A Middleware for Large-scale Simulation Systems and Resource Management

Hemanth Makkapati

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science & Applications

Madhav V. Marathe, Chair
Keith R. Bisset
Eli Tilevich

May 03, 2013
Blacksburg, Virginia

Keywords: Middleware, Cluster, Cloud, Grid, Socio-technical Systems, Simulation Systems, Public Health Epidemiology, Network Science

A Middleware for Large-scale Simulation Systems and Resource Management

Hemanth Makkapati

(ABSTRACT)

Socially coupled systems are comprised of inter-dependent social, organizational, economic, infrastructure and physical networks. Today's urban regions serve as an excellent example of such systems. People and institutions confront the implications of the increasing scale of information becoming available due to a combination of advances in pervasive computing, data acquisition systems as well as high performance computing. Integrated modeling and decision making environments are necessary to support planning, analysis and counter factual experiments to study these complex systems.

Here, we describe SIMFRASTRUCTURE – a computational infrastructure that supports high performance computing oriented decision and analytics environments to study socially coupled systems. Simfrastructure provides a middleware with multiplexing mechanism by which modeling environments with simple and intuitive user-interfaces can be plugged in as front-end systems, and high-end computing resources – such as clusters, grids and clouds – can be plugged in as back-end systems for execution. This makes several key aspects of simulation systems such as the computational complexity, data management and resource management and allocation completely transparent to the users. The decoupling of user interfaces, data repository and computational resources from simulation execution allows users to run simulations and access the results asynchronously and enables them to add new datasets and simulation models dynamically. Simfrastructure enables implementation of a simple yet powerful modeling environment with built-in *analytics-as-a-service* platform, which provides seamless access to high end computational resources, through an intuitive interface for studying socially coupled systems.

We illustrate the applicability of Simfrastructure in the context of an integrated modeling environment to study public health epidemiology and network science.

Dedication

To my family and Shweta Gupta.

Acknowledgments

I truly believe that without the contribution of many outstanding people, my stay at Virginia Tech and the experience of pursuing a Master's wouldn't have been so complete and fruitful! I acknowledge and offer my sincere thanks to the following people who enabled me, directly or indirectly, to reach this milestone:

- My parents - Uma and Ramana Rao - for their unconditional love and support all through. I couldn't have asked for anything more from them.
- My grand parents - Late. Lakshmi Kantamamba and Koteswar Rao - who are easily the people with strongest characters I have ever come across in my life so far. I have very fond memories of their home in Tenali, India, which I frequented every summer to meet my cousins, aunts and uncles.
- My younger brother, Sumanth, who has been terrific since the moment I landed at ROA and helped me in every manner possible. At times, I would wonder if he's my younger brother or elder?
- My cousins - Praveen Kudithipudi, Amarsh Movva, Praneeth Kudithipudi, Aishwarya Movva, Meghna Panda, Svetha Doddapaneni, Juveca Panda, Abhigna Panda and Anuj Panda - for being a constant source of encouragement and support. They always make me feel at home. Thank you very much for being there always!
- My extended family - Anuradha and Prasad Panda, Kamala and Ravindra Kudithipudi, Padma and Prasad Movva, Sita and Sudhakar Panda and Rama Panda - for their constant encouragement and care.
- My advisor, Dr. Madhav Marathe, for being an amazing mentor since day one. I can't thank him enough for making time for me from his extremely busy schedule whenever I needed help. His pro-student attitude makes it a pleasure to work with him.
- My immediate supervisor and committee member, Keith Bisset, for helping me understand almost everything I touched in the last two years. He's been the one-stop solution to all the problems I encountered.

- My committee member, Dr. Eli Tilevich, for his insightful thoughts on the research enterprise and its functioning. His tips on technical writing has gone a long way in helping me write research papers and this thesis.

- Kevin Hall for helping me with the design and implementation of Resource Manager. His attention to detail and helping nature proved to be a big asset!

- Rajesh Subbiah, for giving company during many sleepless nights at NDSSL and encouraging me when things looked grim. Most importantly, I thank him very much for sharing his never-depleting stock of food :)

- Spencer Lee, who shuttled me countless number of times between Foxridge and CRC at unearthly hours. He has been a great support and easily the finest friend I have made in Blacksburg. A true compassionate, benevolent, magnanimous and gadget-loving human being!

- My roommates - Avinash Desai, Rajat Gangrade, Sushrut Shirole and Tanmay Potnis - for putting up with me and fostering a friendly environment all through. They've been very understanding and helpful while I was up against harsh deadlines.

- Shweta Gupta, my friend, philosopher and guide. If it wasn't for her, I probably wouldn't have pursued Master's.

Contents

Contents **vi**

1 Introduction **1**

2 Architecture **3**

2.1 Blackboard 3

2.1.1 Key Features 5

2.1.2 Requests 6

2.2 Brokers 6

2.3 Types of Brokers 7

2.3.1 Edge Brokers 7

2.3.2 Service Brokers 8

2.3.3 Coordination Brokers 8

3 Services **10**

3.1 Job Execution Service 10

3.2 Data Transfer Service 11

3.3 Resource Management Service 11

3.4 Digital Library Service 11

4 Resource Management **13**

4.1 Resource Manager 14

4.1.1 Resource Monitoring 14

4.1.2 Resource Allocation . . . 14
4.2 Design . . . 15
4.2.1 Resource Monitoring Service . . . 15
4.2.2 Resource Allocation Service . . . 19
4.2.3 Naming Scheme . . . 22

5 Case Studies **26**
5.1 Public health epidemiology . . . 26
5.1.1 ISIS Design . . . 27
5.2 Network Science . . . 29
5.2.1 CINET Design . . . 29
5.2.2 Prototype with Resource Manager . . . 31

6 Discussion **35**
6.1 Computational complexity . . . 35
6.2 Scalability . . . 35
6.3 Latency . . . 36
6.4 Fault-tolerance . . . 38

7 Related Work **39**

8 Future Work and Conclusion **41**
8.1 Future Work . . . 41
8.1.1 Blackboard . . . 41
8.1.2 Rule Engine . . . 41
8.1.3 Performance Measurement . . . 42
8.2 Conclusion . . . 42

Bibliography **43**

List of Figures

2.1 A typical set-up of the modeling and simulation environment using Simfrastructure as the central communication and coordination mechanism. Simfrastructure coordinates data and context flow between user interfaces, digital library, compute resources and simulation models and measures. 4

2.2 A generic depiction of actions in a workflow broker: The broker context fetches the desired requests from the blackboard and adds the embedded runners to the runnable queue. Worker threads retrieve runners from the runnable queue and execute them. 9

4.1 The Heartbeat of a computational resource depicting different classes of resource attributes. 16

4.2 A multi-resource set up of Simfrastructure depicting the computational resources – Pecos, Shadowfax and Open Science Grid. An instance of HealthMonitor runs on every computational resource and repeatedly, once every 25 secs, posts/updates the Heartbeat on the Blackboard. The Health Monitor interacts with the Job Manager to collect necessary attributes. 17

4.3 The depiction of Resource Manager Broker with Heartbeat Monitor. The Heartbeat Monitor reads the Heartbeats on the blackboard to obtain the available resources for allocation at any point in time. 18

4.4 The depiction of Resource Manager Broker with Domain Handlers. The Domain Handlers, when invoked by the Resource Manager Broker, read the parameters of a given job and determine the attributes of desired computational resources that are best suitable for job execution. The desired attributes are returned as resource templates. 20

4.5 The depiction of Resource Manager Broker with Resource Allocator and various resource allocation strategies. The Resource Allocator, when invoked by the Resource Manager Broker, invokes a given resource allocation strategy to determine the most appropriate resource from the set of desired resources. 21

4.6 The naming scheme followed for resource allocation. The naming scheme contains three attributes: the organizational domain, resource type and the universal identifier. 22

4.7 A naming scheme indicating a specific cluster identified by "sfxlogin1.vbi.vt.edu" from "vbi.vt.edu" to be ideal for job execution. 23

4.8 A naming scheme indicating a resource of any type from "vbi.vt.edu" organizational domain to be ideal for job execution. 23

4.9 A naming scheme indicating any cluster from "vbi.vt.edu" organizational domain to be ideal for job execution. 23

4.10 A naming scheme indicating a cloud from any organizational domain to be ideal for job execution. 23

4.11 A naming scheme indicating a resource from any organizational domain and type to be ideal for job execution. 24

4.12 A depiction of Resource Manager with all its components. The Resource Manager Broker reads the Job Requests from the Blackboard and hands them over to appropriate Domain Handlers for further processing. The Domain Handlers process the Job Requests and return desired resource templates to the Resource Manager Broker, which then invokes the Resource Allocator with an allocation strategy and the resource template. The Resource Allocator invokes the Health Monitor to fetch all the available resources by reading the Heartbeats from the Blackboard. The Resource Allocator then filters the available resources based on the resource template and invokes the given allocation strategy to pick the best resource from the filtered set. 25

5.1 Sequence of activities in a typical workflow of a multi-cell epidemic simulation executed using ISIS as the simulation tool, EpiFast as the epidemic simulation engine and Simfrastructure as the coordinating middleware platform. 28

5.2 Sequence of activities in a typical workflow of a graph measure analysis using CINET. 30

5.3 Sequence of activities in a typical workflow of a graph measure analysis using CINET in a multiple resource setup. The resource manager is utilized to determine the most appropriate resource for the execution of analysis. 32

5.4 Flowchart depicting the series of actions and decisions performed in Granite Handler to decide an appropriate graph library for the given request. 34

6.1 Performance of *read* and *write* operations with over 200,000 requests divided into 210 buckets, each of size 1000. For each bucket, minimum, maximum and the 95^{th} quantile time values, in milliseconds, are plotted on log scale. 37

Chapter 1

Introduction

Socially coupled systems consist of a large number of interacting physical, technological, and human/societal components. Examples of such systems include urban transportation systems, communication networks, public health, integrated energy systems, etc. All of the above systems share an important common feature: these systems are networked, i.e. individual agents/components interact only with a specified set of components. Understanding the coupled evolution of these interacting networks is critical for situational awareness and decision support for sustainable planning, preparedness and response. For example, human behaviors and day to day activities of individuals create dense social interactions characteristic of modern urban societies. These dense social networks provide a perfect fabric for fast, uncontrolled disease propagation. Conversely, people's behavior in response to public policies and their perception of how the crisis is unfolding as a result of disease outbreak can dramatically alter the normally stable social interactions. Effective planning and response strategies must take these complicated interactions into account. Computer simulation aided decision support tools provide a practical approach for addressing such issues.

Examples of the existing research on such high-resolution models include IBM Smart Cities project [1], weather forecasting models [2], modeling to contain epidemics [3], modeling social mechanisms [4] and so on. Some of the modeling environments we built include EpiFast [5] and EpiSimdemics [6] for epidemiological systems, TRANSIMS [7] for transportation analysis systems, and EpiNet [8] for modeling communication networks. A key characteristic of these modeling environments is the way the underlying socially coupled systems are represented and simulated. The use of high performance computing systems is critical when executing such models. Supporting realistic case studies using these models results in jobs with coupled workflows, as well as several independent tasks similar to a bag-of-tasks (BoT) model. Such diversity in the nature of simulation execution requires deployment of diverse computing resources in the modeling environment. Finally, decision-support environments using high resolution models consume and produce large quantities of structured and unstructured data. Large scale data movement between the data stores and the computing systems needs to be handled carefully in the modeling environment without affecting performance. Furthermore, the modeling environment needs to be adaptable to changes in datasets,

models and computing resources.

In this thesis, we introduce a flexible middleware called SIMFRASTRUCTURE. Simfrastructure provides a computational infrastructure specifically designed to sustain decision support environments aimed at studying socially coupled systems. Simfrastructure provides key services such as data and knowledge management, resource management and allocation, and job execution for setting up modeling environments with ease. These services are stateless and enable many modeling environments and applications to co-exist utilizing the same computational infrastructure.

Simfrastructure architecture is based on the associative shared memory paradigm embodied by tuple spaces. This allows Simfrastructure to serve as a multiplexer: user interfaces, HPC-models, data management systems and computing resources can all be integrated with relative ease. This has two important benefits. First, the loosely coupled architecture implies individual components need not be aware of other components, reducing the programming effort needed to develop and integrate these components. Second, the computational complexity of coordination will then grow linearly as opposed to quadratically. Finally, Simfrastructure enables implementation of a simple yet powerful analytics-as-a-service platform that provides seamless access to high end computational resources through an intuitive analytics interface for studying socially coupled systems. To illustrate the applicability of Simfrastructure in practical settings, we describe recent case studies in public health epidemiology and network science.

The rest of the thesis is organized as follows. We present the architecture of Simfrastructure in Section 2 before describing the core services offered on top of Simfrastructure architecture in Section 3. We present recent case studies of an epidemic simulation system and a cyberinfrastructure for network science developed using Simfrastructure in Section 5. We then present a detailed description of the resource management capabilities of Simfrastructure in Section 4. Then, we offer a discussion on the Simfrastructure architecture in Section 6 before discussing related work in Section 7. Finally, we present our concluding remarks in Section 8.

Chapter 2

Architecture

Simfrastructure is a distributed middleware platform providing a computational infrastructure to enable decision-supporting environments for studying socially coupled systems. The computational infrastructure comprises of two integral parts: infrastructure components and a coordination mechanism. The infrastructure components interact with each other via the coordination mechanism to provide well-defined services such as data and knowledge management, resource management and allocation, and job execution for supporting modeling environments with ease.

Simfrastructure architecture is based on the associative shared memory paradigm embodied by tuple spaces. The services are provided by processes called *brokers* – the infrastructure components – that coordinate over a shared associative memory space called *blackboard* – the coordination mechanism. Figure 2.1 depicts a typical set up of Simfrastructure outlining its key components described in greater detail in the following sections.

2.1 Blackboard

The blackboard is the central communication and coordination mechanism of Simfrastructure. It offers a fundamentally different coordination mechanism from the traditional practices in distributed systems such as message passing and remote invocation of methods. The blackboard embraces a space-based model inspired from the Linda coordination language [] proposed by David Gelernter. The space-based model perceives an application as a collection of coordinating processes exchanging objects via shared virtual – in this case the blackboard.

