PROGRAMMING AND MANAGING DISTRIBUTED SOFTWARE-DEFINED ENVIRONMENTS

By

MOUSTAFA H. ABDELBAKY

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Manish Parashar

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

May, 2017

ABSTRACT OF THE DISSERTATION

Programming and Managing Distributed Software-Defined Environments

By MOUSTAFA H. ABDELBAKY

Dissertation Director:

Manish Parashar

The amount of data generated by applications and digital sources is rising to unprecedented scales. To keep pace, applications and workflows tasked with transforming the data to insight are becoming increasingly dynamic and inherently data-driven. Furthermore, the computational services, i.e., compute, data, and communication, required to run this emerging class of applications are often just as dynamic and heterogeneous. As data sizes continue to grow, one must find new ways of harnessing these services to meet the needs of emerging data-driven workloads.

Building a computational environment capable of supporting these applications presents many complex challenges. For example, there are requirements and dynamic behaviors set forth by multiple components of the environment, i.e., users, service providers, applications, and computational services. Accordingly, an environment must be capable of (1) providing a way for these components to express their requirements at any time in the application lifecycle and (2) reacting in real-time to changes set forth by any of these components by adjusting the service composition. While cloud computing provides the flexibility and diversity of services required by such an environment, determining which services to compose to meet application needs and when to compose them is not supported by current service models and infrastructure.

To address these challenges, this dissertation presents a programming system to enable the creation of a distributed Software-Defined Environment (dSDE); the resulting environment can seamlessly and symbiotically combine compute, data sources, data storage, and network resources. Specifically, this work makes the following contributions: (1) it enables the on-demand aggregation of distributed services while facilitating the continuous deployment of applications on top of them; (2) it provides programming abstractions that allow users, resource providers, and applications to dynamically compose different services based on constraints or requirements; (3) it introduces a runtime framework that can autonomously adapt to changes from any of the components in the environment; and (4) it sets forth a quantification model for application performance and expected quality of service of the resulting distributed Software-Defined Environment, which allows users to reason about trade-offs and requirements with respect to throughput, latency, cost, deadline, etc.

The applicability of this work to real-world scientific applications is validated through a series of experiments where heterogeneous, geographically distributed services are composed based on user, resource provider, and application specifications. The results establish the potential impact of a system capable of real-time adaptability to changes in mixed resource environments, including multiple clouds, grids, clusters, supercomputers, and traditional data centers.

Acknowledgments

Good fortune comes in many forms; mine came in the form of amazing people who have always been there for me. They helped me through my struggles when times were tough and shared my joy in good times. While gratitude is certainly due to them, it is simply not enough. To the following group of people, I owe you more than just an acknowledgment. I am truly grateful for your presence in my life, and I am forever in your debt.

First, I would like to thank my advisor Dr. Manish Parashar for his unconditional support, for being my first advocate, and for giving me the freedom to pursue my ideas yet the guidance to help me not get too lost. Most importantly, thank you for your kindness, patience, and for always giving me a chance. You have been an important role model to me and an excellent example of what an advisor should be.

I would like to thank Dr. Kirk E. Jordan, Dr. Ivan Marsic, and Dr. Deborah Silver for serving on my committee and for their advice and feedback along the way. In particular, I would like to thank Dr. Jordan for his continuous support and for providing me the opportunity to work with his research team at IBM T.J. Watson Research Center. I would like to thank Dr. Marsic for his advice and ideas that helped me strengthen my dissertation. I would also like to thank Dr. Silver for her encouragement and support throughout both my undergraduate and graduate studies at Rutgers.

I would like to thank the following Rutgers faculty (in alphabetical order): Dr. Martin Farach-Colton, Dr. Zoran Gajic, Dr. Marco Gruteser, Dr. Richard Mammone, Dr. Dario Pompili, Dr. Ivan Rodero, and Dr. Wade Trappe, for their valuable time, willingness to help, and for making my experience at Rutgers a pleasant and stimulating one that I will always remember.

I would also like to thank my collaborators whom I have had the pleasure to work with (in alphabetical order): Dr. Ilkay Altintas, Aditya Devarakonda, Dr. Constantinos Evangelinos, Dr. Hani Jamjoom, Dr. Hyunjoo Kim, Dr. Michael Johnston, Dr. Gergina Pencheva, Shweta Purawat, Vipin Sachdeva, Dr. James Sexton, Dr. Zon-Yin Shae, Dr. Malgorzata Steinder, Dr. Reza Tavakoli, Dr. Merve Unuvar, Dr. Jianwu Wang, and Dr. Mary F. Wheeler. I have learned a great deal from all of you.

I would like to thank my colleagues at the CometCloud group, the Applied Software Systems Laboratory, and the Rutgers Discovery Informatics Institute for all the memorable moments during my years at Rutgers.

Finally, I would like to express my deepest gratitude and appreciation to my family for all that they have done for me. Particularly, I would like to thank my parents Hany and Mervat Shaheen, my siblings Abdelhamid and Shymaa AbdelBaky, my sister-in-law Salma Elmallah, my in-laws Albert and Barbara Romanus, and my best friend Mark Wisniowski, for their unconditional love and support. I am where I am today because of you. Last but not least, I would like to thank my wife *(soon to be Dr.)* Melissa for all her help, for always believing in me, and for always being my number one fan.

Dedication

To – *Melissa*. Thank you for always being there for me and for being my better half.

I love you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Increasing computational capabilities coupled with an exponential growth of digital data sources has given rise to a data deluge. Transforming this massive quantity of information from data to insight requires innovations in the computational lifecycle. For example, emerging applications must be able to glean insights from data that is varying in size, distribution, and frequency; i.e., computation is inherently dynamic and driven by data availability. Supporting these data-driven applications requires rethinking the way we store, process, and analyze data as it is generated.

In this dissertation, we propose the creation of a novel and flexible computing ecosystem, called a *distributed Software-Defined Environment (dSDE)*, specifically designed to satisfy the diverse requirements set forth by emerging data-driven applications, end-users, and resource/service providers. The proposed ecosystem opportunistically combines pervasive digital data sources with the distributed compute, storage, and communication services they require over time. As a result, it is capable of automatically evolving over time in response to dynamic application, data, or infrastructure behaviors. Further, the dSDE is exposed programmatically to empower users, resource providers, and applications with control over the ecosystem.

Thesis Statement: *An ecosystem that combines distributed (1) compute, (2) storage, (3) network, and (4) data sources in a flexible, autonomous, and programmable manner can efficiently support emerging data-driven applications and workflows.*

The remainder of this chapter motivates the need for a dSDE using scenarios from diverse areas such as science and engineering, Internet of Things, and business. Next, we describe the challenges that need to be addressed to realize a dSDE, followed by a summary of our approach and results.

1.1 Emerging Dynamic and Data-Driven Applications and Workflows

1.1.1 Computational and Data-Enabled Science and Engineering (CDS&E)

CDS&E application workflows integrate data sources (e.g., monitoring, observations, and experiments) with computational tools (e.g., modeling, analytics, and visualization) to understand natural phenomena, manage complex processes, and accelerate scientific discovery. To prepare for a smarter, more 'connected' world, scientists and engineers are challenging the current methods of simulation by developing new classes of application workflows capable of dealing with real-time data transmissions from a variety of different sources across the globe. Integrating real-time data can enable new opportunities from managing extreme events to optimizing everyday processes and improving the quality of life. For example, an instrumented oil field can (theoretically) achieve robust control and efficient management of diverse subsurface and near subsurface geo-systems by completing the feedback loop between measured data and a set of computational models, thereby providing efficient, cost-effective, and environmentally safe production of oil reservoirs [123]. However, to support emerging CDS&E applications and workflows, one must address the following challenges:

1. **Large scale and dynamic application behavior.** Emerging CDS&E application workflows typically require large capacity and often heterogeneous capabilities that exceed what an average user can expect from a single computational center. Furthermore, these data-driven applications exhibit changing requirements at runtime (e.g., computational demand, data interest, and quality of service levels). Therefore, they require adaptive approaches that can automatically adjust the underlying infrastructure. Finally, these applications require novel methods of combining distributed data sources (e.g., data archives, instruments, experiments, and embedded sensors and actuators) with computational services (e.g., compute, network, storage), for example, when moving data is no longer an option.

   For instance, the Korean Superconducting Tokamak Advance Research (KSTAR) [98] project can generate 3 TB of data during a 100-second experiment. In this multi-location project, there are no local computational services to analyze the generated

data. Moreover, it is critical to promptly analyze certain data (e.g., to detect the creation of anomalies that could damage the experiment), whereas the bulk of data needs to be transferred somewhere else for batch processing using complex models. Therefore, it is essential to dynamically compose services close to the data source to detect anomalies promptly, while coordinating remote services that can store and analyze the bulk of the data to obtain scientific insights.

2. **Resource management.** The complexity and dynamic requirements of emerging CDS&E application workflows, coupled with the abundance and wide variety of available services, makes it very challenging for users to control their execution environments. For example, it is difficult for scientists to programmatically express which services are appropriate for a particular situation or to define the expected behavior of the infrastructure in response to application behavior. Additionally, programmatically exposing the underlying distributed infrastructure can enable new application formulations – those that are primarily driven by the application runtime behavior.

   For instance, molecular dynamics (MD) simulation is an essential part of drug discovery that provides key insights into drug design. A dSDE that combines commodity hardware from multiple sites can effectively support MD simulations (e.g., by accelerating and modeling conformational movement in proteins that are pharmaceutically relevant). Conversely, in the context of protein folding, a dSDE can be programmed to dynamically kill replicas if the protein structures being generated do not progress towards the known structure or show predicted secondary structure features, thus ensuring that CPU cycles are not wasted. Similarly, replicas that show promising results can be automatically moved to supercomputers and run at larger scale, which can speed-up the application execution.

1.1.2 Internet of Things

Emerging next-generation computing paradigms, such as the Internet of Things (IoT), have the potential to fundamentally transform our ability to model, manage, control, adapt, and optimize virtually any sub-system of interest. Examples include the wide variety of sensor-network–based applications, in which sensors interface with real-world artifacts and must

respond to unpredictable physical phenomena. These applications will revolutionize the way we interact with the world around us. For example, these applications can help the concept of "smart" cities become a reality [168]. In such environments, network-connected pervasive devices, ranging from sensors and actuators to special instruments and mobile terminals, interact in different ways to connect the physical world to the virtual world and enable information gathering, processing, and real-time monitoring of a variety of environment variables. Similarly, these applications introduce their own set of challenges.

1. **Unpredictable data patterns.** The amount of data likely to be generated by sensors and the processing requirements of such data cannot be determined in advance as they usually depend on non-deterministic factors. A dSDE can support IoT applications by dynamically adapting the composition of services over the application lifecycle. Moreover, since data generation and consumption patterns are becoming decentralized, it will be increasingly challenging to support latency-sensitive analysis and delivery of data using core-centric approaches. These applications will once again require that we seamlessly and opportunistically combine digital data sources and computational power. In this case, a dSDE can dynamically aggregate services at the edge (e.g., cloudlets [139], micro data centers, gateways) that have limited capabilities and dynamic availabilities, with services along the data path (e.g., SDNs), and services at the core (e.g., data centers, public, or private clouds) in order to process data *in situ* and *in transit*.

   For example, this is relevant to the management of natural disasters, such as monitoring and predicting wildfires [14]. In this case, large amounts of data need to be analyzed at near real-time to predict how a fire would propagate. This data needs to be retrieved from specific locations, and the sources can include field sensors, weather forecasting, satellite images, and social media feeds from an actual evolving crisis. Combining this data with a realistic simulation framework can provide decision support to manage the crisis [138]. However, this data must be analyzed close to the source to obtain timely insights and direct first responders to critical areas. Hence, being able to dynamically compose services (e.g., near these data sources) and automatically (e.g., when an event is detected) is key.

1.1.3 Business

Some enterprise applications are currently being deployed using hybrid infrastructure that combines in-house data centers with public clouds to accommodate for cloud-bursting and provide resilience. However, these solutions mostly rely on a single cloud provider which can lead to "provider lock-in." This prevents users from taking advantage of a competitive pricing market and can limit application deployments in cases of cloud or zone outages [1,2,3]. Once again, enabling the deployment of applications on top of a dSDE that combines services from multiple clouds and data centers can alleviate some of these problems. For example, a dSDE can enable the dynamic selection of services from different providers at runtime (e.g., when an outage is detected) based on various criteria (e.g., cost, performance, etc.), all of which is executed automatically (i.e., without manual user intervention). Moreover, the programmability of a dSDE can enable new scenarios, such as ensuring privacy (i.e., by keeping private data in-house and other services outside) or allowing for *in-situ* data analyses (i.e., by minimizing data transfers and keeping services near data sources). However, supporting business applications in a hybrid/multi-cloud environment can be challenging as well.

1. **Workload scheduling and migration.** Proper placement of service-oriented workloads in a distributed environment is required. For example, one must consider the affinity for a cluster of related services (i.e., should a cluster that provides a single service be placed close together or far apart?). A closely placed cluster provides better performance, whereas a distributed cluster provides better resilience. Similarly, automatic migration of data and services can be tricky, especially when considering workloads with short life cycle (e.g., software container based workloads), which may require fast placement.

---

[1] Microsoft Azure Service Outage in Europe and India https://blogs.msdn.microsoft.com/vsoservice/?p=12295

[2] Amazon S3 Service Outage in the US East Region https://aws.amazon.com/message/41926/

[3] Amazon AWS Service Event in the US East Region https://aws.amazon.com/message/67457/

1.2 Challenges in Achieving a Dynamic Computational Ecosystem

In the previous section, we demonstrated the diverse requirements of emerging data-driven applications and highlighted the shortcomings of traditional infrastructure offerings in supporting them. We then discussed how a dynamic and distributed software-defined computational ecosystem, capable of combining data and compute power on-the-fly, could be used to not only address these shortcomings but also to support novel execution modes and accelerate the application time-to-discovery. In this section, we define the challenges associated with implementing a real-world dSDE.

1. **Service composition.** While the service model (enabled by cloud computing [18]) provides the necessary scale and flexibility required to realize a dSDE, it is not clear how or when to compose different services to meet (or anticipate) the needs of dynamic data-driven applications and workflows.

2. **Interoperability.** The lack of interoperability among cloud providers as well as among traditional data centers (i.e., different providers may use different interfaces and protocols) makes the process of composing various services a challenging issue.

3. **Real-time adaptation.** Dynamic data-driven applications must also contend with an equally dynamic infrastructure – not just in terms of elasticity but also due to the temporal variability of services' properties (e.g., dynamic pricing based on demand, fluctuating performance due to utilization, or time-dependent service availability). Therefore, a dSDE must adapt in real-time to changes in infrastructure or application behaviors.

4. **Satisfying constraints.** A dSDE must also satisfy user and resource provider requirements and objectives, which can change during the application runtime. Similarly, a dSDE must react to changes in objectives or requirements. However, one must also consider how often the ecosystem should adapt to meet these requirements. For example, real-time adaptation might lead to instability (e.g., resulting in a volatile system), whereas periodic adaptation might violate some of the defined requirements.

5. **Unified control.** Providing a unified set of requirements for controlling a dSDE that reflects interests of various actors in the system (i.e., users, applications, or resource providers) is impractical. For example, users might want to manage a dSDE in terms of cost (e.g., budget) or time (e.g., deadline), whereas resource providers might want to govern it based on maximizing utilization. Additionally, applications might want to control a dSDE in terms of types or sizes of the needed computational capacity.

These characteristics make provisioning the appropriate blend of services challenging and requires rethinking how infrastructure is to be delivered to end-users. Current approaches to managing dynamic applications and workflows often target one aspect of the complex environment, namely either scheduling static applications while (1) optimizing user/resource providers Quality of Service (QoS) objectives [167] or (2) deploying dynamic applications by exploiting elasticity of cloud services [169]. However, these approaches do not account for the dynamic behavior of the underlying services. Moreover, they inherently assume that the other elements of the ecosystem are fixed. For example, programming a time-sensitive application capable of reacting to dynamic events (e.g., feature tracking) is of little consequence if the services that it is running on become unavailable or are not capable of scaling up or out at that point in time. Therefore, an effective solution must account for the requirements and dynamic behaviors of the individual elements of the system (i.e., users, service providers, applications workflows, and computational services), as well as their impact on the overall environment (i.e., a solution that may benefit one may not necessarily be the best option for the system as a whole).

1.3 Distributed Software-Defined Environments

The goal of this work is to provide an approach to enable the autonomic composition and continuous adaptation of traditional resources and cloud-based services to support emerging dynamic and data-driven applications and workflows (see Figure 1.1). Our approach leverages concepts from Software-Defined Environments (SDEs) [30, 99] and extends them to enable the creation of a *distributed Software-Defined Environment (dSDE)*. A dSDE creates *distributed programmable infrastructure* that is agile, self-adaptive, and customizable. This infrastructure is then exposed to users, resource providers, and applications using high-level directives, which allows them to programmatically define the desired state of their execution environments (i.e., which services can be part of said execution environment).

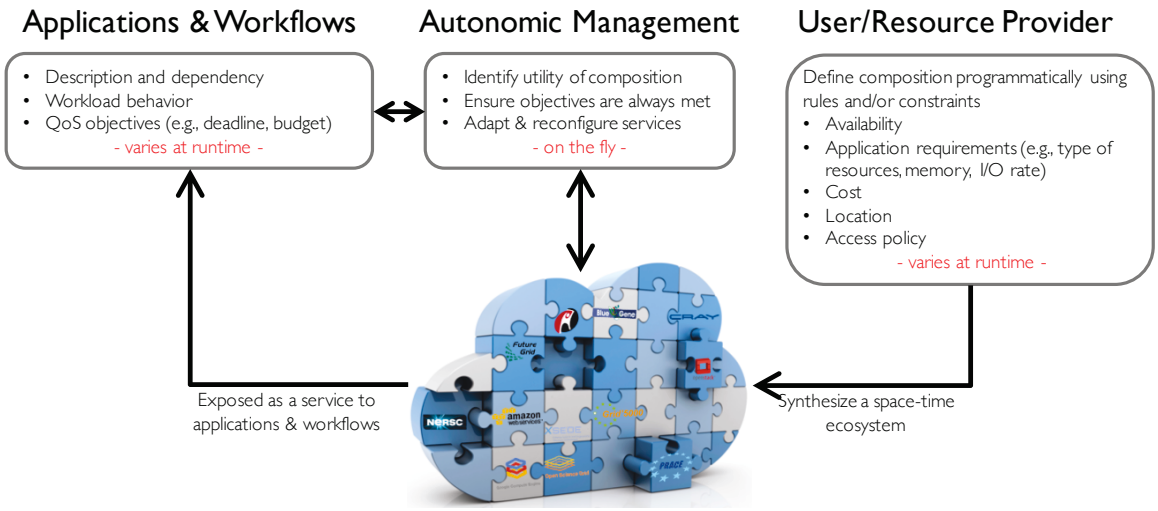


Figure 1.1: An overview of the distributed Software-Defined Environment (dSDE).

1.3.1 Research Approach

This is achieved by the creation of three layers on top of distributed services: (1) a control layer to manage the subset of services composing the execution environment at any moment in an automated fashion; (2) an abstraction layer to programmatically specify criteria, such as performance, capacity, and QoS, without detailed knowledge of the underlying infrastructure; and (3) an orchestration layer that utilizes the information provided by the abstraction layer to enact the necessary operational requests to drive the ecosystem towards the desired state. The abstraction layer is exposed using two different declarative languages

and Application Programming Interfaces (APIs): (1) a rule-engine–based approach that allows fine-grained control over services and (2) a constraint-programming–based approach that provides global control over services.

Once we have defined the desired state of the execution environment and the application requirements, the system can allocate the workload using any state-of-the-art scheduling techniques. We also rely on existing state-of-the-art autonomic mechanisms [112] to ensure that the workload scheduling adapts to changes in the application behavior or deviations from the execution plan. The result is a nimble and programmable environment that automatically evolves over the application lifecycle by dynamically adapting in near real-time to changes in (1) infrastructure behavior, properties, or availabilities; (2) user, application, or resource provider requirements regulating service usage; (3) application behavior or workload; and (4) QoS optimization objectives imposed on the system.

1.3.2 Dissertation Focus

While finding an optimum solution to this problem is desirable, it can take a significant amount of time. Hence, in the case of highly dynamic applications and services, it is more relevant to find the fastest solution that still meets the required criteria, rather than the optimal solution. Thus, the focus of this research is on the *programmable control* of the state of the execution environment and the *continuous orchestration/composition* of services according to dynamic properties and requirements. This approach goes beyond static matchmaking [102] techniques (between users and resource providers) by enabling dynamic directives, which are user– or application–driven and can be expressed as a function of runtime variables (e.g., progress in the execution of an application). In the experimental evaluation, we will show that the ability to react and adapt to changes in the underlying infrastructure while the application is running can reduce monetary cost and/or the time-to-solution for workloads. We use a representative bioinformatics scientific workflow as a driving use case for our evaluation, but this work applies equally to other scenarios such as the IoT scenario mentioned above. Finally, the implementation of the framework also uses only infrastructure services. However, we will show that our approach also applies to other services such as network or storage.

1.3.3 Generalization

We also generalize our approach by modeling the application performance and the expected QoS of a dSDE that includes data, compute, storage, and network services. This allows users to reason about trade-offs and requirements with respect to throughput, cost, deadline, etc. In particular, we quantify the expected Service-Level Agreement (SLA) of the entire ecosystem based on the combined SLAs from different services. For example, the model can estimate the minimum/maximum application throughput or calculate an estimate for the budget or deadline to run a certain workflow given the current composition of services. The model can also evaluate tradeoffs between different scheduling techniques to assist users with scheduling decision. For instance, minimizing cost might reduce throughput, whereas minimizing completion time might increase data transfers or increase cost. Finally, the model can help identify bottlenecks in the current composition to assist users' decision, e.g., help decide whether they need more services or network bandwidth to increase throughput.

1.4 Contributions

A dSDE can redefine how scientists and end-users leverage dynamic services to build data-driven applications and workflows by allowing them to take advantage of the collective capabilities of an aggregated set of distributed and heterogeneous services while tailoring the execution based on the distinct properties and availabilities of these services. This dissertation makes the following contributions:

1. An abstraction layer that enables the programmable and on-demand composition of distributed computational services.

2. Rule-engine–based and constraint-programming–based declarative languages to allow applications, users, and resource providers to express the desired state of the execution environment programmatically.

3. A runtime framework that dynamically synthesizes services in response to rules, constraints, dynamic application behavior, and/or fluctuating state of services.

4. A general mathematical model that considers data, compute, and network to estimate

the expected SLA for a dSDE, which can assist users with resource management decisions.

5. An evaluation of tradeoffs between different scheduling techniques to provide guidance on how and how much to change a dSDE to meet objectives.

1.5 Outline

The rest of the dissertation is organized as follows. Chapter 2 provides a brief background on software-defined environments and the runtime framework that we used for implementing the dSDE. Chapter 2 also provides a literature review of the related work. Chapter 3 presents the architecture and implementation of our programming system. Chapter 4 provides experimental and empirical evaluations of the resulting dSDE using representative scientific workloads. Chapter 5 presents a generalized mathematical model for the distributed software-defined environment. The dissertation concludes in Chapter 6 by outlining future research.

Chapter 2

Background and Related Work

2.1 Background

This dissertation utilizes concepts from software-defined environments and builds on top of an autonomic framework named CometCloud. In this section, we present a brief background on both topics. This chapter contains portions adapted from

2.1.1 Software-Defined Environments

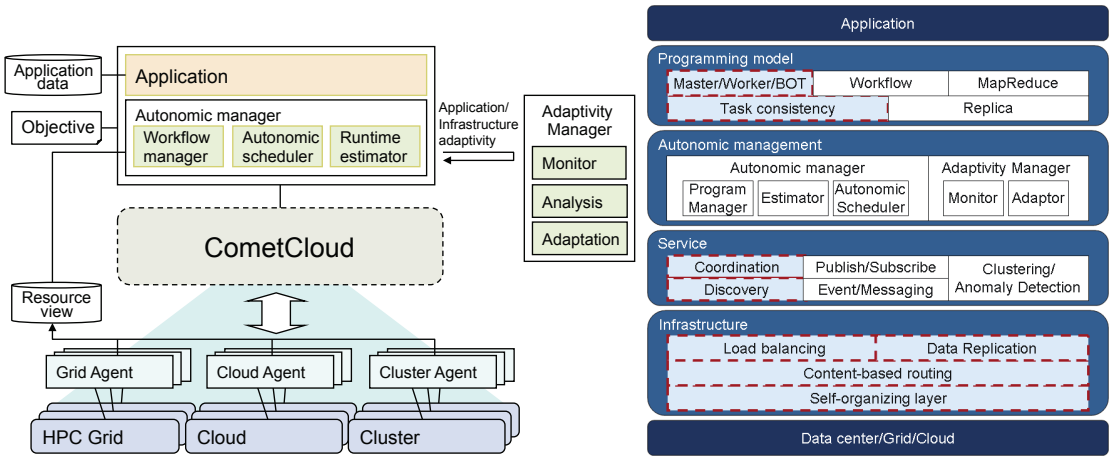
A Software-Defined Environment (SDE) [30, 66, 99], also referred to as a Software-Defined Infrastructure [83, 101], aims to:

1. Expose compute, storage, and network resources of an underlying computing infrastructure in a uniform manner.
2. Dynamically assign workloads to resources based on application characteristics, best-available resources, and service level agreements.
3. Continuously deliver dynamic optimization and reconfiguration to address workload demands and infrastructure issues.

These requirements can be achieved by exposing the computing infrastructure programmatically, which allows separation of the individual resource control mechanisms from the overall orchestration of the environment. The resulting environment is fully customizable and can automatically provision resources as it sees fit while adhering to high-level policies and QoS objectives; it also provides the software abstractions and tools to deploy dynamic workloads on heterogeneous infrastructure within a single data center. As a result, software-defined environments can provide cost savings and better performance across data centers [19], making them an attractive solution for enterprise IT [62].

