SCHEDULING AND FLEXIBLE CONTROL OF WIDE-AREA DATA TRANSPORT SERVICES FOR END-TO-END APPLICATION WORKFLOWS

BY MEHMET FATIH AKTAS

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Manish Parashar

and approved by

New Brunswick, New Jersey

May, 2015

ABSTRACT OF THE THESIS

Scheduling and Flexible Control of Wide-Area Data Transport Services for End-to-End Application Workflows

by Mehmet Fatih Aktas

Thesis Director: Manish Parashar

Emerging end-to-end scientific applications that integrate high-end experiments and instruments with large scale simulations and end-user displays, require complex couplings and data sharing between distributed components involving large data volumes and varying hard (in-time data delivery) and soft (in-transit processing) quality of service (QoS) requirements. As a result, enabling efficient data coupling is a key requirement of such workflows. In this thesis, we try to address this in two levels; in data transport level and in data sharing abstraction level. Firstly, we leverage software-defined networking (SDN) to address issues of data transport service control and resource provisioning to meet varying QoS requirements from multiple coupled workflows sharing the same service medium. Specifically, we present a flexible control and a disciplined resource scheduling approach for data transport services for science networks. Furthermore, we emulate an SDN testbed on top of the FutureGrid virtualized testbed and use it to evaluate our approach for a realistic scientific workflow. Our results show that SDN-based control and resource scheduling based on simple intuitive service models can meet the coupling requirements with high resource utilization. Secondly, we present design and implementation of an asynchronous data sharing framework for application couplings over wide-area network. Specifically, presented framework extends shared space abstractions of Dataspaces, which is a data sharing framework for HPC applications, to wide-area scale using a NUMA-like architecture and implementation that leverages advanced data transport technologies like GridFTP and RDMA, and uses predictive prefetching using Markov based models to bring remote data close to the application in-time. Finally, our initial results evaluating the performance of the presented framework show that given some slack time between the data insertion and retrieval queries, latency due to data transport over wide-area network can be efficiently masked for realistic scientific workflows.

Acknowledgements

I would like to start by acknowledging my family in Turkey who has always been very understanding and encouraging during my studies. I would like to thank my advisor Prof. Manish Parashar for his invaluable guidance, support and encouragement during the course of this work and throughout my graduate studies. I am thankful to Prof. Ivan Rodero and Kristin Dana for being on my thesis committee. I also thank my colleagues in RDI^2, particularly Georgiana Haldeman and Alejandro Pelaez, for their friendship and feedbacks.

The research presented is partially published in [1] and extended version is submitted as a special issue to Future Generation Computer System Journal (FGCS, Elsevier impact factor 2.6). Presented work is supported in part by the US National Science Foundation (NSF) via grant numbers ACI 1339036, ACI 1310283, DMS 1228203, and IIP 0758566; by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the US Department of Energy through the Scientific Discovery through Advanced Computing (SciDAC) Institute of Scalable Data Management, Analysis and Visualization (SDAV) under award number DE-SC0007455; by the Advanced Scientific Computing Research and Fusion Energy Sciences Partnership for Edge Physics Simulations (EPSI) under award number DE-FG02-06ER54857; by the ExaCT Combustion Co-Design Center via subcontract number 4000110839 from UT Battelle; by the RSVP grant via subcontract number 4000126989 from UT Battelle; and by an IBM Faculty Award. The research was conducted as part of the NSF Cloud and Autonomic Computing (CAC) Center at Rutgers University and the Rutgers Discovery Informatics Institute (RDI2).

Dedication

To my parents.

Table of Contents

Abstract ii

Acknowledgements iv

Dedication v

List of Tables viii

List of Figures ix

1. Scheduling and Flexible Control of Bandwidth and In-transit Services for End-to-End Application Workflows 1

1.1. Introduction 1

1.1.1. Motivation 1

1.1.2. Problem Description – Enabling End-to-End Coupled Simulation Workflows 2

1.1.3. Overview and Approach 3

1.1.4. Contributions 4

1.1.5. Outline 5

1.2. Background and Related Work 5

1.3. Scheduling and Flexible Control of Bandwidth and In-transit Services . 7

1.3.1. **In-transit Service Model** 8

1.3.2. **Data Transport Service Control** 10

1.3.3. **Resource Allocation** 12

General Overview 12

Problem Domain-Specific Adaptation 14

Penalty and Utility Functions 16

Deriving models for session penalty and utility 16

Mapping application demands for hard and soft requirements on the models 18

1.4. Experiments and Results 19

1.5. Conclusions and Future Work 24

2. Wide-Area Data Staging 30

2.1. Introduction 30

2.1.1. Motivation 30

2.1.2. Problem Description 30

2.1.3. Overview and Approach 32

2.1.4. Contributions 33

2.1.5. Outline 34

2.2. Background and Related Work 34

2.2.1. Dataspaces 34

2.2.2. Prefetching 38

2.3. Extending Dataspaces Abstractions to Wide-Area Scale 41

2.3.1. A conceptual overview of Wide-Area Dataspaces 41

2.3.2. Architecture and Implementation overview 41

2.4. Wide-Area Data placement 45

2.4.1. Prefetching formulation 48

2.5. Evaluation 54

2.6. Conclusions and Future Work 55

References 58

List of Tables

1.1. Parameters for the Fusion workflow scenario. 22

1.2. Parameters for the random scenario. 25

List of Figures

1.1. The end-to-end connection is broken into three sub-connections, which are represented in the Figure by dotted, dashed and dotted-dashed lines. 9

1.2. Illustration of IT-SP operations at producer, consumer and intermediate host stacks. 9

1.3. Data transport control architecture. Varying-sized squares depict heterogeneous processing/storage capacity of intermediate hosts. 10

1.4. DT-OS operations for a single coupling session – an illustration of the control actions taken to schedule and implement a data walk for a coupling session between source and sink. 12

1.5. Mapping three different coupling types on two dimensional parameter space of m_s^{soft} vs. m_s^{hard} and $\check{x}_s^{soft}$ vs. $\check{x}_s^{hard}$ 19

1.6. A schematic of the ECEI data and diagnostics workflow. The diagnostic routines do not change the size of the data, and comply with the optimization problem formulated in Section 1.3.3. 20

1.7. Schematic overview of the emulation testbed deployed across multiple VMs. 21

1.8. Two topologies used for the experiments. Filled switches are the ones with in-transit host connected. 23

1.9. Results for the first set of experiments using the topology in Figure 1.8a. 26

1.10. Results for the second set of experiments using the topology in Figure 1.8b. 27

1.11. Distribution of the allocation of link and host capacities to active coupling sessions after each scheduling run. Every stacked bar corresponds to allocation to a single session. 28

1.12. Results for the experiments using the random scenario in Table 1.2 and topology in Figure 1.8b. 29

2.1. Organizational representation of Wide-Area Dataspaces. 32

2.2. Representation of wide-area Dataspaces on one-dimensional circular node-identifier space (Left). Each Dataspaces stores preceding segment on the key space over the resources distributed in its locality (Right). 33

2.3. Conceptual view of Dataspaces architecture. 37

2.4. Wide-Area Dataspaces Architecture. DS: Dataspaces, WA: Wide-Area . 41

2.5. Representation of prefetching buffers allocated in the peers of each Dataspaces. 46

2.6. Decision tree constructed by the prefetcher given observed page access history $H = [a, a, b, b, b, a, a, b, a, a, a]$ and order of Markov model is 2 and number of accesses to be predicted is 1. 50

2.7. Representation of application key renaming procedure during prefetching for producer-consumer scenario. Model order k is 2 and number of prefetched data items l is 1. 54

2.8. Comparison of total get time between w/ (Bottom) and w/o (Top) prefetching for the scenario of producer-consumer based sharing of 10 data items of same size. Both X and Y-axes are log-scaled. 57

Chapter 1

Scheduling and Flexible Control of Bandwidth and In-transit Services for End-to-End Application Workflows

1.1 Introduction

1.1.1 Motivation

As scientific discovery is becoming increasingly data driven, scientific applications are moving towards end-to-end workflows that integrate coupled simulations with data sources such as instruments, sensor systems and experiments, and with data analysis and visualization pipelines to facilitate online knowledge extraction. Furthermore, the execution of such workflows often involves geographically distributed resources with runtime interactions, coordination and data exchanges between processes running on these resources [2, 3].

Scientific workflows typically involve complex couplings between the workflow components/services, requiring sharing of large data volumes with varying quality of service (QoS) requirements, and efficient data transport is a key requirement. Specifically, the time between when the data is generated at the producer and when it can be consumed at the consumer can have a significant impact on the execution of the workflow. For example, slower data delivery can throttle the consumer, or faster data delivery may require storing the data at the consumer. Some applications may require that data is delivered at the consumer within a tight time window (for example, data needed to control an experiment), which adds further requirements on the data transport. Additionally, there often exists natural mismatches in the way data is represented at producers and consumers, and the data has to be transformed in a timely manner before it can be consumed. As a result, the data transport has to address multiple challenging requirements based on data sizes, data production and consumption rates, strict constraints on data delivery time and data storage, and managing data transformations between producers and consumers. The data transport medium is also typically shared by multiple application workflows with possibly competing application specific coupling requirements, and best-effort solutions, which inherently can not offer service guarantees, will not be able to achieve high performance when using an over utilized service medium.

1.1.2 Problem Description – Enabling End-to-End Coupled Simulation Workflows

Emerging scientific applications integrate simulations with data sources such as experiments and instruments, and analysis and visualization pipelines, into end-to-end workflows. These workflows exhibit different and varying interaction, coordination and data coupling behaviors, such as:

- **Tight Coupling:** Coupled processes exchange data very frequently. Therefore, when tight coupling is dominant and processes are coupled over the network, total workflow execution time is typically dominated by the data transfer time.
- **Loose Coupling:** Relatively less frequent, asynchronous and possibly opportunistic data exchanges among the coupled processes. Producer and consumer progress at different rates, and may have different data representations making intermediate data transformation necessary.
- **Dataflow Coupling:** Data flows from producers to consumers using publish/subscribe/notify-like semantics, for example, in case of data processing and/or analysis pipelines. Dataflow coupling may often involve strict data transport constraints such as delivery time, data integrity and reliability.

To meet such complex and varying coupling requirements, it is essential for the data transport service to be application aware and self-optimizing in that it must autonomously adapt based on dynamic application requirements and resource states. Meanwhile, the overheads of adaptation and service management on the application should be acceptable.

In this research, we focus on control and scheduling of data transfer (across network switches and links) and in-transit service (intermediate staging and processing hosts) resources to meet varying data transport requirements of differing coupling behaviors, such as those described above. We assume that the different coupling requirements of the application workflow are provided to the scheduler before the coupled data is streamed, as further discussed in Section 1.3.2.

1.1.3 Overview and Approach

The Software Defined Networking (SDN) initiative aims at making network control simpler and more flexible using well-defined abstractions for forwarding and switch configuration. A key idea behind SDN is replacing the existing distributed control with centralized control. This is achieved by implementing network control programs on top of the network operating system (NOS), which in turn provides the control programs with a global view of the network and interfaces for communicating with switches, enabling the control programs to configure the network state.

In this research, we explore how the data transport service can benefit from SDN, and the capabilities that it provides, such as (1) open and programmable control, (2) faster innovation at the networking layer, (3) easy customization and optimization of network resource scheduling via flexible control; all those can be used to enhance data transport service management. Overall, by leveraging SDN-based networking control, we propose a framework for data transport service control and scheduling for end-to-end coupled application workflows.

Our approach for scheduling and flexible control of data transport services is explained in detail in section 1.3. Here a brief overview of it will be given. Data transport service for application workflows over wide-area consists of data transfer and in-transit processing and staging services. We assume no middleware exists to provide data transport services; data transfer between the data producer and consumer is TCP/IP flow over network switches and in-transit processing/staging takes place at available intermediate resources intervening the data flow. Network switches and intermediate resources are distributed and interconnected, and have heterogeneous service capacity i.e. respectively bandwidth, and processing rate and staging duration. As illustrated in Figure 1.3, we propose to implement the control plane for these resources on top of a SDN-Network Operating System (NOS) so that controllers implemented on top of the control plane have global view of resources and achieves resource management using flexible API for controlling data transfer and in-transit processing/staging services. SDN-NOS already provides a flexible API for controlling data transfer so we extend it to Data Transport-OS (DT-OS) by adding "right" abstractions and API for in-transit data processing/staging. We implement a prototype DT-OS by extending a SDN-OS which is actively used by researchers. Finally, we are presenting a disciplined approach using convex optimization to allocate resources (i.e. bandwidth, in-transit processing rate and staging duration) between different data transport sessions and implemented a scheduler using this approach on top of DT-OS.

1.1.4 Contributions

The goal of this project is to explore the scheduling and control of bandwidth and in-transit services to address the challenges of data transport service for end-to-end scientific application workflows. Specifically, we leverage software-defined networking (SDN) to address issues of data transport service control and resource provisioning to meet varying QoS requirements from multiple coupled workflows sharing the same service medium. The specific contribution of this work is a disciplined resource scheduling approach for data transport resources that is both application and network aware, and enables flexible control. The proposed control framework achieves flexibility by using SDN abstractions for managing the network, and further extending these abstractions to manage data transport services and schedule data transport resources. We also develop a model for in-transit data staging and data processing using intermediate resources in the data path using the approach outlined in [3, 4, 5]. Finally, we emulate an SDN testbed on top of the FutureGrid virtualized testbed and use it to evaluate our approach for an end-to-end Fusion workflow. Our results show that SDN-based control and resource scheduling based on simple intuitive models can meet the coupling requirement with high resource utilization.

1.1.5 Outline

The rest of this chapter is organized as follows. Section 1.2 provides some background and summarizes the related work. Section 1.3 presents our approach for scheduling and control of bandwidth and in-transit services for end-to-end scientific application workflows. Section 1.4 describes our emulation testbed and presents the experimental evaluation of our approach. Section 1.5 concludes the chapter, and outlines ongoing and future research.

1.2 Background and Related Work

In [3], Bhat et al. use adaptive buffer management based on proactive and/or online user-defined policies for the QoS management of a self-managing data streaming and in-transit processing service for Grid-based data intensive workflows. In this work, the streaming and in-transit processing components work cooperatively (using feedforward and feedback messages) to meet overall application requirements and constraints. This solution is similar to our solution in that it uses QoS management to meet user-defined requirements. However the approach in this work does not address the management of data transport resources, while our research is founded on scheduling and controlling of the transport resources. Cooperative management strategies presented in [3] could be used to extend our framework to work efficiently on a commodity service medium.

Active networking is a communication pattern [6] for tailoring network service to user requirements as explained in [7]. A network is called active when processing can be done within the network over active elements such as switches that have processing capability. Programming active switches according to application specific needs and taking advantage of packet-based processing within the network has been a key focus of active networking research. In [8], Lefevre et al. present an active network architecture

(A-Grid) that attempts to provide QoS management for Grid data transport services in addition to other data transport services such as reliable multicast and dynamic service deployment. Their architecture employs QoS management at intermediate active routers, and in principal, it is similar to the opportunistic in-transit processing employed by our solution. However, our solution makes use of intermediate hosts to do flow-based processing rather than packet-based processing over active network elements. Moreover, their solution can only provide QoS management per application, but not per workflow, because their architecture lacks a global view of the service medium and of simultaneously running user applications, which is another feature of our solution.

Network resource reservation systems such as ESNET's OSCARS [9] and UltraScience Net [10] provide on-demand dedicated bandwidth channels to user applications. They introduce a virtual single-switch abstraction on top of networks which employ both a bandwidth reservation system and SDN concepts in [11]. The work presented in this chapter is different but complementary. In this thesis, we presented a disciplined scheduling of bandwidth and in-transit processing/staging capacity to meet application requirements for data transport in the context of data-intensive coupled workflows, and utilized SDN-based centralized control concepts to manage resource states according to scheduling decisions. Network reservations and virtualization systems can be used in a complementary manner to improve performance of the scheduling and flexibility of the resource control that we introduced.