The blackboard is a shared, persistent and network-accessible repository of objects. The processes exchange objects over the blackboard instead of directly communicating with each other. Processes use simple operations such as *read*, *take* and *write* to copy an object from the blackboard, remove an object from the blackboard and post an object to the blackboard. To take and read objects, processes use value-based lookup to find objects of interest. A process can wait for a desired

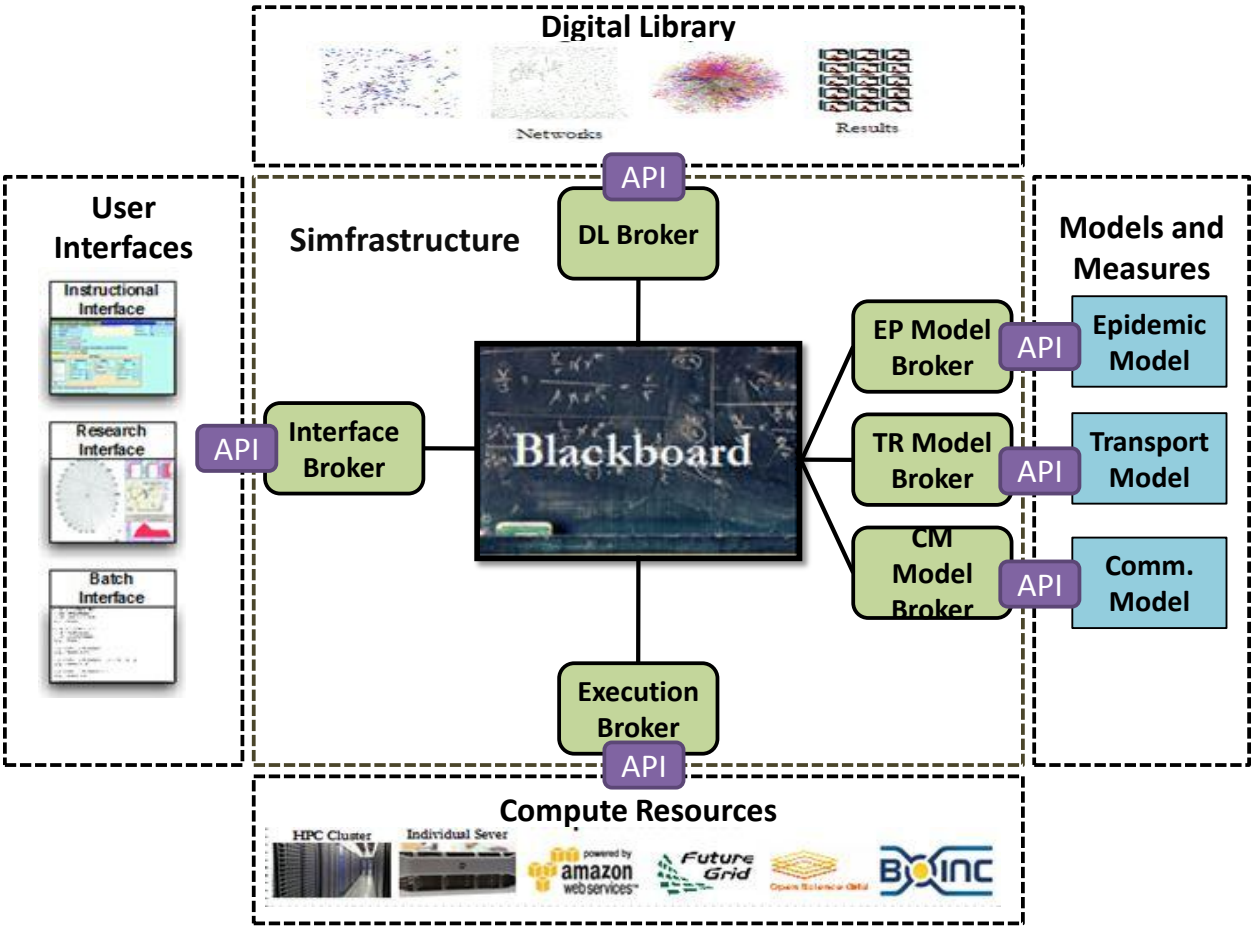

Figure 2.1: A typical set-up of the modeling and simulation environment using Simfrastructure as the central communication and coordination mechanism. Simfrastructure coordinates data and context flow between user interfaces, digital library, compute resources and simulation models and measures.

object to appear on the blackboard, if one is not already present. The objects are not modifiable while they are on the blackboard - they are just passive data. To modify an object, a process must remove it from the blackboard, modify it and post it back onto the blackboard.

The processes follow an application-specific protocol to modify the objects in a manner that achieves the outcome of an application. The protocol encodes the application workflow and any alternative workflows that may be necessary to handle errors. The workflows written over blackboard are loosely-coupled because the processes do not interact with each other directly. Unlike the traditional practices where the sending and receiving processes should know each other explicitly before the invocation, in space-based communication the processes need not know each other's existence. The processes need not even be active at the same time. An object can be written to the blackboard with assumption that some processes somewhere at some point in time will act upon it. This enables easy development of applications and workflows that are simple and flexible.

2.1.1 Key Features

The key features of blackboard, many that are characteristic of the inherent space-based model, are:

- **Blackboard is shared:** Blackboard is a network-accessible shared memory that many processes can interact with remotely and concurrently. Like a space, the Blackboard hides the details of concurrent access, enabling the processes the implement the application specific protocols and workflows. The Blackboard enables multiple processes to create, share, access and modify data structures using objects.

- **Blackboard is persistent:** Blackboard provides reliable storage mechanism for objects. Once objects are posted to the Blackboard, they will remain there until processes explicitly remove them. Processes may also specify leasetime for objects, after which they will be automatically removed from the Blackboard.

  As objects are persistent, they may outlive the processes that posted them. Thus, they remain on the Blackboard even after the processes have terminated. This property plays an important role in enabling the processes to be uncoupled and interact with each other even when they are not alive at the same time. The persistence of objects may also be useful in storing configuration and state information of an application between its invocations.

- **Blackboard is associative:** Objects on the Blackboard are located using associative lookup instead of their memory location or identifiers. The associative lookup mechanism provides a simple way of locating desired objects with respect to their content, without any other knowledge of the object such as its identifier, creator or its memory location. In order to find an object, a template is created with some or all fields of the object set to specific values that are expected in the desired objects. Fields that do not matter may be set to 'null', which acts as a wildcard. Objects on the Blackboard match the template only if their fields match the values in template exactly.

- **Blackboard is transactionally secure:** The Blackboard provides a transaction mechanism to ensure that operations on Blackboard are atomic. This ensures that either an operation succeeds or fails thereby eliminating any possibility of a partial failure leading to an unstable state.

- **Blackboard allows exchanging executable content:** While on the Blackboard, objects are passive data – they cannot be modified or invoked. However, a copy of the object can be made by reading or taking it from the Blackboard. The local copy of the object can be modified and invoked like any other object. This enables extending the behaviour of an application via the Blackboard.

2.1.2 Requests

The objects held on the blackboard, called Requests, represent the need for a service. Requests are typed objects with attributes where the type signifies the service requested and attributes represent the parameters required for the service. The requests are posted to and retrieved from the blackboard to request and fulfill services respectively. Requests are read and retrieved with the use of a template request, with desired type and attributes, and then matched against the existing requests on the blackboard.

Requests are the primary means by which processes coordinate by exchanging information. Typical information contained in the requests are:

- **Parameters:** The parameters required to deliver a particular service. A simulation service may require parameters such as population, simulation model, disease model, infection seeds and interventions etc.
- **State:** A request can be in any one of the following states during its life cycle: new, posted, running, deleted, successful and failed. The state information is used extensively by service providers and consumers to indicate success or failure of a service fulfillment.
- **Workflow:** A workflow contains the details about how a service is to be fulfilled. For instance, a simulation request may contain the workflow to pre-process the input data, locate and fetch the required datasets, run the simulation, and validate the produced output. The workflow is typically specified in the form of an embedded object called the runner. The runners are described in greater detail in Section 2.2.

The blackboard is currently implemented as a JavaSpace [9] holding the request objects as serialized Java objects.

2.2 Brokers

Brokers are the infrastructure components of Simfrastructure responsible for delivering a particular service. They monitor the blackboard for specific requests and deliver the appropriate services. Additionally, brokers post requests to the blackboard to consume any intermediate services required to fulfill a service. A broker essentially comprises of a set of conditions and actions. The conditions, when satisfied, trigger the actions to deliver a service. The conditions embody the matching criterion specified over the type and attributes of requests. The actions, however, greatly vary in complexity and nature based on the service they extend. Some services can be delivered through a simple set of actions, while others may need an elaborate workflow to be employed. For instance, a logging service would require actions to simply read the requests on the blackboard and log appropriate attributes as and when they are posted or removed. However, a simulation service would require an elaborate workflow of actions to pre-process the input data, locate and fetch

the required datasets, run the simulation on computing resources, monitor the simulation execution and validate the output produced.

In Simfrastructure, workflows are encoded in separate objects called runners. The runners are embedded in requests and are executed by brokers to deliver a service as can be seen in the Figure 2.2. The runners perform various actions, including consuming services from other brokers, in order to deliver a service. Runners are typified by short compute portions followed by long idle times as requested services are fulfilled. For instance, the runner for executing a simulation model may use the request parameters to process the input and create a configuration file, determine the datasets needed, spawn new requests to find the datasets and make them available on the local system, if not already present. Further, it may spawn a new request to execute the simulation on available computing resources and, finally, validate the output before converting it into a common format as part of post-processing. In Simfrastructure, runners offer modularity that simplifies addition, update and removal of workflows. Adding a new workflow requires developing a new runner and embedding it in the corresponding request.

Based on their actions, Simfrastructure brokers can be classified into Standalone and Workflow brokers. Standalone brokers deliver the service by executing the actions hardcoded in them. Standalone brokers are typically used to provide simple, well-defined and specific services. On the other hand, Workflow brokers deliver the service by executing an embedded runner. A workflow broker consists of three main parts: the broker context, the runner embedded in the request, and the data contained in the request. The broker context is a generic component that contains APIs that the runners use to interface with the middleware system, such as access to the blackboard and logging system. As depicted in Figure 2.2, the broker context is always running and monitoring the blackboard for suitable requests. It also controls the execution of the runners. The broker context maintains a set of worker threads and runners. When a runner is ready to continue execution, it is placed in a runnable queue. The idle worker threads remove runners from the queue and execute them. The workflow brokers are useful when providing complex and generic services that can be delivered through one or more workflows.

2.3 Types of Brokers

The Simfrastructure architecture comprises of three types of brokers: edge, service and coordination brokers.

2.3.1 Edge Brokers

Edge brokers serve as the link to resources outside of Simfrastructure. They are the only brokers that can communicate directly with system resources or resources not controlled by Simfrastructure. As such, they provide a layer of security for the rest of the system. Some examples of edge brokers are execution brokers and ingestion brokers. Execution brokers are responsible for the

execution of models on resources such as clouds and grids. Ingestion brokers add external data to the system by providing services for transformation. Examples of external resources include computation hardware, multiple data sources (surveillance data, field data, demographic data, etc.) and people. Edge brokers may spawn or may be spawned by service brokers.

2.3.2 Service Brokers

Service brokers provide services within Simfrastructure. An example of a service broker is a model broker that is capable of executing a particular simulation model. It takes as input a set of simulation parameters, runs a specific model, and provides simulation results. Service brokers may run over many iterative steps and replicates to improve the accuracy of the simulation. A service broker may spawn or be spawned by any other type of brokers.

2.3.3 Coordination Brokers

A Coordination broker serves as the coordinator and mediator of services. This broker is necessary when a workflow spans multiple blackboards, to ensure that service and edge brokers remain unaware of services running on other blackboards. It may also enforce data and other restrictions. Coordination brokers can spawn or be spawned by service brokers or other coordination brokers.

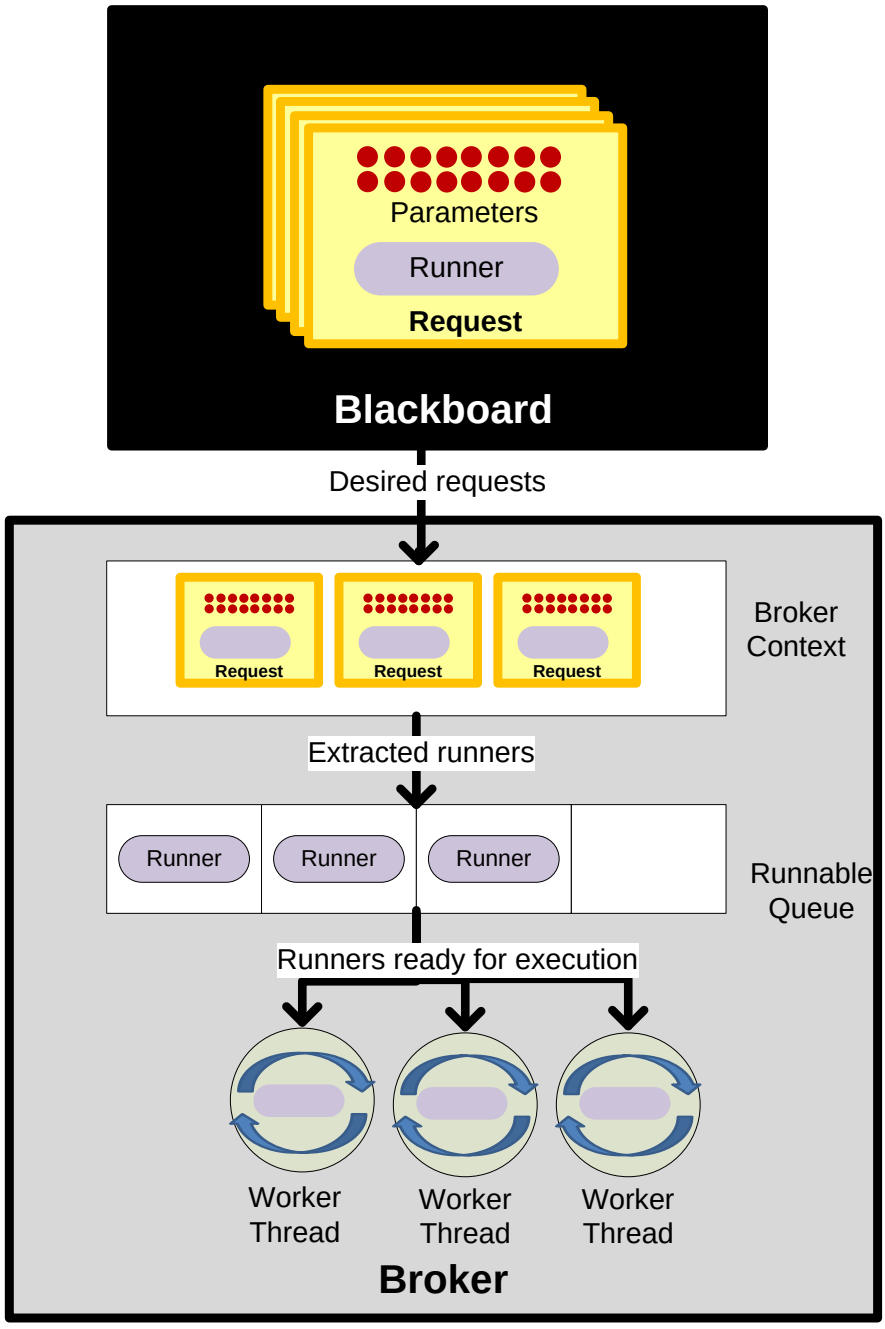

Figure 2.2: A generic depiction of actions in a workflow broker: The broker context fetches the desired requests from the blackboard and adds the embedded runners to the runnable queue. Worker threads retrieve runners from the runnable queue and execute them.