2.1.2 CometCloud

CometCloud [51, 93] is an autonomic framework for enabling real-world applications on federated cyberinfrastructure. CometCloud is based on the Comet [100] decentralized coordination substrate and supports highly heterogeneous and dynamic integration of public and private data centers, clusters, supercomputers, grids, and clouds. The coordination substrate is also used to support a decentralized and scalable task space that coordinates the execution of tasks onto sets of dynamically provisioned workers on available resources based on QoS objectives. CometCloud also provides fault-tolerant mechanisms to guarantee workflow completion despite resource or job failures. Conceptually, CometCloud is composed of a programming model layer, an autonomic management layer, a service layer, and an infrastructure layer. The architecture of CometCloud is presented in Figure 2.1.



(a) Architecture of the CometCloud framework. (b) Layers of the CometCloud framework.

Figure 2.1: The architecture of the CometCloud framework and its layers. ©2011 Wiley, ©2011 Hindawi Publishing Corporation (reprinted with permission) Kim et al. [92, 93].

- The *Infrastructure Layer* manages the dynamic federation of resources and provides essential services for resource discovery and coordination. This layer uses the Chord self-organizing overlay [142] to create a scalable content-based coordination space for wide-area and the Squid information discovery method [140] to create a content-based routing substrate. The routing substrate supports flexible content-based routing and complex querying using partial keywords, wildcards, or ranges. It also guarantees that all peer nodes with data elements that match a query/message will be located [91].

- The *Service Layer* provides a range of services to support autonomics at the programming and application level [125]. This layer supports a Linda-like [35] tuple space coordination model and provides a virtual shared-space abstraction as well as associative access primitives. This layer also supports dynamically constructed transient spaces to allow applications to explicitly exploit context locality to improve system performance. Finally, this layer provides asynchronous (publish/subscribe) messaging and event services.

- The *Autonomic Management Layer* enables users and/or applications to define objectives and policies that drive resource provisioning and execution of application workflows while satisfying user constraints (e.g., budget, deadline). The autonomic mechanisms in place not only provision the right resources when needed but also monitor the progress of the execution and adapt the execution to prevent violations of established agreements.

- The *Programming Layer* provides the basic functionality for application development and management. It supports the Bag of Tasks (BoT) workload model, i.e., Master-Worker. In this model, masters generate tasks and workers consume them. Masters and workers can communicate via the virtual shared space or use a direct connection (i.e., Peer-to-Peer). Scheduling and monitoring of tasks are supported by the application framework. The *Task Consistency Service* handles lost/failed tasks [125].

2.1.2.1 Federation Management in CometCloud

A CometCloud federation is created dynamically and collaboratively; resources or sites can join or leave at any point, identify themselves (using security mechanisms such as X.509 certificates, public/private key authentication), negotiate the federation terms, discover available resources, and advertise their own resources and capabilities [51]. The general overview of the CometCloud federation coordination model is presented in Figure 2.2.

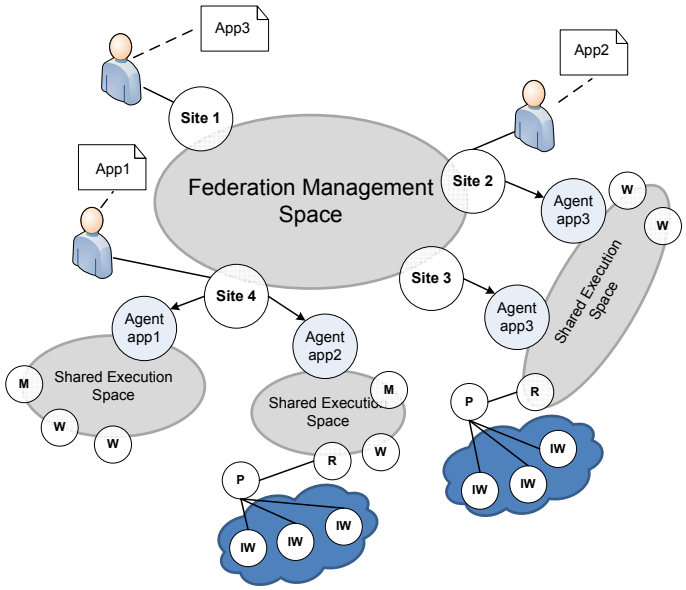


Figure 2.2: The architecture of the CometCloud federation coordination model. ©2015 IEEE (reprinted with permission) Diaz-Montes et al. [51]. Here, (M) denotes a master, (W) is a secure worker, (IW) is an isolated worker, (P) is a proxy, and (R) is a request handler. Arrows represent the deployment of a computational site, while lines show communication.

The Comet coordination space [100], CometSpace, is the key mechanism used to coordinate different aspects of federation and application execution in CometCloud. The CometSpace is a scalable, decentralized, and shared coordination space built on top of a distributed hash table that all resources in the federation can access associatively. It provides a tuple-space–like abstraction for coordination and messaging, enabling coordination in the federation model.

Specifically, the architecture includes two types of spaces (see Figure 2.2). First, the *Federation Management Space* is responsible for creating the actual federation and handling the orchestration of different resources; it does so by exchanging operational messages used to discover resources, announce changes at a site, route users' requests to appropriate sites and initiate negotiations to create ad-hoc execution spaces [4]. The second type of space is the *Shared Execution Space*. Multiple shared execution spaces are created on-demand to satisfy the computational needs of applications. For example, spaces may be created with the objective of (i) provisioning resources in a single resource site; (ii) bursting to other resource sites, such as public clouds or external HPC systems; or (iii) creating private sub-federations across several sites.

The creation and control of each shared execution space are managed by a *federation agent* (or *agent* for short). The agent also coordinates all resources that execute a particular set of tasks on that site (see Figure 2.2). Agents can act as a master of the execution or delegate this duty to a dedicated master (M), e.g., when more complex workflows are executed [4]. Additionally, agents deploy *workers* to perform the execution of tasks. There are two types of workers: Secure Workers (W) and Isolated Workers (IW). Secure Workers are on a trusted network; they are considered to be a part of the shared execution space and can pull tasks and share data accordingly. Isolated Workers, on the other hand, are on a non-trusted network and cannot directly interact with the shared space. Instead, isolated workers depend on a proxy (P) and a request handler (R) to obtain tasks from the space [4]. The distinction between a Secure and Isolated Worker is important because it allows the user to control data access permissions, thereby allowing optimization along the data storage and locality paths. Moreover, this mechanism can be used to define security policies and decide who can access which data.

2.1.2.2 Application Management in CometCloud

CometCloud can be used to manage the execution of a variety of applications, independent of programming language. Integrating an application with the CometCloud framework requires users to develop two simple components – a task generator and a worker. The task generator defines the properties of all application tasks via a simple API. A set of tasks comprises a single stage, while a set of stages comprises the application workflow. Using the CometCloud task generator abstractions, users have the ability to define dynamic applications, where the tasks for each stage are created at runtime depending on previously obtained results. Results of all stages are accessible through the API. This provides tremendous flexibility as an application can evolve in different ways depending on the observed data [112].

On the other hand, the worker's sole responsibility is to execute tasks, either directly or through third-party software. For the latter, the resulting worker becomes a proxy that acts as a wrapper for the target software. The third-party code and workers can be installed directly on the resources (e.g., a local data center cluster), encapsulated using VMs (e.g., in the case of a cloud resource), or containerized using software container services such as Docker. The diversity of options simplifies the migration from traditional environments to the CometCloud federation [4]. The worker component and any other third-party code are made available at each site of the federation and are exposed using a federation agent, as defined above. Workers utilize a pull-based model to pull tasks from the execution space, as opposed to a push-based model because it better supports heterogeneity in resources and tasks. Using this model, the Master inserts tasks into the CometCloud space, and the Workers pull the task by querying for specific task properties, such as keywords, wildcards, or a combination of both.

2.2 Related Work

The work in this dissertation incorporates concepts from several different areas of research, namely resource management, workflow scheduling and execution, and performance and QoS modeling in distributed environments. In this section, we aim to provide an overview of the state-of-the-art research in these areas.

2.2.1 Resource Management in Distributed Environments

2.2.1.1 Federated Computing

Federated computing has been explored over the course of the past decade and has been shown to be an effective model for harnessing the capabilities and capacities of geographically-distributed resources to solve large-scale science and engineering problems [7, 11, 22, 41, 42, 53, 65, 68, 122, 131].

Volunteer Computing and Grid Computing. Two drastically different strategies for federating resources have been proposed: volunteer computing, which harvests donated idle cycles from numerous distributed workstations (e.g., BOINC [16], SETI@home [17]), and HPC grid computing [64, 122], which offers a monolithic access to powerful HPC resources shared by a virtual organization [114] (e.g., EGEE [97] or Open Science Grid [128]). However, both of these concepts have their limitations: volunteer computing is well suited for processing lightweight independent tasks (e.g., HTC applications [105]) in an opportunistic way, but fails when presented with traditional HPC computations. Grid computing, on the other hand, was limited to static resource allocations, non-interactive workloads, and lacks the flexibility of aggregating resources on demand (without complex reconfiguration).

Grid Federation. To increase the scale of grids, researchers looked into grid federation that combines multiple grid systems or Virtual Organizations (VO) to enable e-Science and e-Business. To enable this, researchers have addressed issues related to connecting multiple grid systems [131]. Such connections could be based on economic incentives [49], peer-to-peer agreements among meta-schedulers [29], WS-Agreements and reservation based co-allocation [160], high level abstractions [86], or grid standards [57]. Other researchers have also focused on scheduling across multiple grid systems once they are connected [42, 56, 111].

Hybrid Federation and Cloud Bursting. One issue identified with HPC grids was their perceived complexity and lack of flexibility, as they present users with a pre-defined set of resources and do not allow federations to evolve in response to changing resources or application needs. More recently, hybrid federation approaches have been explored as a way of extending the cloud's as-a-service model to grid federations. A typical hybrid approach is based on extending local cluster/grid resources to cloud resources as needed [47,117,152], i.e., cloud bursting. Federated hybrid grid and cloud environments have also been proposed to create large-scale distributed infrastructure [28]. The goal of these techniques is to complement existing grid infrastructures with cloud resources. M. Parashar et. al [119] proposed different federation models for enabling scientific applications on hybrid infrastructure (e.g., HPC in the Cloud, HPC plus Cloud, and HPC as a Cloud).

Cloud Federation and Inter-Cloud. Further, to maximize the elasticity of the federation, researchers have considered a cloud-based approach named Inter-Cloud, which aims at federating multiple cloud infrastructure [31, 33, 45, 61, 71, 73, 84, 85, 132, 165]. For example, Kertesz et. al [87] aim to enhance the management of federated clouds by utilizing an integrated service monitoring approach. D. Villegas et. al [154] propose federating cloud infrastructures as a single integrated cloud environment using a layered service model. A. Celesti et. al use a customized cloud manager component that is placed inside the cloud architecture to provide a cross-federation model [36]. The reservoir model introduced by Rochwerger et. al [133] aims to enable providers of cloud infrastructure to dynamically partner with each other to create a seemingly infinite pool of IT resources while fully preserving their individual autonomy. Petri et. al [126, 127] propose federation models based on marketplace economics to incentivize collaboration.

Multi-Cloud and Cloud-of-Clouds. Inter-Clouds focus on finding ways for cloud providers to interconnect and share their infrastructure, which requires coordination and can lead to interoperability issues [74]. Alternatively, researchers looked into a Multi-Cloud or a "Cloud-of-Clouds" approach [12,25,46,52,61,73,124], where clients distribute workloads across multiple clouds. This approach avoids Inter-Cloud issues but requires clients to develop extensions for different clouds. For instance, researchers introduced ExoGENI [21,40], an extension of GENI [23], that provides orchestrated provisioning across sites.

2.2.1.2 Resource Description and Matching

Multiple researchers have focused on allowing both users and resource providers to describe resource requirements and match resources based on a given criteria [102,103,155]. However, these approaches were focused on one-time placement between jobs and resources. More recently, Kritikos et. al [95] proposed SRL, a scalability rule language for Multi-Cloud Environments, however, their work is focused on modeling application patterns to specify application behavior across multiple clouds.

2.2.1.3 Constraint Programming

In the context of constraint programming, many researchers have explored using CP for resource allocation. The majority of this research [37, 38, 55, 75, 147, 148, 162], focuses on finding optimum or near-optimal solutions to allocate resources while optimizing single/-multiple objective functions (e.g., cost, deadline, data transfer). The limitation of these approaches is scalability when increasing the number of constraints, objectives, or resources, which increases the search space significantly [67].

2.2.2 Workflow Scheduling in Distributed Environments

Efficiently executing workflows on federated infrastructure remains an active research topic. Workflow execution algorithms produce a mapping of tasks to resources on the cloud. This mapping could be calculated a priori and remain static during the workflow execution [9,10, 24, 32, 34, 80, 81, 104, 150, 167, 169]. It could adapt dynamically at runtime to the changing landscape of resources and tasks [27, 107, 108, 156]. Most research works define the optimization criteria for this mapping as a subset of execution time [108, 129, 151, 159, 163, 164], cost [108,129,151,159,164], resource utilization [172] and makespan [151]. Some algorithms also take scalability [163], scheduling success rate [151], and speed [172] into account. Others consider communication time and deadline constraints [108, 129, 164] as critical factors in scheduling. Fakhfakh et al. [59] and Bala et al. [20] provide detailed surveys of scheduling algorithms in a cloud environment.

Cloud service providers have APIs that enable users to scale resources to meet workload demands automatically. The effect of cloud's dynamic elasticity on performance criteria such

as data-transfer rates, make-span is complex to model for most workflows. This complexity has piqued the interest of the community, resulting in numerous heuristics and approximations. Deelman et al. [48] demonstrate that different schedules of the same application can lead to significantly different costs over the cloud. It establishes that right resource allocation can substantially reduce the overall cost while maintaining performance. Mao et al. [108] utilize auto scaling to dynamically adapt to cost-effective configurations to accommodate changing workload and resource problems while meeting deadlines. They abstract a resource as a Virtual Machine (VM) and characterize each machine using size and cost metrics. The optimization criteria comprise overall cost minimization and soft deadlines. Yazir et al. [166] present an approach for dynamic and autonomous resource allocation handling the constraints and limitations imposed by the resource allocation problem. Wang et al. [159] use a predictive model to generate performance estimate for each task and dynamically finds the best resource configuration. They utilize an iterative control process that uses data mining to map cloud resources while meeting performance and cost constraints continuously.

Varalakshmi et al. [151] perform clustering of sub-tasks and allocate formed clusters to different resources using a heuristic while taking into account the QoS metrics (cost, time and reliability). The approach utilizes resource indexing to find available resources. de Assuncao et al. [47] investigate the performance and cost implication of extending local infrastructure by elastically allocating additional resources from the cloud. Rahman et al. [129] present an Adaptive Heuristic for a hybrid cloud environment that considers workflow level optimization to minimize the cost of execution while meeting other QoS metrics such as budget, deadline and data placement. Bossche et al. [149] formulate the cloud outsourcing problem as a binary integer program and analyzes the cost of running a deadline-constrained application in a hybrid cloud environment. Zhong et al. [172] present an Improved Genetic Algorithm that maximizes resource utilization in the cloud by launching Virtual Machines (VMs) at cheaper sites. Wu et al. [163] approach the cloud-based workflow scheduling problem in a bottom-up manner. They take a hierarchical scheduling approach and find an optimal task to virtual machine mapping while maintaining the QoS requirements. Wu et al. [164] propose a particle swarm optimization based approach that

uses both data transfer and computation cost in consideration for scheduling workflows in the cloud. Chang et al. [39] present a Multiple Criteria Decision Analysis approach and formulate the task of finding the right type and size of resources required for computation as a resource allocation problem with multiplicity. Oprescu et al. [116] present a dynamic scheduler that allocates resources on multiple cloud providers with different cost models while maintaining a user-defined budget constraint.

Finally, some research efforts focus on creating cost models to make sure that the workload is executed according to the defined QoS parameters [8,60,77,171]. Other approaches emphasize resources utilization [26,109,118], although concerns about costs are still present. A specific case for the latter is prioritizing the use of local resources and controlling which tasks are outsourced to remote resources/services [125,149]. There are also research efforts exploring autonomic computing approaches based on concepts of self-managing adaptive resource provisioning, such as Agile [115], PACMan [136], Tiramola [146], and Autoflex [113].

2.2.3 Performance and QoS Modeling in Distributed Environments

Early Research. Modeling and characterizing parallel computing performance in heterogeneous environments have been explored over the past two decades. Early research [70, 137, 170] focused on defining a model for simple parallel applications and predicting their performance on dedicated and non-dedicated heterogeneous networks of workstations using metrics such as speedup, efficiency, scalability, timeliness, precision, and accuracy.

Web Service Modeling. With the emergence of web services [13], many researchers aimed to model the QoS of such services [43, 72, 82, 143]. The focus of this research was on estimating latency (response time to queries) and throughput (transaction processing capacity per time unit), under given application patterns and workload conditions. This research also introduced other QoS parameters such as availability (of web services), accessibility (the degree of which a web service can serve requests), cost, integrity (maintaining correctness), overall performance, reliability (a function of availability and quality of service), verity (an indicator of a service provider truthfulness or reputation), and security, which has led to the introduction of web service level agreements (WSLA) [106]. Other researchers aimed to model the performance of multi-component online services using application profiling [141].

Grid Computing Modeling. At the same time, researchers tried to model QoS in grid environments [94, 110, 145]. However, the majority of this work was focused on scheduling workflows based on QoS requirements [10, 32, 80, 104] or the monitoring, analysis, and management of QoS attributes [145]. Kim *et al.* [94] also introduced a single measure named (FISC) that aims to quantify the collective value of multiple QoS metrics.

Cloud Computing Modeling. More recently, researchers aimed to model the QoS and performance of applications in cloud computing environments. The majority of this work focuses on estimating the performance of different classes of applications in virtualized cloud environments [58, 76, 78, 79], providing CPU-based guarantees via resource-level QoS metrics [69], or modeling the performance of cloud computing services [88–90].

Multi-Cloud Modeling. In the context of Multi-Clouds, where a workload is distributed among independent clouds, Grozev and Buyya [74] modeled the performance of three-tier applications, while taking into consideration the heterogeneity of underlying services.

2.3 Summary

In this section, we detail how this dissertation extends the work described in Section 2.1 and position our work according to the landscape defined in Section 2.2.

2.3.1 Extension of Background Work

Software-Defined Environments. Similar to enterprise IT, SDEs can effectively support emerging data-driven applications and workflows by enabling a programmatic control over resources and services and facilitating dynamic resource management. This dissertation extends the concept of Software-Defined Environments across distributed advanced cyberinfrastructure to empower users with the ability of seamlessly composing services from different providers and geographic locations as well as adapting to changes in near real-time.

CometCloud. This dissertation extends CometCloud by providing a programmable interface for resource management (see Figure 2.3). This interface allows applications, users, and resource providers to declaratively specify resource availability as well as policies and constraints to regulate their use. These constraints collectively define the set of resources that are federated at any time and can be used to deploy the application.

2.3.2 Position in Current Landscape

Resource Management in Distributed Environments. In the context of federated computing, the approach presented in this dissertation leverages the elasticity and scale provided by the Multi-Cloud approach and extends it by applying software-defined concepts to enable a programmatically-defined dynamic and on-demand federation of clouds, grids, and other infrastructure services. Additionally, our model aims to leverage a service model, where computational resources are offered under specific SLAs. In this sense, we diverge from traditional federations that relied on administrative agreements to define how to interconnect distributed resources. Further, the majority of Multi-Cloud approaches (e.g., GENI/ExoGENI [21,40], ORCA [40]) take a resource-provider–centric approach to resource management, where resources from multiple providers are combined into slices. Once a slice is ready, an application can be deployed on top of it.

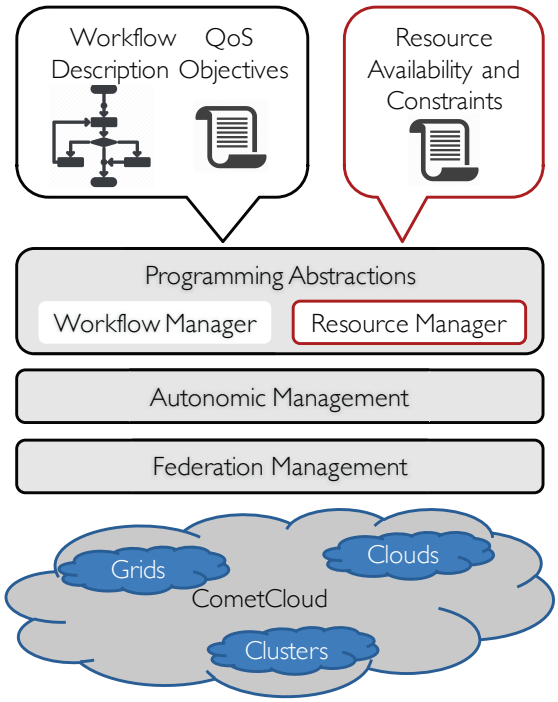


Figure 2.3: The extension of the CometCloud framework to provide a programmable interface for resource management. ©2015 IEEE (reprinted with permission) Diaz-Montes et al. [51]. The red boxes denote new components.

In our approach, we take a user/application-centric approach, where slices are programmatically created using high-level abstractions. Specifically, our approach provides declarative abstractions that allow users to control the slice structure over time by enabling complex constraints, such as *use resource X for a maximum of Y core/hours when workflow priority is low* or *use resource Z only if their price is below a given threshold.* Finally, in our approach, slices can adapt during application runtime to meet the dynamic behavior of applications or accommodate for changes in the infrastructure state.

In the context of resource description and matching, current approaches focus on one-time placement between jobs and resources, and therefore, do not consider the dynamic nature of applications or infrastructure. Our approach supports dynamic and continuous matching of resources to constraints in order to respond to changes in the infrastructure or application requirements/objectives over the application lifecycle. Moreover, our approach allows users to define additional objectives (e.g., maximize data locality, minimize cost), when selecting resources, which cannot be achieved using traditional matchmaking approaches.

Finally, in the context of constraint programming (CP), our approach does not use any optimization techniques (objective functions), but rather uses CP as a declarative language to describe the minimum requirements that resources must have to be part of the composition, i.e., the desired state of the system but not the actual allocation of resources. In doing so, a CP solver can rapidly evaluate a large number of constraints, reducing the search space very quickly and limiting the viable options for the resources to be selected by the scheduler to those that satisfy constraints imposed by users, applications, and resource providers. This allows the scheduler to speed up the process of selecting the proper resources and allocating the workload using given QoS objectives. Additionally, our approach may be used to generate a variety of 'views' of the same underlying federation for different applications, users, or projects.

Workflow Scheduling in Distributed Environments. The focus of this work is not on the optimum scheduling of workloads across clouds, but rather on the adaptability of the solution to dynamic properties and availabilities of the underlying services. In that sense, our approach is complementary to current state-of-the-art scheduling techniques as it offers a way of dynamically creating subsets of resources that are then exposed to any of the previously mentioned schedulers to allocate the application workload.

Performance and QoS Modeling in Distributed Environments. Early QoS models focused on distributed resources within a single location and did not consider network performance, data size, or transfer rates, whereas our model considers them. Our work utilizes some of the QoS metrics introduced in web service modeling but instead uses them to represent resources and infrastructure services. Our work also extends grid computing approaches by modeling the variability of infrastructure services over time, which was not considered before. Moreover, current cloud-based modeling is focused on a single cloud environment or site whereas our work considers multiple geographically distributed sites. The modeling research presented in a multi-cloud environment is the closest to our approach. However, our work differs in that we model batch applications as opposed to internet applications. Finally, the focus of our work is on the expected QoS guarantees from a collection of resources using traditional infrastructure (e.g., grids, supercomputers, clusters), multiple cloud services, and emerging services (e.g., cloudlets, fog and edge clouds).

2.4 Relevant Publications

This chapter contains portions adapted from the following published papers with permissions from the copyright holder.

1. H. Kim, M. AbdelBaky, and M. Parashar. CometPortal: A portal for online risk analytics using CometCloud. In 17th International Conference on Computing Theory and Applications (ICCTA2009), 2009.

2. J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. Software defined federated cyber-infrastructure for science and engineering. In Proceedings of the 2014 ACM international workshop on Software-defined ecosystems, pages 9-12. ACM, 2014.

3. J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. CometCloud: Enabling software-defined federations for end-to-end application workflows. IEEE Internet Computing, 19(1):69-73, 2015.

4. M. AbdelBaky, J. Diaz-Montes, M. Zou, and M. Parashar. A framework for realizing software-defined federations for scientific workflows. In Proceedings of the 2nd International Workshop on Software-Defined Ecosystems, pages 7-14. ACM, 2015.

5. M. AbdelBaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder. Docker containers across multiple clouds and data centers. In 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pages 368-371. IEEE, 2015.

6. J. Wang, M. AbdelBaky, J. Diaz-Montes, S. Purawat, M. Parashar, and I. Altintas. Kepler+ CometCloud: dynamic scientific workflow execution on federated cloud resources. Procedia Computer Science, 80:700-711, 2016.

Chapter 3

Distributed Software-Defined Environments

3.1 Introduction

In this chapter, we present our approach to providing a programmable and dynamic framework that can support data-driven applications by extending software-defined environment concepts to drive the process of dynamically composing infrastructure services from multiple providers. The resulting distributed software-defined environment (dSDE) autonomously evolves over the application life cycle while meeting objectives and constraints set by users, applications, and/or resource providers. In Section 3.1.1, we first discuss the requirements for a dSDE followed by a summary of our methodology in Section 3.1.2. We then present the architecture and implementation of the three layers necessary to realize a dSDE in Sections 3.2-3.4. In Section 3.5, we then demonstrate how the three layers work together to provide a dSDE using a rule-engine–based and a constraint-programming–based approaches. Finally, we provide a summary of the chapter in Section 3.6.