DataCutter [12] is a middleware for filtering scientific data in a Grid environment which enables application developers to explore and analyze data sets stored in archival storage systems in a wide-area network. Data processing takes place within distributed Grid resources to run algorithms for filtering data. These filter-based algorithms are composed by operations like sort, select and join, which are implemented as batch or stream-based processing in the framework. Overall, main focus of this work is data management not data transport – they take data transport into consideration only to develop heuristics to place computation to resources close to data while the main focus of our research is about optimizing the efficiency of data transport over wide-area when it is inevitable. However, they use stream-based processing to offload computation from the data source which is close to opportunistic in-transit processing idea we focus. Control of data transport services we introduce here can be used on extending stream-based processing approach used in DataCutter framework.

AutoFlow [13] project aims to meet performance requirements of distributed information flow applications by achieving awareness between the Grid middleware (including Grid data transport services), resource and applications so that behaviors of interacting elements can be adjusted autonomically based on the runtime states of their horizontal (e.g. data generator application - processing resource) or vertical (e.g. network switch - data transport service middleware) peers. This project is more comprehensive and aggressive than our project in terms of aiming at achieving inter-aware operational Grid. Our work is more focused on enabling efficient wide-area workflow couplings than managing whole workflow execution over the Grid as in AutoFlow.

An active buffering approach is presented in [14] which is mainly about carrying I/O operations from the data source to third-party computing nodes. Their approach achieves this by managing the idle third-party idle memory to form a buffering hierarchy which transports the data to the server side only when the buffer is full and writes data is written to disk at the server side once the server side memory also gets full. Even though active buffering does not address challenges with scientific data transport over wide-area, the idea of using intermediate nodes to stage data in locally available idle resources is similar to in-transit staging we consider.

1.3 Scheduling and Flexible Control of Bandwidth and In-transit Services

In this section, we explore scheduling and flexible control of bandwidth and in-transit services for end-to-end scientific application workflows, and leverage software-defined networking (SDN) to address issues of data transport service control and resource provisioning to meet varying QoS requirements from multiple coupled workflows sharing the same service medium. Specifically, the solution we propose has three key components: (1) A flow-based in-transit service model, (2) Centralized layered architecture for data transport service control, and (3) Application and network aware resource scheduling for workflow couplings.

General solution architecture can be summarized as follows. Given a service medium consisting of interconnected network switches and intermediate resources with heterogeneous staging and processing capability, a centralized controller with capability of managing service medium gets application demands from workflow components (service users) for each data transport session and it tries to schedule total bandwidth, processing and staging capacity between all the active data transport sessions so that maximum ”cumulative” satisfaction of user demands can be achieved. In the following subsections, we will detail the three key components of this general solution idea.

1.3.1 In-transit Service Model

We have observed that in the packet-based intermediate processing/staging model, by intercepting network packets on-the-fly, the end-to-end system design arguments are violated (this has been further discussed in [15], [16] and [17]), which significantly impacts the existing network stack performance. In order to achieve flow-based in-transit processing/staging over the intermediate hosts, we propose a model that uses switch-host coordination, which is conceptually similar to the gateway-host coordination in the Phoebus architecture [18], to break the end-to-end connections into subconnections. A. Brown et al. in [18] demonstrate that the Phoebus architecture preserves the end-to-end design arguments within sub-connections, which ensures that the overall data transport performance is not adversely affected. In the proposed flow-based in-transit processing/staging approach, end-to-end connections between producers and consumers are broken into sub-connections at intermediate hosts so that application data at these intermediate hosts can be treated as a flow rather than packets, as illustrated in Figure 1.1. Autonomic switch-host coordination for creating and managing sub-connections can be handled by the granular forwarding control of SDN – this is further explained in Subsection 1.3.2.

The proposed flow-based in-transit processing/staging can be implemented between the transport layer (TCP) and the application layer using an in-transit service protocol

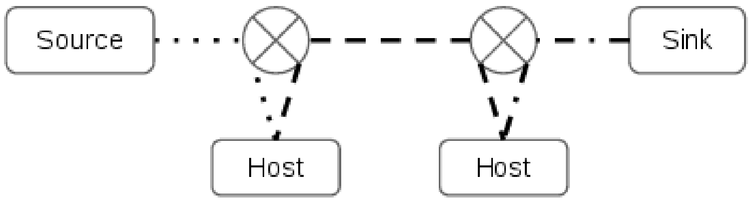


Figure 1.1: The end-to-end connection is broken into three sub-connections, which are represented in the Figure by dotted, dashed and dotted-dashed lines.

(IT-SP). The IT-SP header enables the flow of service specific information (e.g., the list of functions that will be opportunistically executed over the data) between the producer, the consumer and intermediate hosts. As shown in Figure 1.2, the producer encapsulates the application data stream with the IT-SP/TCP header. The intermediate hosts decapsulate the IT-SP/TCP header and execute programmed processing/staging pipelines [1] on the data. When the in-transit pipelines finish executing, the resulting data is encapsulated using the updated IT-SP/TCP header. Finally, the consumer decapsulates the IT-SP/TCP header and pushes the data to the application layer.

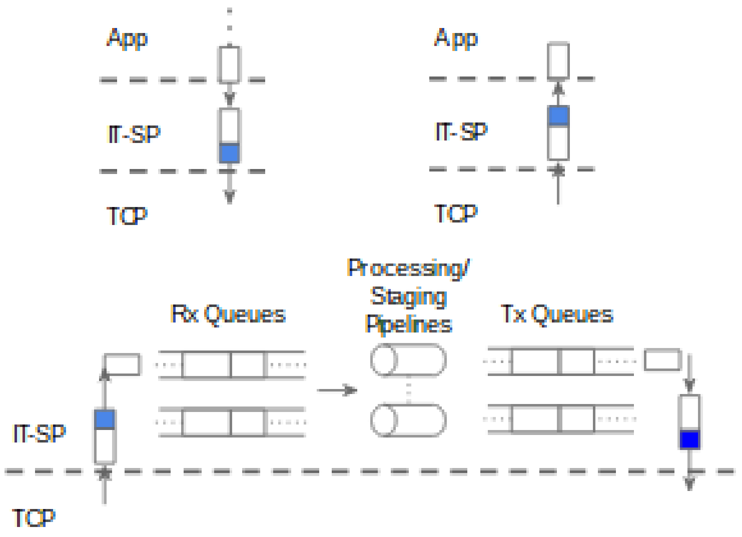


Figure 1.2: Illustration of IT-SP operations at producer, consumer and intermediate host stacks.

[1] We have implemented processing rate and staging duration control using the conventional "Token Bucket Filter" (TBF) approach used for traffic control on packet-switching networks [19].

1.3.2 Data Transport Service Control

In order to manage autonomic switch-host coordination and ensure efficient in-transit service control for flow-based intermediate processing and staging, our approach is to extend SDN-NOS to a Data Transport OS (DT-OS) that offers support for in-transit service management with the help of a controller (scheduler) running above it, as shown in Figure 1.3. The scheduler contains the control logic that is required to decide on the appropriate actions necessary to optimize the allocation of resources. Similar to NOS, DT-OS provides a centralized view of the service medium to the scheduler, and in turn, the scheduler tells DT-OS what resources to allocate to individual coupling sessions according to the application requirements. The resulting control architecture consisting of the DT-OS and the scheduler, preserves the centralized and layered features of the SDN architecture. Overall, our framework adds two additional abstractions to the existing SDN abstractions: (1) Switch-host coordination for flow-based intermediate data processing/staging and (2) Configuration of in-transit processing/staging pipelines at intermediate hosts.

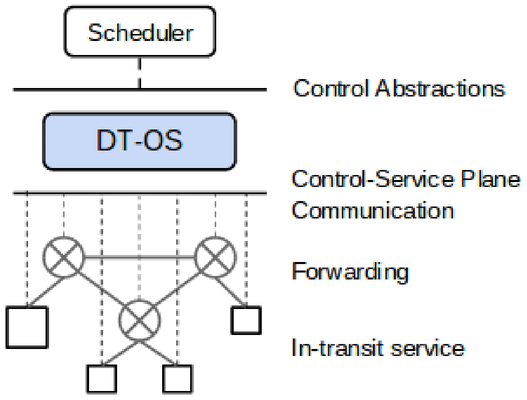


Figure 1.3: Data transport control architecture. Varying-sized squares depict heterogeneous processing/storage capacity of intermediate hosts.

The flow of DT-OS operations for a single coupling session is illustrated in Figure 1.4. The steps include:

- **1-Request:** Application sends a *request* (UDP: sching_port), which consists of application requirements. The gateway switch is programmed in advance to forward packets wild-carded with UDP: sching_port to the scheduler.

- **2-Scheduling:** The scheduler does a feasibility check [2] on the application requirements and then generates rules for data walk route and resource allocation for the session.

- **3-State setup:** DT-OS translates the rules generated by the scheduler into low-level service state update messages, and sets up forwarding and in-transit service tables at the switches and intermediate hosts.

- **4-Reply:** The scheduler sends a *reply* (UDP: sching_port) over the gateway switch where the request was initially received, and the switch forwards it to the initiator of the request (i.e., the source). The reply contains scheduling information (e.g., scheduled TCP destination port stream_port).

- **5-Data streaming:** Once the source receives the reply, it immediately starts streaming data over IT-SP/TCP: stream_port.

In step **3-State setup**, DT-OS configures forwarding tables and in-transit service tables by communicating with switches and intermediate hosts over the control channels. DT-OS programs the switches to recognize session packets wild-carded with TCP: stream_port, modify destination and/or source IP/MAC addresses and forward them accordingly, in order to realize the scheduled data walk (over the source-host and host-sink sub-connections as shown in Figure 1.4). DT-OS also programs the in-transit hosts to listen on TCP:stream_port, and execute scheduled processing/staging pipelines on the session data and forward it to the next hop.

The scheduler treats every application request as a new coupling session. When there are multiple active coupling sessions, the scheduler may need to reallocate resources between them, dynamically update the resource states, and inform the applications about any changes.

[2] If application requests cannot be satisfied as is, they are modified appropriately and the application is notified.

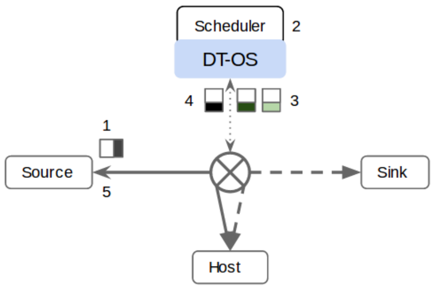


Figure 1.4: DT-OS operations for a single coupling session – an illustration of the control actions taken to schedule and implement a data walk for a coupling session between source and sink.

1.3.3 Resource Allocation

We have not yet addressed the problem of resource allocation, which can become a bottle neck if not done properly. In this section, we systematically formulate the problem of optimizing resource allocation as a convex optimization problem, since convex optimization problems have unique globally optimal solutions that can be obtained with several existing efficient, reliable and robust algorithms [20].

General Overview

Service is a generic term we use to describe capabilities provided by the network infrastructure, which are data transfer or in-transit data processing and staging. Capacity of these services is quantified with data transfer rate, processing rate and staging duration. *Resources* providing data transport services are network switches, links and in-transit hosts. *Coupling sessions* refer to executing coupled simulation workflows that use data transport services. Every coupling session is a stream of application data over the scheduled service resources. *Scheduling* is the primary function of shared service management, and distributes finite capacity of each data transport service between coupling sessions.

The objective during scheduling is to maximize *cumulative user satisfaction*, which quantifies the satisfaction of user application requirements in terms of resource allocation distribution over the coupling sessions. To define it we use total penalty and utility functions $P, U : R^{k\times N} \rightarrow R$ which return numerical penalty and utility scores for a particular resource allocation distribution. Furthermore, we define a penalty and utility function $P_s, U_s : R^k \rightarrow R$ for each coupling session s, which helps to formulate total penalty and utility functions as follows:

$$\begin{aligned} P(\mathbf{r}) &= \underset{s=1,\dots,N}{max} P_s(\mathbf{r}_{1,s}, ..., \mathbf{r}_{k,s}) \\ U(\mathbf{r}) &= \underset{s=1,\dots,N}{min} U_s(\mathbf{r}_{1,s}, ..., \mathbf{r}_{k,s}) \end{aligned} \tag{1.1}$$

where $\mathbf{r}_{i,s}$ denotes allocation of resource i to coupling session s and constitutes an element in the matrix $\mathbf{r} \in R^{k\times N}$, where k is the number of resources and N is the number of coupling sessions. As promised in the last paragraph, we define scheduling objective of maximizing cumulative user satisfaction as minimizing the total penalty and maximizing total utility. Then, the problem of optimizing resource allocation can be expressed as:

$$\begin{aligned} &\underset{w.r.t.R^2_+}{\text{minimize}} && [P(\mathbf{r}), -U(\mathbf{r})] \\ &\text{subject to:} && sum(\boldsymbol{r}[i,:]) \leq C_i,\ i = 1, \dots, k, \\ & && \boldsymbol{r} \succeq 0. \end{aligned} \tag{1.2}$$

where C_i is total capacity of resource i. However, defining objective as above transforms the problem of optimizing resource allocation into a bi-criterion optimization problem, which generally does not achieve global optimal since the objectives of the combining functions (total penalty and total utility functions) may be competing. Therefore, we further fit the bi-criterion optimization problem into a Pareto optimization problem by scalarizing it using the weighted sum objective as in the following:

$$\begin{aligned} &\underset{w.r.t.R+}{\text{minimize}} && f_0(\mathbf{r}) = P(\mathbf{r}) - \gamma U(\mathbf{r}) \\ &\text{subject to:} && sum(\mathbf{r}[i,:]) \leq C_i,\ i = 1, \dots, k, \\ & && \mathbf{r} \succeq 0. \end{aligned} \tag{1.3}$$

where $\gamma > 0$ can be interpreted as the relative weight of total utility relative to total penalty.

In the next subsections, we define the individual components of the construction we just described and explain how they all fit together.

Problem Domain-Specific Adaptation

Fitting the problem of optimizing resource allocation as formulated in the previous subsection strips away some important aspects. For example, resource allocation for coupling sessions assumes end-to-end connectivity, i.e., it assumes that there is a path from the producer to the consumer, and it abstracts all the aspects of routing involved in providing this end-to-end connectivity. Another requirement of resource allocation is data integrity. Guaranteeing data integrity requires some assumptions about the structure of the produced data. We assume that (1) the produced data is in the form of list of chunks of uniform size, (2) in-transit processing does not change the size of the data, and (3) functions operated on data chunks also don't change the size of the data chunks. In addition, optimizing resource allocation over all the physical components (network switches, links and intermediate hosts) for all the coupling sessions using optimization variable matrix $\mathbf{r}$, which contains allocation of actual resources to individual coupling sessions, is an overkill. As a result, for each coupling session our framework schedules network links by selecting one of the available routing paths, and intermediate hosts on the chosen path, which together form the data walk. Then using convex optimization, optimal service capacity (bandwidth, processing rate and staging duration) allocation over the scheduled resources is obtained.

When there are multiple paths available for data transport between the session producer and consumer, our path selection approach attempts to choose the path which minimizes the overall discrepancy between the loads on all of the network links and in-transit hosts in the system. To explain it with an example, suppose a path should be assigned for a new coupling session and there are some number of available paths. Path selection algorithm picks a path and assumes the path resources are allocated for the session and finds the coefficient of variance (the ratio of standard deviation to the mean value) of fair bandwidth of all the network links and fair processing/staging capacity of all the in-transit hosts, and repeats this for every available path. Then, finally picks the path which gives the minimum total coefficient of variance for fair bandwidth and fair processing/staging capacity. Fair capacity of a resource is calculated by dividing the total capacity with the number of coupling sessions using the resource.