Chapter 3

Services

Simfrastructure provides in-built brokers that provide a set of core services such as job execution, data transfer, resource assignment, digital library, etc. These services can be utilized in many ways to realize the capabilities of modeling and analytical systems. We describe the core services provided by Simfrastructure and their details in the rest of this section.

3.1 Job Execution Service

The job execution service provides a mechanism to execute computational jobs on high-end computing resources such as clusters, grids and clouds. The job execution service is fulfilled by the execution broker when an execution request appears on the blackboard. The execution request contains the job, usually an executable, input and output for execution. Any data required is included either directly in the request for small data (e.g., simulation parameters), or indirectly through an identifier called the Simfrastructure ID (SID) for large data (e.g., input datasets). Any data specified by SID is retrieved prior to execution from the digital library. The execution broker retrieves and submits the embedded job to the underlying job scheduling system. While continuously monitoring the job execution, the broker updates the status of the execution request accordingly.

The execution broker plays a crucial role in integrating various computing resources into Simfrastructure. The runner of the execution broker contains the workflow necessary to execute a job on a computing resource. The runner deals with submitting and monitoring the job with the help of an underlying job submission system. Thus, by developing appropriate runners, Simfrastructure can execute jobs on high-performance computing clusters, compute grids, clouds, volunteer computing platforms or dedicated servers. Assuming the availability of appropriate runners, computing resources can be easily added to Simfrastructure.

3.2 Data Transfer Service

The Data Transfer service provides a mechanism to transfer data from one location to another. The data transfer broker monitors the blackboard for data transfer requests that contain the information about the data that needs to be transferred, and the source and destination locations. When a data transfer request appears on the blackboard, the data transfer broker resolves the location of the data, source and destination and initiates a data transfer.

Data Transfer service leverages the broker architecture to support data transfer over a diverse set of protocols. Specific runners are developed to support data transfer over a variety of protocols like SCP, FTP, HTTP and HTTPS etc. Also, a runner can transfer the data, if small, over the blackboard. The data transfer broker can be extended easily to support of a new data transfer protocol by developing a corresponding runner.

3.3 Resource Management Service

The Resource Management service assigns specific resources to an execution request. The resource management broker frequently collects information about the health, load, and available models and datasets on the Simfrastructure computing resources. It uses this information to identify the best resource that can fulfill a given execution request. The resource management broker evaluates the requirements of the request, and matches it to available resources meeting those requirements. The parameters used for matching by this service include current and expected load on the computational resource, availability of models and datasets, deadline for the request to be completed, constraints placed by the initiator of the request and so on.

3.4 Digital Library Service

To manage the large scale of data related to the simulations and the corresponding metadata, it is essential to have a digital library acting as a central repository of information. Many different services can be written on top of the centralized digital library. Simfrastructure provides a digitial library service called the SimDL [10], which enables holistic management of scientific content produced and used during simulation execution.

SimDL provides services to store, manage, retrieve, and compare scientific content produced through simulations. It also provides services such as searching, memoizing, incentivizing, reusing, reproducing, discovering, curation and preservation. The digital library services are offered by the digital library broker that listens for digital library requests. The digital library requests contain the service that is requested and the associated parameters. The primary digital library services provided by Simfrastructure include Storage, Searching, Memoization and Incentivization.

Execution broker, data transfer broker, resource assignment broker and digital library broker described in this section are edge brokers as they interact with resources external to Simfrastructure.

Chapter 4

Resource Management

As discussed earlier, Simfrastructure provides a generic computational infrastructure that can be used by a variety of socially-coupled systems. This leads to different systems resulting in jobs that may vary in their computational characteristics. While some jobs may entail highly-coupled workflows, others may result in numerous independent tasks as in a bag-of-tasks model. In addition, some jobs may be monotonously serial in nature with a high resource footprint. Such diverse computational characteristics call for employing diverse and appropriate computational resources like clusters, grids and clouds etc. Traditionally, clusters are regarded best suitable for highly parallel jobs with intricate workflows whereas, grids are suitable for high-throughput jobs. On the other hand, clouds are best known for their elasticity in catering to higher resource requirements. Simfrastructure deals with this diversity of computational resources elegantly with the help of Execution Broker.

However, the inclusion of multiple resources brings forth three key aspects of resource management: fault-tolerance, flexibility and resource allocation. **Fault-tolerance** plays an important role in keeping the applications robust towards any failures in computational resources. Typically, computational infrastructures such as clusters and grids are prone to frequent downtimes, albeit short, due to internal failures and maintenance activities. Hence, fault-tolerance will enable simulation systems to be oblivious of any failures in the underlying computing infrastructure. **Flexibility** allows computational resources to be added/removed easily without having to intervene with the functioning of other components. This allows for easy and non-intrusive scalability. **Resource allocation** determines the best possible resource for executing a given job. For efficient execution of a job, its computational characteristics may need to be matched with those of the available resources. For instance, grids may not be an efficient match for a job with an aggressive walltime. Furthermore, resource allocation may also take several other factors such as health, load and geographic distance etc into consideration in order to determine the best possible resource. Hence, resource allocation is a critical aspect that provides an efficient way to execute jobs by matching the computational characteristics of both jobs and available resources.

Thus, fault-tolerance, flexibility and resource allocation are three key factors of resource manage-

ment that contribute to the smooth functioning of simulation systems. Also, supporting multiple resources may not be as efficient and useful without fault-tolerance, flexibility and resource allocation.

4.1 Resource Manager

Resource Manager is a Simfrastructure component that manages, allocates and provisions computational resources required to execute jobs. It provides two key services - resource monitoring and resource allocation - that work in conjunction to provide a smooth and efficient execution of jobs on computational resources. Resource monitoring service enables Simfrastructure to be fault-tolerant and flexible with the operation of computational resources. On the other hand, resource allocation service provides a way to allocate the best from the available resources based on the computational characteristics of a job.

4.1.1 Resource Monitoring

The resource monitoring service monitors all the computational resources integrated into Simfrastructure by regularly collecting various parameters such as health, load, wait time, resource utilization, resource type, available data sets and software packages, organizational domain, geographic location etc. With the help of the information thus gathered, the resource monitoring service identifies available resources and their current state at any given point in time. The resource monitoring service thereby maintains a global state of computational resources. This knowledge is further utilized by the resource allocation service to determine the most appropriate resource.

4.1.2 Resource Allocation

The resource allocation service attempts to determine the best possible resource for a given job. It interacts with the resource monitoring service to fetch the current state of computational resources and then matches the job characteristics with that of the available resources to determine the best match. The matching criterion may be specified over a wide variety of rules that mary vary from one simulation system to the other. The allocation rules may be specified over standard parameters – such as available memory, queue wait time, latency, load, geographic location, organizational domain etc. – as well as application-specific attributes and logic. Application-specific rules, typically, are the best judge of the computational characteristics of resultant computational jobs and therefore the ideal candidates for identifying the characteristics of the desired computational resource as well.

4.2 Design

Resource manager is a distributed module designed to utilize Simfrastructure framework for its operation. It uses the blackboard as the communication mechanism to interact with components. The Resource Manager Broker delivers the resource monitoring and allocation services by monitoring the blackboard for appropriate requests. While the Resource Allocation Service may be invoked by any component that intends to execute a job, the Resource Monitoring Service is, typically, used only by the Resource Allocation Service to fetch the available resources and their current states.

4.2.1 Resource Monitoring Service

The basic function of resource monitoring service is to track the availability and health of resources that are hooked to Simfrastructure. Optionally, it collects other attributes of a resource either actively or as exposed by the resources themselves. The Resource Monitoring Service provides the collected attributes to the Resource Allocation Service thereby enabling it to gain further insight into the computational characteristics of the resources and therefore allocate better resource for a given job.

Heartbeat

The resource monitoring service collects information about computational resources through messages called heartbeats. A heartbeat is a short-lived blackboard object – typically with 30 sec. lifetime – that represents the availability of a computational resource for executing Simfrastructure jobs. The computational resources post a heartbeat to the blackboard and renew it repeatedly to suggest their availability. Therefore, at any given instance, only those resources are considered available whose heartbeats are found on the blackboard.

A heartbeat carries information about computational resources in the form of key-value pairs called resource attributes as shown in Figure 4.1. The resource attributes are classified into three categories named: Standard, Custom and Crawled Attributes.

- **Standard Attributes:** These attributes contain essential information of a computational resource such as its name, namespace, ip address, job manager, organization domain, resource type etc. These attributes must be present in every heartbeat.
- **Custom Attributes:** In addition to the standard attributes, a heartbeat may contain attributes a resource may wish to expose. Custom attributes typically include the geographic location, num. of nodes and disk space available etc.
- **Crawled Attributes:** These attributes are special pieces of information collected whenever necessary. The information carried in Crawled Attributes could be different from case to case. Paths and versions of specific software could be collected through this mechanism.

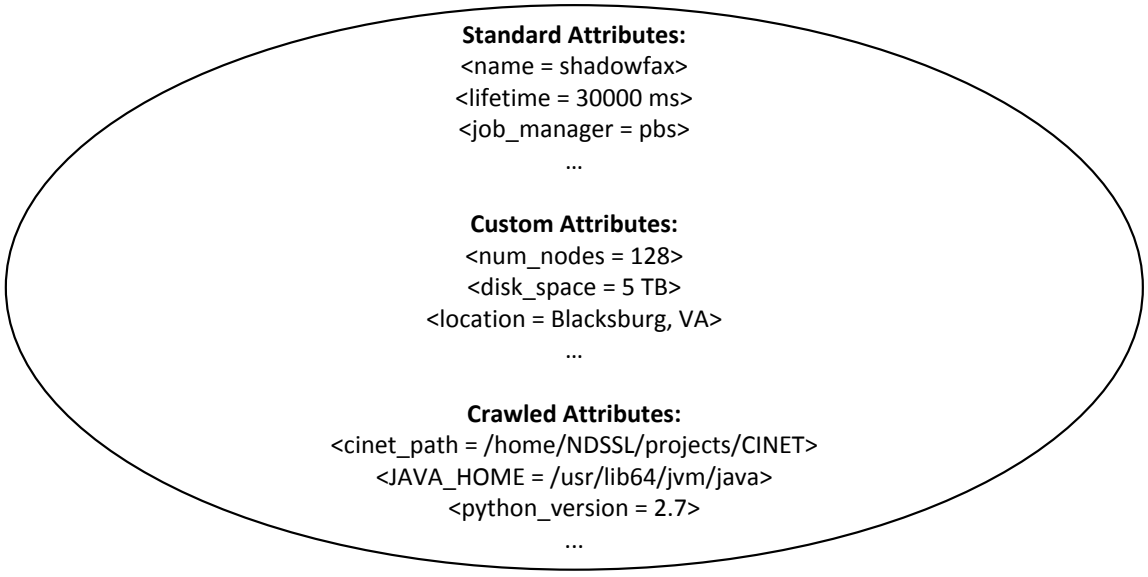

Figure 4.1: The Heartbeat of a computational resource depicting different classes of resource attributes.

Health Monitor

As depicted in Figure 4.2, A Health Monitor is a process that runs on every computational resource and is responsible for posting the heartbeat and repeatedly renewing it. The Health Monitor operates in cycles of short duration called renewal cycles. In each renewal cycle, the Health Monitor posts a heartbeat to the blackboard, if one is not present already, or renews an existing heartbeat. Typically, the renewal cycles are a few seconds shorter than the lifetime of a heartbeat to account for any latencies in the network. The default renewal cycle lasts for 25 sec. – 5 sec. shorter than the default lifetime of a heartbeat i.e. 30 sec.

The Health Monitor, in addition to posting and renewing the heartbeat, populates the standard, custom and crawled attributes of the computational resource into heartbeat. While the standard attributes are predetermined, the custom attributes are read from configuration files. The crawled attributes are obtained through an extension mechanism called Attribute Crawlers. An attribute crawler is a plugin to Health Monitor and collects desired attributes on the computational resource. The attribute crawlers are typically provided by the Resource Manager broker via serialized objects in blackboard entries. The attribute crawlers are unmarshalled and executed by the Health Monitor. The resultant attributes from executing the crawlers are populated into the heartbeats.

The attributes thus collected are used by the Resource Allocation Service during resource allocation.