3.1.1 Requirements

The goal of this work is to provide users, applications, and resource providers with the ability to *programmatically* define the spatial and temporal structure of an environment of geographically-distributed infrastructure services that will be used to execute their workloads. The idea is to enable a nimble software-defined, data-driven service composition process that takes into consideration user needs, QoS objectives, application requirements, resource availability and properties, and resource provider constraints to create a customized environment that is best suited to the application at any given time. To achieve this goal, we have identified a set of requirements that need to be satisfied, as follows.

1. **Support for dynamic composition & programmability:** Existing federation models, that select resources once and begin application execution, are infeasible for dynamic application behaviors. Instead, a new federation model is required, i.e., one that is capable of changing the composition of services at runtime without affecting running applications. This implies that resources can join or leave the environment at any time. Additionally, appropriate APIs are required to allow for the programmatic control of different services in the environment.

2. **Continuous discovery mechanisms:** Emerging infrastructure services typically offer a wide range of capabilities, capacities, and pricing models. Additionally, these services tend to exhibit temporal variability (e.g., dynamic pricing based on demand, fluctuating performance due to utilization, or time-dependent service availability). As a result, effective discovery mechanisms are required to obtain current information about the resources available for usage as well as their characteristics, pricing models, utilization, etc.

3. **Ability to define multiple requirements:** In addition to exhibiting temporal variability, service offerings are typically governed by external factors, such as regulations, privacy concerns, budgets, etc. As a result, applications, users, and resource providers must have a means of specifying their resource requirements and constraints in a simple and flexible manner. The ability to specify multiple requirements allows different actors to simultaneously interact with the system. For example, users may specify requirements based on the type of resources, their costs, their location, the QoS or security/privacy guarantees they can provide. Resource providers can also specify how their resources should be used, for example, by specifying utilization or power consumption thresholds. Unlike traditional grid federation approaches, these requirements may change over the application lifecycle so a system must be capable of taking new requirements as input at any time during workflow and/or application execution.

4. **Support for multiple views:** The system must also support multiple and varied "views" (compositions) over the same or overlapping set of resources while simultaneously supporting different users and applications. Each individual view may be governed by its own set of requirements and can evolve independently.

5. **Dynamic expression of application requirements & QoS:** The requirements and QoS objectives of applications influence all areas of the system. Specifically, they drive resource provisioning, application mapping, scheduling, and execution. For example, an application may require resources with specific capacities or performance, or it may have QoS objectives such as maximizing throughput or minimizing data transfer. Further, QoS objectives may vary during runtime (e.g., over different stages of a workflow), thus the underlying system must be capable of maintaining specified objectives even if there are external changes to the environment (e.g., a resource is removed). As a result, the system must provide a way for applications to dynamically and programmatically express these needs at or during runtime.

6. **Autonomic Scheduling:** The system must provide adaptive mechanisms that allow for the autonomic provisioning of resources to adjust to changes in both the application behavior and the underlying infrastructure that cannot be determined in advance. For example, resources can be adjusted to accommodate a dynamic workload during runtime (e.g., intermediate results may lead to extra workload being generated or different types of analytics being used to process these results). Similarly, the system must also adapt and continue the execution of the application in the case of unforeseen events in the underlying infrastructure (e.g., failures) or change in infrastructure properties (e.g., cost, performance, availability, etc.).

3.1.2 Methodology

In order to support the aforementioned requirements, we enabled the programmatic control and composition of distributed services by creating programmable abstractions on top of distributed infrastructure. These abstractions include declarative mechanisms that allow applications, users, and service providers to control resource behavior over the application

lifecycle by defining high-level policies that translate into actionable events at runtime. This approach goes beyond static matchmaking techniques, such as Condor HTC [105], by enabling dynamic directives, which can be user- or application–driven and are expressed as a function of runtime variables (e.g., progress in the execution of an application). Similarly, we provide infrastructure-driven directives that are expressed as a function of runtime events such as failures, performance degradation, or discovery of resources. This was achieved by adhering to the following design steps.

1. We decoupled applications and workflows from the underlying resources by leveraging software containers (e.g., Docker containers [1]) to package applications with their dependencies and facilitate their execution in heterogeneous environments.

2. We abstracted geographically-distributed sites, containing multiple services, in a modular and uniform fashion.

3. We provided control mechanisms that enable the automatic, continuous, and programmable composition/aggregation of a pool of distributed sites and services.

4. We created orchestration mechanisms, which utilize the control mechanisms, to dynamically manage the environment without interrupting ongoing execution of applications.

Following these steps, we created the conceptual architecture for our distributed software-defined environment, as shown in Figure 3.1a. The architecture is composed of four layers: a Programming Layer, an Orchestration Plane, a Control Plane, and a Federated Infrastructure Layer. A summary of the methodology is also shown in Figure 3.1b.

- The *Programming Layer* enables the programmatic expression of resource requirements, such as performance, capacity, and QoS, without detailed knowledge of the underlying infrastructure. These requirements are then translated into actionable operations at the lower layers to compose resources on-demand and dynamically adapt

[1] Docker Containers: https://www.docker.com

the environment over time. We developed two different approaches for the programming layer: a rule-based engine and a constraint programming method. Both are discussed in more details in Section 3.2.

- The *Orchestration Plane* utilizes a two-step approach, as follows: (1) first, resource requirements are translated to a *virtual slice* – a set of available resources that evolves over time based on changes in resource properties and availabilities or the requirements regulating their usage; and (2) once the first step is complete, a slice is exposed to an autonomic scheduler that provisions resources and maps application tasks based on workload behavior and QoS objectives. The orchestration plane creates operational control requests, which are sent to the control plane to drive the environment towards the desired structure.

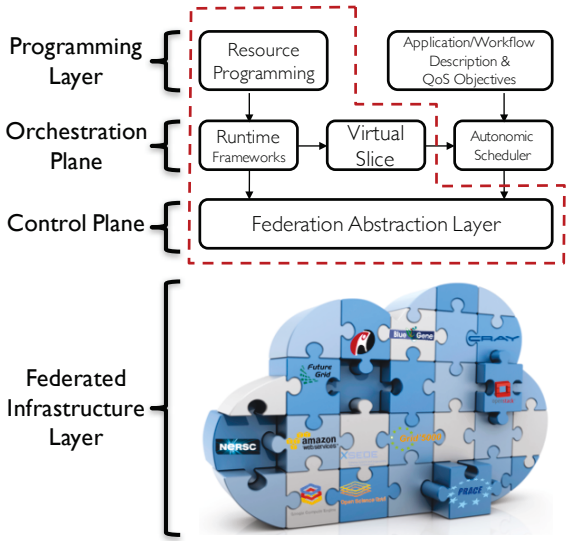
- The *Control Plane* enables the programmatic and on-demand allocation, control, and aggregation of distributed heterogeneous resources and services in an automated fashion. It supports the (1) addition/removal of sites or services and (2) monitoring the status of the federated infrastructure (e.g., available sites, the number of available resources, the number of resources running applications, etc.).

- The *Federated Infrastructure Layer* represents the federated infrastructure and services (i.e., the resulting distributed software-defined environment) available to applications at any time.

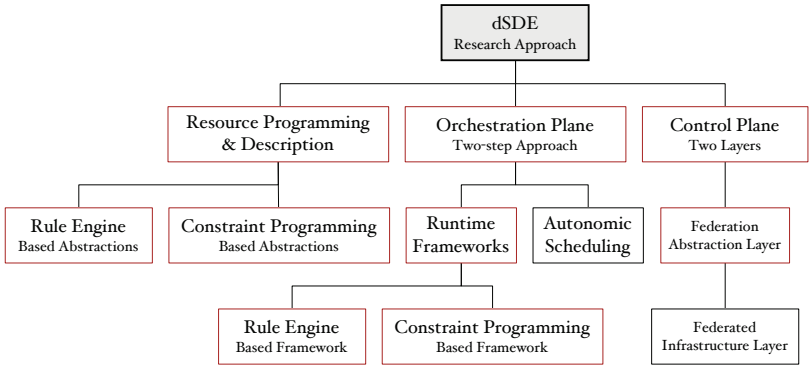
3.2 Resource Programming and Description Abstractions

In this section, we present the detailed architecture and implementation of the top layer of our dSDE. In particular, we present two different approaches for resource programming and description: a rule-engine–based approach and a constraint-programming–based approach. The difference between these approaches can be summarized as follows.

(A) *Rule Engine:* This approach allows fine-grained control over individual resources or sites by defining the availability of resources over time. The availability can be specified directly (e.g., Resource A is available from time *t1* to time *t2*) or by defining rules



(a) The conceptual architecture and main components of a dSDE. The dashed red box denotes this dissertation's contribution.



(b) The overall methodology to achieve a dSDE. The red boxes denote this dissertation's contribution.

Figure 3.1: An overview of the conceptual architecture and main components of a dSDE and our methodology to achieve it.

that are evaluated at runtime (e.g., Resource A is available when its cost is below a certain threshold or when the application reaches a certain stage).

(B) *Constraint Programming (CP):* This approach leverages a CP model and solver to allow applications, users, and/or resource providers to define global constraints that are applied to all available resources. For example, only use resources that have a minimum performance of X. The use of global constraints allows simpler control over resources and provides better techniques to resolve conflicts among competing constraints.

3.2.1 Rule-Engine–Based Abstractions

The rule-engine–based abstractions provide mechanisms for expressing the attributes of the dSDE using rules that define resource availability over time. Specifically, the policy layer combines a set of rules to define the overall spatial and temporal structure of the dSDE. Each rule defines the availability of a single resource/site over time either directly (i.e., by specifying the time the resource is available/unavailable) or indirectly (i.e., by specifying the conditions that need to be true for the resource to be available). Instead of using a resource specification language, the policy layer provides a separate set of Application Programming Interfaces (APIs) for each actor (i.e., applications, users, and resource providers), thereby allowing them to express resource policies in terms that are meaningful to them. In our design, we enable these policies to be specified dynamically and programmatically at any given time. This is achieved by using a RESTful [63] web service (see Figure 3.2), which allows the definition of new rules for existing policies and the creation of new policies at runtime (i.e., during the application execution).

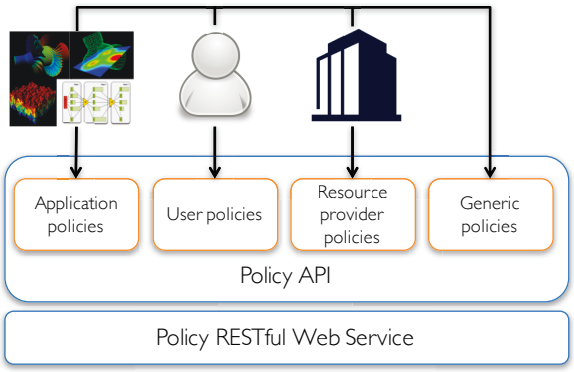


Figure 3.2: An overview of the policy layer for the rule-engine–based resource programming and description abstractions.

Table 3.1 shows some sample rules that we implemented as a function of time to enable the dynamic composition of resources. They are categorized as follows:

- **Generic Policies** support the direct declaration of the structure and behavior of the environment over time and can be used by all actors. For example, the web request `schedule_resource_date` will notify the orchestration plane to schedule a resource to be part of the dSDE during the time specified by the request.

- **User Policies** allow users to define dynamic resource requirements and objectives in a high-level and expressive manner. For example, user-provided policies can define a desired action to take when a resource cost threshold is met or an application/workflow reaches a specific point. For example, the web request `schedule_resource_spot` will notify the orchestration plane to schedule a resource to be part of the dSDE as long as the cost per hour for using that resource is less than a given threshold. These policies may also define more general or non-actionable user-initiated requirements, e.g., privacy-levels required for specific portions of sensitive data.

- **Application Policies** enable applications and workflows to provide requirements to the system about the workload. For example, if a specific stage of a workflow is computationally-intensive, the workflow may specify that a supercomputer should be used for that stage. These requirements may also be dynamic; for example, if a simulation spawns additional analysis tasks in response to a feature-tracking event, it may indicate that these new tasks require graphics processing units (GPUs).

- **Resource Provider Policies** enable resource providers to enact policies for the services and resources they govern. For example, a provider may control the utilization and load-balance of their data center by specifying the rule: use a resource as long as its utilization is below a certain threshold. As with the other policies, provider policies may be enacted at any time during the course of workload execution.

Table 3.1: Sample rules for the rule-engine–based Policy Layer and their description. ©2017 SAGE Journals (reprinted with permission) AbdelBaky et al [2].

API Call	Description
scheduleResourceDate	date/time based rule
rescheduleResourceDate	modify an existing date/time rule
scheduleResourcePeriod	periodic rule (e.g. daily, Mondays, always)
rescheduleResourcePeriod	adjust existing periodic rule
scheduleResourceSpot	resource price based rule
rescheduleResourceSpot	adjust current price threshold
cancelResourceSchedule	cancel a rule for a certain resource
getSchedule	return XML description of the current rules for all resources

3.2.2 Constraint-Programming–Based Abstractions

3.2.2.1 Overview

In this section, we present a different approach for resource programming and description that leverages constraint programming (CP). Constraint Programming [135] is an approach for modeling the relations between variables in the form of constraints and identifying feasible solutions out of a large set of candidate solutions based on these constraints. In our approach, constraints model the relationships of *resources* to (1) *resource properties* and/or (2) *resource availabilities*. The set of resources for which constraints can be defined is limited to those that the user provides valid access credentials for. The candidate solutions generated by the evaluation of these constraints are the subsets of resources that satisfy all of the given constraints. Using this subset of resources, the system creates a customized view (i.e., a virtual slice), which is exposed to the scheduler for use in task placement and mapping.

Constraints may be set forth by users, applications, and/or resource providers at any time before or during workload execution. Similarly, the initiator of a constraint may revoke it during runtime, if it is no longer applicable or desired. In addition to the possibility of varying constraints during workflow execution, resource properties, resource availabilities, and/or application properties can all vary over time as well. Thus, the evaluation of constraints to form resource sets that satisfy all constraints is not a static process but rather one that occurs continuously over the course of workflow execution. Consequently, the solutions of the CP solver drive the service composition process, i.e., seamlessly adapting to the joining and leaving of resources from the solution set. In contrast to the Rule Engine approach, the CP approach can accommodate more complex policies with multiple constraints per resource. This is because the CP solver can create viable views despite conflicts between competing or overlapping constraints.

We classify the dynamic inputs to the constraint programming solver to include:

1. Resource properties and availabilities (e.g., cost, utilization, or performance).

2. User constraints (e.g., security or maximum cost per resource).

3. Resource provider constraints (e.g., maximum utilization or maximum power consumption thresholds).

4. Application workflows constraints (e.g., resources types or minimum performance per resource).

We chose CP as opposed to linear [161] or integer [173] programming (LP, IP) since CP does not require an objective function and therefore can (1) obtain a solution relatively quicker and (2) discover infeasible solutions faster. This enables our framework to respond to changes more effectively and adapt the system accordingly. In addition to these three approaches, optimizing resource selection can be based on multi-criteria algorithms [15, 96, 130, 134], or using advanced reservation mechanisms [153]. However, it is not our goal to find an optimum solution but rather a fast one. Hence, we do not use any CP optimization techniques; instead, we use CP to filter out resources that do not satisfy boolean constraints. Further, we chose CP because it provides a formal declarative language that can be used to define the minimum requirements for resources to be included in the dSDE. Finally, similar to the rule-engine–based approach, we enable constraints to be specified dynamically and programmatically at any given time (i.e., new constraints can be specified during execution and existing constraints can be modified/removed), which is achieved by using a RESTful web service (see Figure 3.3).

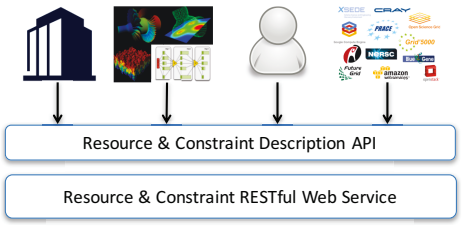


Figure 3.3: An overview of the constraint-programming–based resource programming and description abstractions.

3.2.2.2 Mathematical Model

In this section, we present the mathematical model that we define for our constraint programming approach. In this model, we define a dSDE as a set of n sites $\{S_1, S_2..., S_n\}$. Each *site* is a single physical location or region (e.g., datacenter or a zone within a cloud). We

Table 3.2: Sample resource class properties in the constraint-programming–based mathematical model. ©2017 SAGE Journals (reprinted with permission) AbdelBaky et al [2].

Property	**Description**
Availability (av)	Whether a resource class is operational
Capacity (cp)	Number of instances (e.g. nodes, VMs) in a resource class
Allocation (al)	Number of compute hours available for a shared resource class
Performance (pf)	Average performance of an instance of a resource class
Utilization (u)	Load of a resource class as a percentage of available capacity (0-100%)
Cost (c)	Price per hour for an instance of a resource class. We assume the cost per instance includes both CPU and memory costs
Power (pw)	Power consumption of a resource class
Overhead (o)	Time required to allocate an instance of a resource class
Security (sc)	Whether a resource class is secure or not
Always-on (ao)	Whether a class is provisioned on demand or is always on
Battery level (bl)	The battery charge for a mobile device or an IoT source
Data proximity (dp)	Distance between a data source and a reference location

consider that any given site S_i is composed of a set of m_i *resource classes*, where a *resource class* is defined as a set of resources within such site that share the same properties (e.g., Amazon m4.large instances, or a homogeneous cluster). Table 3.2 presents sample resource class properties that we defined in our model. We introduce linear constraints in the form of $(ax \leq b)$ and we define the decision variable x_{ij} that we use for our CP model as follows:

$$x_{ij} = \begin{cases} 1 & \text{if } i^{th} \text{ site's } j^{th} \text{ resource class} \\ & \text{satisfies all constraints} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{where } \; i = \{1, 2, \ldots, n\} \text{ and } \; j = \{1, 2, \ldots, m_i\}$$

In what follows, we illustrate how to define four different types of constraints in our model. New constraints can be formulated similarly. Simple boolean constraints (e.g., a constraint that holds true if the property is true and false otherwise) can be modeled directly. The availability defined in Table 3.2 is an example of such constraint and can be

modeled as follows. Let av_{ij} represent the i^{th} site's j^{th} resource class's availability. $av_{ij} = 1$ if the resource class is available, otherwise $av_{ij} = 0$. Then,

$$x_{ij} \leq av_{ij} \; \forall j \; \forall i$$

Other types of constraints require the use of minimum thresholds to be modeled as booleans, and a resource is selected if its property is greater than this threshold. Capacity, allocation, performance, and battery level are examples of this type of constraints, and can be modeled as follows. Let cp_{ij}, al_{ij}, pf_{ij}, and bl_{ij} represent the i^{th} site's j^{th} resource class's capacity, allocation, performance, and battery level respectively. Let CP, AL, PF, and BL represent the requested minimum capacity, allocation, performance, battery level thresholds to select this resource class. Then,

$$x_{ij} \cdot CP \leq cp_{ij} \; \forall j \; \forall i$$

$$x_{ij} \cdot AL \leq al_{ij} \; \forall j \; \forall i$$

$$x_{ij} \cdot PF \leq pf_{ij} \; \forall j \; \forall i$$

$$x_{ij} \cdot BL \leq bl_{ij} \; \forall j \; \forall i$$

Similarly, we can use a maximum threshold to specify when a resource class is eligible, i.e. a resource class is selected if its property is less than this threshold. Utilization, cost, power, overhead, and data proximity are examples of this type of constraints, and can be modeled as follows. Let u_{ij}, c_{ij}, pw_{ij}, o_{ij}, dp_{ij} represent the i^{th} site's j^{th} resource class's utilization, cost, power, overhead, and data proximity respectively. Let U, C, PW, O, and DP represent the requested maximum utilization, cost, power, overhead, data proximity

thresholds to select this resource class. Then,

$$x_{ij} \cdot u_{ij} \leq U \ \forall j \ \forall i$$

$$x_{ij} \cdot c_{ij} \leq C \ \forall j \ \forall i$$

$$x_{ij} \cdot pw_{ij} \leq PW \ \forall j \ \forall i$$

$$x_{ij} \cdot o_{ij} \leq O \ \forall j \ \forall i$$

$$x_{ij} \cdot dp_{ij} \leq DP \ \forall j \ \forall i$$

Finally, we can use a request value to see if a resource class is eligible, i.e., a resource class is selected if its property matches the requested value. Security and always-on constraints are examples of this type of constraints and can be modeled as follows.

Let sc_{ij} and ao_{ij} represent the i^{th} site's j^{th} resource class's security level and always-on indicator respectively. $sc_{ij} = 1$ if the resource class is secure, otherwise $sc_{ij} = 0$. $ao_{ij} = 1$ if the resource class is always-on, otherwise $ao_{ij} = 0$. Let SC and AO represent the requested security level and the requested type of resource. Then,

$$x_{ij} \leq (1 - sc_{ij}) \cdot (1 - SC) + (SC \cdot sc_{ij}) \ \forall j \ \forall i$$

$$x_{ij} \leq (1 - ao_{ij}) \cdot (1 - AO) + (AO \cdot ao_{ij}) \ \forall j \ \forall i$$

Depending on the use case, any of these constraints can be added to or removed from the model. Similarly, new constraints can be also formulated and added to the model. Together, a collection of these constraints defines the overall CP model passed to the solver. Users can define complex resource behavior by combining constraints to achieve the desired goal. For example, a user can define the behavior: *use all available resources as long as their utilization is below 80%, their power consumption is below 60%, and their cost per hour is less than $0.10* by combining the three constraints ($utilization < 80, power < 60, cost < 0.10$). The constraints are passed to a CP solver, which returns a list of resource classes that satisfy them. In general, any combination of resource classes that satisfy all constraints is considered a valid solution, therefore, we need to add one final constraint to the CP model to return only the solution that includes all resource classes that satisfy all constraints. This

constraint can be expressed as:

$$\text{maximize} \sum_{i=1}^{n} \sum_{j=1}^{m_i} x_{ij}$$

3.2.3 Abstractions Summary

In summary, the rule-engine–based approach allows users to have fine-grained control over every resource by independently specifying its availability over time (either directly or indirectly). However, this might become cumbersome for a large number of resources. On the other hand, the constraint-programming–based approach allows applications, users, and resource providers to set global constraints that are imposed on all available resource classes, making it easier to control a large number of resource classes. A hybrid approach leveraging both the rule engine and constraint programming can be used to control a larger list of resource classes while providing fine grained control for certain resource classes (e.g., high-end supercomputers). However, coordination between the rule engine and the CP solver must be incorporated in order to avoid conflicts and race conditions among rules and constraints.

Moreover, in the rule-engine–based approach, defining multiple rules per resource requires careful consideration of the entire policy to ensure avoiding conflicts among rules. One way to mitigate this problem is to define rule priorities which ensure that rules with higher priorities are enforced in a case of a conflict. In the constraint-programming–based approach, conflict is resolved by eliminating resource classes that do not satisfy all constraints. This can lead to an empty set of resource classes, i.e. no resource classes satisfy all imposed constraints on the system. Similarly, defining constraint priority can be used to violate some constraints with lower priority when this problem arises.

Finally, one must also consider how often the dSDE should adapt to meet the requirements imposed on the system in both approaches; real-time adaptation might lead to instability (e.g., resulting in a volatile system), whereas periodic adaptation might violate some of the defined rules/constraints. For example, setting the cost threshold very close to a fluctuating spot price point might lead to a resource being available/unavailable very frequently. One solution to this problem can be achieved by defining soft/hard rules/constraints. Using this approach, soft rules/constraints can be temporarily violated in order to keep the system stable (e.g., ignoring a certain percentage of price fluctuation around a defined threshold), whereas the system must react immediately to hard rules/constraints (e.g., exceeding a certain power threshold that can cause damage to the infrastructure).

3.3 Orchestration Plane

In the previous section, we described the resource programming and description abstractions that enable applications, users, and resource providers to describe their resource expectations in a systematic way. In this section, we present the orchestration plane, which is responsible for (1) translating the requirements defined at the top layer to a set of available resources that can be used at a given moment, (2) allocating resources and scheduling application tasks based on available resources, current workload, and QoS objectives, and (3) dynamically managing the dSDE without interrupting ongoing application execution.

This is achieved by using a two-step approach that separates (1) resource description and filtering (i.e., evaluating which resources can be used at a given time) from (2) resource selection and workload allocation. The first step utilizes a runtime framework that translates the requirements specified at the resource programming and description layer into a customized view that we call a "virtual slice", which contains a subset of services that fit the specified base requirements. A virtual slice can evolve over time adapting to changes in the service offering as well as changes in the user and application interests. In the second step, we use autonomic scheduling and matchmaking techniques to allocate the workload among the available services of the virtual slice to ensure that the QoS objectives of an application are always met. If the structure of the virtual slice changes at any time or if the workload or QoS objectives change, the process is reevaluated to adapt the workload allocation and resource provisioning accordingly.

The overall process is depicted in Figure 3.4 and the steps are described in more details below followed by the architecture and implementation of two runtime frameworks: a rule-engine–based and a constraint-programming–based runtime framework.