However, simplifying the problem introduces the need of a mapping from the simplified problem of optimizing over the data walks to the more complex problem of optimizing over the physical resources. Essentially, what we need is a mapping $\mathbf{M}$ that can map the previously defined matrix $\mathbf{r}$ (which abstracts the resource allocation over the physical resources) to a matrix $\mathbf{A}$ (which abstracts the service capacity allocation over the data walks), i.e., we need to construct $\mathbf{A}$ and $\mathbf{M}$ to fit the equality: $\mathbf{A} = \mathbf{Mr}$. We construct matrix $\mathbf{A}$ as:

$$\mathbf{A} = \begin{bmatrix} bw_1 & bw_2 & \dots & bw_N \\ proc_1 & proc_2 & \dots & proc_N \\ dur_1 & dur_2 & \dots & dur_N \\ n_1 & n_2 & \dots & n_N \end{bmatrix}$$

where bw_s, $proc_s$, dur_s represent allocation of bandwidth, processing rate and staging duration for session s, and n_s is a variable to be assigned a value between 0 and 1 by the scheduler; the higher it is, the more processing work is scheduled to be completed in-transit for session s. In order to fit $\mathbf{A} = \mathbf{Mr}$, we extend each element of $\mathbf{r}$ as a tuple $(r_{i,s}^{bw}, r_{i,s}^{proc}, r_{i,s}^{dur})$ such that, if r_i is a network link then $r_{i,s}^{proc}$, $r_{i,s}^{dur}$ are zero, and if it is an in-transit host then $r_{i,s}^{bw}$ is zero. So it follows that:

$$\begin{aligned} bw_s &= \min_{\forall i | r_i \in link_s} r_{i,s}^{bw} \\ proc_s &= \sum_{i | r_i \in host_s} r_{i,s}^{proc} \\ dur_s &= \sum_{i | r_i \in host_s} r_{i,s}^{dur} \end{aligned}$$

where $link_s$ and $host_s$ are sets of network links and in-transit hosts on the data walk of session s.

We first set $\mathbf{M}$ as a constant per scheduling run (it is recalculated for every new scheduling run) and set $\mathbf{A}$ as the optimization variable using which we can model the penalty and utility functions P_s and U_s in a straightforward manner, as presented in the next subsection. Based on this problem domain-specific modification, the formulation of the optimization problem defined in the previous subsection now becomes:

$$\begin{aligned} \text{minimize} \quad & P(\mathbf{A}) - \gamma U(\mathbf{A}) \\ \text{subject to:} \quad & \mathbf{A} = \mathbf{Mr} \\ & \mathbf{A} \succeq 0 \\ & [n_1 \; n_2 \; \ldots \; n_N] \preceq 1 \\ & sum(\mathbf{r}[i,:]) \leq C_i, \; i = 1, \ldots, k. \end{aligned} \tag{1.4}$$

Penalty and Utility Functions

Penalty and utility functions of a coupling session quantify performance of scheduling in terms of its ability to meet the application requirements. They return penalty or utility scores given service capacity allocated and application requirements. Following the methodology suggested in [21], we define a requirement as *hard* if failing to satisfy it returns penalty, and as *soft* if succeeding to satisfy returns utility. In the workflows we have explored (1) there is a defined optimal time for data arrival, and a performance penalty is incurred if the data arrives earlier or later than the optimal time – on-time data delivery is a hard requirement, and (2) applications can benefit from on-the-fly data processing since it can lead to an earlier completion of workflow execution and/or reduced resource requirements, i.e., it incurs utility – in-transit processing is a soft requirement.

Overall, we model session penalty and utility as functions of allocated service capacity and application demands for hard and soft requirements. Two steps to do this are as follows:

Deriving models for session penalty and utility

To obtain a convex optimization objective (see equation (1.1) and (1.4)), P_s and U_s must be respectively convex and concave for each session.

Session penalty score with respect to the hard requirement (in-time delivery) can be modeled as the relative difference of the actual data transport time from the optimal transport time: $|1 - trans_s/opttrans_s|$, where $trans_s$ is actual transport completion time and $opttrans_s$ is the optimal transport completion time for session s. Transport completion time is the sum of data transfer, processing and staging times. Data transfer time increases multiplicatively with data size and additively with latency, and decreases multiplicatively with bandwidth, i.e., $D_s/bw_s + L_s$ where D_s is size of streamed data and L_s is the latency of the network path. Model for data transfer time is a convex function of bw_s since bandwidth is always positive. Data processing time increases multiplicatively with data size (assuming in-transit processing does not change data size), in-transit processing complexity, and decreases multiplicatively with processing rate, i.e., $D_s \times O_s \times (n_s)^2/proc_s$ where O_s is a scalar representing the total processing complexity calculated as the sum of complexity indexes of all processing tasks. Complexity index of a processing routine is a scale measuring how long it takes to execute a routine for unit data at unit processing rate and an oracle model for it can be obtained by running the task on data of varying sizes and interpolating the results. We chose to use n_s^2 as the percentage of processing tasks completed in-transit to make the processing time model quadratic-over-linear function of n_s and $proc_s$, and thus the model is convex. This does not disrupt anything since the value of optimization variable n_s is constrained to be between 0 and 1 in equation (1.4), which constraints also n_s^2 in the same range i.e. if $n_s^2 = 0.4$, then 40% of the total processing work is scheduled to be completed in-transit. Finally, total time that the streamed data is staged over in-transit resources is represented by dur_s. To summarize, we model data transport completion time as:

$$trans_s = \frac{D_s}{bw_s} + L_s + D_s \times \frac{O_s \times (n_s)^2}{proc_s} + dur_s$$

However, substituting this model of $trans_s$ into session penalty model $|1 - trans_s/opttrans_s|$ cannot achieve convexity because the absolute function is not non-decreasing. Therefore, we will introduce a new optimization variable t_s for each session s and replace $trans_s$ with it to get new convex session penalty model $|1 - t_s/opttrans_s|$, and add a new inequality to the optimization problem to reflect the relationship between $trans_s$ and t_s, so that variable t_s will trace epigraph of $trans_s$:

$$\frac{D_s}{bw_s} + L_s + D_s \times \frac{O_s \times (n_s)^2}{proc_s} + dur_s \leq t_s$$

The gap between t_s and $trans_s$ is nonzero when available service capacity exceeds the capacity needed to achieve the optimal objective value. This is not an issue in our case since in data-intensive scientific workflows the network is typically over-utilized by multiple coupling sessions.

Session utility score with respect to the soft requirement (in-transit processing) could be modeled simply as the percent of processing tasks completed in-transit, which we modeled as n_s^2 previously. However, this would violate concavity of session utility function so instead we use square root of this; n_s as the numerical session utility score.

Overall, both session penalty and utility models produce values between 0 and 1 and both roughly represent ratio of two numbers. Therefore, the objective function formulated in equation (1.4) makes sense since it returns the difference between two ratios.

Mapping application demands for hard and soft requirements on the models

We will present a way to implement application demands for hard and soft requirements by setting a few well-understood parameters. To fit application demands on the models discussed in the previous step, we write session penalty and utility functions as follows:

$$\begin{aligned} P_s &= max\{m_s^{hard} \times (|1 - t_s/opttrans_s| - \check{x}_s^{hard}), 0\} \\ U_s &= m_s^{soft} \times (n_s - \check{x}_s^{soft}) \end{aligned}$$

where:

- m_s^{hard}: Rate of multiplicative increase in session penalty score per increase in the relative difference between the actual and optimal transport times.
- $\check{x}_s^{hard}$: Tolerance for in-time delivery; session penalty becomes zero when relative difference between actual and optimal transport time is less than $\check{x}_s^{hard}$.
- m_s^{soft}: Rate of multiplicative increase in session utility score per increase in percent of processing work done in-transit.
- $\check{x}_s^{soft}$: Demand for in-transit processing; session utility becomes negative (penalty) when percent of processing work done in-transit is less than $\check{x}_s^{soft}$.

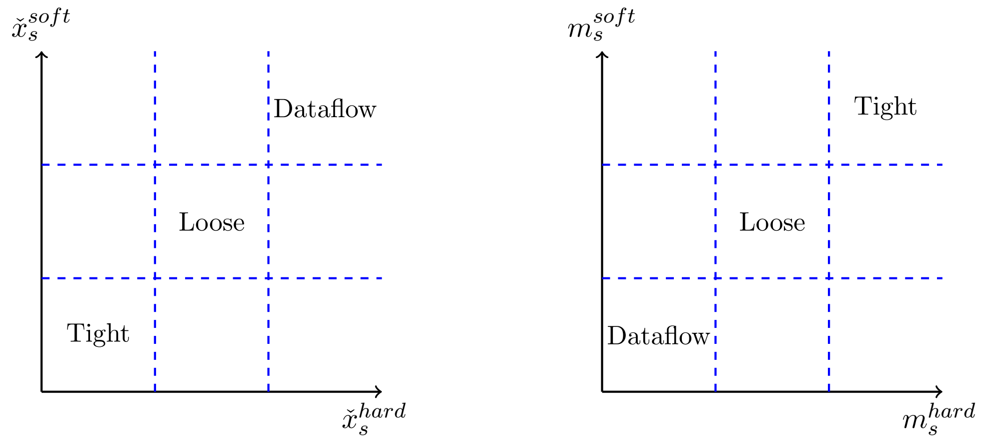


Figure 1.5: Mapping three different coupling types on two dimensional parameter space of m_s^{soft} vs. m_s^{hard} and $\check{x}_s^{soft}$ vs. $\check{x}_s^{hard}$

Overall, session penalty P_s is convex and session utility U_s is concave. The reason we used linear mapping function for session utility rather than piece-wise one as in session penalty, is to make U_s concave. The only effect of this is that $\check{x}_s^{soft}$ is a stronger parameter to increase demand for in-transit processing than $\check{x}_s^{hard}$ is to decrease demand for in-time delivery – if desired, linear mapping function can also be used in penalty function.

Based on the coupling type, these four configuration parameters must be autonomously tuned for each session. Figure 1.5 illustrates the relationship between the coupling types outlined in section 1.1.2 and the four configuration parameters. In short, tight coupling has more strict time constraints, data flow has more opening for opportunistic in-transit processing and loose coupling is in the middle between these two. For practical implementation, the domain of the configuration parameters can easily be limited and discretized.

1.4 Experiments and Results

To evaluate the framework described in this chapter, we used the "Plasma Disruption Analysis" workflow [22] from the KSTAR[3] project to develop synthetic use case

[3]Korea Superconducting Tokamak Advanced Research

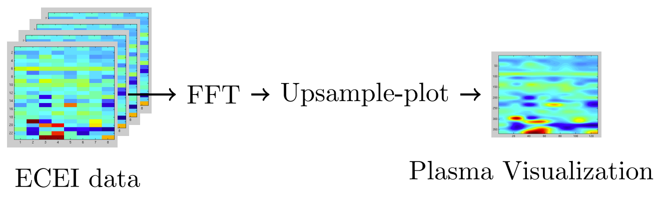


Figure 1.6: A schematic of the ECEI data and diagnostics workflow. The diagnostic routines do not change the size of the data, and comply with the optimization problem formulated in Section 1.3.3.

scenarios. In this application, plasma disruptions occur due to loss of stability and/or confinement of tokamak plasmas and cause a fast thermal and/or current quench within sub-milliseconds, which can damage the expensive (multi-billion dollars) tokamak device. As a result, finding precursors and early prediction of tokamak operation anomalies is a very active research field. One such research effort is the plasma visualization diagnostics system designed to provide direct 2D/3D visualizations of the plasma in a tokamak. Visual plasma images are obtained via monitoring technologies such as Soft X-Ray (SXR), Microwave Imaging Reflectometry (MIR), or Electron Cyclotron Emission Imaging (ECEI). Diagnostic routines are then run over the plasma images to detect precursors of plasma disruption.

For our experiments, we used data, diagnostic and visualization routines for ECEI (see Figure 1.6) to compose synthetic scenarios consisting of multiple coupling sessions with varying requirements sharing a service medium. ECEI generates high resolution 2D images of radiated electron temperature, which provide visualization of plasma instabilities – specifically, a 24x8 float matrix image is generated every 2μs, i.e., 5,000,000 images are generated in 10 seconds resulting in $\approx$ 3.5GB of data. The over reaching goal of KSTAR is to enable a remote scientist, for example, in the US, to monitor the plasma visualizations to monitor tokamak stability and to take regulatory actions if necessary. This requires diagnostic and visualization routines to be run either at US site or opportunistically over the available intermediate resources.

For our experimentation testbed, we implemented DT-OS by extending POX, which is a networking software platform [23], and the scheduler by using POX API. Further, we

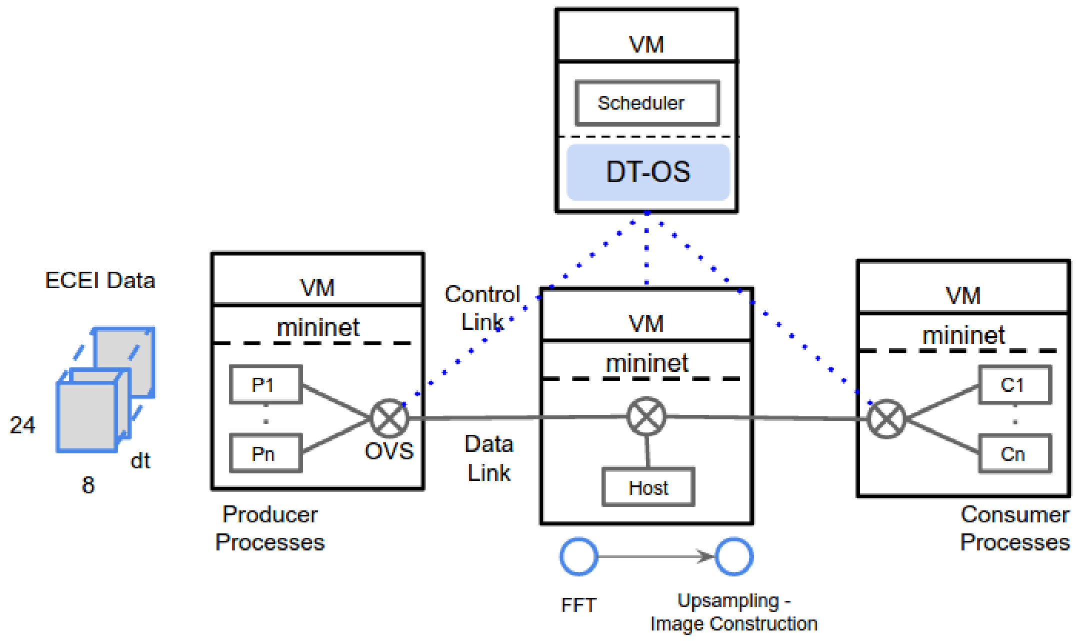


Figure 1.7: Schematic overview of the emulation testbed deployed across multiple VMs.

used CVXPY [24] as the modeling language for the optimization problem for resource allocation. Finally, we implemented the IT-SP layer on top of TCP at the producer, consumer and intermediate hosts.

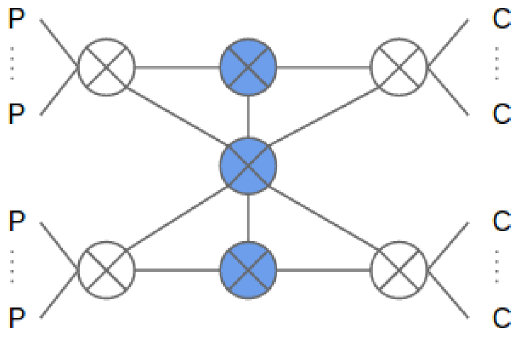
To emulate a coupling session with producer-consumer pairs, network switches-links, and in-transit hosts in our experiments we used mininet [25]. Running the entire system on a single machine is convenient but imposes limitation on switching and in-transit processing capacity. For example, if the machine has 3GHz of CPU, mininet can switch at most 3 Gbps simulated traffic. Moreover, overall available emulation capacity is shared by multiple coupling sessions and resources. Therefore, we used multiple mininets running on different VMs interconnected via GRE tunnels, as illustrated in Figure 1.7. We then deployed our testbed on a distributed OpenStack cloud as part of the FutureGrid testbed [26].