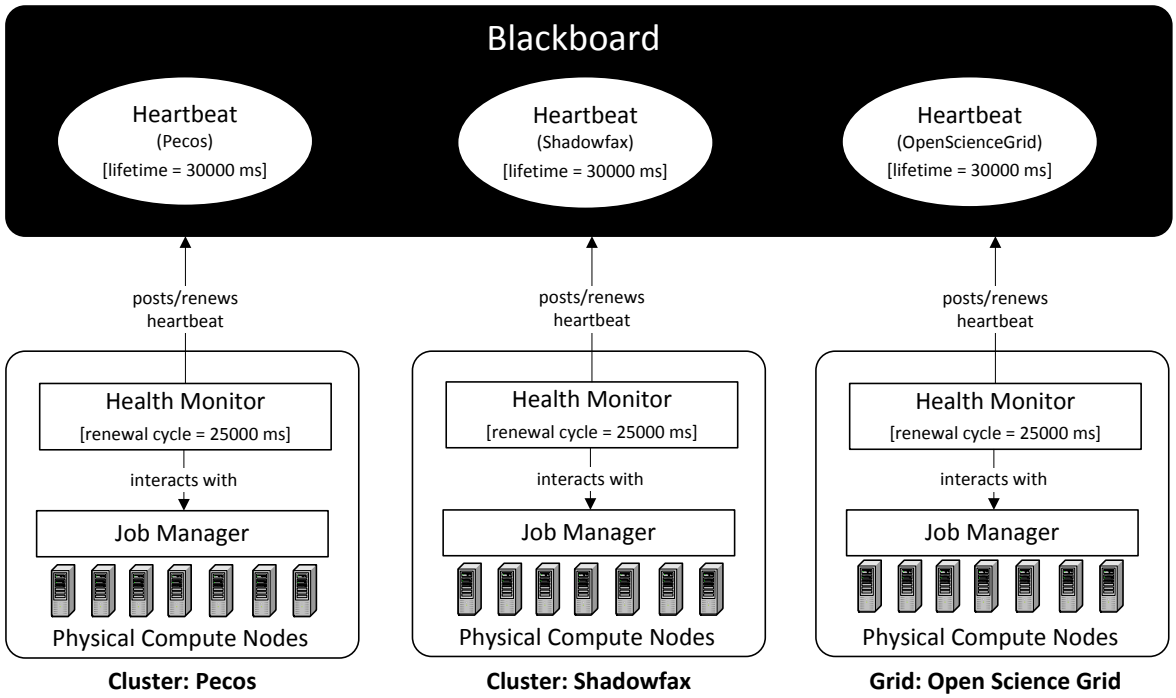

Figure 4.2: A multi-resource set up of Simfrastructure depicting the computational resources – Pecos, Shadowfax and Open Science Grid. An instance of HealthMonitor runs on every computational resource and repeatedly, once every 25 secs, posts/updates the Heartbeat on the Blackboard. The Health Monitor interacts with the Job Manager to collect necessary attributes.

Heartbeat Monitor

The Heartbeat Monitor is an integral part of the Resource Manager broker that helps in tracking the health and load of the computational resources by reading the heartbeats present on the blackboard. As depicted in Figure 4.3, the Heartbeat Monitor reads and converts the information present in the heartbeats into a form the Resource Manager can understand and make decisions.

The mechanism of heartbeats and their repeated exchange between computational resources and Resource Manager helps achieve fault-tolerance and flexibility.

Fault-tolerance

If and when a computational resources goes down, for maintenance or due to internal issues, its Health Monitor would be killed. Once the Health Monitor is nonfunctional, the heartbeat of the corresponding resource would eventually expire its lifetime and disappear from the blackboard.

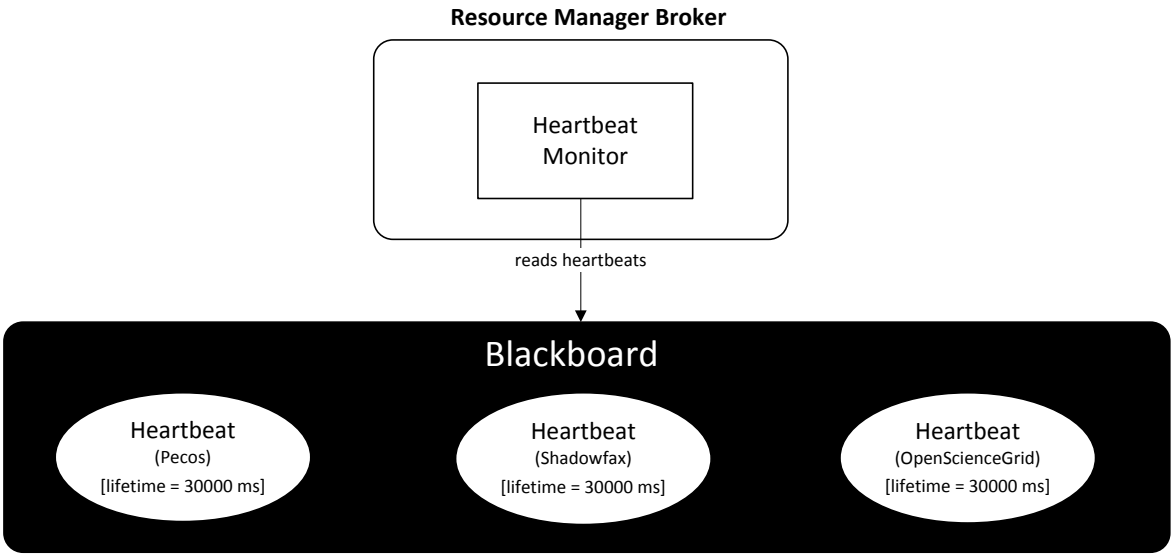

Figure 4.3: The depiction of Resource Manager Broker with Heartbeat Monitor. The Heartbeat Monitor reads the Heartbeats on the blackboard to obtain the available resources for allocation at any point in time.

Thus, the unavailable resource wouldnt not be considered. This way, Simfrastructure jobs are always allocated to run on resources that are currently active. Also, the simulation systems themselves need not actively keep a track of the computational resources and their availability. Furthermore, Health Monitors can be made intelligent to detect partial failure of hardware on computational resources and accurately reflect it in the health information provided with the heartbeats.

Flexibility

The utilization of computational resources in simulation systems is often unpredictable. For instance, during the outbreak of an epidemic, simulation systems may experience an usually high load and consequently higher need for resources when evaluating various countermeasures to curb the contagion. Otherwise, the resource requirements may not be as high and fairly stable. Due to such unstable resource requirements, simulation systems would be benefitted with flexible addition and removal of computational resources.

With Resource Manager, addition and removal of computational resources is easy and non-intrusive. A computational resource would be considered for executing Simfrastructure jobs just by adding an appropriate heartbeat to the blackboard. Similarly, a computational resource may be removed just by taking the heartbeat off the blackboard.

Thus, Simfrastructure systems avail the benefits of fault-tolerance and flexibility with regards to computational resources at no extra cost.

4.2.2 Resource Allocation Service

The resource allocation service aims to determine the most appropriate computational resource to execute a given job. It allocates a resource by determining the appropriate resource type and also a specific resource of the identified type. Also, it takes the attributes and computational characteristics of the job into consideration to determine the attributes of desired computational resources. However, to determine the computational characteristics of a job, the resource allocation service often requires the assistance of application-specific knowledge. The Resource Allocation service embraces a plug-in based design to accommodate application-specific knowledge into resource allocation decisions. The desired attributes thus obtained form application-specific knowledge act as a filter to reduce the available set of computational resources to smaller set before actually allocating a resource.

Also, The Resource Allocation service comprises of several standard resource allocation strategies like Least Loaded, Minimum Completion Time, Least Recently Used etc to determine a suitable computational resource from the filtered set of available resources. Hence, the Resource Allocation service takes two distinct steps: Resource Filtration and Resource Allocation to arrive at an allocation. Domain Handlers and Resource Allocator are the two distinct components of Resource Manager that offer Resource filtration and allocation services, respectively.

Domain Handlers

Domain Handlers are pluggable application-specific modules that understand the parameters of a job in a particular Domain. A domain is typically an application or a class of applications that share similar job characteristics. For instance, many public health epidemiology systems may have similar job parameters and hence share the same Domain Handler. As depicted in Figure 4.4, Domain Handlers, when invoked by the Resource Manager broker, read the parameters of a given job and determine the attributes of a desired computational resource. The attributes thus obtained act as resource templates to discover suitable resources among the available ones. The resource templates are returned to the Resource Manager broker for further processing.

The Domain Handlers may determine attributes of several kind varying from the type of desired computational resource to the required software installed on it. For instance, simulation systems that intend to run user-provided intervention scripts may wish to run the simulation in a sandbox environment such as that provided in clouds due to security reasons. In such a case, the Domain Handler could add the type of desired computational resource to be a cloud. Furthermore, simulation systems may specify specialized software and libraries required for execution as desired attributes in the resource template.

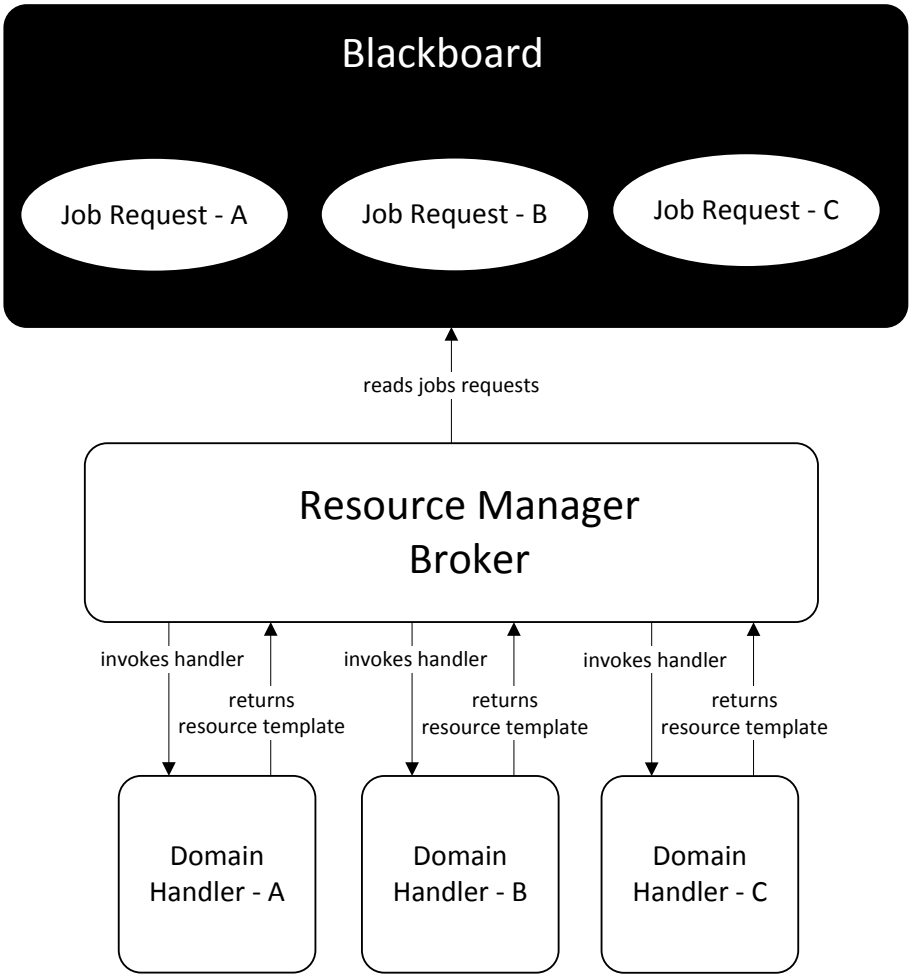

Figure 4.4: The depiction of Resource Manager Broker with Domain Handlers. The Domain Handlers, when invoked by the Resource Manager Broker, read the parameters of a given job and determine the attributes of desired computational resources that are best suitable for job execution. The desired attributes are returned as resource templates.

Resource Allocator

The Resource Allocator allocates a desired resource from the available resources based on the given resource template. The allocation is performed in two steps: Filtration and Allocation. During the filtration step, the Resource Allocator determines a smaller set of resources by matching the resource template attributes with those of the available resources. Any resource from this set of resources is a desired match with respect to the template. However, to determine the best match, the Resource Allocator invokes one of the many allocation strategies, as depicted in Figure 4.5, as

a part of the allocation step. The Resource Allocator ships with allocation strategies such as Least Load, Random and Least Recently Used etc.

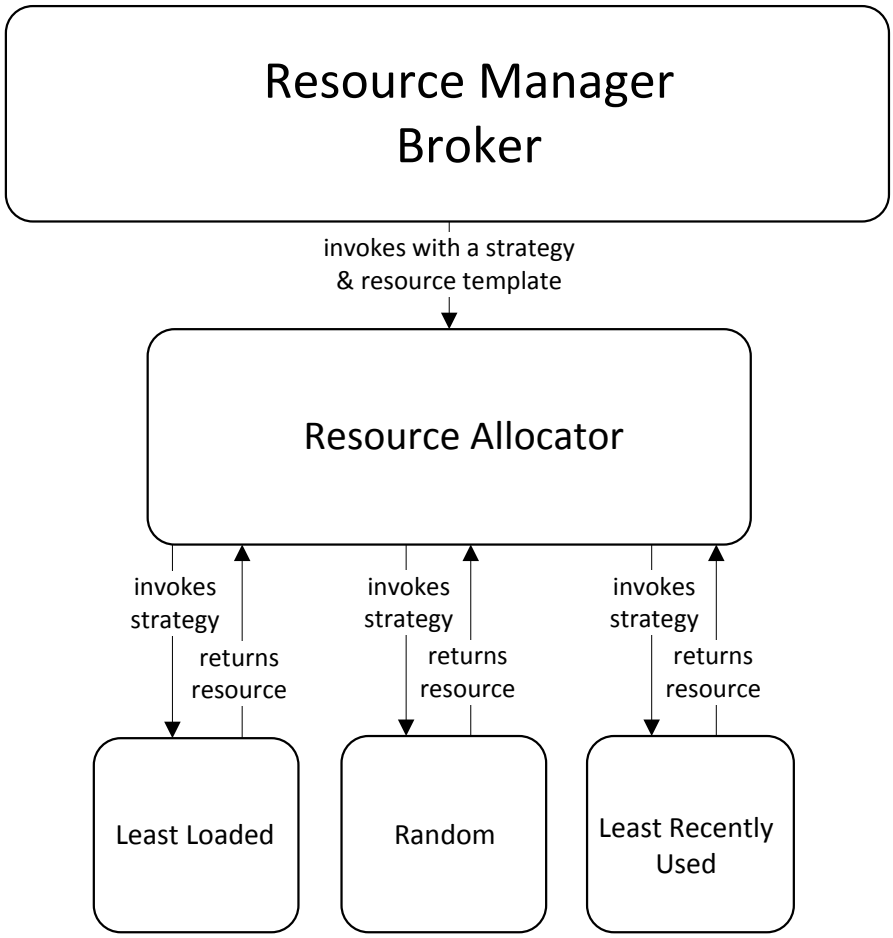

Figure 4.5: The depiction of Resource Manager Broker with Resource Allocator and various resource allocation strategies. The Resource Allocator, when invoked by the Resource Manager Broker, invokes a given resource allocation strategy to determine the most appropriate resource from the set of desired resources.