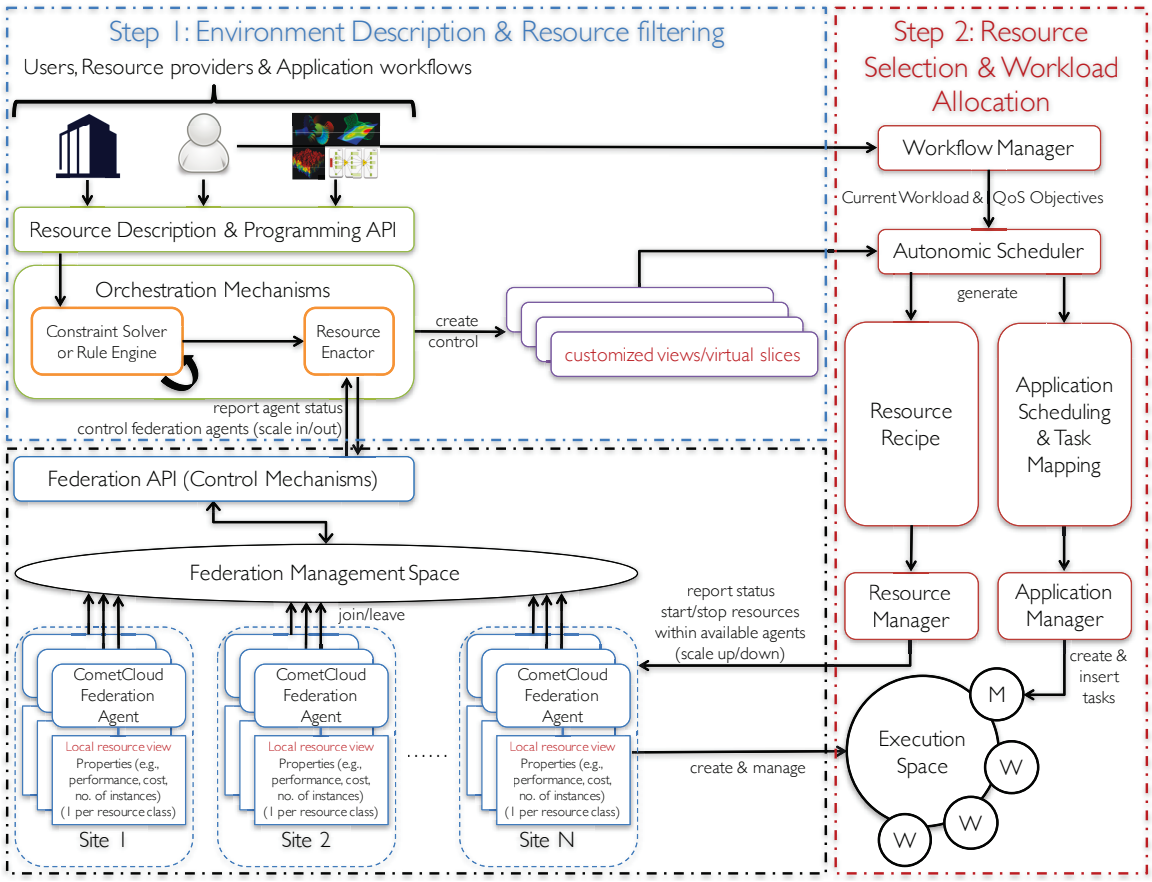


Figure 3.4: An overview of the service composition and workflow execution in a distributed software-defined environment. First, we provide federation abstraction mechanisms that enable the dynamic composition of distributed services. On top of this layer, we utilize a two-step approach for orchestrating the federation. The output of Step 1 is a virtual slice – a filtered set of resource classes that satisfies all rules/constraints, i.e., the current available resource classes at each site. The output of Step 2 is a resource recipe, which contains the exact type/number of resources to be provisioned based on the application workload and QoS objectives. ©2017 SAGE Journals (reprinted with permission) AbdelBaky et al [2].

3.3.1 Two-step Approach

3.3.1.1 Environment Description and Resource Filtering

The first step uses the resource programming and description layer to allow applications, users, and resource providers to define the desired state of the execution environment (i.e., what type of resources should be exposed to the scheduler). We begin by considering a global list of resources that the user has access to. This list is generated by actively combining information obtained by discovery mechanisms and users' knowledge. The global list can vary over time (e.g., due to the discovery of new resources or the removal of existing ones).

Additionally, resource classes in the global list are characterized by their availability, which can vary over time, and a set of properties (e.g., cost, utilization, or performance). These properties can be static (e.g., the number of cores) or dynamic (e.g., dynamic pricing). Using these properties and availabilities, we then utilize a rule-engine–based or a constraint-programming–based runtime frameworks to identify a subset of resource classes and sites that satisfy rules/constraints set forth by applications, users, and/or resource providers (e.g., minimum performance, only GPU resources, resources cheaper than $X), which creates a transient customized view, i.e., a virtual slice (see Step 1 in Figure 3.4).

3.3.1.2 Resource Selection and Workload Allocation

In the second step, the virtual slice is exposed to an autonomic scheduler [92]. The scheduler evaluates the application workload (e.g., the number of tasks in a workflow stage) and objectives (e.g., maintain data locality) to appropriately provision resources and map, schedule, and execute the application workload according to its QoS requirements (see Step 2 in Figure 3.4). Contrary to the first step, the scheduler is workload-aware in that it selects the most appropriate resources from the current slice that satisfies the given QoS objectives for the application.

The autonomic scheduler can use any state of the art scheduling strategy to, for example, exploit data locality, minimize data transfer, ensure an overall budget, etc. We integrate the scheduler into our approach to ensure that, at all times, only resources that are part of the current virtual slice can be allocated, and they are only allocated according to the application QoS objectives. For example, resources that are deemed viable in step one are not necessarily selected by the scheduler in step two, if using them would violate a QoS objective. On the other hand, resources that are not selected in step one cannot be selected by the scheduler in step two.

The scheduler then generates a list of resources to be provisioned. We call the subset of resources identified in step two the *resource recipe*, where each recipe is composed of different *ingredients* (i.e., the exact number and types of resources and services). The recipe can change over time due to changes in step one (i.e., the virtual slice), or due to changes in step two (i.e., the behavior or objectives of the application workload).

The scheduler also generates the placement of the workload tasks to be deployed on the provisioned resources, which can be achieved using a range of techniques, from simple matchmaking [102] to advance autonomics [92]. Once the schedule is determined, we use the underlying federation abstraction mechanisms, i.e., the control plane, to allocate resources and deploy the workload on top of them.

Finally, we continuously monitor the state and availability of services, applications, users, and resource providers rules/constraints, and the applications' workload and objectives, which allows us to react to changes in an online manner. The resource filtering phase (Step 1) generates a new virtual slice whenever resources join or leave, resources characteristics change, or if the rules/constraints governing resources change. As a result, the workload allocation phase (Step 2) has to be reevaluated to ensure the application QoS is not violated with the new changes in the available resources. Reevaluating the second step can also help find better allocations by using newly discovered resources. Additionally, the workload allocation (Step 2) is reevaluated if the application workload or QoS objectives change. This leads to a new resource recipe and a new workload map and schedule. The recipe and schedule are then communicated to the underlying framework, which adapts the resources and the application execution accordingly.

3.3.1.3 Two-step Approach Summary

The above process leads to the creation of a *living* distributed software-defined environment that changes and evolves in both time and space, constantly adapting to the dynamic state of the available resources, and to the constraints, requirements, and objectives imposed on the system. The framework automates the process in a way that is transparent to running applications. Further, using this two-step approach, allows users and resource providers to have better control over resources, define fine-grained requirements, and be able to easily express resource behavior and availability over time. For example, different slices can be created for different applications, projects, or users by controlling what the scheduler can see. Each slice can independently evolve over time to respond to changes in the resource properties or the requirements regulating their usage.

3.3.2 Runtime Frameworks

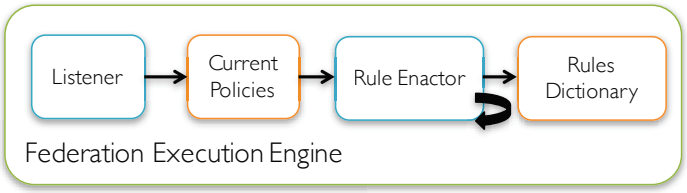
3.3.2.1 Rule-Engine–Based Framework

The rule-engine–based runtime framework enables the management of the composition process based on high-level policies. The framework is responsible for (i) translating high-level policies at runtime into a set of resources (i.e., *a virtual slice*), (ii) ensuring the orchestration of federated sites over time according to these policies, (iii) executing the application on top of the resulting dSDE, and (iv) monitoring the environment over time and modifying it as necessary based on existing or new policies.

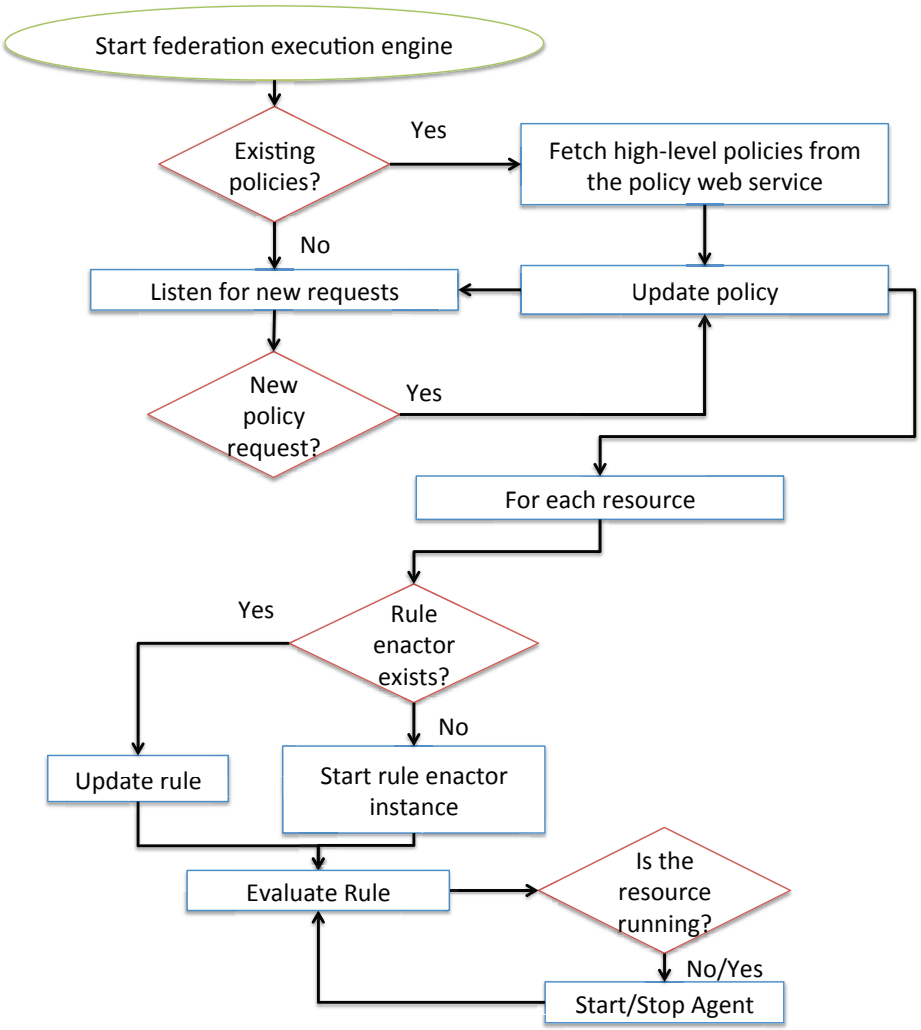
In particular, we implemented a rule-based execution engine, which is depicted in Figure 3.5a. The execution engine provides the orchestration mechanisms needed to support a dSDE that could dynamically evolve in terms of size and capabilities. The engine collects all rules from the policy layer and translates any indirect rule into an availability over time (e.g., by evaluating the rule condition during runtime). These rules are then used to create a virtual slice that defines the available resource at any given time. The engine uses this slices to create controls and operational requests that manage the dSDE and alters its overall structure by adding, modifying, or shutting down resources/sites. These requests are passed along to the underlying layers that know how to coordinate different types of resources. The engine interacts with the policy layer to maintain an up-to-date list of rules.

The execution engine is composed of two main components, a listener, and a rule enactor. Their behaviors are depicted using the flowchart in Figure 3.5b. The listener is a socket-based server responsible for communicating with the policy web service. The rule enactor is responsible for ensuring the execution of a rule for a single resource/site. An instance of the enactor is started for every available resource/site in the policy list. Currently, the execution engine is limited to a single rule per resource to avoid conflicts and race conditions (e.g., if two rules contradict). When the listener receives a new request from the policy web service, it starts a new instance of the enactor (using a unique `resource_id`) if the resource is not already running, or submits the new rule to the existing enactor responsible for the given resource otherwise.

The enactor translates a given rule using a rule dictionary and creates/manages/terminates the CometCloud federation agent responsible for this resource accordingly. The enactor controls the dSDE using the federation abstraction layer. The enactor remains in a loop while monitoring the status of the corresponding agent until a new rule is submitted or the rule is complete (e.g., available allocations on a given resource are exhausted).



(a) The overall architecture of the federation execution engine.



(b) The flowchart operation of the federation execution engine.

Figure 3.5: The rule-engine–based runtime framework architecture and its operation flowchart. ©2015 ACM (reprinted with permission) AbdelBaky et al [6].

3.3.2.2 Constraint-Programming–Based Framework

The constraint-programming–based runtime framework operates similarly to the rule-engine–based one. However, instead of a rule engine, the CP approach utilizes a constraint programming solver, which is used to generate the virtual slice (see Figure 3.6). The solver takes as input the global list of resources and the constraints regulating their usage. The solver then generates a solution including all resource classes that satisfy all constraints (i.e., a virtual slice). The slice is used to create multiple instances of the enactor, where each enactor is responsible for a resource class and creates/manages/terminates the CometCloud federation agent accordingly.

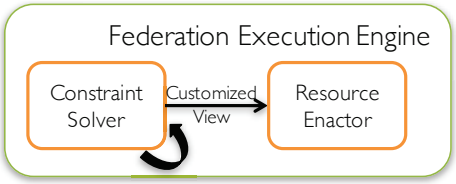


Figure 3.6: The constraint-programming–based execution engine architecture.

3.4 Control Plane

In this section, we present the bottom layers to realize a dSDE. The goal of the control plane is to expose distributed computational resources (e.g., clouds, grids, clusters, cloudlets) as *programmable infrastructure*. The control plane consists of two layers: a federation abstraction layer and a federated infrastructure layer. Their overall architecture is shown in Figure 3.7 and explained in more details below.

3.4.1 Federated Infrastructure Layer

The federated infrastructure layer provides mechanisms for creating the federated infrastructure available to an application at any time and supports the application execution using this infrastructure. This layer is responsible for federating infrastructure with different ownership, and finding an agreed interface and language in order to negotiate and establish access and usage rights based on mutual contracts. The federated infrastructure layer is built on top of the CometCloud framework and extends the CometCloud federation

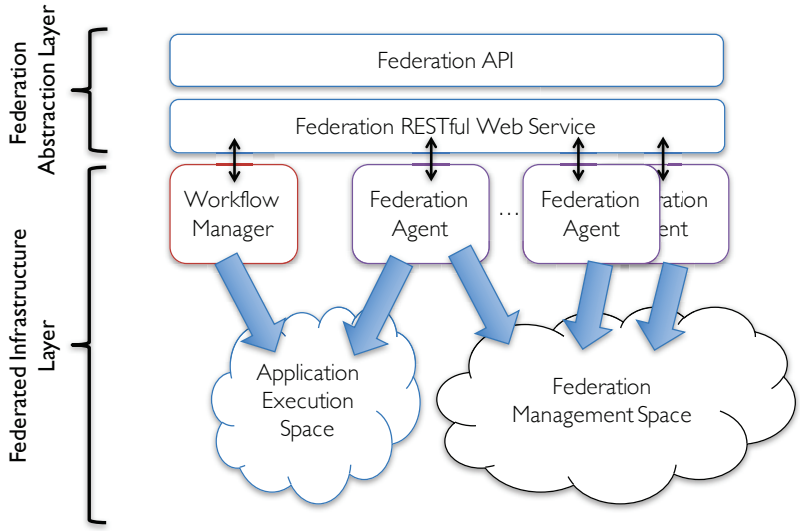


Figure 3.7: An overview of the control plane necessary for a dSDE, which utilizes the CometCloud framework [51, 93]. The control plane consists of two layers: a federation abstraction layer, and a federated infrastructure layer.

model, described in Section 2.1.2, to provide fine-grained control of the resources composing such federation by providing dynamic join/leave mechanisms for federated sites, fault tolerance, and scale up/down/out mechanisms within the federation.

3.4.2 Federation Abstraction Layer

The goal of the federation abstraction layer is to provide uniform programmatic access to the federation management. This layer supports the programmatic addition/removal of sites and provides abstractions for monitoring the status of the federated infrastructure (e.g., available sites, the number of available resources within a site, etc.). This layer accepts operational control requests to interact with the federated infrastructure layer. These requests are then translated into specific actions that are used by the federated infrastructure layer to drive the dSDE towards the desired structure. In particular, we have built a RESTful web service that exposes the necessary functionality to monitor and control the dSDE. The resulting federation abstraction API supports a set of functions that are listed in Table 3.3 and are categorized as follows:

- *Resource description operations:* These operations are used to compile a list of available resources and can be used for resource registration or discovery. For example, the

web request `register_resource` registers a resource using the provided parameters and returns a unique `resource_id` that can then be used for further operations.

- *Status operations:* These operations report the status of the dSDE (e.g., a list of available sites, a list of running sites, the status of each federation agent, and the status of applications that are in execution). These operations can be used to monitor the structure of the dSDE as well as the usage level. For example, the web request `get_reg_res` returns an XML description of all available resources (including active and inactive sites).

- *Application execution operations:* These operations are used to deploy the necessary entities that enable the execution of applications on top of the dSDE. This is achieved by creating an execution space on different sites and interacting with a workflow manager that monitors the execution of the end-to-end workflow. For example, the web request `start_workflow_manager` starts a workflow manager on a given resource (identified by a `resource_id`). The manager can then receive subsequent requests to execute different workflows on the current dSDE. Note that this assumes that applications are already deployed in CometCloud and compiled for each site that desires to support them (see Section 2.1.2.2).

- *CometCloud federation agent operations:* These operations support starting/stopping a CometCloud agent at any given site. An agent is a federation entity that provides access to specific resources and knows how to deal with the particularities of heterogeneous resources. For example, the web request `start_agent` starts the CometCloud agent on a given site (identified by a `resource_id`). Agents join or leave the federation management space when requested, ensuring that their resources are accessible to the federation only when specified. Additionally, we can pause and resume federation agents to temporally disable access to the resources they represent. This keeps agents as part of the federation space but disables their ability to allocate resources to external users and applications. This functionality can be useful to minimize joining and leaving overheads of resources that have very dynamic availability.

Table 3.3: The Federation Abstraction Layer APIs and their description. ©2017 SAGE Journals (reprinted with permission) AbdelBaky et al [2].

API Call	Description
registerResource	add a new resource class to the global list of resources
updateResource	update the information (properties or availabilities) for an existing resource class
deregisterResource	remove a resource class from the global list of resources
startAgent	start the federation agent for a given resource class
terminateAgent	terminate the federation agent for a given resource class
disableAgent	disable a resource class but do not terminate the agent (pause)
enableAgent	enable a resource class that was disabled earlier (resume)
startWorkflowManager	start the workflow manager on a given resource
terminateWorkflowManager	terminate the workflow manager on a given resource
checkWorkflowManagerStatus	check the status of the workflow manager
submitWorkflow	run a new workflow
checkWorkflowStatus	check the status of an existing workflow
cancelWorkflow	terminate an existing workflow
startCometCloudSpace	start the comet execution space on a given resource
getRunningResources	get the current running resources
getFederatedResources	get the current federated resources (includes paused resources)
checkAgentStatus	check if the federation agent is started or stopped on a given resource class
checkAgentEnabled	check if the federation agent is paused or resumed on a given resource class
getAvailableResources	return XML description of all registered resource classes

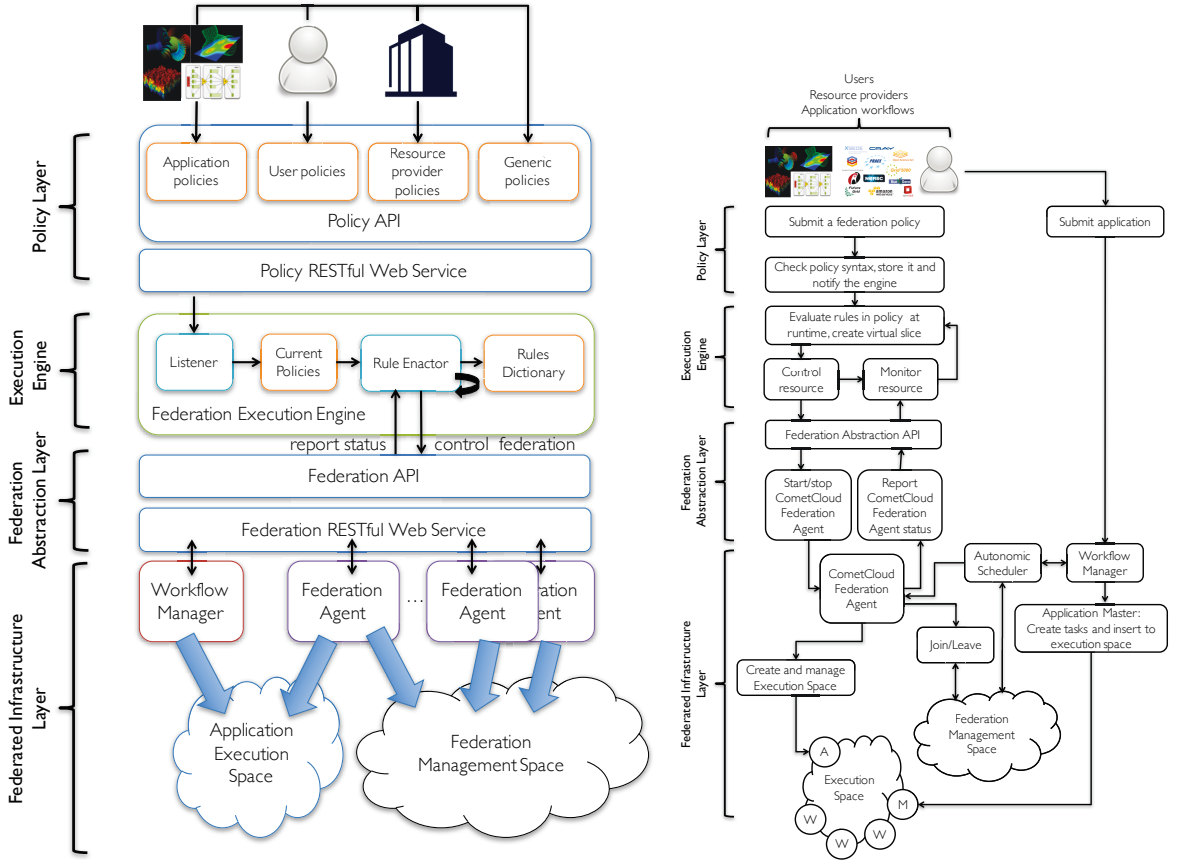
3.5 Realizing A Distributed Software-Defined Environment

In this section, we show how the three layers (i.e., resource programming and description layer, orchestration plane, and control plane) work together to create a dSDE using both the rule engine and constraint programming approaches.

3.5.1 Rule-Engine–Based dSDE

The overall architecture of the rule-engine–based approach and a flowchart diagram describing the overall operation of the rule-engine–based dSDE are shown in Figure 3.8. First, applications, users, or resource providers use the policy layer to submit a policy that contains resource usage rules. These rules can be modified during application runtime. The policy layer verifies these rules and notifies the execution engine. The execution engine evaluates the rules during runtime and creates virtual slices accordingly. The execution engine then

starts, stops, and monitors CometCloud federation agents according to the resources available in the virtual slice. The agents join/leave the federation management space, which is exposed to the autonomic scheduler. The agents also create local execution spaces at their sites that can be used to execute a workload. The user also submits the application or workflow to be executed along with its QoS objectives to a workflow manager. The manager communicates the current workload and QoS objective to the autonomic scheduler. The scheduler generates the resource recipe and task placement. The scheduler notifies the federation agents to allocate resources and workers according to the resource recipe. The scheduler also provides the workflow manager with the task placement. The scheduler communicates with the application master to create tasks and insert them into the execution space. Finally, the workers execute the tasks.



(a) The overall rule-engine–based dSDE architecture.

(b) The overall rule-engine–based dSDE operation flowchart.

Figure 3.8: The overall rule-engine–based dSDE architecture and operation flowchart. ©2015 ACM (reprinted with permission) AbdelBaky et al [6].

3.5.2 Constraint-Programming–Based dSDE

The overall system architecture of the constraint-programming–based approach is shown in Figure 3.9. The main components are described below.

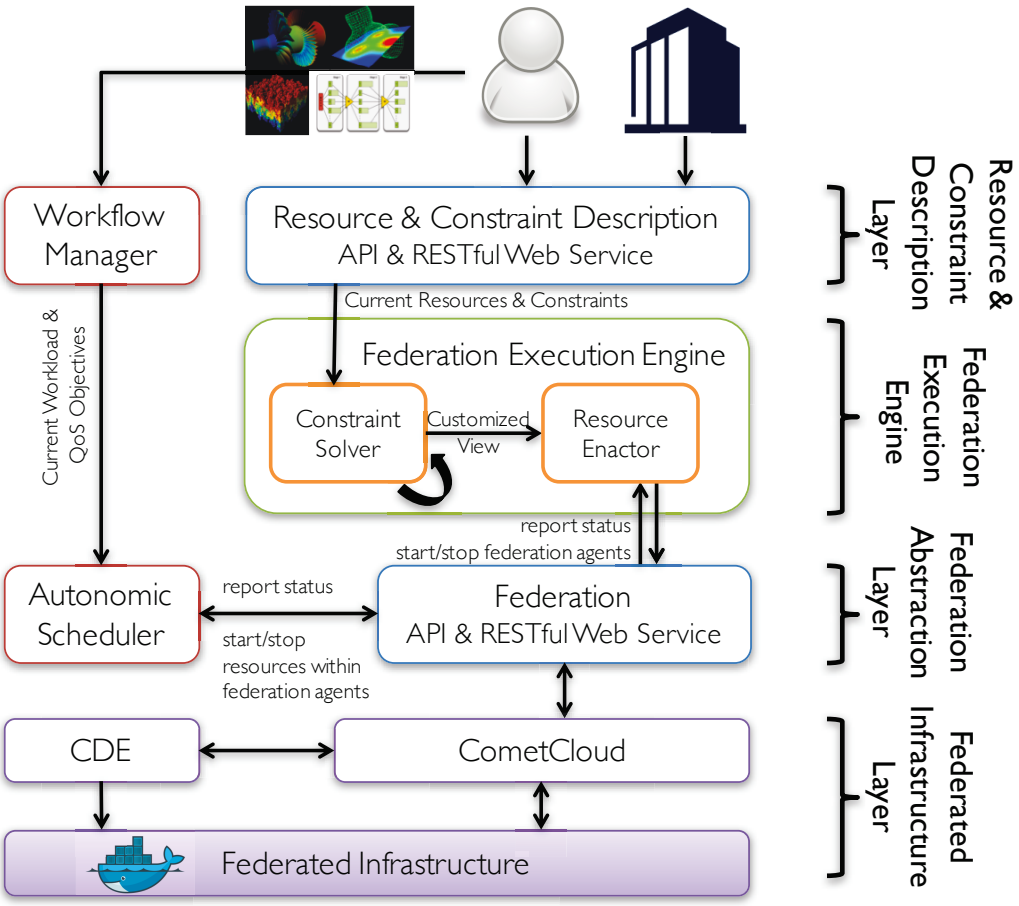


Figure 3.9: The overall architecture of the constraint-programming–based dSDE. ©2017 IEEE (reprinted with permission) AbdelBaky et al [5]. The federation execution engine is responsible for starting/stopping/pausing/resuming CometCloud federation agents based on constraints. Each federation agent is responsible for a single resource class within a single site. The Autonomic Scheduler is responsible for allocating resources within each resource class.