In our experiments, we used a use-case scenario which we obtained by monitoring the Fusion workflow, and adapted it appropriately for our virtualized emulation setup, which operates at a lower bandwidth (e.g. network links of 10Mbps) and processing rate (e.g. in-transit hosts of 100Mfps). We label the coupling sessions taking place during the workflow execution as tight, loose or dataflow based on the application characteristics i.e. demand for in-time delivery or in-transit processing for a particular coupling session. Relevant parameters of the scenario are summarized in Table 1.1, where arrival time is when application sends the request for the coupling session, data size is the size of data to be streamed over the coupling session and optimal transport time is the optimal time for completing data transfer and in-transit processing.

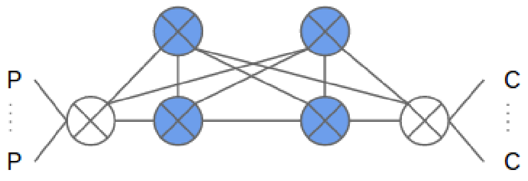
Session Id	Arrival Time (sec)	Coupling Type	Data Size (MB)	Optimal Transport Time (sec)
0	0	Loose	20	50
1	10	Dataflow	15	60
2	20	Tight	15	20
3	30	Dataflow	60	80
4	100	Loose	50	100
5	110	Dataflow	50	100
6	120	Dataflow	55	120
7	130	Loose	80	150
8	230	Tight	20	200
9	240	Dataflow	20	200
10	250	Loose	20	50

Table 1.1: Parameters for the Fusion workflow scenario.

We run the experiments on two topologies shown in Figure 1.8. Performance results of the experiments are presented in Figure 1.9 and 1.10. Figure 1.9a and 1.10a show that as expected from the coupling models discussed in Section 1.3.3, the amount of processing completed in-transit is maximum for dataflow couplings and intermediate for loose and the least for tight coupling sessions. More specifically, session 1-2, 8-9 are dataflow-tight and 4-5 is loose-dataflow coupling pairs and the order for in-transit processing completed is dataflow > loose > tight. Figure 1.9b and 1.10b show that the relative error between actual and optimal transport time can be slightly high for relatively short-duration couplings. This difference is mainly caused by the discrepancy between the models for data transfer and in-transit processing and the actual values



(a)



(b)

Figure 1.8: Two topologies used for the experiments. Filled switches are the ones with in-transit host connected.

but the average error is approximately 5% which is acceptable for the Fusion workflow execution. Figure 1.9c and 1.10c show that scheduling overhead is also naturally higher for relatively short-duration couplings, in this case inherent network link latencies seems to have most effect on the state setup time, and the average overhead is close to 3% which does not also disrupt Fusion workflow execution. Moreover, scheduling overhead can be masked by the application via sending request for data transport earlier than the actual starting time.

In the experiments, scheduling is initiated by the DTS every time a new request for data transport received from the application, and each scheduling run uses all the resource capacities available to achieve its optimization goal (i.e., highest cumulative satisfaction of application requirements). Figure 1.11 plots the logs of a network link and an in-transit host loads after every scheduling run; link capacity of 10Mbps and host capacity of 100 Mfps are fully distributed between active user sessions after every scheduling. Note that since we assume the network is exclusively used by the scientific workflows and the bandwidth is sliced between different coupling sessions by the scheduler, even though resources are exhaustively used there is no congestion in the network. This explains why the simple transfer completion model used in Section 1.3.3 works very accurately – the TCP windows rapidly converge to the optimal (allocated bandwidth) without suffering from congestion.

To show the performance of the introduced scheduling and control approach for a general use-case scenario, using the topology in Figure 1.8b, we run experiments with a random scenario where the coupling sessions arrive as a Poisson process of rate 0.04; one arrival every 25 seconds on average. Independent of the arrival process, every session is of uniformly one of the coupling types considered in this work; tight, loose and dataflow. Also independent of the coupling type, every session streams data of size uniformly distributed between 10MB and 100MB, and has optimal transport time which is Gaussian given the data size. One realization of such scenario is given in Table 1.2. Figure 1.12 shows the results. As in Fusion workflow scenarios, bandwidth, processing rate and staging capacity are distributed for each coupling session to maximize the specific demands in terms of in-time delivery and in-transit processing. Discrepancies between modeled and actual transport times are around %5 on average, scheduling overhead is around %2 on average.

1.5 Conclusions and Future Work

This chapter presented the architecture and design of a scheduling and control framework for data transport service management of data intensive scientific workflows which is also partially published in [1]. Our framework leverages software-defined networking (SDN) to address issues of data transport service control and resource provisioning to meet varying QoS requirements from multiple coupled workflows sharing the same service medium. Furthermore, we addressed both, data transfer and in-transit processing/staging services. The presented framework has three key components: (1) A flow-based opportunistic in-transit processing/staging service model, (2) SDN-based centralized architecture for data transport service control, and (3) Application and service medium aware scheduling. Overall, the framework attempts to address complex, dynamic and varying data transport requirements of coupled application workflows by

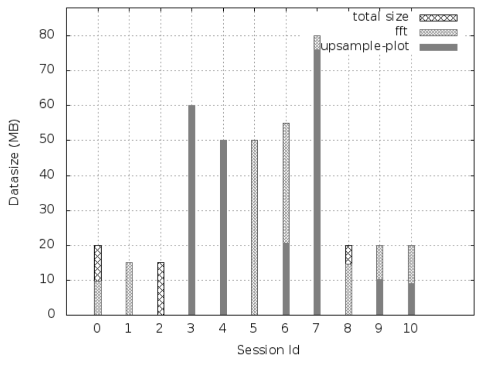
Session Id	Arrival Time (sec)	Coupling Type	Data Size (MB)	Optimal Transport Time (sec)
0	4	Loose	88	118
1	40	Dataflow	71	106
2	60	Tight	88	145
3	80	Dataflow	67	95
4	220	Loose	30	44
5	238	Dataflow	70	108
6	285	Dataflow	66	122
7	293	Loose	84	125
8	328	Tight	49	80
9	335	Dataflow	62	101
10	376	Loose	53	87

Table 1.2: Parameters for the random scenario.

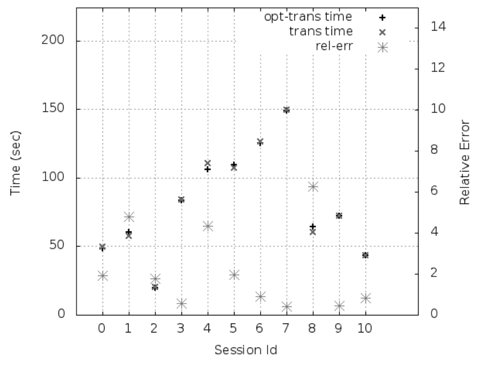
scheduling bandwidth and in-transit processing/staging capacity. Scheduling is formulated by using intuitive models for data transfer and in-transit processing/staging, and implemented as a disciplined convex optimization problem.

We also presented an experimental evaluation using an emulated SDN testbed on top of the FutureGrid virtualized testbed; for the evaluation we used synthetic application workflow scenarios (derived from real Fusion simulations workflows and random processes) to demonstrate that our framework can meet the coupling requirements and achieve high resource utilization.

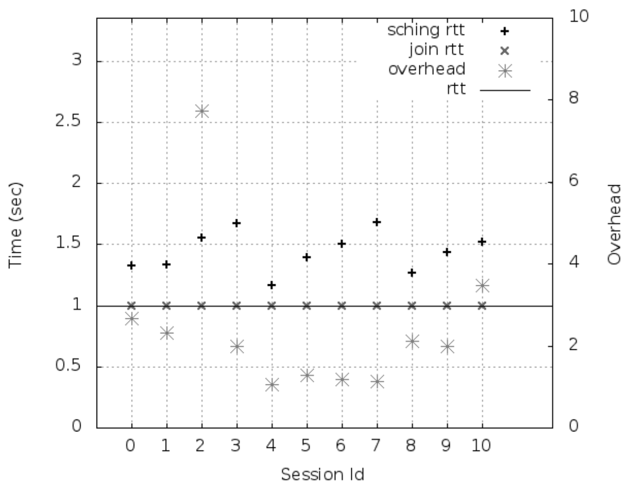
Our current work is focused on deploying the framework in a HPC Grid environment and using it to support real application workflows. We are also working on getting the optimization problem to work for more complex situations, such as with in-transit routines which change data size.



(a) Size of the total delivered data and the size of the portions that are opportunistically processed.

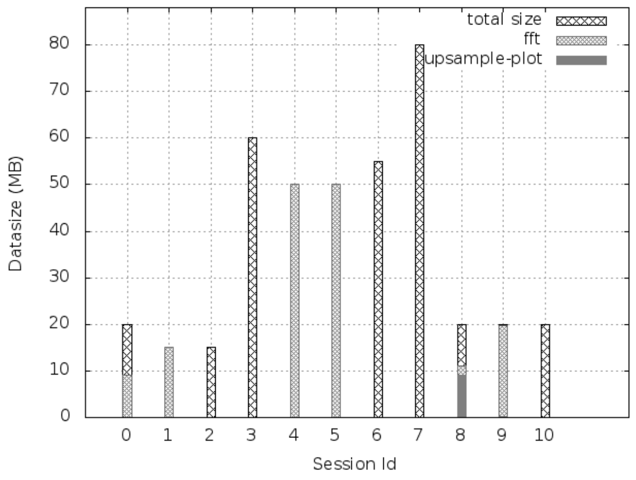


(b) Actual transport time: *trans time*, optimal transport time: *opt trans time* and relative error between actual and optimal transport time: *rel-err*.

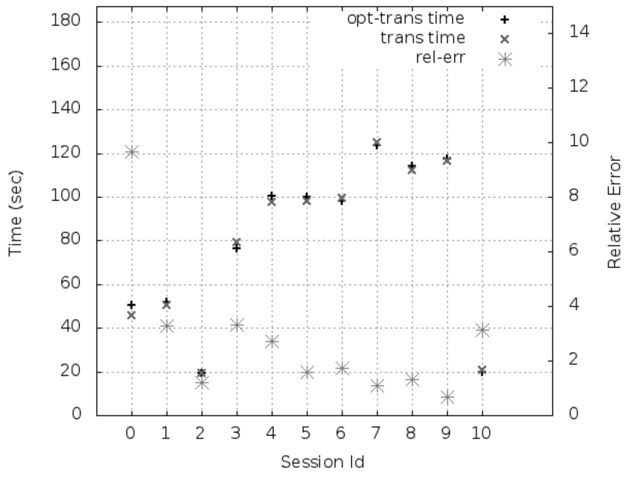


(c) Time between join request-reply: *join rtt* and scheduling request-reply: *sching rtt*. Scheduling overhead: *overhead* is the relative error between *sching rtt* and actual data transport time. Horizontal line: *rtt* is the round-trip time between session producer and scheduler due to network link latencies.

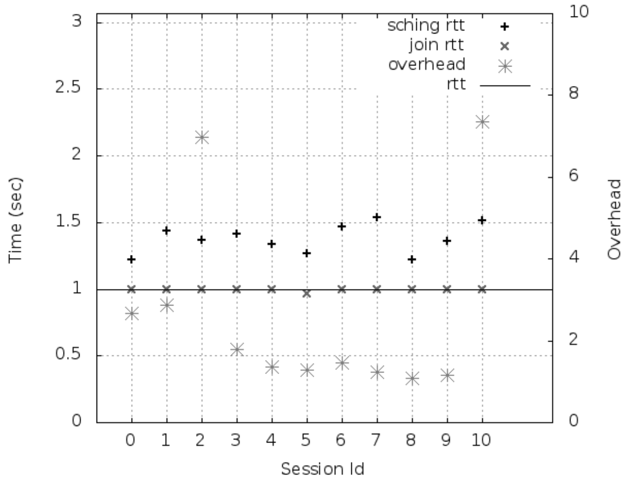
Figure 1.9: Results for the first set of experiments using the topology in Figure 1.8a.



(a)

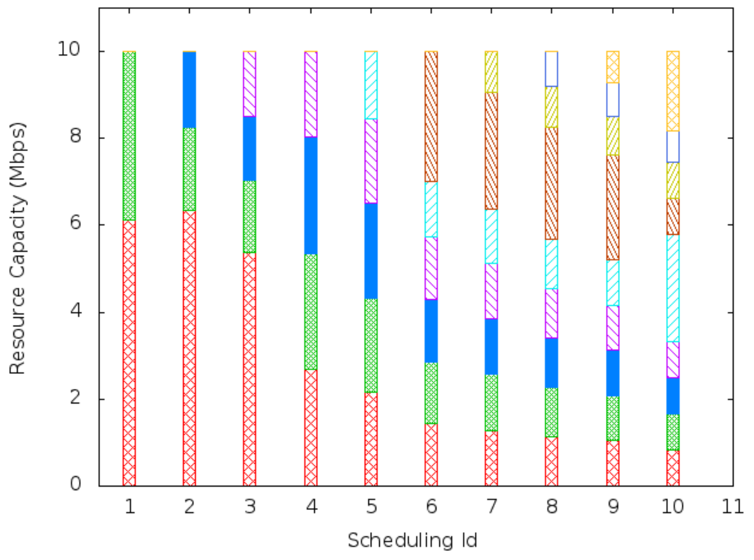


(b)

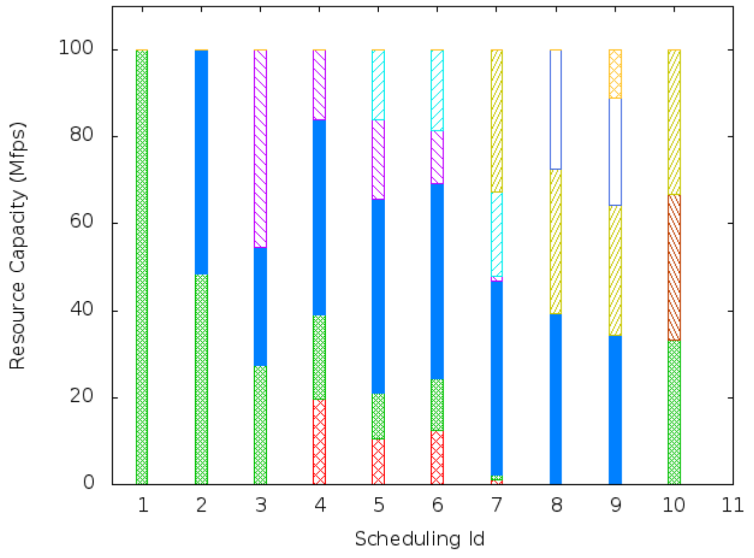


(c)

Figure 1.10: Results for the second set of experiments using the topology in Figure 1.8b.

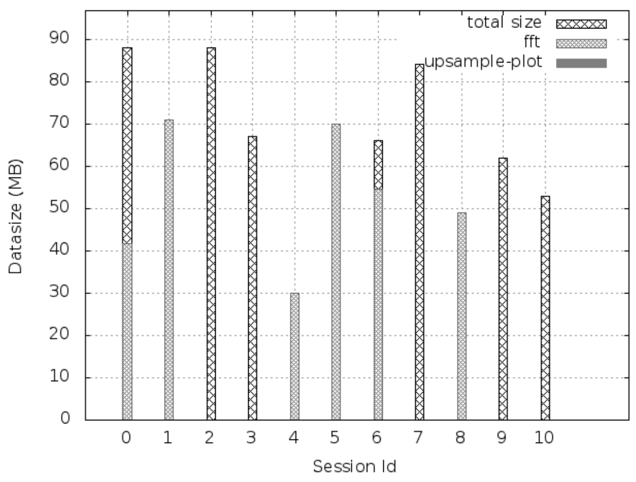


(a) Distribution of bandwidth allocation for a network link.