The Least Loaded allocation strategy takes into consideration the load statistics collected from all the available resources, through heartbeats, and allocates a resource that is least loaded with jobs. For instance, one of the load statistics considered is the number of jobs currently queued and running. The Random allocation strategy picks a resource from the available resource in a random fashion without considering any load statistics. The Least Recently Used strategy aims to allocate a resource that is currently available and least recently used. To determine the last usage

of a resource, the time of submission of the most recently submitted job is considered. In addition to the default allocation strategies, custom resource allocation strategies may be added easily.

In some cases, when there is no specific Domain Handler available to determine a resource template, an empty template is created by the Resource Manager broker. In such a case, the filtration step has no effect as all the resources would be considered a match for the empty resource template. Going further, all the resources would be considered for resource allocation and the best match is determined by the resource allocation strategy employed.

Allocation

With the integration of multiple resources and diverse computational infrastructures, allocation takes paramount importance in order to achieve efficient performance. The Domain Handlers and Resource Allocator work in tandem to provide a comprehensive allocation mechanism that takes into consideration the computational characteristics of the job along with the availability, load statistics and other attributes of the computational resources. This allocation mechanism enables Resource Manager to determine and allocate the ideal resource for job execution.

4.2.3 Naming Scheme

To assist with resource allocation and identification, a three part naming scheme is introduced to uniquely identify the computational resources. Each part of the naming scheme expresses an attribute of the computational resource. As depicted in the Figure 4.6 , three attributes of the namings scheme are: the organizational domain of the resource, the type of computational resource and the universal identifier of the resource. The organizational domain of the resource represents the domain in which the computational resource is hosted. An organizational domain can be represented by a URL. For instance, the organizational domain for Shadowfax would contain vbi.vt.edu as it is hosted in VBI domain. Secondly, the type of the resource would indicate the kind of computational resource such as cluster, cloud, grid and server etc. Finally, the universal identifier would contain the ip address or hostname of the resource itself.

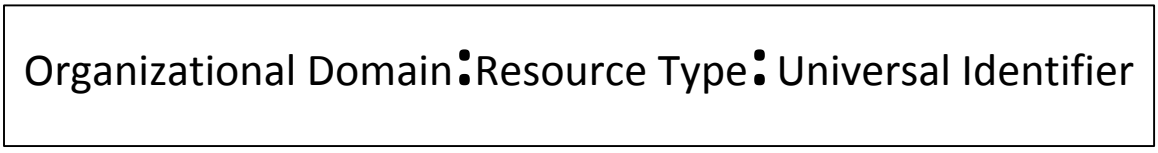

Figure 4.6: The naming scheme followed for resource allocation. The naming scheme contains three attributes: the organizational domain, resource type and the universal identifier.

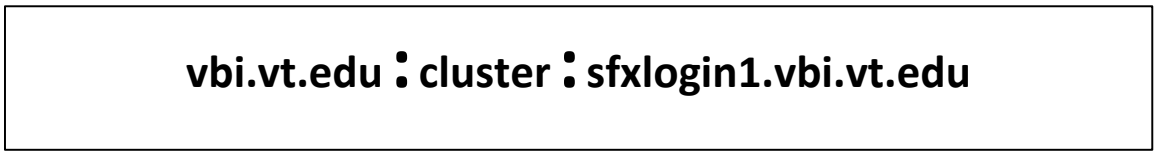

Figure 4.7: A naming scheme indicating a specific cluster identified by ”sfxlogin1.vbi.vt.edu” from ”vbi.vt.edu” to be ideal for job execution.

Figure 4.8: A naming scheme indicating a resource of any type from ”vbi.vt.edu” organizational domain to be ideal for job execution.

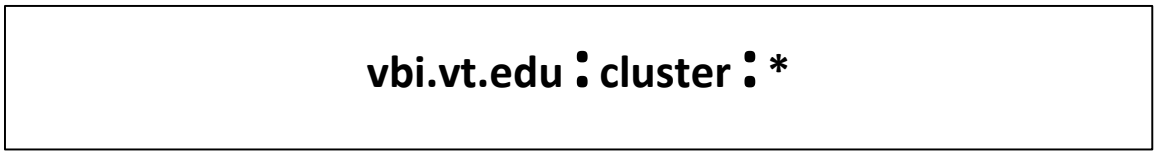

Figure 4.9: A naming scheme indicating any cluster from ”vbi.vt.edu” organizational domain to be ideal for job execution.

Figure 4.10: A naming scheme indicating a cloud from any organizational domain to be ideal for job execution.

The three attributes of the naming scheme determine the genericness of the allocation. When a

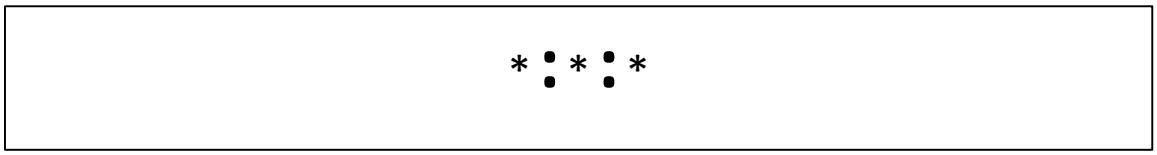

Figure 4.11: A naming scheme indicating a resource from any organizational domain and type to be ideal for job execution.

job goes through the allocation process, it is marked with a template that indicates the resource allocated for its execution. The template may contain zero or more non-empty fields of the naming scheme. The various possibilities of the template are shown in Figures 4.7-4.11. An empty template indicates that a job could be served by any resource. On the contrary, a fully populated template identifies a particular resource that is the most suitable for executing the job. Moreover, partially populated templates may identify one or more resource that could serve the request. Thus, with the help of this naming scheme, allocations in Simfrastructure can be flexible and dynamic.

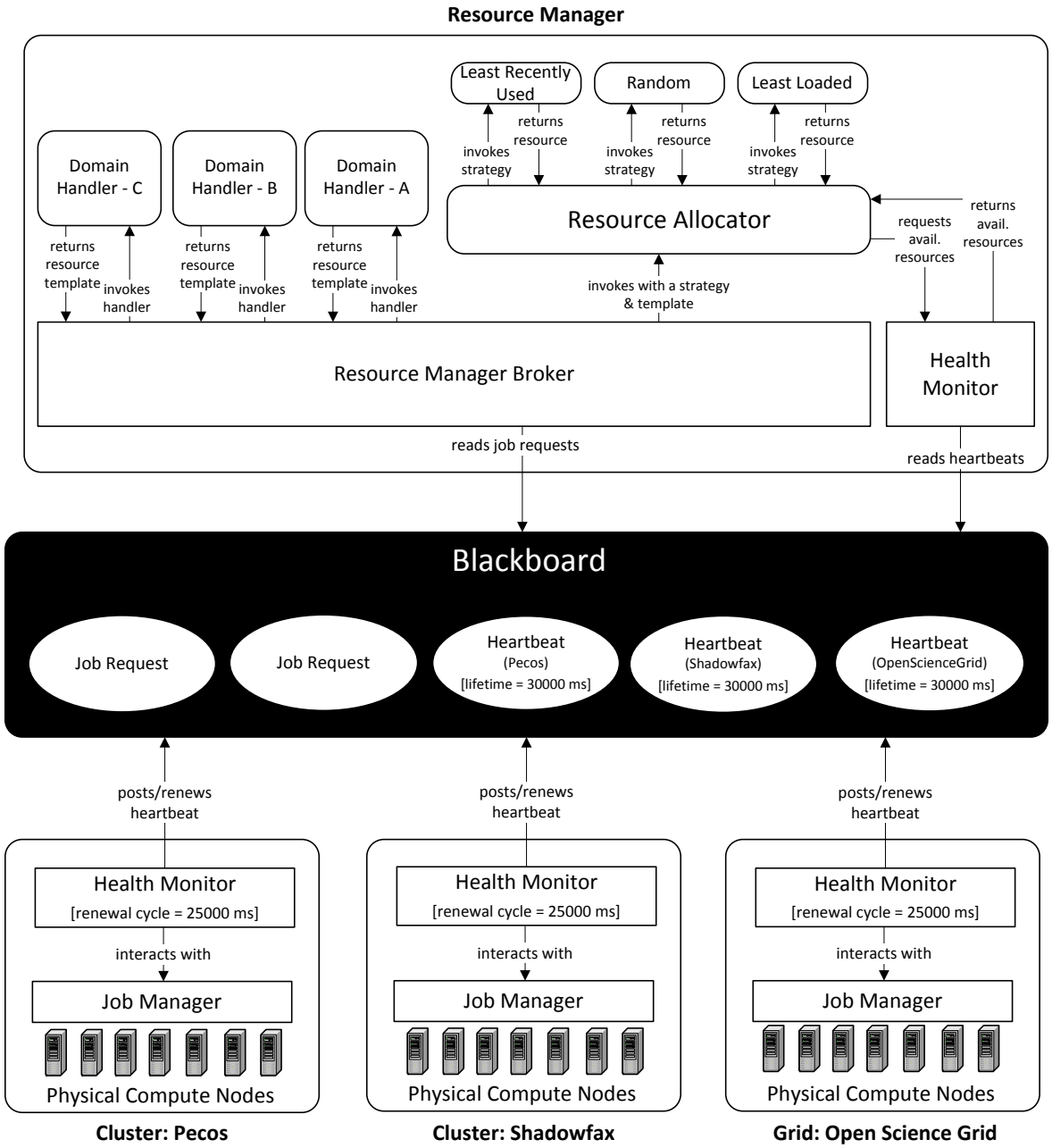

Figure 4.12: A depiction of Resource Manager with all its components. The Resource Manager Broker reads the Job Requests from the Blackboard and hands them over to appropriate Domain Handlers for further processing. The Domain Handlers process the Job Requests and return desired resource templates to the Resource Manager Broker, which then invokes the Resource Allocator with an allocation strategy and the resource template. The Resource Allocator invokes the Health Monitor to fetch all the available resources by reading the Heartbeats from the Blackboard. The Resource Allocator then filters the available resources based on the resource template and invokes the given allocation strategy to pick the best resource from the filtered set.

Chapter 5

Case Studies

Simfrastucture has been used as the middleware platform for developing simulation and analytical systems in different domains. In this section, we present the implementation of a modeling environment extensively used for analytical training and pandemic planning purposes in public health epidemiology.

5.1 Public health epidemiology

Public health decision makers need to use simulation systems as decision support instruments to carry out what-if analysis experiments. In the event of an epidemic, decision makers wish to study the interaction effects of various mitigation strategies and determine the optimal strategy to contain the epidemic effectively. The main challenge for a decision-supporting epidemic simulation system is to provide users with a coherent view of the actual epidemic situation while continuously updating the views as new information becomes available. Thus, the decision support systems should be able to act as cognitive augmentation tools that can offload the cognitive task of forecasting on to the simulation system.

Based on years of expertise in epidemiology, the Network Dynamics and Simulation Science Laboratory developed a simulation system, called Interface for Synthetic Information Systems (ISIS), to assist public health domain experts in planning and decision making. ISIS is equipped with a web-based user interface that allows users to set-up and manage experimental case studies. ISIS enables users to evaluate various intervention strategies by designing epidemiological experiments with various configurations and disease models that execute on synthetic population data of different regions.

To enable coordination between the web-based interface and different simulation models at the back-end, Simfrastructure has been used as the primary coordinating mechanism in ISIS. The simulation models used as the simulation execution engines include EpiSimdemics [6], EpiFast [5]

and Indemics [11]. By virtue of the flexibility offered by Simfrastructure, new epidemic models, datasets and other dynamic information can be quickly incorporated into the ISIS system and thus enable future pandemic preparedness, planning and analysis. Many real experimental case studies have been executed using the ISIS system including [12] and [13]. A typical workflow of an epidemic simulation in ISIS is as follows.

5.1.1 ISIS Design

Figure 5.1 depicts the workflow of a multi-cell epidemic simulation executed using ISIS as the simulation tool and Simfrastructure as the coordinating middleware platform. The epidemic model used in this workflow is EpiFast. Typically, an epidemiological experiment has three major steps - executing the multi-cell simulation, computing the summary of cells and analyzing the results. Due to space constraints, we skip the description of the analysis step, which takes a similar course of action as the other two steps.

An analyst accesses the ISIS web-based interface through a web browser to set-up and design experiments for an epidemiological study. Experiment parameters such as the epidemic model, disease model, number of cells, initial seeds, etc. are selected through the interface. Once the experiment is submitted, the selected parameters are sent to the server. Based on the experiment parameters, the ISIS server creates an appropriate request and places it on the blackboard. As the epidemic model is EpiFast in this case, an EpiFast request is created holding all the parameters required for the EpiFast model and a corresponding embedded workflow inside the EpiFast runner.

The EpiFast broker is a service broker configured to monitor and serve the EpiFast requests. In order to complete the simulation, the EpiFast broker has to perform two major steps: running the multi-cell EpiFast simulation and summarizing the results from each cell.