- *Resource and Constraint Description Layer:* provides mechanisms for expressing the attributes of the dSDE in terms of resource availabilities, properties, and constraints. This layer is exposed using a RESTful web service and can be accessed programmatically. Applications, users, and resource providers use this layer to add/update/remove resource classes and/or constraints. Updates at this layer are propagated to the federation execution engine.

- *Federation Execution Engine:* manages the composition process based on the properties, availabilities, and constraints specified at the description layer. It consists of two main components, a CP solver, and a Resource Enactor. The solver generates a virtual slice of resources that satisfies all constraints. We currently use a Python Constraint Solver [2]. The solver runs whenever updates are submitted to the description layer. The virtual slice is then passed on to the resource enactor and the autonomic scheduler and they adapt the environment accordingly.

- *Workflow Manager:* receives information about application workflows including their requirements, objectives, and other details such as the number of tasks per stage, stage dependencies, etc. The workflow manager passes application stages that are ready to execute along with their associated QoS objectives to the scheduler.

- *Autonomic Scheduler:* is responsible for deploying the current workload, encapsulated inside containers, on a set of selected resources based on different optimization objectives. The scheduler receives the application information as well as the currently available resources from the federation abstraction layer to create the resource recipe and the workflow placement. The scheduler continuously monitors the execution of the applications as well as the current virtual slice and adapt the dSDE as needed.

- *Container Deployment Enactor (CDE)* deploys containers directly on any allocated resource or interacts with the local scheduler to submit container jobs. CDE is built on top of CometCloud to support the deployment of Docker containers using the native docker environment [3] or alternatively using Cloud Foundry [4], and can be easily extended to support other container deployment schedulers such as Kubernetes [5]. The CDE can be used to deploy any container image in a public/private registry (as long as the user has access to the registry). New workflows can be easily deployed without any modification to our framework once they are 'containerized'.

---

[2]Python Constraint Solver: https://labix.org/python-constraint

[3]Docker: https://www.docker.com

[4]Cloud Foundry: https://www.cloudfoundry.org

[5]Kubernetes: http://kubernetes.io

3.6 Summary

In this chapter, we presented an approach that enables the programmatic control of services to realize a distributed software-defined environment. Our approach is based on an abstraction that enables dynamic composition of services driven by high-level rules or constraints. Rules and constraints can be programmatically defined as a function of time or runtime events. We compared the tradeoffs of both resource description approaches.

Our approach enables elastic and on-demand aggregation of distributed services and resources from multiple providers (e.g., clouds, grids, data centers). First, we used a software-defined programmable abstraction to allow applications, users, and resource providers to programmatically express resource requirements, which are then translated into a customized view of the federation, called a virtual slice. A virtual slice can evolve over time adapting to changes in the service offering as well as changes in the user and application interests. We presented two ways of implementing our software-defined approach, using a rule engine and constraint programming, respectively. Finally, we used autonomic scheduling and matchmaking techniques to allocate the workload among the available services of the virtual slice to ensure that application QoS objectives are always met. If the composition of the virtual slice changes or if the application workload or QoS objectives change, the process is reevaluated to adapt the workload allocation and resource provisioning accordingly. As a result, our approach creates a nimble and dynamically programmable environment that evolves over time, adapting to changes in infrastructure state and application requirements.

Our prototype leverages Docker containers for encapsulating and deploying application workflows across the dynamically federated infrastructure. We utilized Docker for our work for multiple reasons. First, the agile nature of container workloads coupled with their requirement for fast deployment and adaptation to changing environments makes them a good candidate to test the usability and adaptability of our framework, which we will show in Chapter 4. Furthermore, the use of containers adds another dimension of accessibility and portability to our framework (i.e., the entire application environment is encapsulated inside containers and run anywhere using our framework). Lastly, current solutions for deploying Docker containers focus on single-site implementations, whereas our work enables Docker users to dynamically run across multiple sites or clouds.

3.7 Relevant Publications

This chapter contains portions adapted from the following published papers with permissions from the copyright holder.

1. M. AbdelBaky, J. Diaz-Montes, M. Zou, and M. Parashar. A framework for realizing software-defined federations for scientific workflows. In Proceedings of the 2nd International Workshop on Software-Defined Ecosystems, pages 7-14. ACM, 2015.

2. M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder. Docker containers across multiple clouds and data centers. In 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pages 368-371. IEEE, 2015.

3. M. AbdelBaky, J. Diaz-Montes, M. Unuvar, M. Romanus, M. Steinder, and M. Parashar. Enabling distributed software-defined environments using dynamic infrastructure service composition. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2017 – to appear.

4. M. Abdelbaky, J. Diaz-Montes, and M. Parashar. Towards distributed software-defined environments. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2017 – to appear.

5. M. AbdelBaky, J. Diaz-Montes, and M. Parashar. Software-defined environments for science and engineering. International Journal of High Performance Computing Applications, 2017 – to appear.

Chapter 4

Evaluation of the Distributed Software-Defined Environment

4.1 Introduction

In this chapter, we present an empirical and experimental evaluation of the programming system that enables our distributed Software-Defined Environment (dSDE). In particular, we evaluated our approach using user-driven use case scenarios. The experimental evaluation used a scientific application workflow as a driving use case but this work applies equally to other scenarios such as the IoT scenario described in Chapter 1. The evaluation of the system also used only infrastructure resources and services, however, we will show in Chapter 5 that our approach applies to other services such as network or storage. In the remainder of this chapter, we first present the experimental evaluation of the rule-engine–based approach using a user-driven use case scenario. Next, we present the empirical evaluation of the scalability and correctness of the constraint programming solver for resource filtering. Finally, we present an extensive evaluation of the constraint-programming–based dSDE using multiple experiments.

4.2 Experimental Evaluation of the Rule Engine Approach

In this section, we evaluated the rule-engine–based dSDE through a user-driven use case scenario where users have fine-grained control over available resources. We first present the scenario details and our evaluation goals, followed by the experimental setup, results, and conclusion.

4.2.1 Use Case Scenario

Consider a scientist who has access to several geographically distributed computational resources – e.g., XSEDE [144] supercomputers or grid resources (with limited allocation units), a local campus cluster, and public clouds. The scientist wants to combine these resources to run a large scale application workflow. Further, they want to define how these resources should be used to run their application.

For example, a scientist may prefer to use a certain type of resources over others (e.g., HPC versus clouds or "free" HPC systems versus the allocation-based ones); they may want to reserve XSEDE resources for specific projects or restrict the maximum amount of grid allocations that a single workflow can consume; or they may be only allowed to use certain resources at certain times of the day. More complex requirements may involve allowing the use of public clouds only if the local cluster becomes unavailable during the execution of a workflow and the price per cloud instance is under a certain threshold – additionally, this threshold should be easily adjusted/removed to meet a deadline. Finally, a scientist may want to specify how to react to unexpected changes in the resource availability or performance or they may want to only use resources within the US or Europe due to the laws regulating data movement across borders. Therefore, by allowing users to define this kind of rules in a formal and abstract manner, they can achieve unprecedented control over the resources, which can improve the efficiency of resource use and maximize the application's outcome.

The rule-engine–based dSDE provides users with two different ways of specifying the usage policy for a list of available resources. In the first method, the scientist can declare a policy that specifies the exact composition of the dSDE over time, whereas in the second method, they can define the desired behavior of the dSDE but not its exact composition. The two examples below demonstrate the use of these two types of policies.

Suppose a scientist has access to the following resources:

1. Resource A (local cluster) - full access
2. Resource B (supercomputer) - 5000 allocation units
3. Resource C (public cloud) - pay per usage

In the first scenario, the scientist defines the following policy:

- Run on Resource A always
- Run on Resource B daily from 9:00 AM to 5:00 PM
- Run on Resource C from January 8, 2015, 9:00 AM to January 10, 2015, 12:00 PM

Based on the described policy, the system will start resource A immediately and continuously monitor its status until the policy is changed or the resource is removed from the list of available resources. Further, the system will start resources B and C at their respective `start_time`, monitor their status, then stop them at their respective `end_time`.

In the second scenario, the scientist defines the following policy:

- Run on Resource B using 2500 allocations
- Run on Resource C when the dynamic price per instance is less than $0.1 per hour and not exceeding a budget of $50

In this case, the system will start resource B, monitor the number of allocations used, and release it when the target allocation is met. For Resource C, the system will monitor its dynamic price and keep it running as long as the dynamic price is below the specified threshold and the total budget allocated is not yet exhausted.

Evaluation Goals. Using this scenario, we set up an experiment to evaluate the rule-engine–based dSDE. The goal of this experiment was to show that the system can (1) combine multiple geographically distributed resources and use them to execute an application and (2) automatically adapt the composition of resources based on user-driven rules.

4.2.2 Experimental Setup

We executed a synthetic bag-of-tasks application on a dynamic federation of resources. The QoS objective defined for this application was maximizing throughput, i.e., aggregating as much computational power as possible. Our environment was composed of four geographically distributed sites. Table 4.1 describes the resources utilized as well as their

computational characteristics. These are all the resources that the user had access to. However, the usage of some of these resources was limited by resource providers. As a result, the user specified some rules using time-based generic policies as follows.

- Run on Future Systems always
- Run on Spring daily from 11:05 AM to 11:40 AM
- Run on Green from 02/28/2015 11:15 AM to 02/28/2015 11:30 AM

Table 4.1: Rule engine evaluation: resources available at each site and their characteristics. ©2015 ACM (reprinted with permission) AbdelBaky et al [6].

Future Systems - OpenStack Cloud				
Resource	#Cores	Memory	Performance	Max. VMs‡
VM_Medium	2	4 GB	1.36	3
VM_Small	1	2 GB	0.69	6
Spring - HPC Cluster				
Resource†	#Cores	Memory	Performance	Max. Machine‡
Bare-metal	8	24 GB	1.42	16
Green - HPC Cluster				
Resource†	#Cores	Memory	Performance	Max. Machine‡
Bare-metal	8	24 GB	0.42	16
Chameleon - OpenStack Cloud				
Resource	#Cores	Memory	Performance	Max. VMs‡
VM_Medium	2	4 GB	1	3
VM_Small	1	2 GB	0.5	4

Note: ‡ – Maximum number of available VMs/bare-metal per type

In addition, ten minutes after the experiment started, the user added a new cloud resource, which offered a dynamic pricing model similar to Amazon EC2 spot instance model [1]. In this experiment, we emulated this model on one of the cloud infrastructures provided by the Chameleon Project [2]. The varying price of the instances is shown in Figure 4.1a. In our experiment, the user wanted to take advantage of the dynamic pricing

[1]Amazon EC2 Spot Price Model: http://aws.amazon.com/ec2/purchasing-options/spot-instances

[2]Chameleon Cloud: https://www.chameleoncloud.org

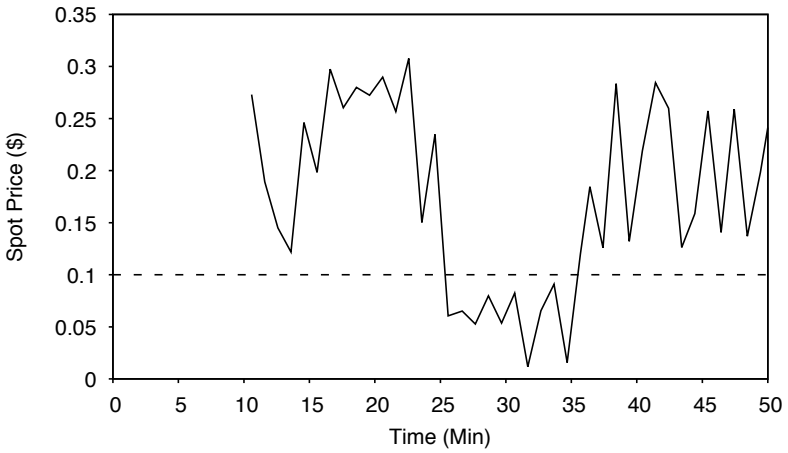
model and established a policy that defined the maximum price they were willing to pay for these instances. The rule was as follows.

- Run on Chameleon when the dynamic price is less than $0.1 per hour

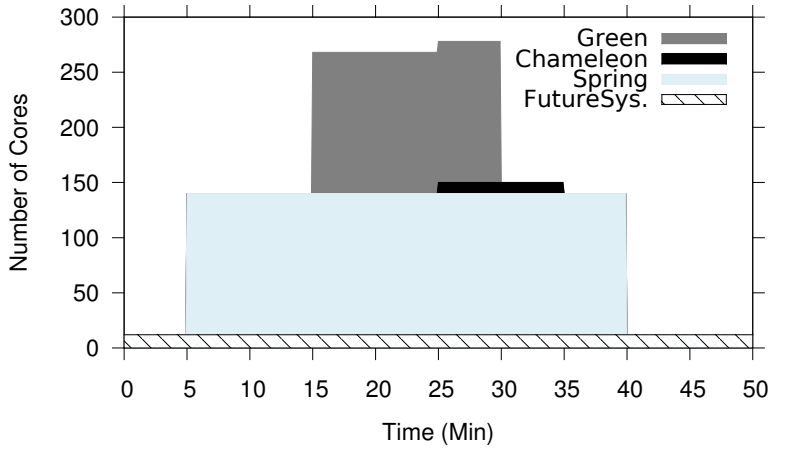
4.2.3 Results

We started the experiment on 02/28/2015 at 11:00 AM, which is referred to as time 0 henceforth. Table 4.2 summarizes the different actions taken during the experiment. The first action (at time 0) was deploying the appropriate services and creating the environment by registering the resources and their rules, as the user defined. Next, the experiment is started by launching the application and deciding which resources can be allocated to execute the application. Once the experiment was running, the system periodically evaluated the rules, creating a new virtual slice when modifications in the environment were needed and allocating/deallocating resources to maximize the overall aggregated power. Figure 4.1b shows how these events changed the composition of the environment over time according to the specified rules.

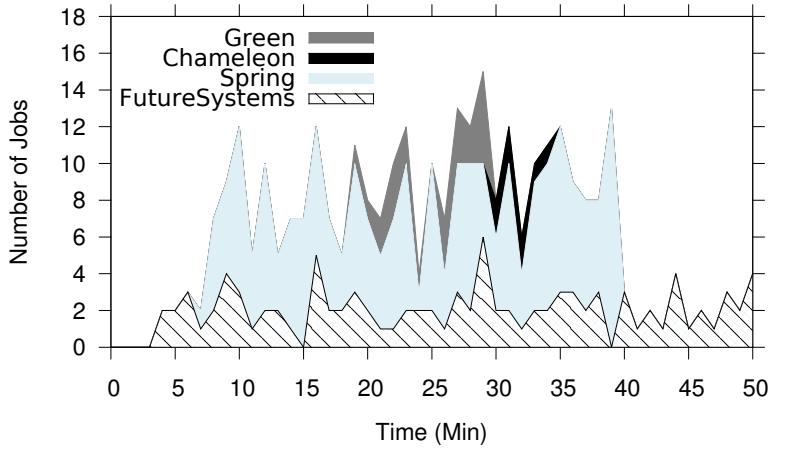
Since the objective of the application was to maximize throughput, we can observe, in Figure 4.1c, how the throughput varied adapting to the changes in the environment. Specifically, Figure 4.1c shows the throughput per site in a stacked area chart. We can observe how the throughput dramatically increased when the Spring cluster, with powerful resources, joined the environment. Similarly, we see that the peak performance of the dSDE occurs when all four sites were contributing resources. Following the dynamic pricing model presented in Figure 4.1a and the price defined specified by the user, we can observe how Chameleon resources joined the environment between time 25 and 35 of our experiment, which was the time period where the instance price was below 0.1 dollars.



(a) Emulated price of a spot instance over time



(b) Execution time



(c) Throughput, measured as the number of tasks completed over time

Figure 4.1: Summary of experimental results for the rule-engine–based dSDE. ©2015 ACM (reprinted with permission) AbdelBaky et al [6].

Table 4.2: Rule engine evaluation: actions triggered by the rule engine that created the environment and modified it over time to comply with established rules. ©2015 ACM (reprinted with permission) AbdelBaky et al [6].

Time (Min)	Event
00	Start services
	(WFM, CS, Execution Engine)
	Register and schedule future systems
	Register and schedule spring
	Register and schedule green
	Start experiment
	Future Systems starts
05	Spring starts
10	register and schedule Chameleon
15	Green starts
25	Chameleon starts
30	Green ends
35	Chameleon ends
40	Spring ends
50	Workflow complete
	Future Systems ends

4.2.4 Conclusion

We evaluated the rule-engine–based dSDE framework using a user-centric use case and demonstrated how our framework allows users to programmatically and dynamically adapt the environment over time without interrupting application execution. Similarly, the framework can be used by application workflows to allow them to negotiate their own resources, or by resource providers to allow them to define the environment to meet different requirements. While we have used relatively simple rules in these experiments, they do serve as a proof of concept to validate the essential mechanisms to enable our distributed software-defined environment.

4.3 Empirical Evaluation of the Constraint Programming Solver

4.3.1 Heuristic

To evaluate the performance of the CP-based approach, we developed a simple heuristic for resource filtering. The heuristic approach calculates a single solution that includes all viable resource classes. In contrast, the CP solver provides every possible combination. The heuristic approach iterates over all resource classes in each site, and filters out resource classes that do not satisfy all constraints. The algorithm for the heuristic approach is shown below and uses the same notations defined in Section 3.2.2.2.

Algorithm 1 Algorithm for the Heuristic Approach. ©2017 IEEE (reprinted with permission) AbdelBaky et al [5].

1: **for** $i = \{1, 2, \ldots, n\}$ **do**
2: **for** $j = \{1, 2, \ldots, m_i\}$ **do**
3: $x_{ij} = 0$
4: **if** $av_{ij} == 1$ *and* $CP \leq cp_{ij}$ *and* $AL \leq al_{ij}$ *and* $PF \leq pf_{ij}$ *and* $BL \leq bl_{ij}$ *and* $u_{ij} \leq U$ *and* $c_{ij} \leq C$ *and* $pw_{ij} \leq PW$ *and* $o_{ij} \leq O$ *and* $dp_{ij} \leq DP$ *and* $sc_{ij} == SC$ *and* $ao_{ij} == AO$ **then**
5: $x_{ij} = 1$
6: **end if**
7: **end for**
8: **end for**

4.3.2 Experimental Setup and Evaluation Criteria

In this experiment, we evaluate the performance and correctness of the CP resource filtering. We compare it to heuristic, predefined, and random resource filtering. Predefined and random filtering do not consider constraints when selecting resource classes. We used two types of predefined filtering, `predefined_select_half`, which always returns the first half of the global list, and `predefined_select_all`, which returns the full list. Random filtering returns a random selection of resource classes from the list. We ran a simulation on a list of resource classes with sizes varying from one to one million, using a normal distribution of random properties and availabilities. We varied the number of imposed constraints as follows: no constraints, five constraints, and ten constraints. We ran each of the solvers and measured the performance and the correctness of the solution for each case. The simulations ran on a machine with a 1.7 GHz Intel Core i5 CPU and 4 GB of memory. To evaluate correctness, we use a utility function that calculates a *correctness factor*, which is defined as the percentage of constraint violation (if any). The correctness factor is normalized

over the total number of selected resource classes and the total number of constraints. We assume that constraints and resource classes have equal weight. For example, if the input to the utility function is five constraints and a correct solution includes 20 resource classes, then each resource class satisfying one constraint counts for 1% of the overall correctness factor. We have also developed a second utility function that calculates a *missed resources factor*, which is defined as the percentage accounting for resources that should have been selected (i.e., they satisfy all constraints) but was not selected. The missed resources factor is normalized over the total number of selected resource classes.

4.3.3 Results

Performance. The results are plotted in Figure 4.2 and show strong linear scaling for all four approaches when increasing the number of resources. As expected, increasing the number of constraints increases the time-to-solution. The CP solver finds the solution for up to 10,000 resource classes and 10 constraints in less than 1.5 seconds. Given that each class is composed of one or more resources, this shows that the CP approach is suitable for a very large number of resources. The CP approach takes more time because the CP solver generates every possible combination of resource selection, and then returns the combination with the most resources that satisfy all constraints. This requires creating a large search space that can accommodate all combinations. The heuristic approach performs very close to the predefined selection and better than the CP-based approach.

Correctness. The correctness and missed resources factors were evaluated for 1000 and 1,000,000 resource classes using five and ten constraints. The results are plotted in Figure 4.3. The figure shows that the selections generated by both the CP and heuristic approaches always satisfy all constraints with a correctness factor of 100%. Additionally, neither of them miss any resources with a missed resources factor of 0%. The random and predefined selections have an average correctness factor of 78%; (i.e., on average the selections generated by these approaches violate 22% of the given constraints). Additionally, the predefined (select_first_half) and the random selection have an average missed resources factor of 50% (i.e., on average the selections generated by these approaches miss half of the viable resources).

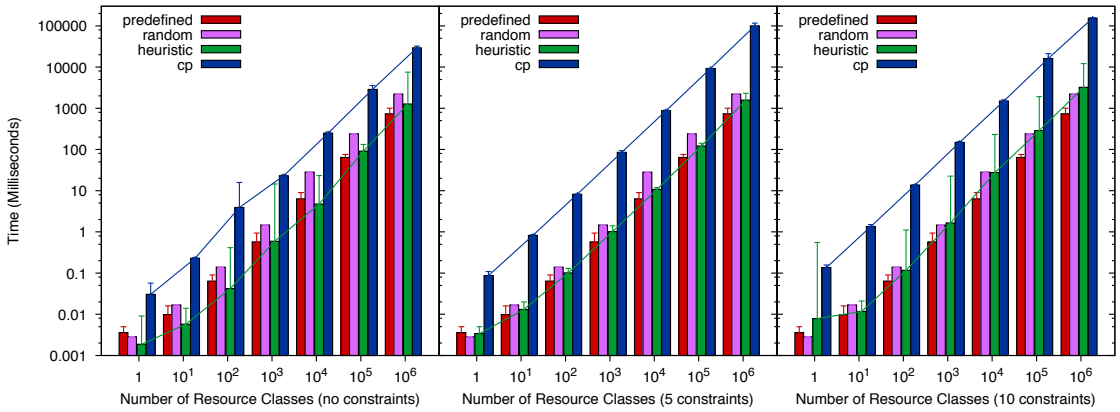


Figure 4.2: Performance of the constraint-programming–based and heuristic-based resource filtering approaches. Given a global list of resources; random filtering returns a random selection of resource classes. Predefined filtering includes (predefined_select_half), which always returns the first half of the list, and (predefined_select_all), which returns the full list. ©2017 IEEE (reprinted with permission) AbdelBaky et al [5].

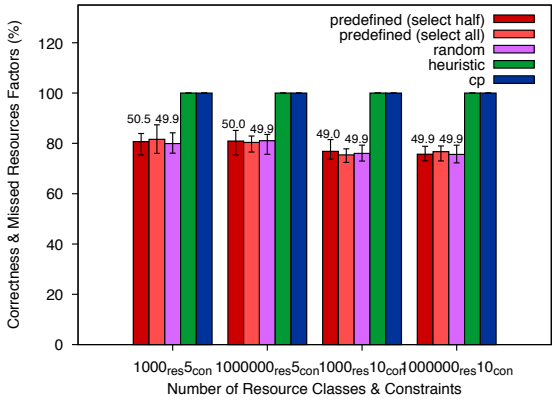


Figure 4.3: Correctness and missed resources factors, showing the percentage of satisfied constraints and the percentage of missed resources for each resource filtering approach. ©2017 IEEE (reprinted with permission) AbdelBaky et al [5].

4.3.4 Conclusion

While the heuristic approach provides better performance, a key limitation of this approach is that modeling constraints can be cumbersome when considering more complex scenarios. Additionally, the heuristic approach cannot return different combinations of viable solutions, which can be used to introduce different QoS points with different tradeoffs. A key advantage of the CP approach is its ability to easily add or remove constraints without modifying the rest of the framework. Finally, constraint programming provides a formal language and syntax that can be used by users, applications, and/or resource providers

to easily express constraints and define the desired state of the execution environment. A hybrid approach that mixes the heuristic and CP solvers can be used to provide simple ways of specifying constraints, yet provide better performance at extreme large scales.

4.4 Experimental Evaluation of the Constraint Programming Approach

In this section, we evaluate the constraint-programming–based dSDE using a bioinformatics workflow and a user-driven scenario where users have global control over available infrastructure services from multiple cloud providers. We first present the scenario details, evaluation goals, and driving application, followed by the experimental setup, results, and conclusion.