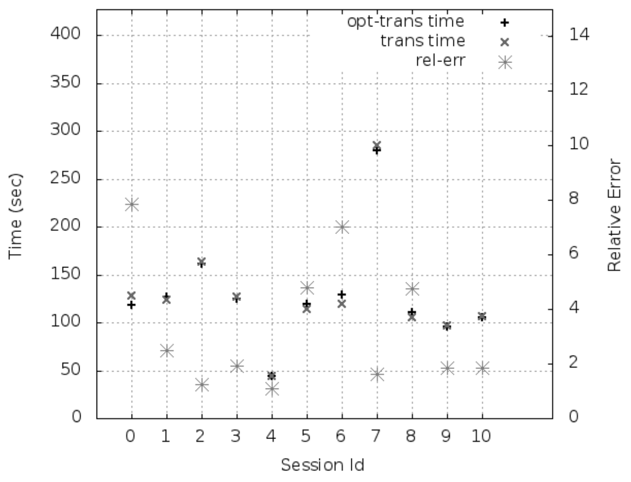


(b) Distribution of processing rate allocation for an in-transit host.

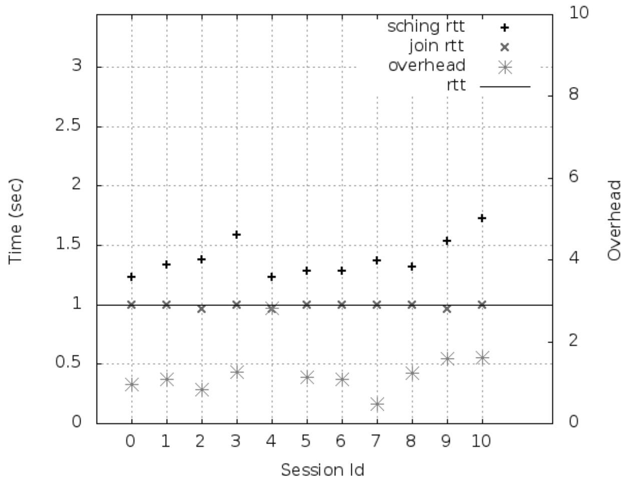
Figure 1.11: Distribution of the allocation of link and host capacities to active coupling sessions after each scheduling run. Every stacked bar corresponds to allocation to a single session.



(a)



(b)



(c)

Figure 1.12: Results for the experiments using the random scenario in Table 1.2 and topology in Figure 1.8b.

Chapter 2

Wide-Area Data Staging

2.1 Introduction

2.1.1 Motivation

Dataspaces provides a shared space abstraction to scientific applications. Shared-space is modeled as tuple-space which can accessed with flexible interaction interface of put/pub(key:data) and get/sub(key) – it enables asynchronous coordination and data sharing between coupled applications. Dataspaces can support only local-couplings (i.e. couplings on LAN e.g. processes running on two different nodes of a cluster) since shared-space abstraction is implemented using the memories of local staging nodes.

Depending on the need for computational or storage capacity and resource availability, applications may have to be run on resources distributed over wide-area. With this project, we are investigating if we can extend Dataspaces flexible programming system to enable wide-area couplings (i.e. workflow couplings over WAN). A programming system like Dataspaces for wide-area scenario would enable scientists to work with large-scale workflows using a simple put/get based API and then can run their applications over the resources distributed over WAN. Moreover, since Dataspaces framework achieves high data insertion and retrieval performance via in-memory data staging, extension of the principles of Dataspaces is promising for wide-area data staging scenario as well.

2.1.2 Problem Description

Goal of this project is to extend Dataspaces abstractions for in-memory data staging for enabling efficient coupling between applications running on resources distributed over wide-area. Dataspaces framework has been used in scientific discovery since 2010 [27]. Its flexible API has been proven to be developer-friendly and efficient over the years in terms of the needs for scientific applications. To mention some instances of its usage: (1) Couplings between the gyrokinetic PIC edge simulation code XGC0 and the MHD code M3D-OMP, and also direct numerical simulations (DNS) code S3D and the data analytics pipeline of the turbulent combustion workflow, is implemented using Dataspaces, (2) Dataspaces is an integrated part of Adaptive IO System (ADIOS) framework distributed by Oak Ridge National Laboratories (http://www.olcf.ornl.gov/center-projects/adios/), which is an open source I/O middleware package that has been shown to be highly scalable (i.e., scales to hundreds of thousands of cores) and is being used to enable many scientific workflows. Therefore, first challenge while extending Dataspaces to wide-area would be preserving Dataspaces' flexible API.

Workflows are composed by multiple applications running independently and coordinating with each other based on their runtime state. Coordination between applications is required to be handled in a timely manner for the sake of finishing the whole task workflow is running to solve, as fast as possible by minimizing the duration that applications stay in the idle state. Equivalently, one can state this requirement as: scientific applications are required to utilize computational resource capacities exhaustively. This translates into the following requirement in data management layer, which is data transport between peer applications should be fast, robust and efficient so that computational resources can be exhaustively utilized by the workflow components. Equivalently, this requirement can be thought as minimal overhead to the workflow execution due to the application couplings where overhead is defined as the total duration that applications stay in the idle state waiting for the completion of data transport.

Another challenge connected to the minimal overhead requirement discussed in the previous paragraph is to achieve scalable performance with respect to both data and problem size of the workflow. This is very crucial for scientific workflow execution since application couplings are needed to realize over wide spectrum of data size, and often frequency and requirements of application couplings vary (see SubSection 1.1.2 for more detail). Dataspaces framework achieves scalable data insertion and query/retrieval performance by using in-memory data staging and novel distributed hash table approach as further discussed in Section 2.2.

2.1.3 Overview and Approach

Following NUMA-like memory design, we implemented a wide-area Dataspaces abstraction by interconnecting multiple Dataspaces instances over WAN as illustrated in Figure 2.1. This architectural choice is mainly because of the large distance between each Dataspaces for WAN scenario. In this case, choice of designing the shared memory abstraction with a UMA-like approach would have required the revaluation of Dataspaces principles and abstraction, and possibly result in the re-implementation of most of the framework. NUMA-like architecture allows the design and inner workings of individual components of the whole system be decoupled from each other; e.g., optimization of data distribution in local (i.e., in each Dataspaces) and wide area scale (i.e., between Dataspaces instances) can be addressed independently from each other in terms of semantics and dynamics of the solution.

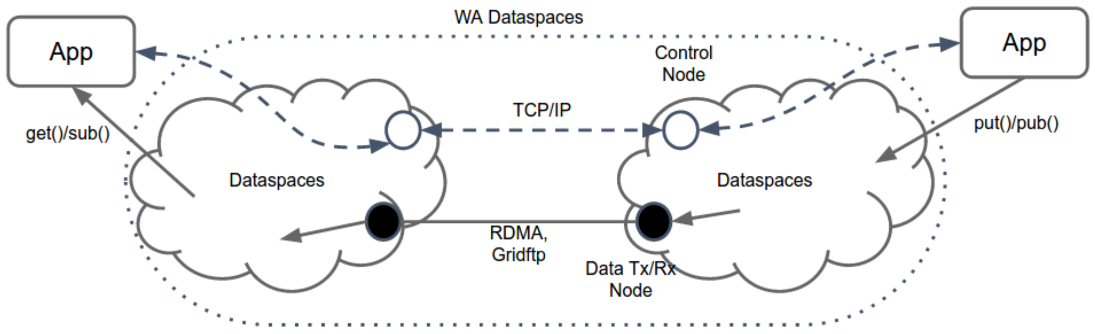


Figure 2.1: Organizational representation of Wide-Area Dataspaces.

Considering one-dimensional circular node identifier space, individual Dataspaces instances act like individual staging nodes storing the keys mapped to the segment of the curve between itself and the predecessor node (Figure 2.2, Left), then each Dataspaces stages its segment over locally available staging nodes (Figure 2.2, Right). Since in a typical scenario, the number of Dataspaces to be interconnected would be low, constant-time (i.e., O(1)) peer-to-peer lookup between Dataspaces is feasible. WAN can negatively affect the performance of data lookup and data transport, which can be addressed by (1) using local caching of remote queries and predictive queries before the actual application request to decrease the negative effect of WAN on data lookup, (2) supporting high-performance data transport technologies and protocols like GridFTP and RDMA to achieve faster data transport between Dataspaces, and (3) prefetching remote data to local Dataspaces before the application makes a get request for it. These three items will be expanded in Section 2.3

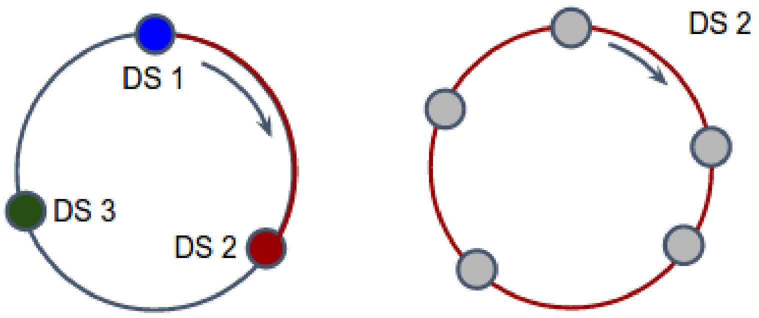


Figure 2.2: Representation of wide-area Dataspaces on one-dimensional circular node-identifier space (Left). Each Dataspaces stores preceding segment on the key space over the resources distributed in its locality (Right).

2.1.4 Contributions

The goal of this project is to explore NUMA-like architecture for extending Dataspaces principles and abstractions to wide-area scale, and implement and test performance of varying policies and strategies for distributed data management in wide-area scale. Current implementation supports non-blocking Dataspaces API of put/get data and also additional blocking get. Non-blocking get routine returns no matter if data is available or not; returns data if available or returns false otherwise, blocking get blocks the execution of applications until the data is available and finally returns data. Blocking get helps application developers to listen for data availability using multi-threaded programming; a thread to listen data can be created and a callback function may be called once the data is available to the application. Specifically, embracing NUMA-like architectural choice for designing a big shared memory abstraction to applications, we implement an O(1) peer-to-peer asynchronous overlay network between Dataspaces instances to carry control messages. Then each Dataspaces communicate with each other over this control network through a JSON-based control protocol which defines the semantics of data distribution in wide-area scale. Data distribution is managed according to two following policies (1) Demand-based: If requested data is not available in a local Dataspaces it is transported and copied into the local one from the remote Dataspaces where the data is available, if data is not available and a blocking get request is done by the application, requested data is copied into the local one once it becomes available in any of the remote Dataspaces instances, (2) Prefetching: Data to be requested next is predicted using the history of access sequence initiated by the application and prefetching buffers in the local Dataspaces is filled with predictive data entries. Within our limited time, we focused on data prefetching strategies and evaluated the performance of our implementation of a Markov-based prefetching strategy to enable couplings of Fusion workflow and also random workflows.

2.1.5 Outline

The rest of this chapter is organized as follows. Section 2.2 provides some background and summarizes the related work. Section 2.3 presents our approach for wide-area Dataspaces architecture and data prefetching for improved data retrieval time. Section 1.4 describes our experimental setup and presents the experimental evaluation of our approach. Section 2.6 concludes the chapter, and outlines ongoing and future research.

2.2 Background and Related Work

2.2.1 Dataspaces

Dataspaces is a distributed interaction and coordination substrate for scientific workflows [28]. Dataspaces has a client-server architecture; an application using Dataspaces runs on two components; *DataspacesServer* and *DataspacesClient*. Server component runs separately and independently from the user applications on the dynamic set of cluster nodes assigned for staging and provides the shared space abstraction to the client components. Client components are integrated with the user application and provides the interfaces for interacting with the server. User application implements application couplings within the workflow by using this flexible API provided. Dataspaces' interface supports non-blocking and re-entrant put and get data operations, which enable the application to put and get data asynchronously to and from the shared space. As outlined in Listing 2.1, put and get operations correspond to *dspaces_put* and *dspaces_get* routines.

Listings 2.2 and 2.3 show sample data producer and consumer applications using Dataspaces client component. As the names explain itself, in this sample scenario producer application is producing and sharing data with the consumer application. Since data put into the server is shared *dspaces_lock* and *dspaces_unlock* calls guarantee exclusive access on the data for put and get operations. Additional to the name and the version of the data, Dataspaces uses a geometric descriptor as the input from application to *dspaces_put/get* routines to describe the bounding box identifying the range of the data that application wants to put or get. For example, data put to space can be identified in 3-dimensional space as the cube with volume diagonal from (0, 0, 0) to (x, y, z) and consumer application can get only a part of this data (or all of it) e.g., range identified as the cube with volume diagonal from (x/3, y/3, z/3) to (x/2, y/2, z/2).

```
/* Insert the data specified by the geometric descriptor and
 * pointed by var_ref. */
int dspaces_put(char* var_name, int version,
                <geometric descriptor>, void* var_ref);


/* Retrieve the data specified by the geometric descriptor and
 * pointed by var_ref. */
int dspaces_get(char* var_name, int version,
                <geometric descriptor>, void* var_ref);
```

```
// Request exclusive access to the specified data
void dspaces_lock_on_[read/write](const char *lock_name);

// Release the exclusive access to the specified data
void dspaces_unlock_on_[read/write](const char *var_name);
```

Listing 2.1: Dataspaces API

```
...
// Some computational routine producing the data
void* var_ref = produce();

dspaces_lock_on_write(var_name);
dspaces_put(var_name, version,
            <geometric descriptor>, void* var_ref);
dspaces_unlock_on_write(var_name);
...
```

Listing 2.2: Data producer example using Dataspaces

```
...
dspaces_lock_on_read(var_name);
dspaces_get(var_name, version,
            <geometric descriptor>, void* var_ref);
dspaces_unlock_on_read(var_name);

// Some computational routine consuming the data
consume(var_ref);
...
```

Listing 2.3: Data consumer example using Dataspaces

Dataspaces' architecture is layered and consists of data communication, data storage and data lookup layer as illustrated in Figure 2.3. Data communication layer leverages standard communication technologies like TCP/IP and also advanced communication technologies like RDMA for low-latency, high-throughput data transfers. Data communication layer enables asynchronous data transport between application and staging nodes, and between staging nodes. Data storage layer manages the memory of staging nodes and creates a shared, unified, in-memory storage space. Data lookup layer translates geometric descriptors input by the application to DHT indexes and routes application queries to the nodes responsible for staging the queried space.

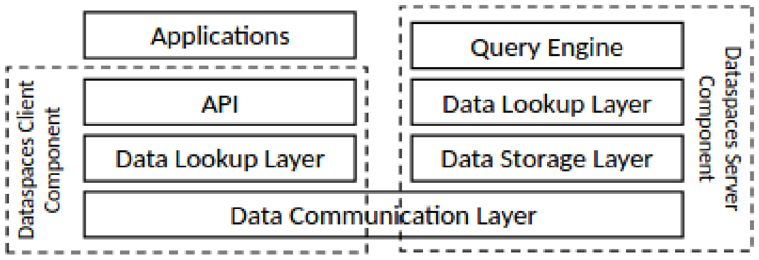


Figure 2.3: Conceptual view of Dataspaces architecture.