Firstly, the EpiFast broker aims to the run EpiFast simulation on a computing resource. The EpiFast broker extracts the simulation parameters embedded in the request and writes them to a configuration file on the file system. To initiate the simulation, the EpiFast broker posts an execution request to the blackboard with an embedded job script to run the EpiFast binaries on the underlying compute resource. The execution broker picks up the execution request and serves it by submitting the embedded job script to the underlying job scheduling system for execution. The EpiFast binary starts the simulation with regards to the configuration earlier written to the file system and writes the simulation output back to the file system. Once the simulation terminates successfully, the execution broker, while actively monitoring the job execution, updates the status of the execution request as completed.

Secondly, the EpiFast broker monitors for the success of the earlier execution request and posts another execution request to the blackboard for cell summarization. This time, the EpiFast broker embeds a new execution request with a job script to invoke the cell summarization binaries. The execution request takes a similar course of action as in the earlier step and reports the success of cell summarization. Going further, the EpiFast broker updates the status of the initially placed Epi-

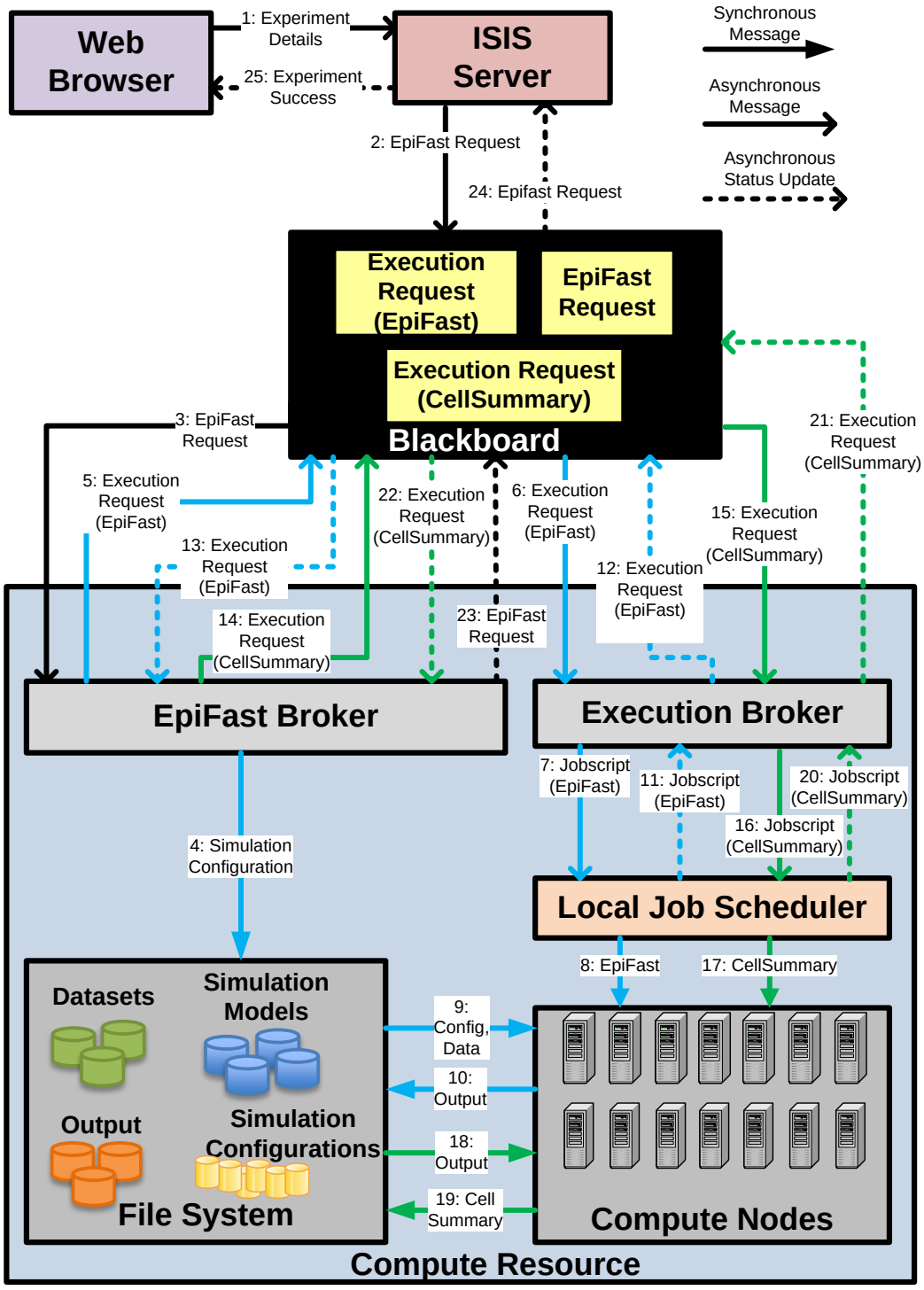

Figure 5.1: Sequence of activities in a typical workflow of a multi-cell epidemic simulation executed using ISIS as the simulation tool, EpiFast as the epidemic simulation engine and Simfrastructure as the coordinating middleware platform.

Fast request as completed, which triggers the ISIS server to convey the success of the experiment to the end-user.

The workflow described in this section has been followed by many experimental case studies in the domain of public health epidemiology.

5.2 Network Science

Many socio-technical systems such as the urban transportation systems, electric power grids, communication and computer networks, social sciences and economics etc employ graph abstractions, theory and algorithms to understand their structural properties and dynamical behavior. To study such complex socio-technical systems, graphical modeling tools with mechanisms for representing and manipulating large scale graphs are essential. CINET, A CyberInfrastructure for Network Science, is developed to help with large-scale graph processing and dynamic graph computations. It includes several graph measures such as the vertex degree, edge degree, clustering coefficient and betweenness centrality etc provided by various graph libraries such as GaLib [14], NetworkX [15] and SNAP [16]. CINET enables users to take advantage of the cyberinfrastructure by enabling the addition of custom graphs and measures. This leads to an increase in the number of graphs and measures over time. Moreover, the computational requirements of the graph processing algorithms may vary significantly based on the complexity of the measure and the size of the input graph. For instance, to find the clustering coefficient of a densely connected graph, with a million nodes and up to 10 million edges, in a reasonable time period, a high-end computing resource like a high-performance cluster may be required. Whereas, to find the vertex degree of a 100 node graph, the computing capacity of a desktop computer may just be sufficient. Also, as discussed in the previous sections, different graph measures and analyses may result in jobs of different nature. While, some graph analyses may result in many serial jobs others may result in massively parallel jobs. Furthermore, CINET is expected to deal with security issues that may arise in executing user-submitting graph analysis programs. To develop such complex large-scale graph analysis infrastructure over a diverse set of graph measures and computational resources, we utilized Simfrastructure and its resource management capabilities.

5.2.1 CINET Design

Figure 5.2 depicts the workflow of a graph-measure analysis performed using Granite as the application front-end and Simfrastructure as the coordinating middleware platform. The user selects graphs and measures from the user interface after which a series of jobs, one for each graph and measure selected, are submitted to the computational resource – Shadowfax, an HPC cluster hosted at VBI, in this case – for analysis. Once the analysis is completed, the results are transferred back to the user and shown via the user interface.

Firstly, an analyst accesses the Granite web-based interface through a web browser to set-up a

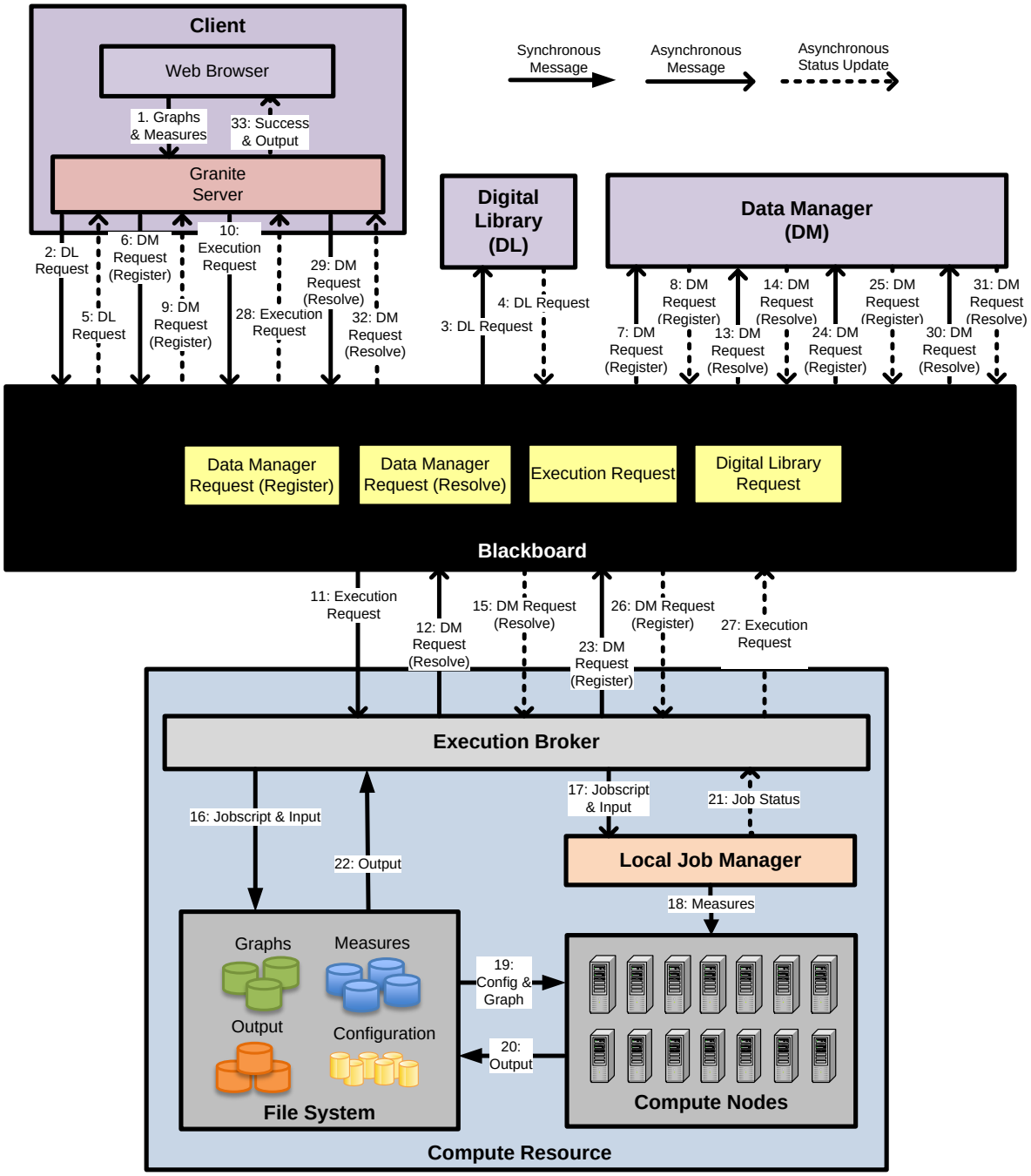

Figure 5.2: Sequence of activities in a typical workflow of a graph measure analysis using CINET.

graph analysis request. The user selects a set of graphs and measures from a list shown on the user interface. Along with graphs and measures, users may also set parameters for measures through the

user interface. Once the analysis is submitted, the selected graphs, measures and their parameters are sent to the server. The Granite server then creates and posts a Digital Library (DL) Request to the blackboard to fetch additional information about the selected graphs and measures.

The Digital Library Broker, configured to respond to DL Requests, picks up the DL Request and fetches the information related to graphs and measures from the database. This retrieved information is sent to the Granite server by updating the DL Request appropriately. Typically, the information sent in the DL Request contains details about the paths, names, description of the graphs and measures. The Granite server utilizes this information in creating appropriate Execution Request. However, before creating the Execution Request, the Granite server registers any data that needs to sent over for execution with the Data Manager. The Data Manager broker, configured to serve the Data Manager Requests, reads the request off the blackboard and registers the data included in the request. In return, the Data Manager broker returns a symbolic ID that could be used to resolve the data back to its original form.

The Execution Broker reads the Execution Request from the blackboard and initiates the analysis. It starts off by resolving the data embedded in the Execution Request by placing a Data Manager Request on the blackboard. The Data Manager updates the request with data corresponding to the IDs. Then, the Execution Broker writes the resolved data on the file system for later use. Following this, the Execution Broker submits the job script that encodes the commands to run the analysis to the job manager. Once the job is submitted, the Execution Broker monitors the status of the job continuously. Upon completion, the outputs are registered with the Data Manager and the resultant IDs are updated in the Execution Request. The Granite server, then, reads the completion status of the request and resolves the IDs of output. The output thus obtained is shown to the user via the web browser.

5.2.2 Prototype with Resource Manager

Figure 5.3 depicts the workflow of a similar graph-measure analysis described in Figure 5.2 in the context of the proposed design that utilizes the services of Resource Manager. In addition to the Simfrastructure components described earlier, the proposed setup includes the Resource Manager that enables efficient utilization of multiple resources – Pecos, Open Science Grid (OSG) and FutureGrid (FG) in addition to Shadowfax. While Pecos is an HPC cluster hosted at Virginia Tech, OSG is a computational grid and FG is a grid testbed that provides a platform for on-demand resource provisioning and utilization. Both OSG and FG are hosted by their respective consortiums to in an effort to further scientific research.