4.4.1 Use Case Scenario and Evaluation Goals

Consider a scientist who has access to infrastructure services from multiple cloud providers. The scientist wants to combine these services to run multiple dynamic workloads with varying constraints and QoS requirements. Given these services and workloads, oftentimes, the scientist needs to decide how to distribute the available resources to maximize the outcome across several projects. Using our dSDE this can be achieved by creating multiple virtual slices of the same underlying resources, which are then exposed to different applications. Further, our approach allows users to control each slice individually where each slice can evolve separately depending on the workload behavior and imposed constraints.

Evaluation Goals. The goal of the following experiments is to show that our system can (1) dynamically compose distributed services from multiple cloud providers, (2) automatically adjust the composition in response to dynamic application workload, (3) automatically adjust the composition due to changes in the infrastructure properties or the constraints regulating their usage, and (4) create multiple views/compositions of the same underlying services, which is then used to execute similar or independent workloads.

4.4.2 Driving Application

We containerized a synthesized workflow based on a histopathology image analysis application and used it as a driver for our experiments. Specifically, the application is an

automatic Gleason-grading workflow of prostate malignancy using pathology specimens of prostate tissues, which provides critical guidance for prostate cancer diagnoses and further treatment [157]. This workflow is composed of two stages: 1) region segmentation, which extracts image patches from whole slide imaging (WSI) of the sample slides and locates regions of interest (ROI); and 2) grade classification, which gives a Gleason grade for each segmented ROI by a 3-level classifier. There is a barrier between the two stages (i.e., all tasks in S1 must finish before S2 is executed). The characterization of this workflow showed that each completed job from stage one generated six jobs for stage two. The execution time of jobs within the first stage was between 10 and 250 seconds, and the execution time for each job in stage two was between two and 100 seconds. Since the application was containerized, every job was executed inside a container that is instantiated on demand.

4.4.3 Experimental Setup

We conducted our experiments in a distributed multi-cloud environment with resources from five different cloud providers at seven independent sites: an OpenStack community cloud from the Chameleon project [3], located at TACC in Texas; two different zones (us-west-2 and us-east-1 regions) from the public cloud provided by Amazon Web Services [4], located in Oregon and N. Virginia respectively; two different zones (East US and West US) from the public cloud provided by Microsoft Azure [5], located in Virginia and California respectively; one zone (us-east1-b) from the public cloud provided by Google Cloud Platform [6] located in S. Carolina; and one zone (us-south) from the public cloud provided by IBM Bluemix [7]. Additionally, we used a local cluster located at Rutgers University in New Jersey to host our framework and web services. We ran Docker containers inside virtual machines using the native Docker environment in all clouds except for IBM Bluemix. IBM Bluemix supports deploying Docker containers directly on bare metal hardware. We also used separate virtual machines to deploy CometCloud agents for a total of 15 instances. Detailed characteristics of

[3]Chameleon Cloud: https://www.chameleoncloud.org

[4]Amazon Elastic Compute Cloud: https://aws.amazon.com/ec2/

[5]Microsoft Azure: https://azure.microsoft.com/en-us/

[6]Google Compute Engine: https://cloud.google.com/compute/

[7]IBM Bluemix: https://console.ng.bluemix.net

the resources used at each site are presented in Table 4.3. The performance of the resources is represented by the speedup and has been experimentally calculated as a function of the performance of the Azure_East instance using the UnixBench benchmark [8], which was used to characterize the bioinformatics workflow. The cost ($) per hour for all cloud providers is based on their real pricing models, except Chameleon which is estimated using AWS pricing model. In our experiments, we scheduled the workload with a QoS objective of maximizing throughput by allocating jobs to the resources with the minimum estimated time of completion.

4.4.4 Results

4.4.4.1 Experiment 1: Single Slice - Varying Workload - Static Resources and No Constraints

In the first experiment, named cancer dynamic run (CDR), the resource class availabilities, properties, and constraints were fixed, whereas the application workload varied over time. In particular, two cancer application workloads were executed back to back. The first workload was submitted at time (t=0) and had two stages, Stage one (S1) had 10 jobs, Stage two (S2) had 60 jobs. The second workload was submitted at (t=11) and had two stages, Stage one (S1) had 100 jobs, and Stage two (S2) had 600 jobs.

Figure 4.4 shows the provisioned resources (left) and the job throughput (right) per resource class over the course of the experiment. The graphs show the provisioned resources successfully adapted to the workload by allocating and deallocating resources to meet the demand. The graphs show all resources were deallocated after the completion of the first workload (t=8) and did not start again until the submission of the second workload (t=11). Since resource availability and constraints were fixed, this adaptability was achieved by only reevaluating the workload allocation step (step two of our two-step approach) whenever changes in the workload were detected. The specific blend of resources depended on the scheduling policy used. In our case, we aimed at maximizing throughput and for that reason, Chameleon site was preferred as it had the most powerful resources.

[8] UnixBenchmark: https://github.com/cloudharmony/unixbench

Table 4.3: Constraint programming evaluation: resources available at each site.

(a) Experiment 1, 2, and 3

Site Name	VM Type	Cores	Memory	Max. VMs‡	UnixBench Score	Speedup	Cost($)◇	Overhead	Conts.△
AWS US East	t2.micro	1	1 (GB)	10	1785.6	2.39	0.013	30 (secs)	1
	t2.small	1	2 (GB)	10	1785.1	2.39	0.026	30 (secs)	1
	t2.medium	2	4 (GB)	10	2499.9	3.35	0.052	30 (secs)	2
	t2.large	2	8 (GB)	10	2588.2	3.47	0.104	30 (secs)	2
AWS US West	t2.micro	1	1 (GB)	10	1881.1	2.52	0.013	30 (secs)	1
	t2.small	1	2 (GB)	10	1736.9	2.33	0.026	30 (secs)	1
	t2.medium	2	4 (GB)	10	2573.3	3.45	0.052	30 (secs)	2
	t2.large	2	8 (GB)	10	2586	3.47	0.104	30 (secs)	2
Chameleon	m1.small	1	2 (GB)	8	1856.7	2.49	0.026	15 (secs)	1
	m1.medium	2	4 (GB)	6	2979	3.99	0.052	15 (secs)	2
	m1.large	4	8 (GB)	4	4380.2	5.87	0.209	15 (secs)	4
Azure US East	Standard-A1	1	1.75 (GB)	3	746.2	1.00	0.044	5 (secs)	1
Azure US West	Standard-D1	1	3.5 (GB)	3	1271.4	1.70	0.077	5 (secs)	1
Google US East	n1-standard-1	1	3.75 (GB)	3	1793.1	2.40	0.05	5 (secs)	1
IBM US South	Bluemix	N/A	N/A	3∗	N/A	N/A	0.028	45 (secs)	3∗

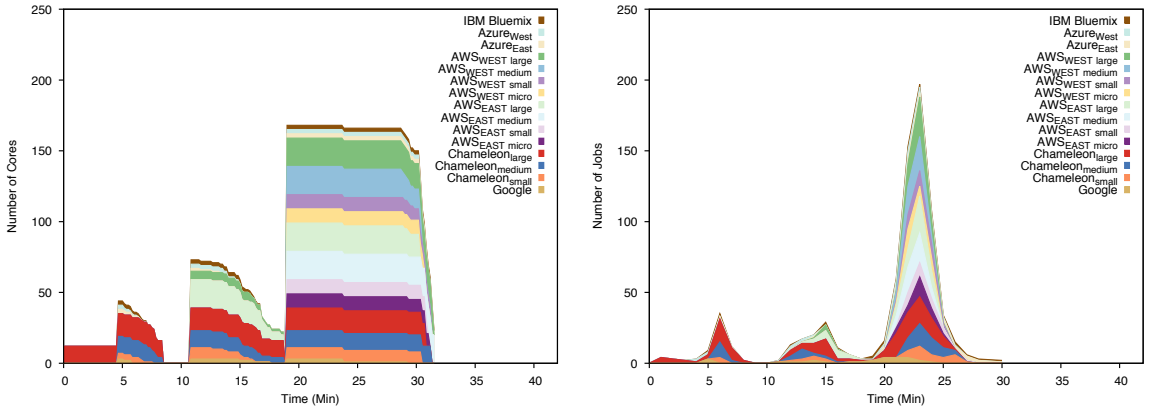
(b) Experiments 4 and 5

Site Name	VM Type	Cores	Memory	Max. VMs‡	UnixBench Score	Speedup	Cost($)◇	Overhead	Conts.△
AWS US East	t2.micro	1	1 (GB)	10	1785.6	2.39	0.013	30 (secs)	1
	t2.small	1	2 (GB)	10	1785.1	2.39	0.026	30 (secs)	1
	t2.medium	2	4 (GB)	10	2499.9	3.35	0.052	30 (secs)	2
	t2.large	2	8 (GB)	10	2588.2	3.47	0.104	30 (secs)	2
AWS US West	t2.micro	1	1 (GB)	10	1881.1	2.52	0.013	30 (secs)	1
	t2.small	1	2 (GB)	10	1736.9	2.33	0.026	30 (secs)	1
	t2.medium	2	4 (GB)	10	2573.3	3.45	0.052	30 (secs)	2
	t2.large	2	8 (GB)	10	2586	3.47	0.104	30 (secs)	2
Chameleon	m1.small	1	2 (GB)	4	1856.7	2.49	0.026	15 (secs)	1
	m1.medium	2	4 (GB)	4	2979	3.99	0.052	15 (secs)	2
	m1.large	4	8 (GB)	4	4380.2	5.87	0.209	15 (secs)	4

Note: ‡ – Maximum number of available VMs per type.
◇ – Actual cost per hour except for Chameleon, which was based on Amazon pricing.
△ – Number of containers deployed per instance.
∗ Max number of containers for Bluemix.

4.4.4.2 Experiment 2: Single Slice - Varying Workload - Varying Resources and Constraints

In this experiment, we used a large workload to force the scheduler to initially allocate all available resources in the virtual slice. We ran two scenarios, a cancer base run (CBR) scenario, where the virtual slice contained all resources from the global list and did not



(a) CDR Resources (b) CDR Throughput

Figure 4.4: Constraint programming evaluation: Experiment 1 – the provisioned resources and throughput for the cancer dynamic run (CDR). Two workflows each composed of two stages with varying workloads were executed in this experiment.

change during runtime. In the second scenario, named cancer constrained run (CCR), the virtual slice changed as we added/removed resources, and adjusted resource class properties, availabilities, and constraints during runtime. The workload for both scenarios was the same and was composed of two stages, Stage one (S1) had 1000 jobs, Stage two (S2) had 6000 jobs. Figure 4.5 shows the changes imposed on the system in the constrained run. Every time a change was submitted to the system, the first step of our two-step approach was repeated and a new virtual slice was generated. Once the new slice was generated, the second step of our approach reevaluated the newly available resources as well as the current workload to create a new resource recipe and a workload placement. Figure 4.6 shows our results.

In Figure 4.6 we can observe the provisioned resources for CBR (top left) and for CCR (top right), which are separated by resource class, and the job throughput for CBR (bottom left) and for CCR (bottom right), which are aggregated by provider zone, over the course of the two scenarios. In these figures, the colored area represents allocated resources, dashed area represents unavailable resources (e.g. due to some constraint or resource unavailability), and empty area represents the available resources that were not allocated. Figure 4.6b (CCR) shows that the provisioned resources in the constrained run successfully adapted to all the changes imposed on the system. For example, IBM Bluemix joined the environment (t=10) after it was added to the global list, AWS East Small left the environment (t=26)

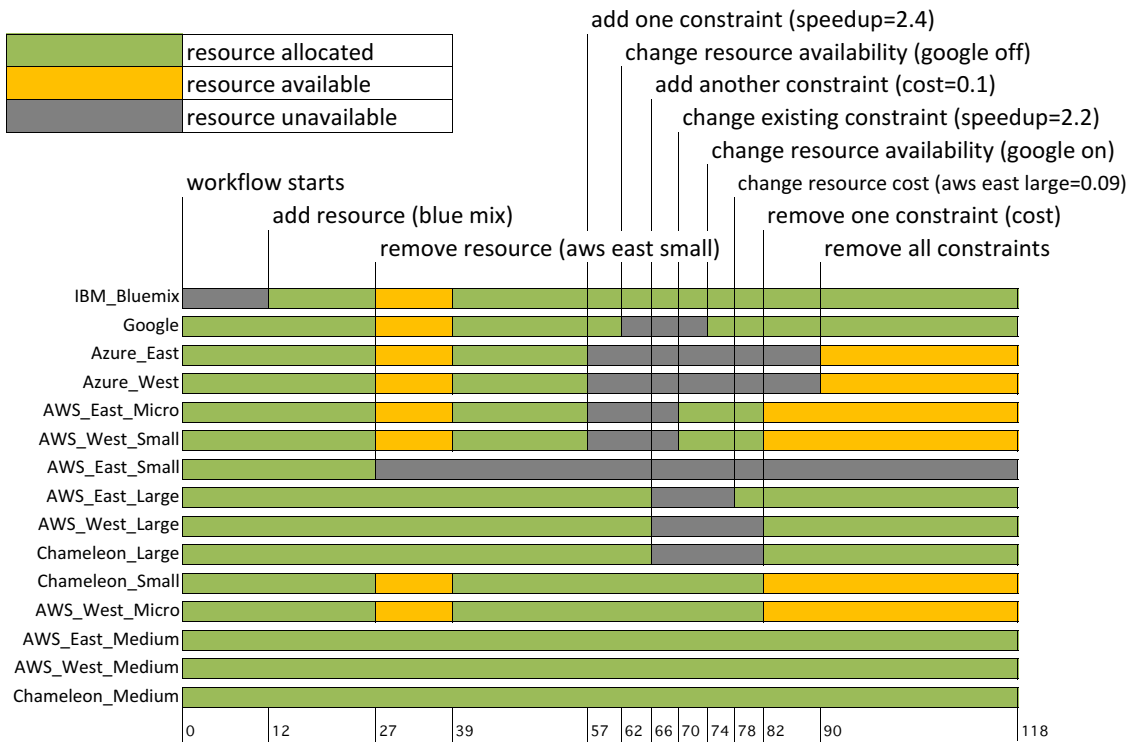
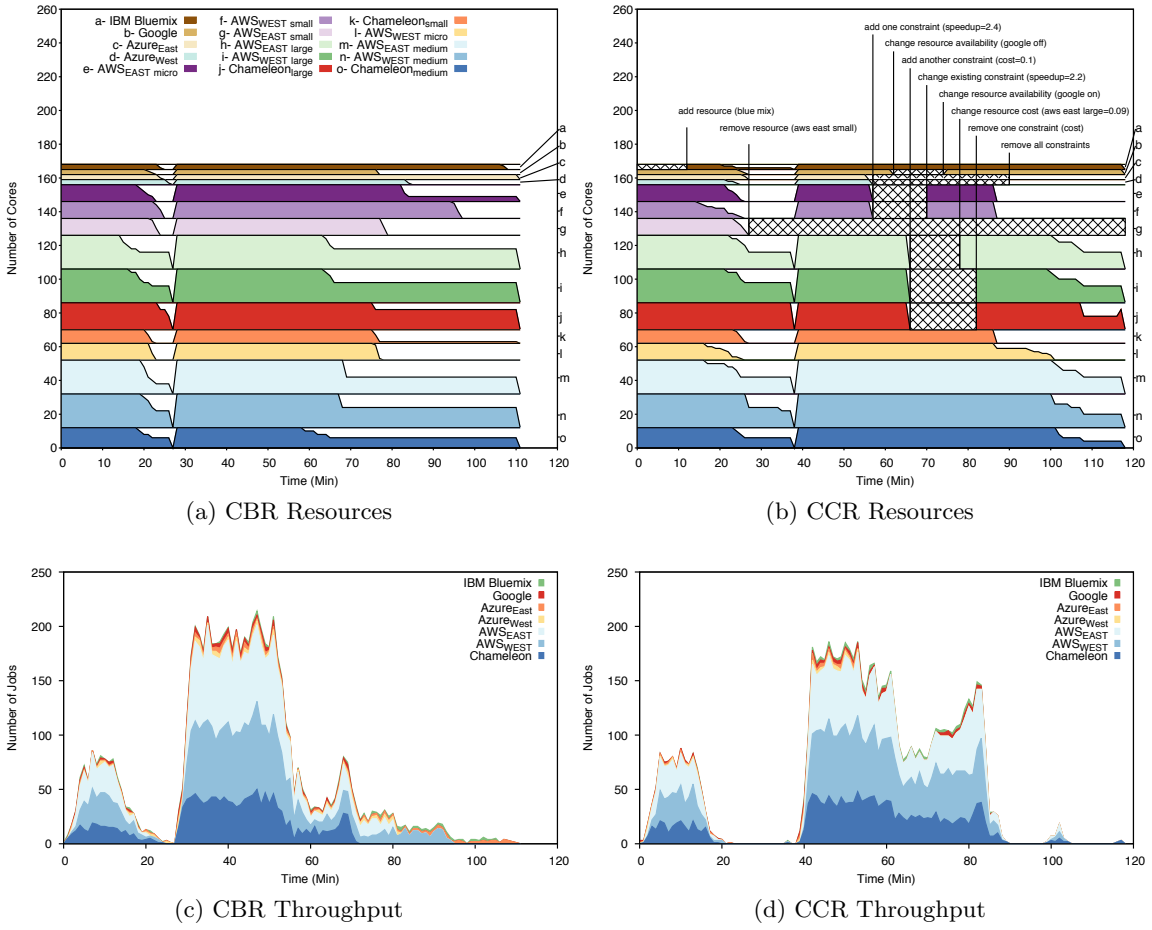


Figure 4.5: Constraint programming evaluation: Experiment 2 – constraints imposed on the system. ©2017 IEEE (reprinted with permission) AbdelBaky et al [5].

after it was removed from the global list, Google was disabled (t=66) when it became unavailable, then enabled (t=72) when it became available again, AWS East Large was disabled when a cost constraint (cost = \$0.1) was imposed on the system, and enabled when its dynamic cost was changed (reduced to \$0.09 from \$0.104). Figures 4.6c and 4.6d show the workload (job throughput) adapting to the available resources. The results show all resources were deallocated after the completion of the first stage and did not start again until the submission of the second stage (t=27 for CBR and t=38 for CCR). The gaps in throughput in Figure 4.6d (between t=23-36, t=90-98 and t=105-115) are due to the failure of some tasks which were reinserted and executed at the end of each stage. The total workflow execution time in CBR was 111 minutes and in CCR was 118 minutes for a total of 6% increase in duration. The total cost for CBR was \$8.93 whereas the total cost for CCR was \$8.68 for a total of 3% decrease. As can be expected, CBR took less time and more cost to execute since all of the resources were available to use. We limited the scheduler in CBR to a one-time placement, where the schedule was generated once at



(a) CBR Resources

(b) CCR Resources

(c) CBR Throughput

(d) CCR Throughput

Figure 4.6: Constraint programming evaluation: Experiment 2 – the provisioned resources and job throughput for the cancer base run (CBR) and cancer constrained run (CCR). The key in Figure 4.6a also applies to Figure 4.6b. The second Y-axis in both figures 4.6a and 4.6b matches the key to facilitate finding resource classes in black and white print. ©2017 IEEE (reprinted with permission) AbdelBaky et al [5].

the beginning of S1 and S2, whereas in CCR, the schedule was continuously reevaluated whenever a change was detected. The long tail in the throughput in S2 in Figure 4.6c was due to the load imbalance resulting from the one-time placement at the beginning of S2. This imbalance was partially reduced in CCR due to the continuous reevaluation of the schedule whenever a change was detected. In fact, the second stage took 84 minutes to execute on CBR and only 80 minutes to execute on CCR. This 5% improvement was achieved despite task failures and fewer resources in CCR. In summary, this experiment shows how our framework was able to dynamically compose resources during runtime according to the resource class properties, availabilities, and constraints. More importantly, it was able to

adapt the workload placement according to the available resources in the virtual slice and continue executing the application. Finally, the ability to react and adapt to changes in the underlying infrastructure coupled with the continuous evaluation of the scheduling can reduce cost and/or the time-to-solution.

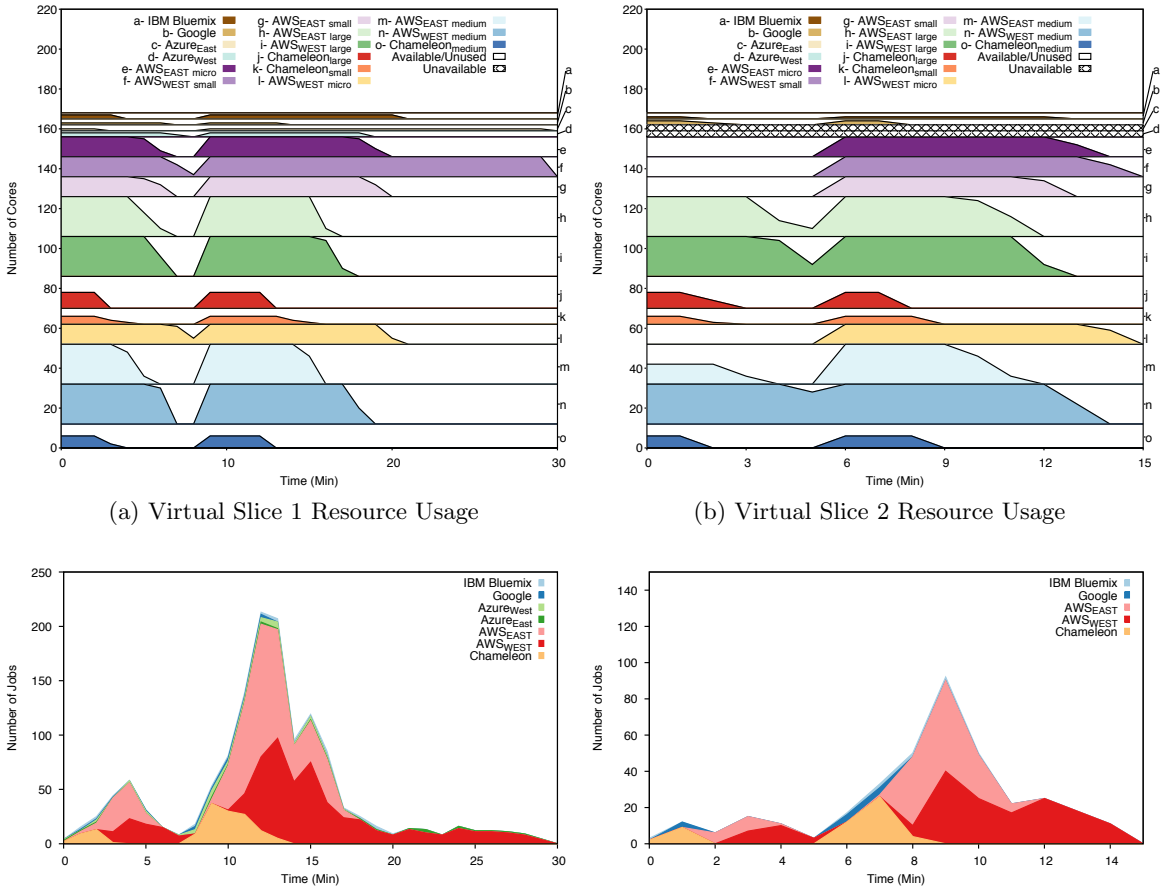
4.4.4.3 Experiment 3: Multiple Slices - Different Workloads

In this experiment, we created two different virtual slices from the same underlying resources that were used to run two independent workloads. Each slice was different and consequently, the subset of resources that each workload was able to see was different. The first slice had no constraints and therefore was able to use any of the resource classes in the global list. The workload running on the first slice was composed of two stages, Stage one (S1) had 200 jobs, Stage two (S2) had 1200 jobs. The second slice had a constraint to only use resources with a minimum speedup threshold of 2.2. The workload running on the second slice was composed of two stages, Stage one (S1) had 50 jobs, Stage two (S2) had 300 jobs. The results of this experiment are shown in Figure 4.7.

Figures 4.7a and 4.7b show the provisioned resources for virtual slices one and two, respectively. The graphs show that different blend of resources was provisioned. For example, the second virtual slice did not include any resource classes from Microsoft Azure since their speedup was lower than the required threshold. On the other hand, Figures 4.7c and 4.7d show the job throughput, which is aggregated by provider zone, over the course of the experiment for virtual slice one and two, respectively. The graphs show the workload (job throughput) adapted to the available resources in both slices. This experiment demonstrates that our framework can support multiple users or projects at the same time (using different slices) and it can adapt to the workload of each slice independently.

4.4.4.4 Experiment 4: Single Slice - Same Workload - Static Resources and No Constraints

In this experiment, there were no constraints or changes in the underlying infrastructure properties or availabilities. We used a workload that consists of two stages, Stage one (S1) had 250 jobs, Stage two (S2) had 1500 jobs. This experiment was executed as a base run



(a) Virtual Slice 1 Resource Usage

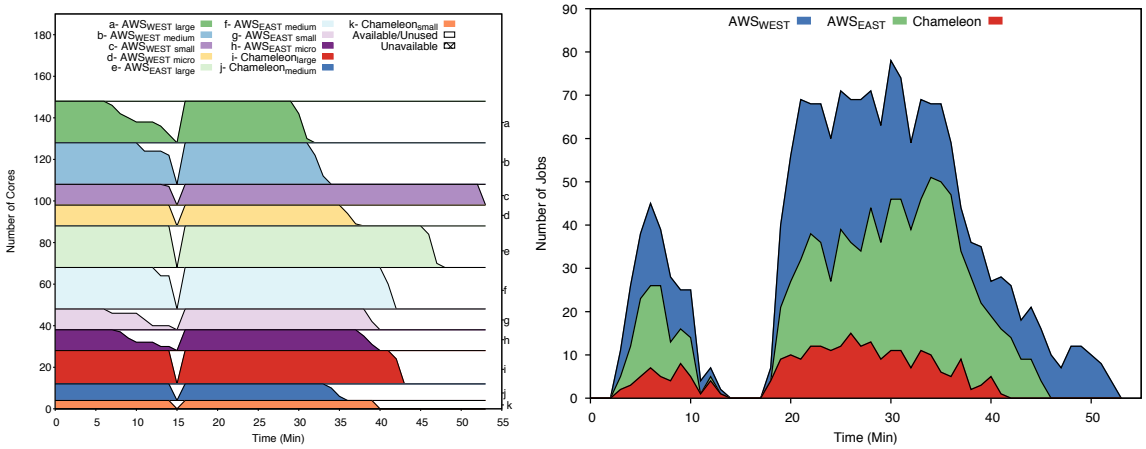
(b) Virtual Slice 2 Resource Usage

(c) Virtual Slice 1 Throughput

(d) Virtual Slice 2 Throughput

Figure 4.7: Constraint programming evaluation: Experiment 3 – The provisioned resources and task throughput for each virtual slice. In this experiment, two independent workloads each composed of two stages with varying workloads were executed on two slices created from the same underlying set of resources. ©2017 IEEE (reprinted with permission) AbdelBaky et al [5].

for experiment 5. The user was able to execute their workload across all resources available in the environment, see Table 4.3a. The results of this experiment are shown in Figure 4.8. We can observe in Figure 4.8a how the system uses as many resources as are available across all infrastructure, releasing them when they are not needed anymore. Similarly, in Figure 4.8b we can observe the throughput of the experiment over time. As we can see it follows the resource allocation shown in Figure 4.8a. The throughput is defined as the number of completed tasks per minute.



