You-Wei Cheah, Devarshi Ghoshal, Scott Jensen, and Yuan Luo. Visualization of Network Data Provenance, Workshop on Massive Data Analytics on Scalable Systems, co-located with HiPC, Pune, India, Dec. 18-21, 2012.

2. Plale, B., Withana, E. C., Herath, C., Chandrasekar, K., Luo, Y. Effectiveness of Hybrid Workflow Systems for Computational Science, International Conference on Computational Science (ICCS), Omaha, Nebraska, Jun 4-6, 2012.

3. Luo, Y. and Plale, B. Hierarchical MapReduce Programming Model and Scheduling Algorithms, Doctoral Symposium of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Ottawa, Canada, May 13-16, 2012.

4. Luo, Y., Guo, Z., Sun, Y., Plale, B., Qiu, J., Li, W. W. 2011. A Hierarchical Framework for Cross-Domain MapReduce Execution. In Proceedings of the Second International Workshop on Emerging Computational Methods for the Life Sciences (ECMLS '11). ACM, New York, NY, USA, 15-22.

5. Wei, X., Luo, Y., Gao, J., et al. The Session Based Fault Tolerance Algorithm of Platform EGO Web Service Gateway, Proceedings of International Symposium on Grid Computing (ISGC2007), Academia Sinica, Taipei, Taiwan, March 26-29, 2007.

6. Ding, Z., Luo, Y., Wei, X., Misleh, C., Li, W. W., Arzberger, P. W., Tatebe, O. My WorkSphere: Integrative Work Environment for Grid-unaware Biomedical Researchers and Applications, 2nd International Workshop on Grid Computing Environments (GCE06) at SC06, Tampa, FL. Nov. 12-13, 2006.