As in the previous case, the user starts off an analysis by selecting a set of graphs and measures from the web interface. The Granite Server, instead of creating the Execution Request itself like in the earlier case, creates and posts a Resource Manager request indicating the graphs and measures it intends to process. The Resource Manager is tasked with creating the appropriate Execution Request with a resource allocation. In order to do so, The Resource Manager reads the request from the blackboard and delivers the request to the Granite handler, which is the Domain Handler

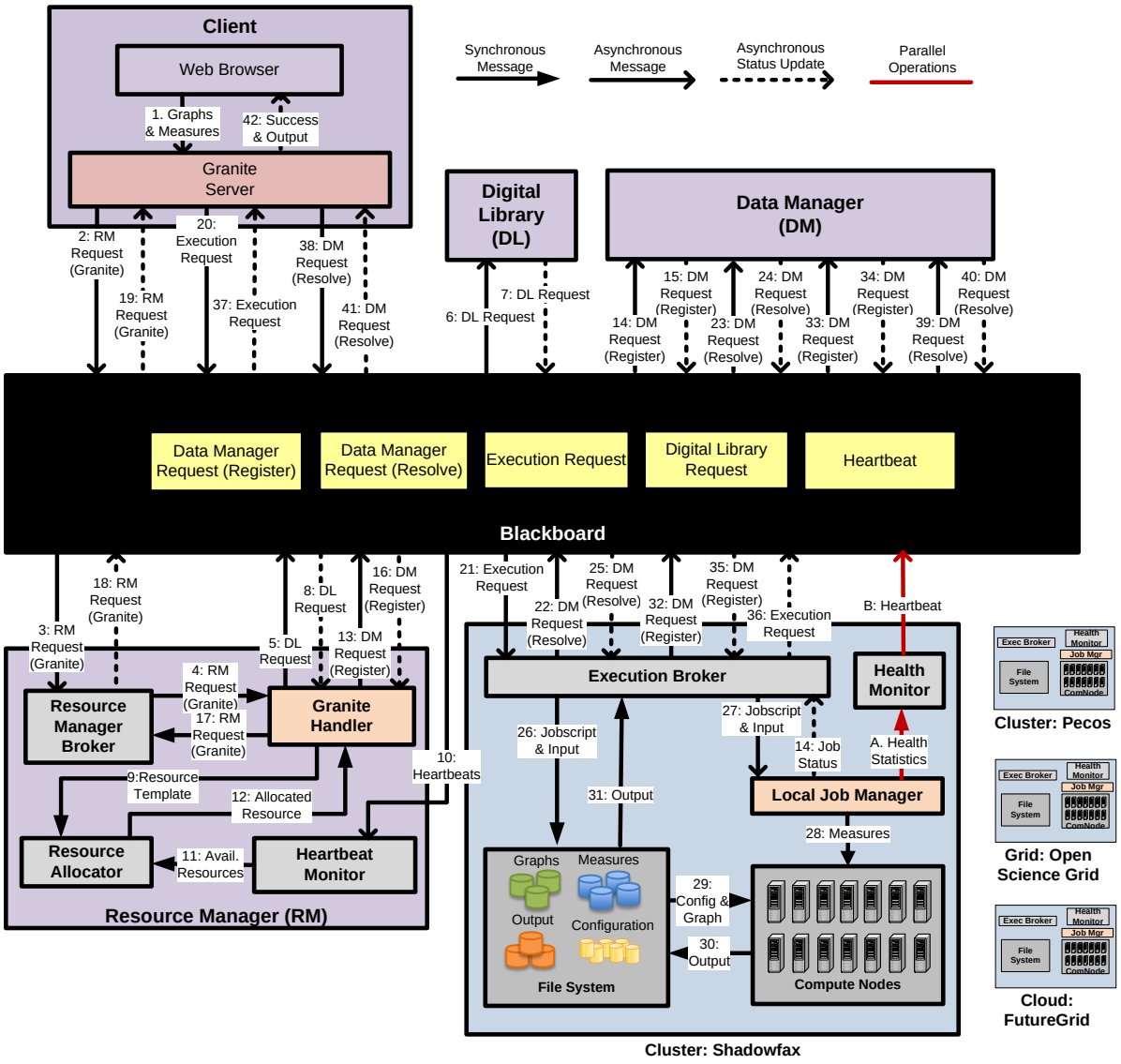

Figure 5.3: Sequence of activities in a typical workflow of a graph measure analysis using CINET in a multiple resource setup. The resource manager is utilized to determine the most appropriate resource for the execution of analysis.

for Granite. The Granite Handler, as in the previous case, tries to fetch more information about the selected graphs and measures by placing a DL Request on the blackboard. Upon fetching the required information, the Granite Handler creates a resource template, indicating the ideal resource for its execution, and hands it over to the Resource Allocator to find a suitable resource that fits the template. Following this, the Resource Allocator invokes the Heartbeat Monitor to collect all the available Heartbeats from the blackboard. By reading the Heartbeats and the properties included

in them, the Resource Allocator determines the most appropriate resource that fits the template. When the Resource Allocator returns the allocated resource, the Granite Handler registers any input data with Data Manager before creating the Execution Request marked with the allocation name of the allocated resource. The Execution Request thus created is updated on the Resource Manager Request originally posted on the blackboard by the Granite Server. Upon reading the response, the Granite Server posts the Execution Request on the blackboard for execution. From this point onwards, the flow continues as in the earlier case.

The Health Monitor process installed on the computational resources repeatedly posts and updates the Heartbeats on the blackboard. The Health Monitor interacts with the local job manager to collect load statistics and reflect them in the Heartbeats.

Granite Handler Figure 5.4 depicts the series of actions and decisions taken in the domain handler for Granite called Granite Handler. The Granite Handler reads the graph analysis request and determines a graph library that is best suited to serve the request. The Granite Handler creates a resource template indicating the presence of chosen graph library in the desired resource.

The Granite Handler starts off by reading the graph and measure contained in the request. If a measure is available in only one graph library, it is chosen and included as a desired characteristic in the resource template. On the other hand, if a measure is available in multiple libraries, the size of the graph is taken into account. The library GaLib is chosen if a graph is big, otherwise NetworkX is chosen. The 'big'ness of the graph is determined by the number of nodes and edges contained in it. Also, if the chosen measure is one of those uploaded by the user, it is indicated to in a sandbox environment on cloud for security. Otherwise, any resource with the chosen library should suffice.

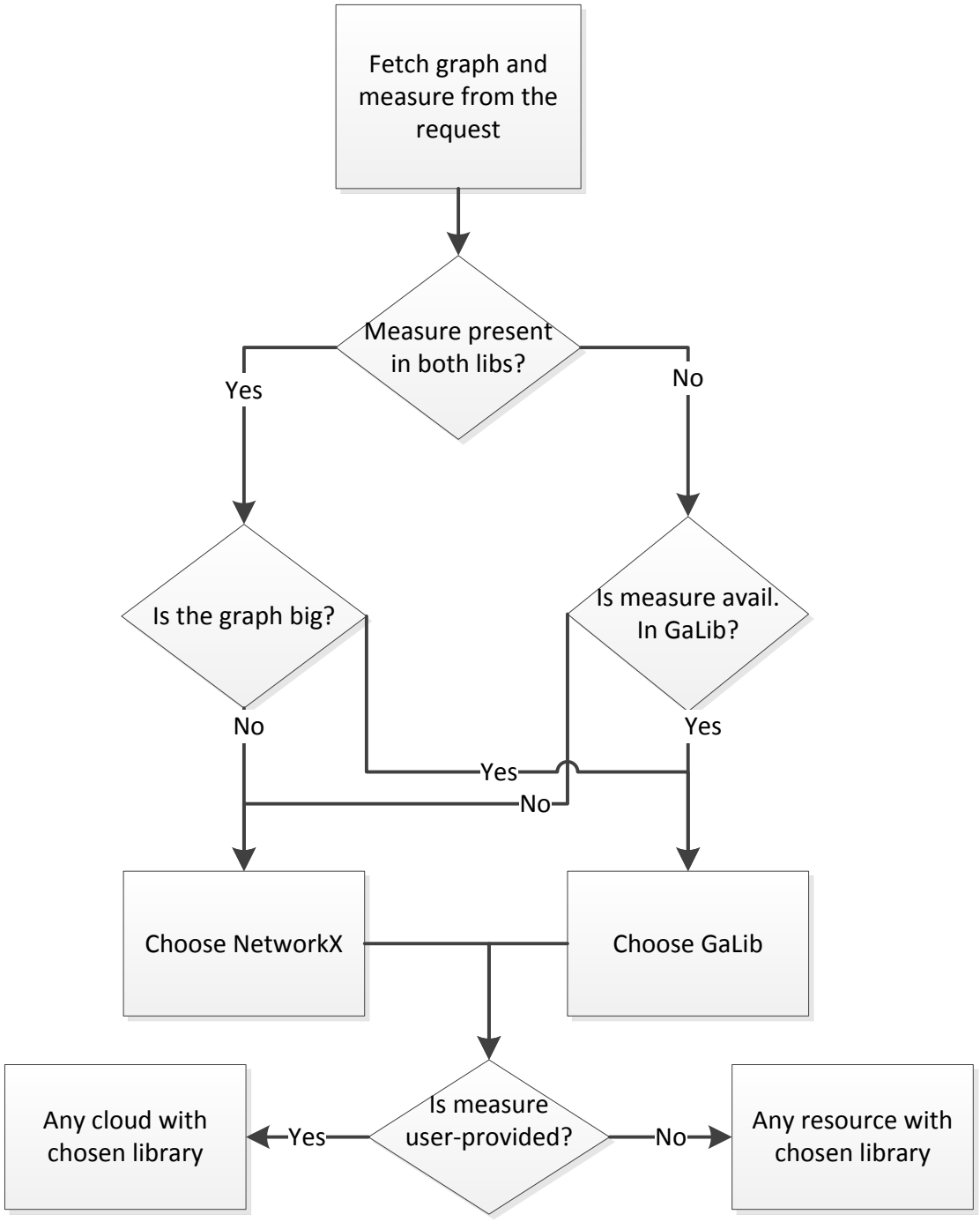

Figure 5.4: Flowchart depicting the series of actions and decisions performed in Granite Handler to decide an appropriate graph library for the given request.

Chapter 6

Discussion

In this section, we discuss how the loosely coupled architecture of Simfrastructure provides essential qualities of scalability, fault tolerance, quality of service and load balancing, in development of integrated modeling environments. We present our evaluation of the framework on each of the performance characteristics below:

6.1 Computational complexity

Simfrastructure employs a space-based architecture, enabling easy coordination between the components of a simulation system through anonymous communication. This greatly simplifies coordination among components since they need not be aware of the existence of other components in the system. Components are then self-contained as the computational complexity of each component is hidden within itself such as the simulations. Also, this achieves loose-coupling among components thereby improving the maintainability and usability of the system.

6.2 Scalability

In Simfrastructure, as the components are loosely-coupled, they can be added, removed and changed transparently. For instance, additional computing resources may be added to the simulation system to increase computational capacity without having to interrupt any running jobs and components. Similarly, computing resources may be removed if they are underutilized. Moreover, any request submitted to the blackboard can be served by any available broker relevant to the request. Hence, scalability with respect to the service provided is inherently handled by our system.

However, as the blackboard holds all the requests for further processing, the scalability in terms of the number of requests served simultaneously may be limited by the total memory available

to the blackboard. To study this further, we carried out an experiment to evaluate the capacity of blackboard by hosting a JavaSpace with a 1 GB Java heap size. Further, we created a typical Simfrastructure request and populated the blackboard to its capacity, i.e., until it ran out of memory. Before going out of memory, the JavaSpace could hold 200,000 requests, which translate to around 40,000 simultaneous experiments, assuming every experiment generates four other requests, each corresponding to the services mentioned in Section 3. This limitation of 200,000 requests would become a bottleneck when the underlying computing hardware is capable of serving more than 40,000 experiments simultaneously. Also, increasing the heap size of JavaSpace will further increase the capacity of blackboard and consequently the number of experiments handled by Simfrastructure. However from our experience, though the scalability of blackboard is an important factor, it does not play a significant role as in the most cases, scalability is limited by the throughput of underlying computing infrastructure.

6.3 Latency

One of the sources of latency in Simfrastructure is caused by the *read* and *write* operations on the blackboard. To study such latencies, we measure the time taken to read and write the 209,700 requests created in the previous experiment to measure the capacity of blackboard. Time measurements of all the requests are divided into 210 buckets, with each containing the time measurements of 1000 requests. For each bucket, we plot the minimum, maximum and the 95^{th} quantile of time measurements on a log scale for read and write operations as shown in Figure 6.1. As shown by the *Min* and *95^{th} Quanile* lines, the time taken to read and write requests increases gradually as the number of requests on the blackboard increases. So, a higher load on the blackboard may lead to higher latency in the end-to-end processing of a simulation. We attribute the abnormally high values of *Max* plots to time incurred by the Java garbage collector.

As explained in the Architecture section, the request for a given service might include an identifier to the data stored in the digital library, if the scale of data is large. This look-up operation can be a bottleneck and lead to latency. However, for socially coupled systems, the latency incurred due to look-up operations is several orders of magnitude smaller than the total execution time of simulations. Hence, the latency in look-up operations is not perceivable during the simulation execution. For instance, the latency incurred for look-up operation in an epidemic simulation using EpiFast, on the Miami region with a population of around 2 million, is 3 seconds. While, the simulation running time is approximately 15 minutes. Thus, the latency is negligible compared to the simulation execution time and is not a significant factor for middleware latency consideration. The workflow of this use-case using EpiFast is explained in Section **??**.