Dataspaces uses distributed hash table (DHT) for fast data lookup operations. The principle behind Dataspaces' DHT system is explained in great depth in [29] which is the paper introducing framework Squid enabling search in DHT-based systems. In Dataspaces, index for DHT is derived from the geometric descriptor of the object put into space by the application. This index is used as the address space of the shared space and also it is used to map range of data on the staging nodes by the Dataspaces server. In particular, for multiple dimensional data, Hilbert spaces-filling curve (SFC) is used to generate to index the data. SFC is basically the trajectory of a point traveling every point in the multi-dimensional volume without visiting any point twice. In other words, SFC is a continuous function mapping a multi-dimensional volume to a single-dimensional one. One important feature of Hilbert SFC, which is the main reason picked for indexing data put to Dataspaces server, is that it preserves data locality i.e., points close to each other in multi-dimensional space is mapped to adjacent or points close to each other in single-dimensional space. To sum up, Dataspaces indexes multi-dimensional application data with a single-dimensional curve, which is used to map and distribute ranges of data to different staging nodes in the cluster.

2.2.2 Prefetching

Prefetching is used as a way of masking the latency of bringing data to the consumer component. Prefetched data usually gets kept in an intermediate staging area, which is small (not enough to fit all data of interest) and consequently faster (compared to the actual storage area) in terms of data insertion, lookup and retrieval, and it is usually named as cache or buffer depending on the context. Prefetching is heavily used in hardware or software level in wide range of contexts to improve the overall performance of the system. For example, instruction caches are used to speedup execution in microarchitecture context, link prefetching is used to improve web browsing experience, prefetch buffers are used within many modern OS to load machine code from disk into the memory in advance to speedup core operations and many others within memory, database, and in general, system management context. In this work, predictive prefetching is used to bring remote data to local staging buffer to decrease the time application experiencing to get data from wide-area Dataspaces.

There is a large literature about prefetching in the microarchitecture and the database context. One of the most interesting techniques is using data compression techniques for achieving optimal prefetching, which is introduced in [30] and is explained with good practical examples in [31]. The intuition is that data compressors typically operate by postulating a dynamic probability distribution on the data to be compressed. Data expected with high probability (low entropy) are encoded with few bits, and unexpected (high entropy) data with many bits. Thus, if a data compressor successfully compresses the data, then its probability distribution on the data must be realistic and can also be used for effective prediction. Assuming the prefetching buffer can buffer at most some given number of pages [1], it is shown in [30] that any optimal character-by-character compressor for strings can be converted to a prefetcher that has an optimal (minimum) page fault rate.

[1]In database context, word *page* is used to refer to data to be accessed.

The cache architecture discussed in [32] represents the commonly used architectural model for predictive prefetching. It describes a predictive cache that uses associative memory to recognize access patterns and does online prediction and prefetching accordingly. Introduced predictive cache consists of two main components: (1) Predictor; the component that recognizes and learns the pattern in the access sequence and predicts the next access, (2) Cache replacement manager; the component that applies a policy for managing the replacement of prefetched objects within the arrival of newly prefetched objects. As discussed in Section 2.3, we use a similar architecture for predictive prefetching remote data within wide-area Dataspaces framework.

A predominantly used technique to recognize access patterns and make predictions accordingly is modeling access sequence as a Markov process. In this approach, set of pages that can possibly be accessed form the set of Markov states. State transition probabilities are computed and dynamically updated online using the observed access sequence. Markov chain and the extracted transition probabilities are then used to find out likelihood of particular patterns to emerge in the future, which is used to predict future accesses. Usually a "hit or miss" [2] cost function is used as the objective to minimize while optimizing the prediction outcome, which tells to predict the access pattern with maximum likelihood. Within wide-area Dataspaces framework, we also use Markov models for predicting future data to be accessed by the application and a formal description of our formulation will be given in Section 2.3.

Web prefetching (i.e., prefetching links in world wide web (WWW)) is an interesting field to research for getting ideas about prefetching within wide-are shared space abstraction context. Some similarities between two fields regarding prefetching are as follows: (1) WWW is a distributed system that provides data in varying formats like text, image, stream etc. to users via servers all of which are interconnected over WAN. Wide-area shared space abstraction aims at providing a similar distributed data access service to applications via multiple Dataspaces instances all of which are interconnected over WAN, (2) Access sequence in WWW generated by the users is sequence of links

[2]For hit or miss function $C(\epsilon)$ where ϵ is error between the true and predicted (or estimated) value of a parameter, $C(\epsilon) = 1$ for $|\epsilon| > \delta$ for $\delta > 0$ and $C(\epsilon) = 0$ otherwise.

which is similar to tuple space model of Dataspaces in that data is accessed using key and version numbers. Similar arguments could be made for comparing wide-area shared space abstraction with large database systems.

Probabilistic models are applied extensively in the field of world wide web for prefetching links, especially Markov models have been the focus as further discussed in [33]. First-order Markov models are proposed for using speculative document prefetching in WWW in [34]. Later it is shown that even though first-order models are good formulations for recognizing and computing the probabilities for patterns in the access sequence, they cannot achieve high predictive accuracy because of the usage of limited information about the past which can be solved by using higher order Markov models which enable building models looking at as far in the past as enough to make an accurate prediction. As an example application of higher order Markov models, Ngram model is employed for mining user behavior patterns in [35] and shown to display better predictive accuracy. However, higher order models result in (1) higher state-space complexity due to exponentially increasing number of states with the model order, (2) Many states in higher order model may not find a use case in the test set, which reduces the coverage of the model; especially short-duration user access sessions can suffer from low coverage dramatically, (3) Length of the access sequence should be longer to compute transition probabilities for higher order models which may decrease the predictive accuracy given a training set of limited size. Low coverage problem of high order Markov models is solved by "All-Kth Order Markov Model" introduced in [36] which combines varying order Markov models for higher predictive accuracy and coverage. However, using multiple variable-order Markov models increases the state-space complexity even further which is proposed to be tackled in [37] by selective Markov models in that multiple varying order Markov models are combined intelligently in a way to keep state-space complexity low while achieving high predictive accuracy.

2.3 Extending Dataspaces Abstractions to Wide-Area Scale

2.3.1 A conceptual overview of Wide-Area Dataspaces

Wide-area Dataspaces (WA-Dataspaces) is a distributed framework which extends flexible API of Dataspaces to enable application interaction and coordination couplings over wide-area network. It preserves the semantically specialized (i.e., addressing of the data is specialized using geometric descriptor by the application as described in SubSection 2.2.1) virtual shared space abstraction of Dataspaces. Goal of WA-Dataspaces is to provide efficient put data to the space and get data from the space services. Following NUMA-like design, WA-Dataspaces consists of multiple Dataspaces that can dynamically join the space and are orchestrated to create the virtual shared space abstraction in that data put and get operations are realized.

2.3.2 Architecture and Implementation overview

WA-Dataspaces has client-server architecture as Dataspaces. It is built on top of Dataspaces stack and has a layered architecture as illustrated in Figure 2.4. Client components are integrated with the user application as in Dataspaces framework, and provide an API for wide-area data insertion and retrieval similar to Dataspaces API. WA-Dataspaces server is a combination of Dataspaces server, a Dataspaces client extended with wide-area data management stack. Server component runs independently from the user applications and provides the wide-area shared space abstraction to the client components.

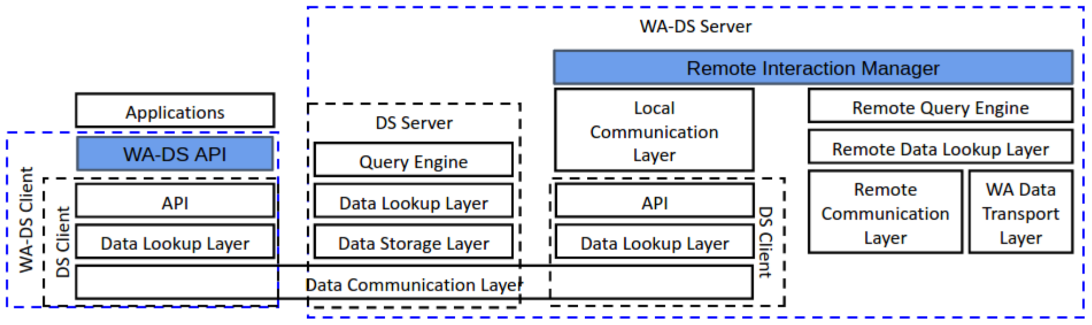


Figure 2.4: Wide-Area Dataspaces Architecture. DS: Dataspaces, WA: Wide-Area

As described in SubSection 2.2.1, Dataspaces API is non-blocking in that put or get operations return immediately to the application. WA-Dataspaces API is almost same with Dataspaces API except it also provides a blocking get operation, in which application thread making a blocking get call is blocked until the requested data becomes available and then is returned to the application. Blocking get aims to enable applications to access the data in a more timely and asynchronous manner. For example, if an application requires a particular data set to advance with a part of the execution, while doing some other task in the main thread, a listener thread can be created to make a blocking get for the data of interest which would be used a call back mechanism to the application once the data becomes available and retrieved from the space. Such a behavior of listening for a particular data set cannot be possible to implement in a asynchronous manner with a non-blocking get; a listening loop may be used which would cause unnecessary burden on both application and the system. As outlined in Listing 2.4, put and get operations correspond to *wa_dspaces_put* and *wa_dspaces_get* routines. In addition, we made the locking operations of Dataspaces API implicit in the put and get routines so application does not need to worry about exclusive access to the shared data because it is guaranteed by WA-Dataspaces framework.

```
/* Insert the data to wide-area shared space which is specified
 * by the geometric descriptor and pointed by var_ref. */
int wa_dspaces_put(char* var_name, int version,
                   <geometric descriptor>, void* var_ref);


/* Retrieve the data of type var_type from the wide-area shared
 * space which is specified by the geometric descriptor and
 * pointed by var_ref in a blocking or non-blocking manner. */
int wa_dspaces_get(bool blocking, char* var_name, int version,
                   <geometric descriptor>, void* var_ref);
```

Listing 2.4: Wide-Area Dataspaces API

Most important component of the WA-Dataspaces server is the "Remote Interaction Manager" (*RIManager*), which realizes the wide-area shared space abstraction. *RIManager* is roughly a Dataspaces client which is extended with the wide-area data management stack. There are important tasks that *RIManager* accomplish to achieve the wide-area shared space abstraction, which are (1) Application-Space coordination, (2) Inter-Dataspaces coordination. Unlike Dataspaces server component, *RIManager* is highly multi-threaded to be able to provide asynchronous and efficient service.

Communication between the user applications and *RIManager* is implemented by the "Local Communication Layer (LCL)" which defines the primitives for exchanging control messages. LCL is implemented on top of Dataspaces client and it enables asynchronous communication using custom locking Dataspaces server provides, which enforces writer/reader synchronization; locks for writing and reading shared data can be acquired always in order such that write should come before read, and the read should happen before the next write. This blocking nature of Dataspaces server's custom locking enables LCL to implement asynchronous communication primitives.

RIManager connects one Dataspaces to another over a peer-to-peer asynchronous overlay network which we built on top of TCP/IP, which together with a custom defined JSON-based communication protocol implements the "Remote Communication Layer (RCL)". RCL is a crucial part of the overall architecture since it defines the communication primitives for control messages and for exchanging them between between Dataspaces instances. RCL primitives enable asynchronous communication between multiple Dataspaces instances. Within the same level of architectural hierarchy, "Wide-Area Data Transport Layer (WA-DTL)" resides and implements efficient transport of application data between Dataspaces peers. WA-DTL leverages advanced network technologies such as RDMA, legacy network stack of reliable data transfer (TCP/IP), and also a high-throughput data transport stack GridFTP [38]. Unlike LCL, RCL implements synchronous communication primitives. One main reason for synchronous implementation of data transport services is to manage resources efficiently such that data transport resources (e.g., communication ports, RDMA memory) are used whenever it is required.

Since at runtime space is not aware of future application actions, default data placement policy of WA-Dataspaces is inserting the data put by the application to the local Dataspaces server. As discussed further in Section 2.4, data is moved between Dataspaces instances with demand and/or predictive prefetching. "Remote Data Lookup Layer (RDLL)" implemented on top of RCL and WA-DTL extends Dataspaces' data lookup principles and operations to wide-area scale by managing the metadata. Metadata in this case is the availability and the location. RDLL associates every data element put into space with the id of the local Dataspaces data is inserted with default behavior. Once the data is moved or copied to other Dataspaces instances metadata is updated accordingly and shared with every peer. "Remote Query Engine (RQE)" builds on top of RDLL and provides a simple API for both asynchronous and synchronous data query over the wide-area shared space. Using this API of RQE, *RIManager* implements O(1) data lookup which checks and returns the availability and the location of the queried data in a very fast manner (i.e., constant time). To briefly describe how O(1) data lookup is implemented by *RIManager*, whenever a new data element is put into space, availability and location information relevant to the data is broadcasted to all Dataspaces peers which update their state accordingly. This way, metadata for data lookup always becomes locally available for fast data lookup operations. In case an application queries for an unavailable data set and makes a blocking get request, every Dataspaces instance is programmed to transport the data to the Dataspaces local to the application once it becomes available.

Consistency models for shared distributed memory systems are essential to achieve predictable results for memory operations [39]. Consistency of the shared data is ensured by these models which are basically offline contracts between the system and the programmer. Dataspaces employs a high-level consistency model in two levels: (1) Data communication layer ensures coherency of shared data by DHT created from the geometric descriptors of the data, (2) DHT does not govern the dynamics of data access which may create coherency issues (e.g., two applications using Dataspaces, using the same data descriptor, one may want to write and the other may want to read at the same time, or both may want to write simultaneously) which is addressed by locking operations as described in SubSection 2.2.1. Currently, WA-Dataspaces does not employ any consistency model beyond Dataspaces' consistency model which ensures data coherency within Dataspaces servers; we assume applications are aware of the lack of consistency model for wide-area shared space scenario and should avoid the following operations for data shared between applications coupled over wide-area (i.e., applications are running on sites remote from each other): (1) Overwriting a data set put into the space by another application, (2) Multiple writing by using the same data descriptor simultaneously. While this assumption of application developer awareness is not unreasonable and can be accommodated by the application but addressing data coherency inside the framework can potentially increase the practicability and overall efficiency of the framework (e.g., wide-area data placement policies discussed next SubSection would be required to change with the consistency model employed).

2.4 Wide-Area Data placement

Default behavior of WA-Dataspaces framework is to initially place the data to the Dataspaces server which is local to the application putting the data. Therefore, if a remote application makes a get request for a particular data, if it is (or when it is in case of blocking get request) available in a remote Dataspaces server, it is copied to the Dataspaces server which is local to the application requesting it. This default behavior of data fetching from a remote site based on the application demand is called *demand-based fetching.* Even though, demand-based prefetching alone enables applications to implement asynchronous interactions over wide-area, it is inefficient in terms of enabling the interaction since it reflects the overhead of data movement to the application execution i.e., the time between get request and data retrieval is large due to slow data transport. This is bad for performance especially in the case of workflows requiring application coupling over WAN with large data because bulk data transport over WAN is inherently slow and in this case execution time mostly gets dominated by the data transport time. Overhead due to data transport between remote and local servers is impossible to eliminate when the difference between the time that a particular data is put and get request is made, which we will call as slack time, is small. In this section, we will consider the case interesting from research perspective in which slack time is enough for moving data close to application before the application makes a get request for it.

The optimal strategy in this case would be to replicate the data to all of the Dataspaces servers once it is put into space so that when a get request is made, the data would be locally available. However, staging capacity at each site may be different and not enough for staging all the data in each Dataspaces server. Especially considering that scientific applications usually work on big data, in-memory staging capacity of modern clusters are usually not enough to stage all of the application coupling data within memory and even though one site may have enough memory available it is very unlikely that all the sites workflow is running on will meet this requirement. As discussed in SubSection 2.2.2, size-limited prefetching buffers (or cache) can be used for hiding the latency of data access by bringing the data of interest close to the consumer before the consuming state actually starts (i.e., in-time). In our case, every Dataspaces server has a prefetching buffer to its every peer (see Figure 2.5), which tries to hide the wide-area data transport latency by effective prefetching.