(a) Resource Usage (b) Throughput

Figure 4.8: Constraint programming evaluation: Experiment 4 – The provisioned resources and task throughput for experiment 4. In this experiment, a workload composed of two stages was executed on a virtual slice without constraints, hence using all available resources in the global list. ©2017 SAGE Journals (reprinted with permission) AbdelBaky et al [2].

4.4.4.5 Experiment 5: Multiple Slices - Same Workload - Varying Resources and Constraints

In this experiment, we use the same workload as in Experiment 4, i.e., a workflow with two stages, where Stage one (S1) had 250 jobs, Stage two (S2) had 1500 jobs. In this case, we consider two users that are going to use the infrastructure at the same time to execute their workflows. Each user defines a different set of constraints, which results in two different virtual slices from the same pool of underlying resources. Each slice was subject to different constraints and consequently, the resource available in each slice changed over time (independent of one another). The constraints defined for each slice are presented in Table 4.4. Additionally, during execution, resource availabilities and properties were changed to show how the system adapts both slices according to these changes. The results of this experiment are shown in Figure 4.9.

Figures 4.9a, and 4.9b show the provisioned resources for virtual slices one and two, respectively. In these figures, the colored area represents allocated resources, dashed area represents unavailable resources (e.g. due to some constraint or resource unavailability), and empty area represents the available resources that were not allocated. The graphs show that

Table 4.4: Constraint programming evaluation: Constraints for each slice in Experiment 5. ©2017 SAGE Journals (reprinted with permission) AbdelBaky et al [2].

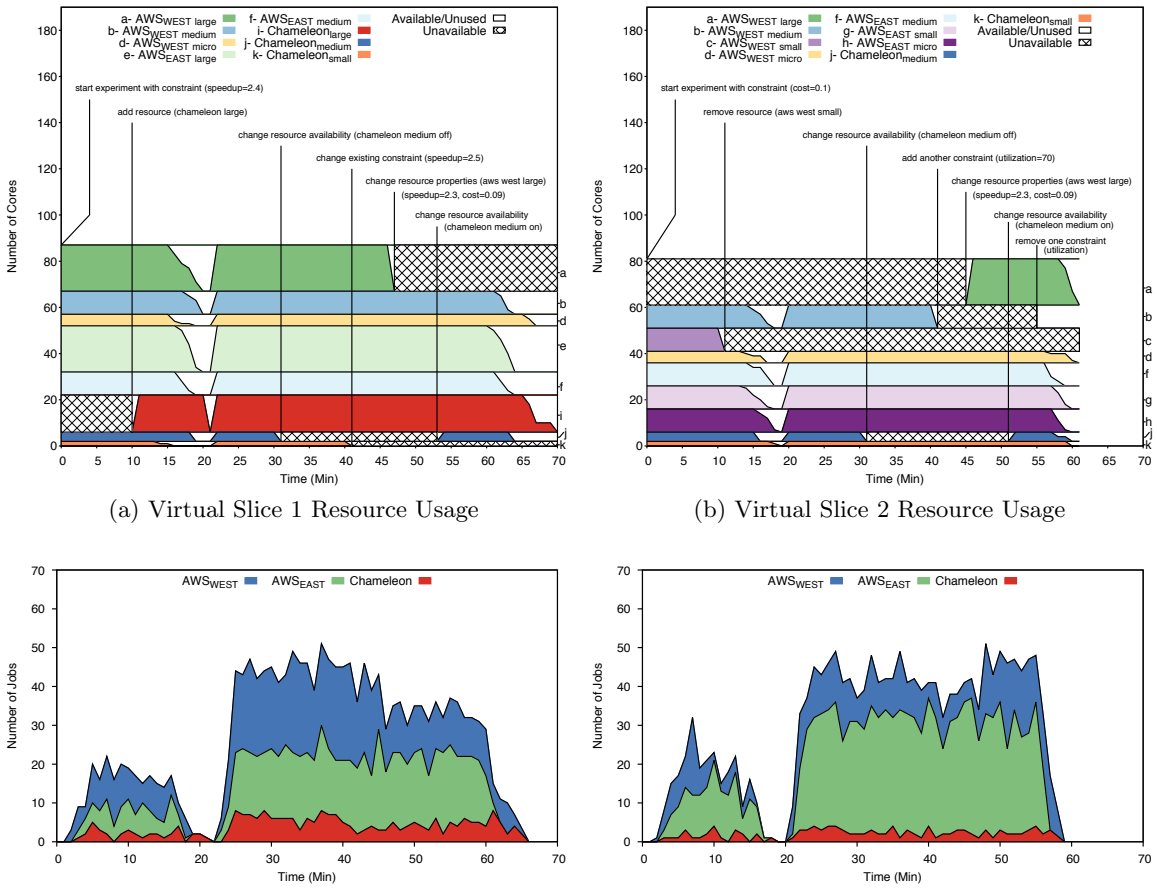
(a) Virtual Slice 1

Time (min)	Constraints
0	Start experiment with constraint (speedup $\geq$ 2.4)
10	Add resource class (chameleon large)
30	Change resource availability (chameleon medium off)
40	Change existing constraint (speedup $\geq$ 2.5)
45	Change resource properties (AWS west large:speedup=2.3, cost=0.09)
50	Change resource availability (chameleon medium on)

(b) Virtual Slice 2

Time (min)	Constraints
0	Start experiment with constraint (cost $\leq$ 0.1)
10	Remove resource (AWS west small)
30	Change resource availability (chameleon medium off)
40	Add another constraint (utilization $\leq$ 70)
45	Change resource properties (AWS west large:speedup=2.3, cost=0.09)
50	Change resource availability (chameleon medium on)
55	Remove one constraint (utilization)

different blends of resources were provisioned upon availability. For example, the first virtual slice did not include any resource classes from AWS_West_small, AWS_East_small, or AWS_East_micro since their performance was lower than the required threshold. The results also show that around minute 30 of the experiment, Chameleon_medium became unavailable, hence both slices were affected and the AS rescheduled their workloads to adapt to that event. Similarly, around minute 45 of the experiment, the properties of AWS_West_large changed (cost and speed up) and two things happened: a) as its performance went down, it became a nonviable option for slice one and hence it was removed from slice one; and b) as its cost went down, it became a viable option for slice two and thus it was added to slice two at the same time. Finally, Chameleon_medium instances joined back both slices when it became available again around minute 50. Figures 4.9c and 4.9d show the aggregated job throughput per site over the course of the experiment for virtual slices one and two,



(a) Virtual Slice 1 Resource Usage

(b) Virtual Slice 2 Resource Usage

(c) Virtual Slice 1 Throughput

(d) Virtual Slice 2 Throughput

Figure 4.9: Constraint programming evaluation: Experiment 5 – The provisioned resources and task throughput for each virtual slice. In this experiment, two identical workloads each composed of two stages were executed concurrently using two different views of the same underlying set of resources using different constraints (virtual slices 1 and 2). ©2017 SAGE Journals (reprinted with permission) AbdelBaky et al [2].

respectively. The graphs show the workload (job throughput) adapted to the available resources in both slices. We can observe that due to the constraints and events that occurred during this experiment, the overall workload execution time was between 58 and 65 minutes versus the 52 minutes that took in the experiment presented in Experiment 4, where we did not have any constraints or events. This experiment demonstrates that our framework can create different slices from the same underlying resources and control them independently.

4.5 Summary

In this chapter, we presented the evaluation of the distributed software-defined environment framework, which leverages a Rule Engine or Constraint Programming (CP) for resource programming and description, and implements a two-step approach to space-time infrastructure service composition. The framework provides near real-time adjustments to the environment in response to changing stimuli (e.g., changes in resource properties or availabilities, or the rules or constraints regulating their use). The resulting amorphous ecosystem is transparently exposed as a service to running applications. The ability to react and adapt to changes in the underlying infrastructure while the application is running can reduce cost and/or the time-to-solution for workloads. We evaluated the rule-engine–based approach using a synthetic bag of task applications and showed how the framework was able to combine resources from multiple distributed sites based on user-driven rules. We used simulations to evaluate the performance and correctness of the CP-based resource filtering approach and compared it to a simple heuristic algorithm. We also evaluated the constraint-programming–based approach using a cancer informatics workflow simulating a Gleason-grading system for prostate pathology image analysis, which was packaged inside Docker containers. We demonstrated the operation of our framework using resources from five different cloud providers that are composed based on user and resource provider constraints. Further, we showed how our framework allows users to programmatically and dynamically define multiple virtual slices of the same underlying resources, and adapt each slice over time without interrupting application execution. Our results show that the framework can adjust the environment based on application behavior (experiments 1 and 4), adapt the application based on the available resources (experiment 2), and support multiple, separately evolving, slices over the same set of resources to support different users and applications (experiments 3 and 5). The result is a fully customizable and synthesized environment that can automatically provision resources as it sees fit.

Finally, we demonstrated how our dSDE framework can enable the deployment and orchestration of containers across a heterogeneous and geographically distributed infrastructure. The agile nature of container workloads coupled with their requirement for fast deployment and adaption to changing environments made them a good candidate to test

the usability and adaptability of our framework. Utilizing Docker did not only demonstrate the flexibility and adaptability characteristics of our federation model, but it also posed a breakthrough [9] towards generalizing the use of Linux containers, and Docker in particular, in multi-cloud environments.

4.6 Relevant Publications

This chapter contains portions adapted from the following published papers with permissions from the copyright holder.

1. M. AbdelBaky, J. Diaz-Montes, M. Zou, and M. Parashar. A framework for realizing software-defined federations for scientific workflows. In Proceedings of the 2nd International Workshop on Software-Defined Ecosystems, pages 7-14. ACM, 2015.

2. M. AbdelBaky, J. Diaz-Montes, M. Unuvar, M. Romanus, M. Steinder, and M. Parashar. Enabling distributed software-defined environments using dynamic infrastructure service composition. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2017 – to appear.

3. M. AbdelBaky, J. Diaz-Montes, and M. Parashar. Software-defined environments for science and engineering. International Journal of High Performance Computing Applications, 2017 – to appear.

[9] Docker containers across clouds in Fortune http://fortune.com/2015/08/27/ibm-deploys-containers-across-clouds/

Chapter 5

General Model and Quality of Service Quantification

5.1 Introduction

In order to effectively support application deployment in a dSDE, we need models and tools that can evaluate application performance and Quality of Service of different compositions (from the same pool of distributed heterogeneous services). Previous work focused on modeling embarrassingly parallel applications with no data dependencies or data movement requirements on distributed heterogeneous resources [70, 94, 110, 137, 145, 170]. However, since a dSDE exhibits variation in space and time and emerging workflows are becoming more dynamic and data-driven, the existing models do not apply anymore.

The goal of this work is to generalize the approach presented in Chapter 3 to include other services besides compute (e.g., data storage, network, etc.). Moreover, we aim to quantify the QoS of a dSDE as well as evaluate the impact of alternative compositions on the application performance. This work aims to answer questions such as:

1. What is the expected SLA of the entire ecosystem based on the combined SLAs of different services. For example, what are the minimum/maximum overall expected application throughput or what is the estimated budget or deadline to run a certain workflow given the current composition of services

2. What are the tradeoffs between different scheduling objectives (e.g., min completion time, min cost, min data transfer). For example, minimizing cost might reduce throughput whereas minimizing completion time might increase data transfers or increase cost.

3. Can we identify bottlenecks in the current composition. For example, do we need more resources or more network bandwidth to increase the application throughput.

4. What to expect from the system if a QoS objective changes. For example, if we reduce the budget, how would system adapt and how would the application performance change.

Generating these metrics and tradeoffs for different scheduling techniques can assist users with decisions in selecting the right composition. In the remainder of this chapter, we present a general model for applications, workflows, and distributed infrastructure services that include compute, data sources, storage, and network. We also present a model that can be used to evaluate a dSDE QoS and application workflow performance on top of it. Finally, we conclude this chapter by evaluating the estimated application workflow throughput and comparing it to the results from the experimental evaluation in Chapter 4.

5.2 General Model

5.2.1 Application Model

We model an application using a directed weighted graph (see Figure 5.1), where each node represents a single **task** and the edges represent the **data flow** among tasks. The flow implicitly specifies the order of execution of these tasks. A task t_i represents the smallest granularity of work for which a resource can be allocated. An application can be represented using a set of n tasks $\{t_1, t_2, \ldots, t_n\}$. Furthermore, for each task t_i, we characterize it using a minimum execution time $T_{exec}(t_i)_{min}$ and a maximum execution time $T_{exec}(t_i)_{max}$. Additionally, we also characterize each task by an input data size $D_{in}(t_i)$ and an output data size $D_{out}(t_i)$, where $i = \{1, 2, \ldots, n\}$. Finally, the graph contains o edges $\{e_1, e_2, \ldots, e_o\}$ where the weight of each edge e_j represents a value that we define as the **data weight** $D_{weight}(e_j)$; which is the size of the input/output data between two connected tasks $D_{in/out}(e_j)$ relative to the total sum of input/output data sizes.

$$D_{weight}(e_j) = \frac{D_{in/out}(e_j)}{\sum_{l=1}^{o} D_{in/out}(e_l)},\ j = \{1, 2, \ldots, o\} \tag{5.1}$$

$$\text{where } \sum_{j=1}^{o} D_{weight}(e_j) = 1$$

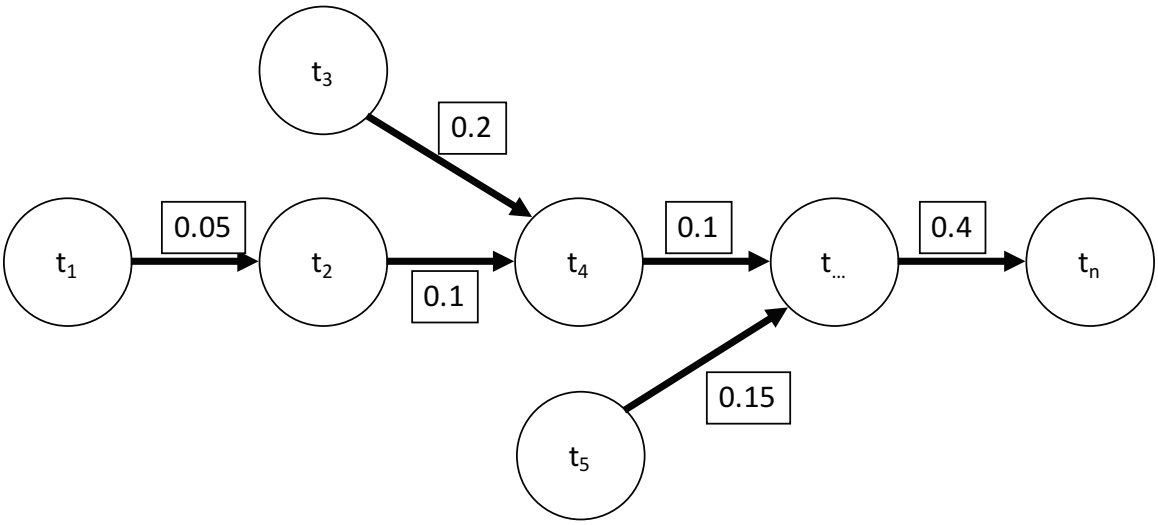


Figure 5.1: An example of a directed weighted graph that represents an application. Nodes represent tasks and edges represent data flow. The weight of an edge represents the input/output data weight.

5.2.2 Workflow Model

A workflow can be represented using a set of m stages $\{St_1, St_2 \ldots, St_m\}$, where each stage St_k is represented by a set of n_k tasks $\{St_k(t_1), St_k(t_2) \ldots, St_k(t_{n_k})\}$ (see Figures 5.2 and 5.3). Similarly, a workflow task $St_k(t_i)$ is characterized by a minimum execution time $T_{exec}(St_k(t_i))_{min}$, a maximum execution time $T_{exec}(St_k(t_i))_{max}$, an input data size $D_{in}(St_k(t_i))$, and an output data size $D_{out}(St_k(t_i))$, where $k = \{1, 2, \ldots, m\}$ and $i = \{1, 2, \ldots, n_k\}$. The data weight is the same as in equation (5.1)

5.2.3 Infrastructure Model

In our model, we extend the definition for a distributed software-defined environment from Section 3.2.2.2 to include other services besides compute (see Figure 5.4). In particular, we define a dSDE as a set of x sites $\{S_1, S_2, \ldots, S_x\}$. Each *site* is defined as a collection of services in a single physical location or region (e.g., data center, cloud provider, or a single zone within a cloud). We consider that any given site S_i is composed of a set of y_i resource classes, where a resource class is defined as the aggregation of resources within such site that share the same properties (e.g., Amazon EC2 m4.large instances, or a homogeneous cluster). Table 5.1 presents resource class properties that we defined in our model. For

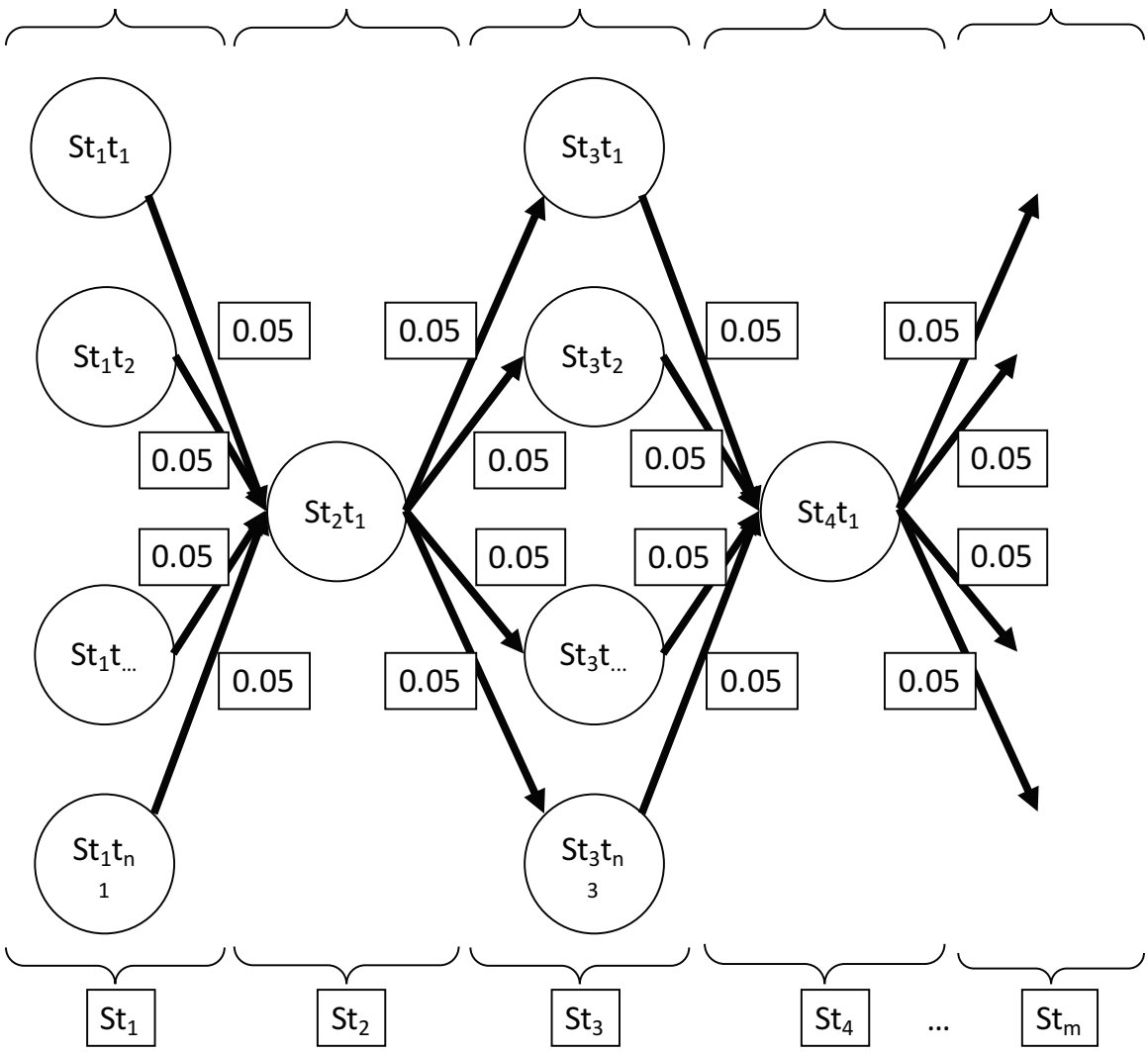


Figure 5.2: An example of an ensemble application workflow, which is represented as a directed weighted graph.

simplicity, the performance of a resource class is represented by the speedup, which can be experimentally calculated as a function of the performance of a reference resource class using various methods from application profiling [54] to benchmarking [44]. Moreover, we also assume that inter-site and intra-site data transfer rates and costs are the same for each resource class. While this assumption is not accurate (i.e., intra-site rates and costs are usually a lot higher than inter-site ones), the procedure for modeling both types of data transfer rates and costs is the same, and thus was omitted for simplicity.

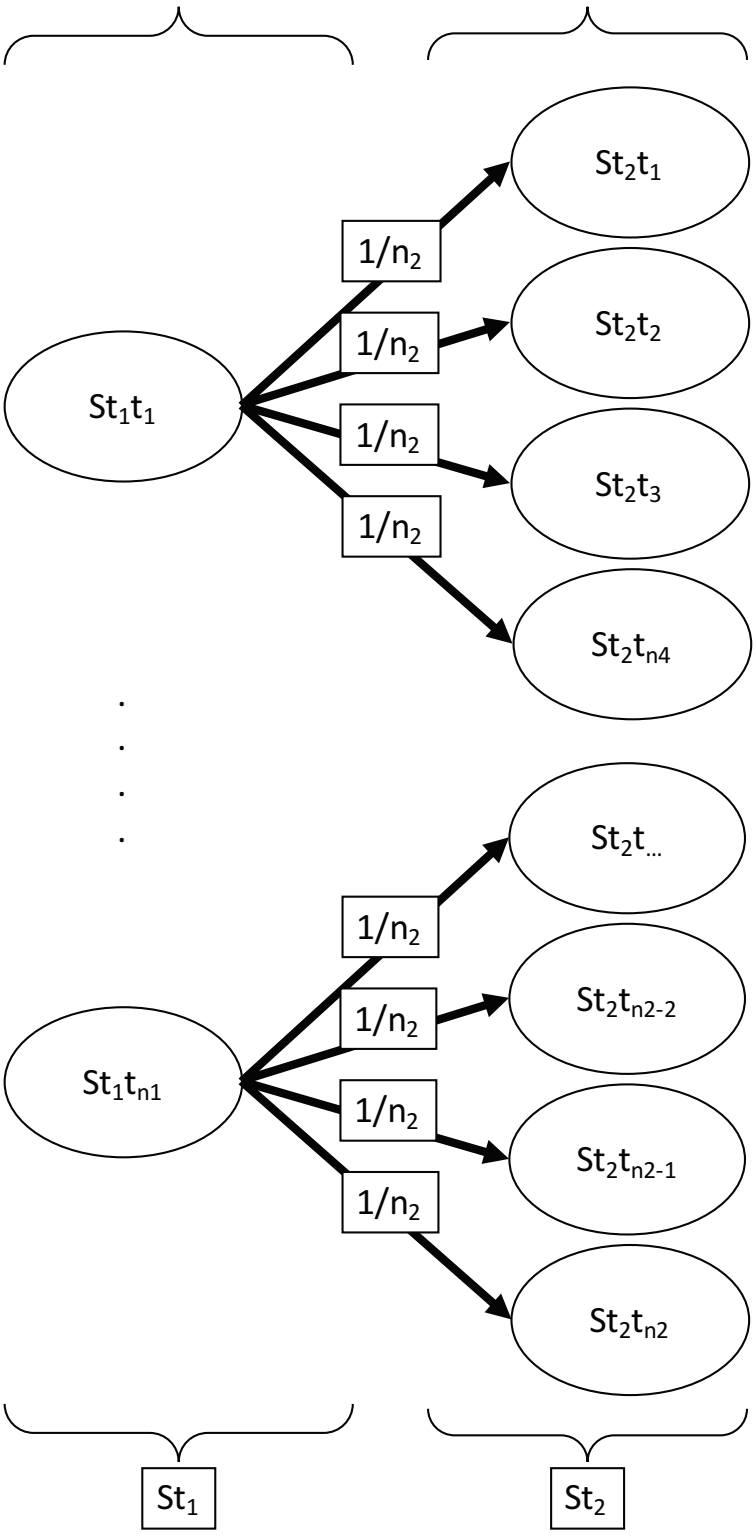


Figure 5.3: Another example based on a bioinformatics workflow [157], where each task in Stage 1 generates multiple tasks in Stage 2.