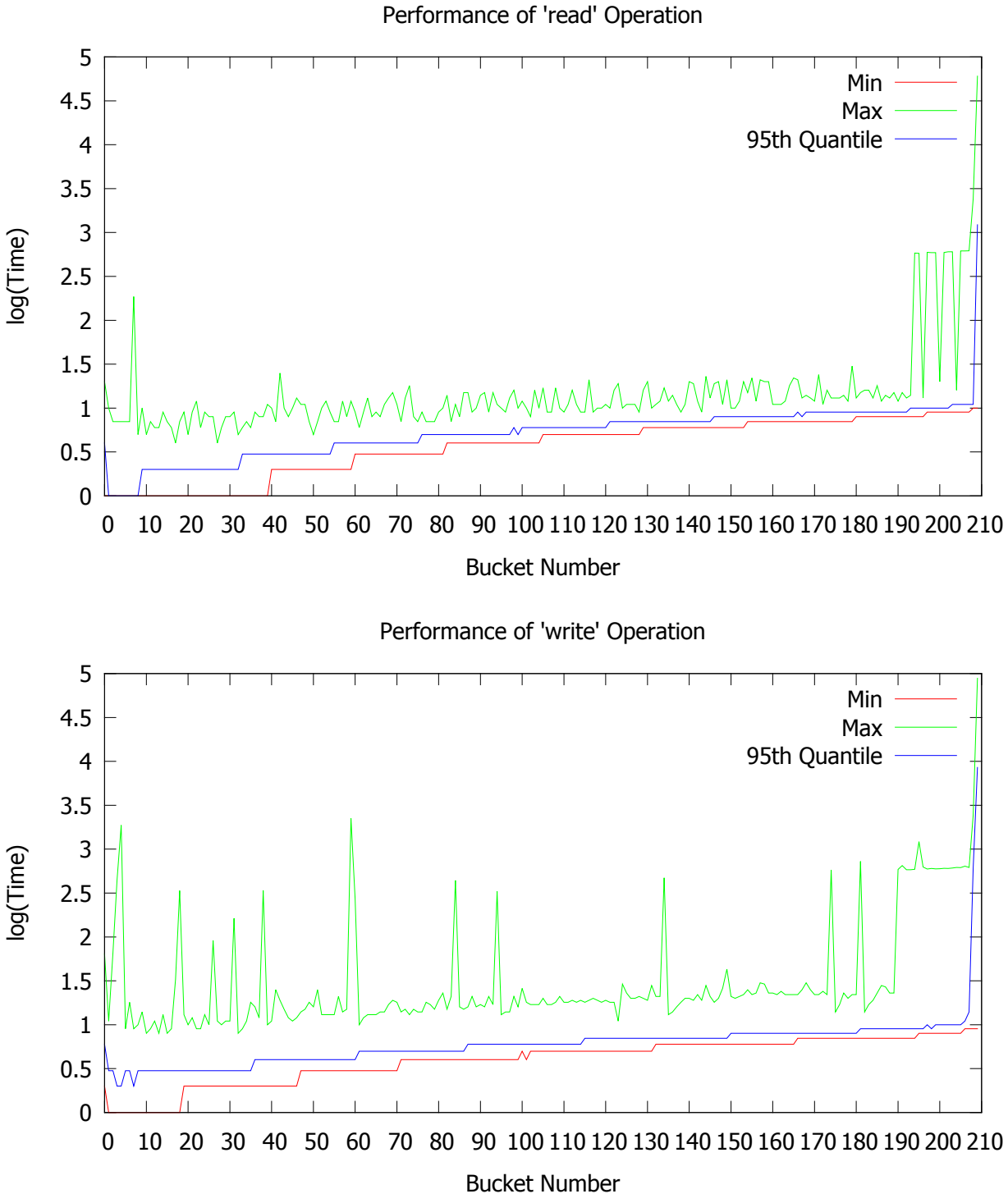

Figure 6.1: Performance of *read* and *write* operations with over 200,000 requests divided into 210 buckets, each of size 1000. For each bucket, minimum, maximum and the 95^{th} quantile time values, in milliseconds, are plotted on log scale.

6.4 Fault-tolerance

In the current architecture of Simfrastructure, as many brokers can co-exist and be dynamically added to offer a particular service, if one broker is down, another broker can serve the same request without the loss of continuity. However, a crash in the blackboard, which is the central communication mechanism, may critically affect the system. Presently we are studying various fault-tolerance strategies like adding redundancy in the shape of a secondary blackboard, and employing hierarchical and distributed spaces to give the illusion of a single blackboard. Currently we use watchdog scripts to regularly probe the status and take corresponding actions with regards to brokers and blackboard.

Chapter 7

Related Work

There have been many recent advancements in the field of high performance computing systems and middleware platforms to support coordination between the components of such systems. However, to the best of our knowledge, our work on Simfrastructure is the first effort of its kind in developing a middleware platform that aims at providing seamless access to powerful computational models and resources for use by subject matter experts. In this section, we list some of the recent research relevant to our work.

In paper [17], the authors argue the need for delivering high performance computing resources as a service to scientists and domain specialists. The authors provide a prototype implementation of an elastic cloud that provides high performance computing infrastructure as a service using the IBM BlueGene Supercomputer. This new innovation called HPCaaS (HPC as a Service) derives from the positives of both cloud computing and high-end cluster computing. In our approach, we deliver high performance computing services as well, but as an indirect consequence. Our focus is primarily on delivering higher level services, in particular analytics and modeling for socially coupled systems. We also differ in the types of computational resources we address including clusters, clouds and grids. On top of providing an easy coordinating mechanism between user-interfaces, data stores and computing resources, our space-based architecture is also used to provide resource allocation and management, data transfer and digital library services.

Some of the other related work in this domain includes the work on Narada brokering architecture and Granules. The Narada brokering architecture [18] provides a distributed brokering system with a brokering middleware that combines a hybrid environment of peer-to-peer systems and grids to provide web services. "Granules" [19], a streaming based runtime environment, extends the basic brokering architecture from "Narada" for executing complex scientific applications on the cloud using a peer-to-peer system of communication between its broker components. It also supports the existing MapReduce framework [20] and variants of it.

Our approach greatly differs from these approaches in that our work does not focus on any particular programming model such as MapReduce to provide services. Instead, we provide a platform

for large scale socially coupled systems to provide and access services without having any constraints on the models used, type of data required or computing resources available. For instance, our approach does not require the simulations to follow certain norms of execution parallelism whereas MapReduce operations typically need the datasets to be highly parallel in nature. Hence our approach can be used in cases where large scale sequential execution has to be supported, along with cases where massively parallel operations are to be executed on the grid. Moreover, our approach uses tuple space based architecture, instead of earlier approaches used in the literature such as peer-to-peer systems.

Space-based architectures have gained popularity and applicability in various domains ever since the work of Gelernter et al at Yale [21]. The papers [22], [23], [24], [17] and [25] explore space-based architectures for building adaptive distributed systems. The novelty of our approach lies in bringing together a comprehensive set of services involving simulations, data stores, computing resources and user interfaces required for automating socially coupled systems. Resource management, job monitoring, data transfer management and digital library service are a few examples of the services. Our approach also makes it possible to provide a powerful analytical platform for domain-experts.

Space-based architectures provide a natural way for coordination in a spatially, temporally and logically decoupled manner. However, space-based architectures are known to have certain limitations, some of which are thoroughly studied and addressed in the literature [23, 26, 27]. In the following sections of the thesis, we argue some of these limitations are not applicable in our context, given the requirements of our system.

Chapter 8

Future Work and Conclusion

8.1 Future Work

Before concluding the work presented in this thesis, we describe a few areas that can be further improved upon.

8.1.1 Blackboard

Though the Blackboard provides a loosely-coupled way of communication and coordination, it stands as a single point of failure. Traditional strategies like adding redundancy, in the shape of a secondary Blackboard, may be employed to increase the robustness of Blackboard. Further, Blackboard can be conceived as a virtual space that is realized by many spaces to increase the capacity of the system. Blackboard may be distributed into several spaces based on the type of objects they store. This allows for better scalability and robustness in the Blackboard and also the system as a whole.

The inherent communication model of Blackboard, through object exchanges, may introduce inefficiencies due to the latency of *read*, *take* and *write* operations. As the latency of operations vary with the size and number of objects held on the Blackboard, a distributed caching mechanism maybe employed to store objects of interest at every broker's location. Objects in the cache maybe stored based on the type of objects frequently accessed. However, this may need distributed cache updating mechanism to maintain the state of an object across the system.

8.1.2 Rule Engine

Currently, all the domain-specific rules of resource allocation are encoded in Domain Handlers. Though Domain Handlers are designed to work in a pluggable mechanism, adding a new domain or

rules, and changing or removing existing rules may require code changes and hence re-deployment. Thus, externalizing resource allocation rules may be helpful. A rule engine driven approach may offer greater flexibility in terms of adding, changing and removing existing domains and rules. This approach also enables users without any knowledge of the code or administering the system to easily change the rules through a user-interface.

8.1.3 Performance Measurement

The performance of the computational infrastructure as a whole is yet to studied thoroughly. The performance of Simfrastructure needs to be evaluation at both component-level and system-level. The performance of each individual component such as the Blackboard, Data Manager, Digital Library, Execution Brokers etc needs to be studied and understood in the context of the whole system. Also, measuring the performance under varying workloads may give an insight into the effectiveness of Resource Manager in handling different resources and jobs.

8.2 Conclusion

In this thesis, we have described a flexible middleware platform named Simfrastructure, which automates the set-up, management and execution of large-scale modeling environments, based on different models and runs on a range of computational resources including clouds, clusters and grids. The computational infrastructure of Simfrastructure provides key services such as data and knowledge management, resource management and allocation, and job execution for setting up modeling environments with ease. This enables easy integration of modeling environments into the computational infrastructure as front-end applications.

The space-based architecture of Simfrastructure enables easy development and maintenance of components. With the resource management capabilities of Simfrastructure, the front-end applications may leverage the computational capabilities of diverse computational resources at no extra cost. We are working on many improvements to the basic architecture to enhance its scalability, efficiency and flexibility. In our experience, modeling environments are typically constrained by the complexity of analysis rather than scalability or latency limitations imposed by the middleware platform.

Bibliography

[1] S. Dirks and M. Keeling, "A vision of smarter cities: How cities can lead the way into a prosperous and sustainable future," *IBM Institute for Business Value. June*, 2009.

[2] A. Holtslag, D. BRUIJN, and H. Pan, "A high resolution air mass transformation model for short-range weather forecasting," *Monthly Weather Review*, vol. 118, no. 8, pp. 1561–1575, 1990.

[3] J. M. Epstein, "Modelling to contain pandemics," *Nature*, vol. 460, no. 7256, p. 687, August 2009.

[4] D. Helbing, *How to Do Agent-Based Simulations in the Future: From Modeling Social Mechanisms to Emergent Phenomena and Interactive Systems Design*.

[5] K. R. Bisset, J. Chen, X. Feng, V. S. A. Kumar, and M. V. Marathe, "EpiFast: a fast algorithm for large scale realistic epidemic simulations on distributed memory systems," in *Proc. the 23rd International Conference on Supercomputing*, 2009, pp. 430–439.

[6] C. L. Barrett, K. R. Bisset, S. G. Eubank, X. Feng, and M. V. Marathe, "Episimdemics: an efficient algorithm for simulating the spread of infectious disease over large realistic social networks," in *Proc. ACM/IEEE conference on Supercomputing*, 2008, pp. 290–294.

[7] R. Jacob, M. V. Marathe, and K. Nagel, "A computational study of routing algorithms for realistic transportation networks," *ACM JOURNAL OF EXPERIMENTAL ALGORITHMS*, vol. 6, 1998.

[8] K. Channakeshava, D. Chafekar, K. Bisset, V. S. A. Kumar, and M. Marathe, "Epinet: a simulation framework to study the spread of malware in wireless networks," in *Proc. of the 2nd International Conference on Simulation Tools and Techniques*, 2009, pp. 6:1–6:10.

[9] Sun Microsystems Inc. Javaspaces service specification version 2.2.

[10] J. Leidig, E. A. Fox, K. Hall, M. V. Marathe, and H. S. Mortveit, "Simdl: a model ontology driven digital library for simulation systems," in *JCDL*, 2011, pp. 81–84.

[11] K. Bisset, J. Chen, X. Feng, Y. Ma, and M. Marathe, "Indemics: an interactive data intensive framework for high performance epidemic simulation," in *Proceedings of the 24th International Conference on Supercomputing (ICS)*, 2010, pp. 233–242.

[12] J. Chen, A. Marathe, and M. Marathe, "Coevolution of epidemics, social networks, and individual behavior: a case study," in *Proc. of the 3rd International Conference on Social Computing, Behavioral Modeling, and Prediction*. Springer-Verlag, 2010, pp. 218–227.

[13] J. Chen, F. Huang, M. Khan, M. Marathe, P. Stretz, and H. Xia, "The effect of demographic and spatial variability on epidemics: A comparison between beijing, delhi, and los angeles," in *Critical Infrastructure (CRIS), 2010 5th International Conference on*, sept. 2010, pp. 1 –8.

[14] (2013, May) Galib. [Online]. Available: http://cinet.vbi.vt.edu/cinet_new/node/10

[15] (2013, May) Netoworkx. [Online]. Available: http://networkx.github.io/

[16] (2013, May) Stanford network analysis platform. [Online]. Available: http://snap.stanford.edu/snap/index.html

[17] M. AbdelBaky, M. Parashar, H. Kim, K. E. Jordan, V. Sachdeva, J. Sexton, H. Jamjoom, Z.-Y. Shae, G. Pencheva, R. Tavakoli, and M. F. Wheeler, "Enabling high-performance computing as a service," *Computer*, vol. 45, pp. 72–80, 2012.

[18] S. Pallickara and G. Fox, "Naradabrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids," in *Proc. of the ACM/IFIP/USENIX International Conference on Middleware*, 2003, pp. 41–61.

[19] S. Pallickara, J. Ekanayake, and G. Fox, "Granules: A lightweight, streaming runtime for cloud computing with support, for map-reduce," in *CLUSTER*. IEEE, 2009, pp. 1–10.

[20] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[21] D. Gelernter, "Generative communication in linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, Jan. 1985.

[22] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: an interaction and coordination framework for coupled simulation workflows," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 25–36.

[23] A. K. Atkinson, "Tupleware: A distributed tuple space for cluster computing," in *Proc. of the 9th IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2008, pp. 121–126.

[24] F. B. Engelhardtsen, T. Gagnes, F. Boger, and E. T. Gagnes, "Using javaspaces to create adaptive distributed systems," 2002.

[25] J. Batheja and M. Parashar, "Adaptive cluster computing using javaspaces," in *Proc. of the 3rd IEEE International Conference on Cluster Computing*, 2001, pp. 323–.

[26] A. N. Bessani, "BTS: A Byzantine Fault-Tolerant Tuple Space," in *In Proceedings of the 21st ACM Symposium on Applied Computing - SAC 2006*, 2006, pp. 429–433.

[27] D. Fiedler, K. Walcott, T. Richardson, G. M. Kapfhammer, A. Amer, and P. K. Chrysanthis, "Towards the measurement of tuple space performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 3, pp. 51–62, Dec. 2005.