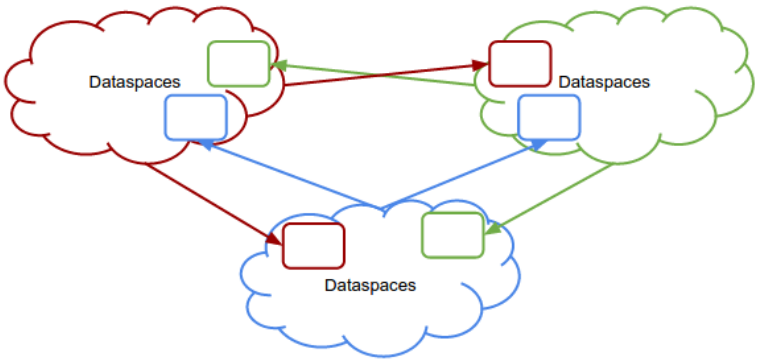


Figure 2.5: Representation of prefetching buffers allocated in the peers of each Dataspaces.

The goal of prefetching is to keep the buffer filled with the correct data in the right time. Here we will use database terminology in that page will be used for data elements and page fault defines the event when the page is not available in the local buffer once a get request is made for it (like cache miss). Rephrasing the research problem, limited size of the prefetching buffers should be filled with the correct pages which are going to be requested soon. Finding out which page to prefetch is a prediction problem.

It is shown in [30] that a prefetcher designed by converting character-based data compressors is optimal for a Markov source in the sense that minimal page fault rate is achieved as the number of observed page accesses goes to infinity. Source in this context used as the generator of page accesses. As presented in [31], converting a character-based data compressor to a pure prefetcher can be done quite simply. For example, an algorithm called LZ is introduced for prefetching which is based on the character-based version of the Lempel-Ziv algorithm [40] for data compression. LZ prefetcher is basically using the probability model built by an encoder which is using the Lempel-Ziv algorithm on the observed access sequence, and prefetching is done for the page which is maximizing the likelihood of being the next access in the sequence.

Although LZ prefetcher has a practical implementation and it is theoretically optimal in the limit [41], algorithm cannot converge to probability model which is giving the optimal prediction fast enough. In WA-Dataspaces scenario, we can say by inspecting the behavior of HPC applications that fast convergence is rather more important than optimal prediction in the long run since usually number of couplings to be executed over wide-area is much lower compared to other applications of prefetching such as web link prefetching or instruction caching. Also the penalty of a page fault is high due to slow data transport over wide-area i.e., workflow execution usually stalls until the required data is retrieved to continue with the next computational task. Following a similar motivation, it is suggested in [31] to use a prefetcher which is obtained by converting from prediction-by-partial-match (PPM) data compressor rather than Lempel-Ziv, which is shown to perform better for ”ill-case scenarios” with short data access observance possibility. PPM algorithm is nothing but an implementation of prediction by high-order Markov model, which we also use (see SubSection 2.4.1) in formulating the prefetching problem for WA-Dataspaces.

2.4.1 Prefetching formulation

Here we will give a formal description of the problem formulation for prefetching using Markov models as promised in SubSection 2.2.2. Let *alphabet* be the set of page ids that a page access sequence can consist of. We will generally denote page ids using characters such as $A = \{a, b, c\}$. Prefetching problem is predicting the next page access P_{n+1} given an observed access sequence $H = [p_1, p_2, \ldots, p_n]$ where P_n is a random variable representing the nth page access in the sequence and with value $p_n \in A$. A first-order Markov model suggests that probability of a particular value that next page access takes $Pr\{P_{n+1} = p_{n+1}\}$ depends only on the last observed page access $P_n = p_n$; $Pr\{P_{n+1}|P_n, \ldots, P_1\} = Pr\{P_{n+1}|P_n\}$. A kth-order Markov model suggests, next page access depends on last k observed page accesses; $Pr\{P_{n+1}|P_n, \ldots, P_1\} = Pr\{P_{n+1}|P_n, \ldots, P_{n-k+1}\}$.

PPM algorithm presented in [31] builds a kth-order Markov and then makes the prediction in favor of the page value which maximizes the value of $Pr\{P_{n+1}|P_n, \ldots, P_{n-k+1}\}$:

$$\hat{p}_{n+1} = \underset{p_n \in A}{\text{argmax}}\; Pr\{P_{n+1} = p_n | P_n = p_n, \ldots, P_{n-k+1} = p_{n-k+1}\} \tag{2.1}$$

where $\hat{p}_{n+1}$ is the predicted value of P_{n+1}. Given the access sequence $H = [p_1, p_2, \ldots, p_n]$, probability model is built by computing the conditional probabilities of P_n given the values of last k page accesses by using the frequency of observed patterns in H as follows:

$$\begin{aligned} &Pr\{P_{n+1} = p_{n+1} | P_n = p_n, \ldots, P_{n-k+1} = p_{n-k+1}\} \\ &\qquad = \text{frequency of "}p_{n-k+1}, \ldots, p_n, p_{n+1}\text{" in H} \\ &\qquad = \frac{\text{number of "}p_{n-k+1}, \ldots, p_n, p_{n+1}\text{" in H}}{\text{number of "}p_{n-k+1}, \ldots, p_n\text{" in H}} \end{aligned} \tag{2.2}$$

To predict multiple pages that may be accessed in close future, conditional probability of l pages to be accessed given last k observed using the unordered set of random variable $\hat{\mathbf{P}} = \hat{P}_{n+l}, \ldots, \hat{P}_{n+1}$ where $\hat{P}_{n+i}$'s for $i \in [1, l]$ are random variables representing the predicted pages to be accessed within next l access steps. Important thing to note is $\hat{\mathbf{P}}$ is unordered e..g, for l equals to 3, predicted set of pages a, b, c and permutations of it are equal. Set of pages to be prefetched can be determined by maximizing the conditional probability $Pr\{\mathbf{P}|P_n, \ldots, P_{n-k+1}\}$ as follows:

$$\hat{\mathbf{p}} = \operatorname*{argmax}_{\mathbf{p} \in A^l} Pr\{\mathbf{P} = \mathbf{p}|P_n = p_n, \ldots, P_{n-k+1} = p_{n-k+1}\} \tag{2.3}$$

where A^l is the $l-ary$ cartesian power of set A and $\hat{\mathbf{p}}$ is the predicted set of l pages. As in the case for predicting only the next page access, given the access sequence $H = [p_1, p_2, \ldots, p_n]$, probability model can be built by computing the conditional probabilities of $\mathbf{P}$ given the values of last k page accesses by using the frequency of observed patterns in H as follows:

$$\begin{aligned} & Pr\{\mathbf{P} = \mathbf{p}|P_n = p_n, \ldots, P_{n-k+1} = p_{n-k+1}\} \\ & \qquad = \text{frequency of } "p_{n-k+1}, \ldots, p_n, permutation(\mathbf{p})" \text{ in H} \qquad (2.4) \\ & \qquad = \frac{\text{number of } "p_{n-k+1}, \ldots, p_n, permutation(\mathbf{p})" \text{ in H}}{\text{number of } "p_{n-k+1}, \ldots, p_n" \text{ in H}} \end{aligned}$$

Conditional probabilities can be built in an online fashion by updating them every time a new page access is observed. To explain with a simple example, consider the alphabet a, b and the observed page access sequence $H = [a, a, b, b, b, a, a, b, a, a, a]$ of length 7. Our prefetcher builds a decision tree using H for k (order of Markov model) and l (number of accesses to be predicted) equal to 2 and 1. Figure 2.6 shows the decision tree in which conditional probabilities associated with leaf edges are calculated by computing the frequency of patterns in the access sequence as described by the equation 2.2. Every time a new page access is desired to be predicted, the pointer walks on the tree starting from the root and goes down following the last k accesses, which in this case is $[a, a]$. If the walk cannot be completed due to missing nodes in the tree (possible when the pattern e.g., "a, a" int this case, is not observed yet), prediction cannot be made and instead a default policy can be used to randomly pick one of the pages in the alphabet to prefetch. If the walk can be completed as the walk ended on the node marked with "X" in the example shown in Figure 2.6, prediction can be made by picking the edge among the connected edges which is assigned highest probability and finally assigned page to the picked edge is prefetched by the prefetcher. For example, page b is prefetched in this example since edge corresponding to b is assigned with 2/3 which is greater than 1/3 of the edge corresponding to page a.

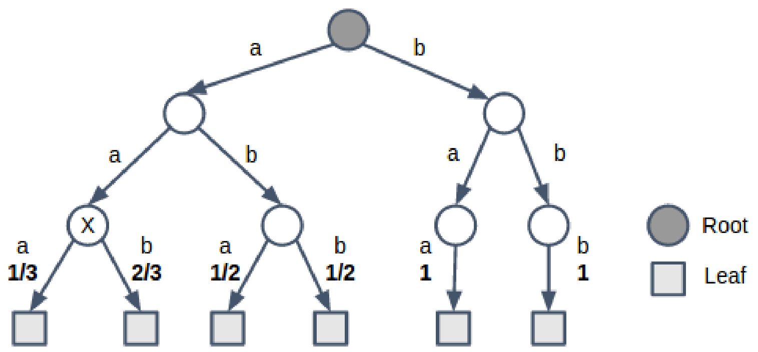


Figure 2.6: Decision tree constructed by the prefetcher given observed page access history $H = [a, a, b, b, b, a, a, b, a, a, a]$ and order of Markov model is 2 and number of accesses to be predicted is 1.

Decision tree as exemplified above for storing and updating the conditional probability values is fairly easy to implement. Naturally, implementation and computational complexity of the decision tree increases proportional with k and l. Complexity is not an issue in WA-Dataspaces case because prefetching algorithm will be run on a compute node of a HPC cluster. Actually, even if prediction algorithm takes considerable time to make a decision, it still would not introduce high relative overhead to the workflow execution since data transport time over WAN is already much higher in practice. Criteria for picking the right values for k and l are outlined next as follows:

- **Model coverage:** As discussed in SubSection 2.2.2, coverage of the Markov model decreases with the increasing model order k. This affects the predictive accuracy and consequently prefetching performance adversely for especially short-duration user sessions where the length of the access sequence is ”low or not enough” to recognize patterns in a high-order Markov model in a timely manner. Model coverage can also be thought in terms of convergence of the decision tree associated with the model to the optimal. It is easy to observe that higher the coverage is (i.e., the lower the order of the model k), faster the convergence to optimal is. Currently, WA-Dataspaces framework does not employ any autonomic decision mechanism to tune the value of k and we assume application developer picks a value for k by considering application characteristics and the corresponding required coverage. However, as again discussed in SubSection 2.2.2, low coverage problem of high order Markov models is solved by "All-Kth Order Markov Model" introduced in [36] and we will investigate possibility of using this idea to integrate a more autonomic prefetcher for WA-Dataspaces.

- **Prefetching buffer size and data replacement:** Prefetching buffer is is meant to store only the pages that are going to be accessed soon. Pages in the buffer are replaced in many contexts using policies such as Least Recently Used (LRU) policy, First-in First-out (FIFO) policy, random policy, and many other more complicated derivatives of those. Goal of replacing prefetched pages with the newly prefetched ones is both to keep buffer small and optimal strategy would be to replace the pages which are prefetched but never used and will not be used due to misprediction. Decision of the number of pages to be prefetched at a time l should be done based on the available buffer size and the replacement policy which are not addressed in this thesis. Therefore, we will assume also the value of l is tuned by the application developer who is assumed to be aware of its effects on the prefetching performance.

- *Prediction Accuracy:* Most of important criterion for picking the order of the model k is the desired prediction accuracy. Given the characteristics of the user process, different access patterns may emerge. Some patterns are easier to recognize than others by using a Markov model because prefetcher uses the sequentiality within the observed page access while building the decision tree and sequentiality is affected by the alphabet size and the length of the observed access sequence. For example, given alphabet of a, b and observed sequence $[a, b, a, b, a, b, a]$ sequential pattern of "a, b" and "b, a" can be captured using a first-order Markov model and since order is low, model coverage is high and also decision tree converges to the optimal very fast. However, when the scenario is not as simple as in the last example, order of the model k should be carefully selected since model coverage and accuracy usually compete since the choice of k affects both oppositely i.e., if coverage increases, accuracy will decrease. Therefore, process of selecting a "good" value for k requires Pareto optimal-curve analysis. In WA-Dataspaces case, we do not address the autonomic management of l with respect to application characteristics and heterogeneity and the limits on the buffer size, and again we let application developer tune the value of l assuming s/he know its discussed effects.

To conclude, current prefetcher implementation in WA-Dataspaces framework, mostly relies on developer's knowledge about choosing important model parameters k and l and then implements a kth-order Markov model for predictive prefetching l data tuples every time a new data access is observed from the application. We do not investigate and leave autonomic tuning of these model parameters and buffer management as future work.

As discussed previously in this section, sequentiality in the observed access sequence is important for Markov models to recognize patterns and compute conditional prediction probabilities accurately. However, Dataspaces API and consequently WA-Dataspaces API does not constrain the set of possible keys to identify the shared data although whole discussion about Markov-based model assumed that user accesses are identified by a finite set of identifiers (i.e., alphabet). Therefore, WA-Dataspaces prefetcher performs an online mapping application keys to the elements of an alphabet of finite size as described next. Starting with the observation that scientific applications usually interact with each other at particular time steps for varying purposes such as synchronization, computational validation of a model, visualization, stream-based processing pipelines etc. This characteristic of scientific applications results in a trend of data access behavior in that the data that is most recently produced and put into the space is most likely going to be consumed soon, in other words, one can say probability of a data item to be consumed within a time interval since its arrival can be described with an exponential distribution. Given this observation one can make the following claim that if data put into the space is identified by the sequence of keys $[s_1, s_2, \ldots, s_n]$, it is highly likely that access sequence observed later in time will be the same. WA-Dataspaces prefetcher tries to maximize the chance of recognizing this access behavior by mapping data keys put into the space, which can be arbitrarily any char array, to a one-dimensional circular sequence in a sequential manner using an alphabet of finite size. We call this procedure as *application key renaming.* Size of the circular alphabet used depends on the model order k and number of items to be prefetched at a single prefetching session l. As a heuristic to optimize the model coverage and prediction accuracy, our prefetcher sets the size of the alphabet equal to maximum of k and l.

To explain with an example, suppose k is 2 and l is 1, and consequently generated alphabet is of size; suppose it is $A = a, b$. Application scenario is fairly simple such that producer application puts data sequentially into the space which is identified by the following key sequence $S = [s_1, s_2, \ldots, s_{10}]$ in which s_i represents an arbitrary char array, and consumer application accesses the data put by the producer by following the same sequence S. As can also be followed on the representation given in Figure 2.7, prefetcher maps each item in the access sequence S to a new key in alphabet A by traversing the alphabet in a sequential and circular manner, and generates the new access sequence $S^c = [a, b, a, b, a, b, a, b, a, b]$. There is a strong sequentiality in the new sequence S^c which can be easily detected by the second-order Markov model and high coverage and predictive accuracy can be obtained i.e., in this example starting with the fifth key, model predicts each of the following keys in the sequence accurately by making a prediction every time true value of a new access is observed. However, one problem that comes with application key renaming procedure described here is that data elements in the new sequence S^c cannot be differentiated after more than 2 keys have to be staged before the consumer gets them which can also be called computational phase lag between the consumer and the producer component. This should be addressed in the renaming process since applications cannot give any guarantee on the computational phase lag proactively – it can only be detected at runtime. Therefore, prefetcher labels each key in the newly generated sequence with a *version* which can be computed using a counter being incremented each time a circular traverse on the alphabet is completed, so with labeling generated sequence turns into $S^c = [a^0, b^0, a^1, b^1, a^2, b^2, a^3, b^3, a^4, b^4]$.

Prefetching logic at every staging site is realized by the corresponding *RIManager*, which is explained in detail in SubSection 2.3.2. Application data access sequence is observed by the prefetcher through "Local Communication Layer (LCL)" and *RIManager*

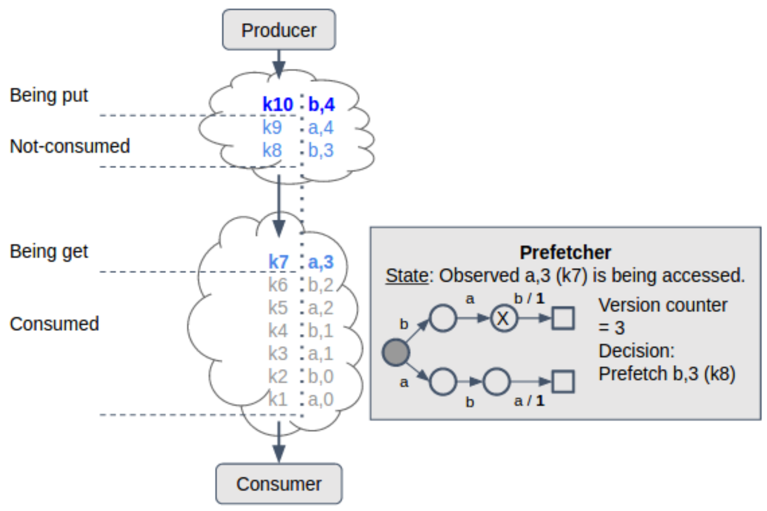


Figure 2.7: Representation of application key renaming procedure during prefetching for producer-consumer scenario. Model order k is 2 and number of prefetched data items l is 1.

implements prefetching logic and operations on top of ”Remote Query Engine (RQE)”.

2.5 Evaluation

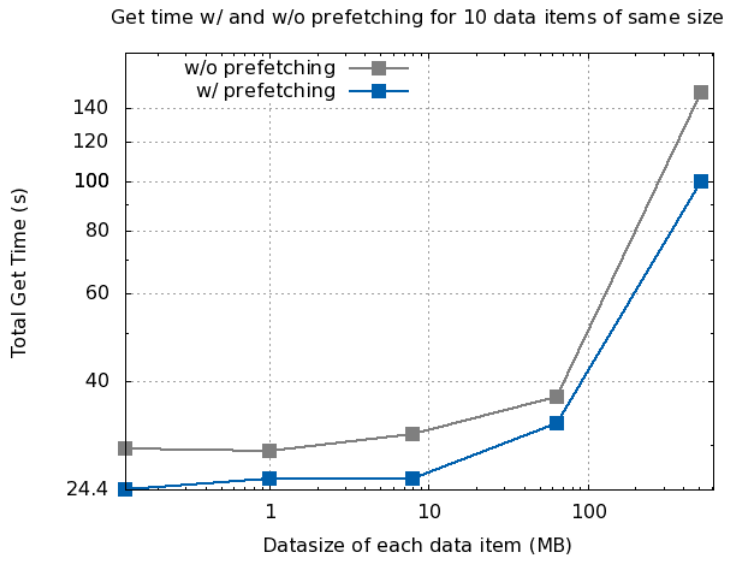
To evaluate the framework for wide-area data staging described in this chapter, we used the synthetic workflows which comply with the characteristics (i.e., data access patterns) of ”Plasma Disruption Analysis” workflow we used in the previous chapter (see Section 1.4). The application scenario we run on the framework is as follows. As in the example given in previous SubSection 2.4.1 while describing prefetching, we consider one producer-consumer pair exchanging data over the two Dataspaces servers residing at corresponding sides and exchanged data is a sequence of 10 (key, value) pairs. Consumer application accesses the data sequentially and in the same order as it is put into space by the producer – staging space plays the role of a queue in this case as shown for the previous example in Figure 2.7. We also assume application developer sets the model order to 2 and number of prefetched (key, value) pairs at once to 1. The infrastructure framework and the applications run on is an RDMA cluster at Rutgers – nodes of the cluster is separated into producer and consumer sites, and data transport between the two sides over WA-Dataspaces framework is done on RDMA stack.

First, we assume there is no *slack time* between the get requests that consumer application initiates. As introduced previously, prefetching in wide-area data staging context tries to use the slack time between the application requests to fetch data of interest before the application request arrives. We run the workflow multiple times by varying the size of a single (key, value) pair (i.e., data item) is put and retrieved by the coupled applications. Figure 2.8a shows that there is still improvement with prefetching even with the lack of slack time. This is due to earlier start of handling next get request using prefetching while the previous get is still being served. Therefore, we expect prefetching to improve data retrieval time more for the cases where get requests take longer to handle i.e., due to slow large data transport over WAN. Second, we set a fixed slack time between get requests consumer application initiates. In practice, the slack time between the data retrieval sessions of workflows is due to a computational task which takes a random duration but fixed slack time we set tries to enforce this nature of workflows. Figure 2.8b shows that compared to scenario without slack time, gain in total time to get data with prefetching is much higher for the case with slack time. One point that may look unexpected at first is that relative gain in total get time by prefetching seems to decrease as the data item size increases. This is expected since the fixed slack time is enough to finish fetching all the data of interest for relatively smaller data items but as data items get larger, during the available slack time not all the data can be transported to the consumer site which decreases the relative gain obtained from prefetching.

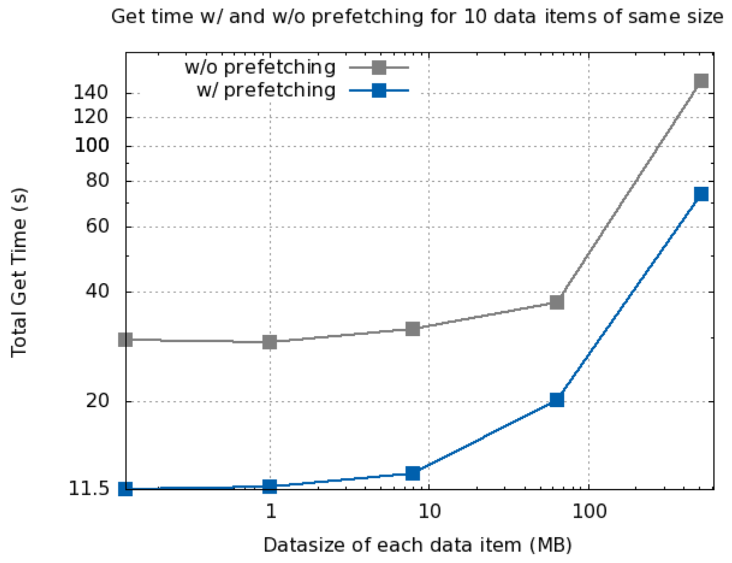
2.6 Conclusions and Future Work

This chapter presented the architecture, design and implementation of an approach for wide-area shared space abstraction for scientific workflows. Proposed framework WA-Dataspaces extends currently available data sharing framework Dataspaces, which provides a semantically specialized shared space abstraction for application coupled over LAN. WA-Dataspaces leverages NUMA architectural design principles and it has a layered architecture which puts semantics for remote and local data management operations together. Furthermore, implementation of the presented framework leverages different data transfer technologies (i.e., TCP/IP, RDMA) and high performance data transport protocols (i.e., GridFTP). To enable fast data retrieval from remote applications, we explored Markov-based predictive prefetching models and strategies, and presented an approach for prefetching with partly application developer-driven Markov models where some model parameters are set by the application. Finally, integration of the prefetching approach with the overall framework is presented. Also we presented an evaluation of the framework for real producer-consumer based Fusion Workflow scenario in terms of end-to-end data retrieval performance and showed considerable improvement is achieved with prefetching due to efficient recognition of the application access pattern by renaming keys associated with data by the application.

WA-Dataspaces has enabled application couplings in the Fusion Workflow in a demo presented during SC'2014. Fusion workflow was run between clusters at Singapore and Georgia Tech sites which are connected over wide-area RDMA technology (more information can be found in [42]). Our current work is focused on implementing a more autonomic prefetching model and semantics to achieve runtime awareness for the application characteristics and resource constraints. We are investigating how to apply All-kth order Markov models on the wide-area remote data prefetching problem for achieving both high model coverage and predictive accuracy at the same time, and exploring how to tune model order and number of data elements prefetched at a time in terms of temporal application data access behavior change and dynamic staging capacity needs and constraints.



(a)



(b)

Figure 2.8: Comparison of total get time between w/ (Bottom) and w/o (Top) prefetching for the scenario of producer-consumer based sharing of 10 data items of same size. Both X and Y-axes are log-scaled.

References

[1] M. F. Aktas, G. Haldeman, and M. Parashar, "Flexible scheduling and control of bandwidth and in-transit services for end-to-end application workflows," in *Proceedings of the Fourth International Workshop on Network-Aware Data Management.* IEEE Press, 2014, pp. 28–31.

[2] L. Zhang, C. Docan, and M. Parashar, "The seine data coupling framework for parallel scientific applications," *Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications*, p. 283, 2010.

[3] V. Bhat, M. Parashar, and S. Klasky, "Experiments with in-transit processing for data intensive grid workflows," in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing.* IEEE Computer Society, 2007, pp. 193–200.

[4] V. Bhat, M. Parashar, H. Liu, M. Khandekar, N. Kandasamy, and S. Abdelwahed, "Enabling self-managing applications using model-based online control strategies," in *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on.* IEEE, 2006, pp. 15–24.

[5] V. Bhat, S. Klasky, S. Atchley, M. Beck, D. McCune, and M. Parashar, "High performance threaded data streaming for large scale simulations," in *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on.* IEEE, 2004, pp. 243–250.

[6] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," in *DARPA Active NEtworks Conference and Exposition, 2002. Proceedings.* IEEE, 2002, pp. 2–15.

[7] T. M. Chen and A. W. Jackson, "Commentaries on" active networking and end-to-end arguments"," *Network, IEEE*, vol. 12, no. 3, pp. 66–71, 1998.

[8] L. Lefèvre, C.-d. Pham, P. Primet, B. Tourancheau, B. Gaidioz, J.-P. Gelas, and M. Maimour, "Active networking support for the grid," in *Active Networks.* Springer, 2001, pp. 16–33.

[9] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston, "Intra and interdomain circuit provisioning using the oscars reservation system," in *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on.* IEEE, 2006, pp. 1–8.

[10] N. S. Rao, W. R. Wing, S. M. Carter, and Q. Wu, "Ultrascience net: Network testbed for large-scale science applications," *Communications Magazine, IEEE*, vol. 43, no. 11, pp. S12–S17, 2005.

[11] I. Monga, E. Pouyoul, and C. Guok, "Software defined networking for big-data science," *SuperComputing 2012*, 2012.

[12] M. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz, "Datacutter: Middleware for filtering very large scientific datasets on archival storage systems," in *IEEE symposium on mass storage systems.* Citeseer, 2000, pp. 119–134.

[13] K. Schwan, B. F. Cooper, G. S. Eisenhauer, A. Gavrilovska, M. Wolf, H. Abbasi, S. Agarwala, Z. Cai, V. Kumar, J. Lofstead *et al.*, "Autonomic information flows," 2005.

[14] X. Ma, M. Winslett, J. Lee, and S. Yu, "Faster collective output through active buffering," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM.* IEEE, 2001, pp. 8–pp.

[15] J. S. Plank and M. Beck, "The logistical computing stack–a design for wide-area, scalable, uninterruptible computing," in *Dependable Systems and Networks, Workshop on Scalable, Uninterruptible Computing (DNS 2002).* Citeseer, 2002.

[16] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *Communications Magazine, IEEE*, vol. 35, no. 1, pp. 80–86, 1997.

[17] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura, "Active networking and the end-to-end argument," in *Network Protocols, 1997. Proceedings., 1997 International Conference on.* IEEE, 1997, pp. 220–228.

[18] A. Brown, E. Kissel, M. Swany, and G. Almes, "Phoebus: A session protocol for dynamic and heterogeneous networks," *University of Delaware, Tech. Rep*, vol. 2008, p. 334, 2008.

[19] "Token bucket filter," http://lartc.org/howto/lartc.qdisc.classless.html/.

[20] J. Mattingley and S. Boyd, "Automatic code generation for real-time convex optimization," *Convex optimization in signal processing and communications*, pp. 1–41, 2009.

[21] S. L. Bird and B. J. Smith, "Pacora: Performance aware convex optimization for resource allocation," in *Proceedings of the 3rd USENIX Workshop on Hot Topics in Parallelism (HotPar: Posters)*, 2011.

[22] J. Lee, J. Kim, C. Kessel, and F. Poli, "Simulation study of disruption characteristics in kstar," in *APS Meeting Abstracts*, vol. 1, 2012, p. 8047P.

[23] "Pox," http://www.noxrepo.org/pox/about-pox/.

[24] S. Diamond, E. Chu, and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization, version 0.2," http://cvxpy.org/, May 2014.

[25] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks.* ACM, 2010, p. 19.

[26] G. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo, S. Smallen, W. Smith, and A. Grimshaw, "Futuregrida reconfigurable testbed for cloud, hpc and grid computing," *Contemporary High Performance Computing: From Petascale toward Exascale, Computational Science. Chapman and Hall/CRC*, 2013.

[27] "Dataspaces: An extreme-scale data management framework," http://personal.cac.rutgers.edu/TASSL/projects/data/index.html.

[28] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: an interaction and coordination framework for coupled simulation workflows," *Cluster Computing*, vol. 15, no. 2, pp. 163–181, 2012.

[29] C. Schmidt and M. Parashar, "Squid: Enabling search in dht-based systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 7, pp. 962–975, 2008.

[30] J. S. Vitter and P. Krishnan, "Optimal prefetching via data compression," *Journal of the ACM (JACM)*, vol. 43, no. 5, pp. 771–793, 1996.

[31] K. M. Curewitz, P. Krishnan, and J. S. Vitter, "Practical prefetching via data compression," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 257–266, 1993.

[32] M. Palmer and S. B. Zdonik, *Fido: A cache that learns to fetch.* Brown University, Department of Computer Science, 1991.

[33] N. R. Mabroukeh and C. I. Ezeife, "Semantic-rich markov models for web prefetching," in *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on.* IEEE, 2009, pp. 465–470.

[34] A. Bestavros, "Using speculation to reduce server load and service time on the www," in *Proceedings of the fourth international conference on Information and knowledge management.* ACM, 1995, pp. 403–410.

[35] J. Borges and M. Levene, "Data mining of user navigation patterns," in *Web usage analysis and user profiling.* Springer, 2000, pp. 92–112.

[36] J. Pitkow and P. Pirolli, "Mininglongestrepeatin g subsequencestopredict worldwidewebsurfing," in *Proc. USENIX Symp. On Internet Technologies and Systems*, 1999, p. 1.

[37] M. Deshpande and G. Karypis, "Selective markov models for predicting web page accesses," *ACM Transactions on Internet Technology (TOIT)*, vol. 4, no. 2, pp. 163–184, 2004.

[38] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, "Gridftp: Protocol extensions to ftp for the grid," *Global Grid ForumGFD-RP*, vol. 20, 2003.

[39] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," *computer*, vol. 29, no. 12, pp. 66–76, 1996.

[40] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *Information Theory, IEEE Transactions on*, vol. 24, no. 5, pp. 530–536, 1978.

[41] P. Krishnan and J. S. Vitter, "Optimal prediction for prefetching in the worst case," *SIAM Journal on Computing*, vol. 27, no. 6, pp. 1617–1636, 1998.

[42] "Industry collaboration drives 100g foundation for supercomputer infrastructure," http://www.obsidianresearch.com/archives/all/2014/100G-SC.html.