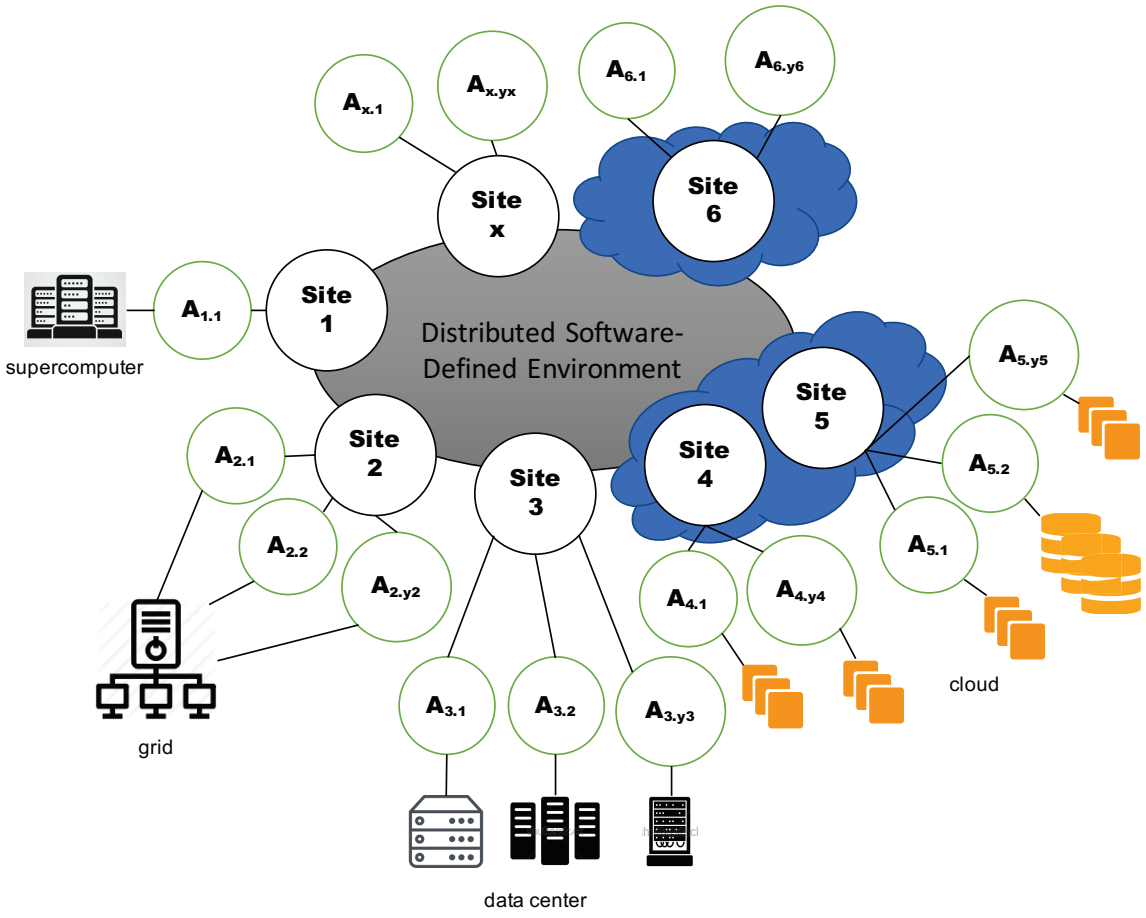


Figure 5.4: An example of a distributed software-defined environment based on the CometCloud framework [2,5,51,92]. Each site contains a collection of resource classes, where each class represents a set of resources (e.g., compute or storage) with the same properties. Each resource class is controlled using an Agent (A). Lines represent network services.

Table 5.1: Resource Class Properties for a generalized distributed Software-Defined Environment.

(a) Compute resource properties

Property	**Description**
Capacity (cp)	Number of instances (e.g. nodes, VMs) in a resource class
Compute cost (cc)	Price per unit time for an instance of a resource class. We assume the cost per instance includes both CPU and memory
Allocation (al)	Number of compute units consumed per unit time for a shared resource class
Performance (pf)	Average performance of an instance of a resource class
Overhead (o)	Time required to allocate an instance of a resource class
Security (sc)	Whether a resource class is secure or not

(b) Data storage properties

Property	**Description**
Data capacity (dcp)	Maximum capacity in unit data size (e.g., GB) for a data storage resource class
Data cost (dc)	Price per unit data size (e.g., GB) for storing data in a data storage resource class
Security (dsc)	Whether a data storage resource class is secure or not

(c) Network resource properties

Property	**Description**
Network bandwidth (np)	Maximum data link transfer capacity in unit data size (e.g., GB) per unit time (e.g., second) for a network resource class

(d) Common properties

Property	**Description**
In-data transfer cost (dic)	The cost of incoming data transfer for a resource class
Out-data transfer cost (doc)	The cost of outgoing data transfer for a resource class
In-data transfer rate (ditr)	The speed of incoming data transfer for a resource class
Out-data transfer rate (dotr)	The speed of outgoing data transfer for a resource class

5.3 QoS Quantification

5.3.1 QoS Metrics

In our model, we define two types of QoS metrics: (1) application and scheduling independent metrics, which represent some qualities of the dSDE that do not depend on any application characteristics or scheduling techniques and (2) application and scheduling dependent metrics, which depend on the current dSDE composition (i.e., current available services in a virtual slice), the application workflow characteristics and workloads, the QoS objectives, and the scheduling techniques. Some of these metrics represent the expected peak performance, i.e., the upper bound, while others represent the minimum expected performance, i.e., the lower bound, of a given composition.

1. *Availability:* measures the degree to which a resource class is accessible and operational.

2. *Security:* measures whether a resource class is secure or not.

3. *Reliability and failure rate:* measure the ability of a resource class to keep operating with a specified level of performance over time.

4. *Total compute cost:* measures the total cost of using all available compute services per unit time.

5. *Average compute cost per instance:* measures the average cost per compute service per unit time.

6. *Total allocation:* measures the allocation cost of using all available allocation-based resources per unit time.

7. *Average allocation per instance:* measures the average allocation cost per allocation-based resource per unit time.

8. *Total performance:* measures the speedup of using all available resources. The speedup is calculated by running benchmarks on all available resources and using a single resource class as a reference point.

9. *Average performance up per instance:* measures the average speedup of a resource.

10. *Workflow time-to-completion:* measures the time to finish executing a workflow using the current dSDE.

11. *Total workflow cost:* measures the total cost of executing a workflow using the current dSDE.

12. *Average workflow cost:* measures the average cost per unit time for executing a workflow using the current dSDE.

13. *Throughput:* measures the average task throughput per unit time using the current dSDE.

5.3.2 QoS Objectives

Finally, we define some QoS objectives that can be used to drive the application scheduling.

1. *MinRunningTime:* aims to maximize application throughput by allocating jobs to resources with the minimum estimated time of completion.

2. *MinCost:* aims to minimize cost by allocating jobs to resources with the cheapest cost.

3. *BudgetConstraint:* finds a solution that minimizes the total execution time (i.e., critical path) while keeping cost within a given budget.

4. *DeadlineCost:* aims to complete the application execution within a given deadline while minimizing cost.

5. *DeadlineLocalityAware:* aims to complete the application execution within a given deadline while minimizing data transfer.

5.3.3 QoS Model

In this section, we utilize the application workflow model defined in Section 5.2.2, the infrastructure model defined in Section 5.2.3, and the $MinRunningTime$ QoS scheduling objective to model the previously defined metrics. Given a dSDE composition with x sites, where each site S_i contains y_i resource classes and $i = \{1, 2, \ldots, x\}$, we define the following QoS model.

Availability (AV). The availability of a single resource class can be obtained from a resource provider's SLA or calculated as follows. Let av_{ij} represent the i^{th} site's j^{th} resource class's availability.

$$av_{ij} = \frac{\text{runtime}}{\text{runtime+downtime}}, \text{ where } i = \{1, 2, \ldots, x\} \text{ and } j = \{1, 2, \ldots, y_i\}$$

Given the individual availability of a resource class, the following metrics can be defined for the dSDE.

The availability of at least one resource class is:

$$\max\{av_{ij}\}, \text{ where } i = \{1, 2, \ldots, x\} \text{ and } j = \{1, 2, \ldots, y_i\}$$

The probability that all resource classes in a dSDE are available is:

$$\Pi_{i=1}^{x} \Pi_{j=1}^{y_i} av_{ij}$$

The probability that none of the resource classes in a dSDE are available is:

$$\Pi_{i=1}^{x} \Pi_{j=1}^{y_i} (1 - av_{ij})$$

The availability of the dSDE (AV) is:

$$AV = 1 - (\Pi_{i=1}^{x} \Pi_{j=1}^{y_i} (1 - av_{ij})) \tag{5.2}$$

Security (SC). The security of a single resource class can be defined as follows. Let sc_{ij} represent the i^{th} site's j^{th} resource class's security level. $sc_{ij} = 1$ if the resource class is secure, otherwise $sc_{ij} = 0$. Consequently, the overall security level of the dSDE (SC) can be defined as follows.

$$SC = \frac{\sum_{i=1}^{x} \sum_{j=1}^{y_i} sc_{ij}}{\sum_{i=1}^{x} y_i} \tag{5.3}$$

Reliability (R) and Failure Rate (FR). Let r_{ij} and fr_{ij} represent the i^{th} site's j^{th} resource class's reliability and failure rate respectively.

$$r_{ij} = \frac{\text{number of executed tasks}}{\text{number of assigned tasks}}$$

$$fr_{ij} = \frac{\text{number of failed tasks}}{\text{time period}}$$

Consequently, the overall dSDE reliability (R) and failure rate (FR) can be defined as follows.

$$R = \frac{\text{total number of executed tasks}}{\text{total number of assigned tasks}} \tag{5.4}$$

$$FR = \sum_{i=1}^{x} \sum_{j=1}^{y_i} fr_{ij} \tag{5.5}$$

Total Compute Cost (TCC). Let cc_{ij} and cp_{ij} represent the i^{th} site's j^{th} resource class's compute cost per unit time and capacity respectively. Then, the total cost per unit time of the dSDE (TCC) is:

$$TCC = \sum_{i=1}^{x} \sum_{j=1}^{y_i} (cc_{ij}.cp_{ij}) \tag{5.6}$$

Average Compute Cost (ACC). The average compute cost per instance per unit time (ACC) is:

$$ACC = \frac{\sum_{i=1}^{x} \sum_{j=1}^{y_i} (cc_{ij}.cp_{ij})}{\sum_{i=1}^{x} \sum_{j=1}^{y_i} cp_{ij}} \tag{5.7}$$

Total Allocation (TA). Let al_{ij} represent the i^{th} site's j^{th} resource class's allocation cost per unit time. Then, the total allocation per unit time of the dSDE (TA) is:

$$TA = \sum_{i=1}^{x} \sum_{j=1}^{y_i} (al_{ij}.cp_{ij}) \tag{5.8}$$

Average Allocation (AA). The average allocation cost per instance per unit time (AA) is:

$$AA = \frac{\sum_{i=1}^{x} \sum_{j=1}^{y_i} (al_{ij}.cp_{ij})}{\sum_{i=1}^{x} \sum_{j=1}^{y_i} cp_{ij}} \tag{5.9}$$

Total Performance (TPF). Let pf_{ij} represent the i^{th} site's j^{th} resource class's performance. Then, the total performance of the dSDE (TPF) is:

$$TPF = \sum_{i=1}^{x} \sum_{j=1}^{y_i} (pf_{ij}.cp_{ij}) \tag{5.10}$$

Average Performance (APF). The average performance per instance (APF) is:

$$APF = \frac{\sum_{i=1}^{x} \sum_{j=1}^{y_i} (al_{ij}.pf_{ij})}{\sum_{i=1}^{x} \sum_{j=1}^{y_i} cp_{ij}} \tag{5.11}$$

Workflow time-to-completion. Given the workflow model defined in Section 5.2.2, the overall workflow time-to-completion ($WFTTC$) is equal to the sum of the time-to-completion ($STTC$) of all stages.

$$WFTTC = \sum_{k=1}^{m} STTC(St_k) \tag{5.12}$$

For each stage, the scheduler generates the resource recipe containing the exact number/types of services to be provisioned within each site and the task placement within each resource class. The scheduler generates this information based on the current available services (i.e., in the virtual slice), the workflow stage characteristics (e.g., number of tasks and data sizes), and the QoS objective defined for that stage (e.g., MinRunningTime). Let x and y_i represent the total number of sites and the total number of resource classes within each site S_i respectively.

Let cp_{ij}, $ditr_{ij}$, and $dotr_{ij}$ represent the i^{th} site's j^{th} resource class's compute capacity (e.g., number of instances), in-data transfer rate, and out-data transfer rate respectively. Let $n_{k_{ij}}$ represent tasks in the k^{th} stage mapped to i^{th} site's j^{th} resource class. Then, the total time of execution for each stage is bound by the maximum time-to-completion ($RCTTC$) of the slowest resource class.

$$STTC(St_k) = \max\{RCTTC_{ij}\}$$

$$\text{where } k = \{1, 2, \ldots, m\},\ i = \{1, 2, \ldots, x\}, \text{ and } j = \{1, 2, \ldots, y_i\}$$

Further, to calculate the time-to-completion for each resource class ($RCTTC_{ij}$), we first calculate the time-to-completion ($TTTC_{ij}$) for each task ($St_k(t_l)$) in the i^{th} site's j^{th} resource class, which includes the time to transfer data to/from each instance (TD_{in} and TD_{out}), and the minimum or maximum computation time (TC_{min} or TC_{max}).

$$TD_{in}(St_k(t_l)) = \frac{D_{in}(St_k(t_l))}{ditr_{ij}}$$

$$TD_{out}(St_k(t_l)) = \frac{D_{out}(St_k(t_l))}{dotr_{ij}}$$

$$TC(St_k(t_l))_{min} = \frac{T_{exec}(St_k(t_l))_{min}}{pf_{ij}}$$

$$TC(St_k(t_l))_{max} = \frac{T_{exec}(St_k(t_l))_{max}}{pf_{ij}}$$

$$TTTC_{ij}(St_k(t_l))_{min} = TD_{in}(St_k(t_l)) + TD_{out}(St_k(t_l)) + TC(St_k(t_l))_{min}$$

$$TTTC_{ij}(St_k(t_l))_{max} = TD_{in}(St_k(t_l)) + TD_{out}(St_k(t_l)) + TC(St_k(t_l))_{max}$$

$$\text{where } i = \{1, 2, \ldots, x\},\ j = \{1, 2, \ldots, y_i\},\ k = \{1, 2, \ldots, m\}, \text{ and } l = \{1, 2, \ldots, n_{k_{ij}}\}$$

The time-to-completion for each resource class ($RCTTC_{ij}$) includes the overhead (o) for allocating instances of that type and the minimum or maximum time-to-completion for all tasks ($TTTC_{ij}$). Given that some tasks are executed in parallel (based on the capacity of each resource class), then the upper and lower bounds for the overall time-to-completion for each resource class ($RCTTC_{ij}$) can be calculated as follows.

$$\text{Upper bound for } RCTTC_{ij} = o_{ij} + \frac{\max\{TTTC_{ij}(St_k(t_l))_{max}\} * n_{k_{ij}}}{cp_{ij}} \tag{5.13}$$

$$\text{Lower bound for } RCTTC_{ij} = o_{ij} + \frac{\min\{TTTC_{ij}(St_k(t_l))_{min}\} * n_{k_{ij}}}{cp_{ij}} \tag{5.14}$$

where $i = \{1, 2, \ldots, x\}$, $j = \{1, 2, \ldots, y_i\}$, $k = \{1, 2, \ldots, m\}$, and $l = \{1, 2, \ldots, n_{k_{ij}}\}$

Total workflow cost. The total workflow cost ($TWFC$) can be estimated using the time-to-completion for each resource class ($RCTTC_{ij}$).

$$TFWC = \frac{\sum_{k=1}^{m}\sum_{i=1}^{x}\sum_{j=1}^{y_i}(RCTTC_{ij} * cp_{ij} * (cc_{ij}|al_{ij}))}{\text{unit time}} + \frac{\sum_{k=1}^{m}\sum_{l=1}^{n_{k_{ij}}} D_{in}(St_k(t_l))}{dic_{ij}} + \frac{\sum_{k=1}^{m}\sum_{l=1}^{n_{k_{ij}}} D_{out}(St_k(t_l))}{doc_{ij}} \tag{5.15}$$

Average workflow cost. The average workflow cost ($AWFC$) can be calculated as follows.

$$AFWC = \frac{TWFC}{\text{unit time}} \tag{5.16}$$

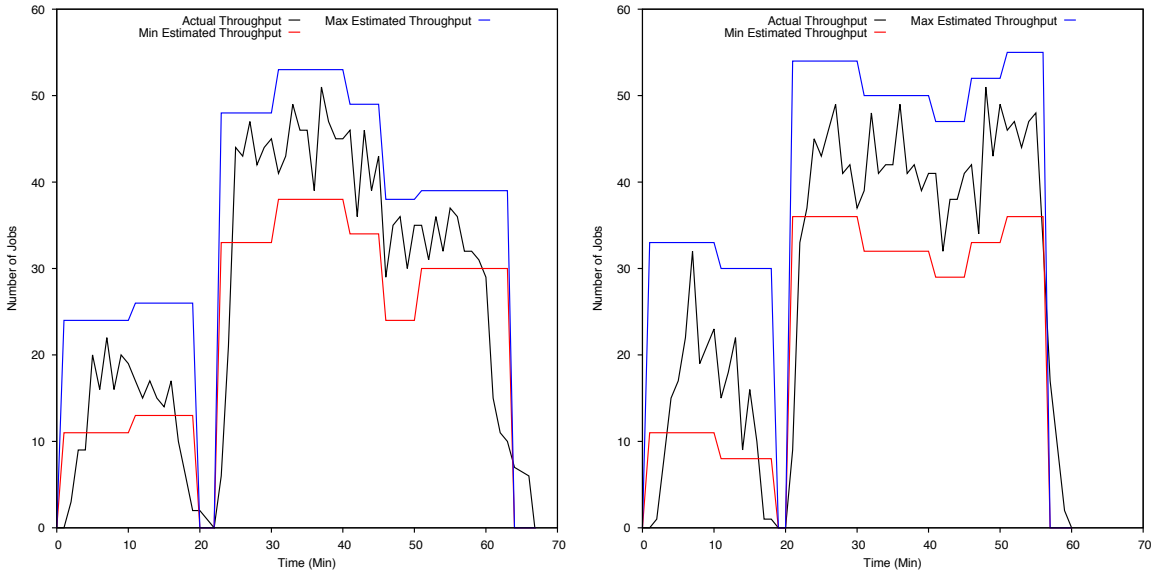
Throughput. The workflow throughput per stage ($WFTS$) and the overall workflow throughput (WFT) can be calculated as follows.

$$WFTS_k = \frac{n_k}{STTC(St_k)} \text{ where } n_k \text{ is the number of tasks in stage k and } k = \{1, 2, \ldots, m\} \tag{5.17}$$

$$WFT = \frac{n}{WFTTC} \text{ where } n \text{ is the total number of tasks in a workflow} \tag{5.18}$$

5.4 Evaluation and Summary

We evaluated the QoS model by calculating the minimum and maximum estimated throughput using the defined model and compare the results to the actual throughput measured in Section 4.4.4.5. The results are shown in Figure 5.5. The results show that experimental throughput was within the bounds estimated by the QoS model.



(a) Virtual Slice 1 Throughput

(b) Virtual Slice 2 Throughput

Figure 5.5: The evaluation of the QoS model. The red line shows the estimated minimum throughput and the blue line shows the estimated maximum throughput using the defined QoS model. The results are compared to the actual run from Experiment 5 in Section 4.4.4.5 shown in the black line.

In this chapter, we presented a generalized model for a distributed Software-Defined Environment that includes other services such as storage or network. We also presented a QoS quantification model for the dSDE, which can be used to estimate the application performance at any given time based on available services. The model takes in consideration the SLAs of individual services and uses it to estimate the overall SLA of the dSDE. Further, this model can be used to calculate the tradeoffs between different scheduling techniques,

which can generate different compositions from the underlying pool of services based on multiple factors (e.g., current resource availabilities, application workload, QoS objectives, etc.). The evaluation of these tradeoffs can then facilitate user's decision when considering application deployment.

Chapter 6

Conclusion

6.1 Summary

Service-based access models can support emerging dynamic and data-driven applications and workflows. However, this requires rethinking traditional federation models to support dynamic (and opportunistic) service compositions, which can adapt to evolving application needs and the state of the underlying infrastructure. In this dissertation, we presented a programmable dynamic service composition approach that uses software-defined environment concepts and implements a two-step approach to space-time service composition. This approach takes into consideration the interface, access, and usage rights of different resources, and provides an ad-hoc federation model. The resulting distributed software-defined environment autonomously evolves in near real-time over the application lifecycle in response to changing stimuli (e.g., changes in resource properties or availabilities, or the rules/constraints regulating their use). The framework automates this process, which is transparent to the applications running on the dSDE, i.e., without any loss of progress or a need to restart the application.

Moreover, we presented and compared two different approaches for programming resources and controlling the environment, one that is based on a rule engine and another that leverages constraint programming. We compared the tradeoffs of both approaches and showed how they can be used to create multiple views of heterogeneous and geographically distributed services. These views contain dynamically aggregated services, which evolves based on user and resource provider specifications. We also presented the design and implementation of the dSDE and demonstrated its operation using simulations and real experiments running a representative scientific workflow across multiple distributed infrastructure. The workflow was encapsulated using Docker containers to facilitate deployment

in a heterogeneous environment.

Our results show that the framework can adjust the composed services based on application behavior and available resources. Further, the ability to react and adapt to changes in the underlying infrastructure while the application is running can reduce cost and/or the time-to-solution for workloads. Finally, we presented a general mathematical model that includes additional services such as storage and network, which can be used to assist users' decision. Using this model, we provided upper and lower bounds on the expected SLA and QoS of the dSDE and the tradeoffs between different scheduling techniques.

6.2 Broader Impact

A distributed software-defined environment can efficiently serve the needs of dynamic and data-driven applications ranging from large-scale science and engineering to pervasive computing over distributed data sources. We foresee that a distributed software-defined environment can improve resource management and usage by simplifying how services (compute, data, and networking) are federated and customized. For example, a dSDE can provide the large scale required for emerging science and engineering applications [1, 7, 158]. It can also support business applications and prevent vendor lock-in by allowing users to cloud burst their applications while taking advantage of different services and price points from multiple cloud providers [4]. A dSDE can also allow users to easily define the distribution of different workloads across multiple resources or allow resource providers to easily control what resources are available to users [2].

More importantly, this work enables the composition of distributed services based on application runtime behavior [3, 5, 6, 50], for example, by composing different types of resources based on different stages of a workflow or based on application results that cannot be determined beforehand. Furthermore, this work can support emerging data-driven applications in CDS&E and IoT [51, 120] by allocating resources along the data path [121] or enabling new scenarios such as (1) follow the sun: a time-based service composition – where resources can be composed following the sun/time of day; (2) follow the user: a location-based service composition – where resources can be dynamically provisioned to remain within a certain proximity of a moving user (e.g., a mobile user); and (3) follow

the data: a data-based service composition – where resources can be provisioned based on proximity to data sources/sinks.

6.3 Future Work

Analogous to how clouds revolutionized the way we acquire and use IT resources, we envision distributed software-defined environments as a game changer in the way federations can be built, especially, if we combine it with other approaches such as software-defined networks to customize and optimize the communication channels. This is especially important for upcoming infrastructure models where non-trivial computational power and ubiquitous data require novel abstractions to enable complex trade-offs, including data movement, performance, energy efficiency, and failures.

While this dissertation provides a framework for the dynamic and on-demand composition of services, more work is required to support emerging data-driven applications. For example, one must find near-optimal solutions for composing services when the behaviors of both the infrastructure and the workloads are highly volatile. In particular, can we dynamically adapt applications in cases where resources are limited? Can we anticipate the application workload or the future state of the underlying ecosystem? Can we use this information to optimize different objectives (e.g., maintain data locality, real-time processing, or minimize cost)? The research presented in this dissertation can be expanded in several areas to answer these questions.

1. **Programming Models.** Existing programming models and systems for parallel and distributed computing must be extended and redesigned to address the dynamic adaptation and uncertainty of the underlying infrastructure. This will allow information-driven applications to detect and adapt to changes in both, the state of the execution environment and the state and requirements of the application. Specifically, abstractions and mechanisms are required to enable the specification (often imprecise or fuzzy) of components adaptations and interactions, and knowledge-driven dynamic compositions, implementations, and deployments. As in any real ecosystem, behaviors and resource usage of all components have to be mutually balanced in order to

obtain a stable and sustainable level of system operation.

2. **Application-driven QoS and Malleability.** We envision domain-driven platforms, where scientists and end-users define their QoS requirements in terms of science or domain-specific metrics. In particular, users can define different configurable parameters in their workflows and the expected results. For example, there may exist some operations that have certain degrees of freedom in a workflow or they may potentially use different methods with a similar outcome. In such cases, a user can define different levels of accepted QoS for the solution, e.g., accuracy, error margins, etc., together with other requirements such as budget or deadline. Consequently, we can enable an organic platform that considers all the variables or "knobs" offered by the application and troes to allocate the workload in the best possible way by initiating a bidirectional negotiation between the workflow management framework and the underlying execution environment. In this way, not only the execution environment can adapt to meet application needs, but applications can also adapt by modifying previously defined "knobs" to facilitate finding a solution that is feasible given the currently available services in the environment.

3. **Machine-Learning–Based Optimization.** We can use Machine Learning (ML) techniques to predict the behaviors of applications and underlying services, which can then be used to optimize the service composition and workload placement. In particular, we can model the applications' behavior (e.g., workload size, data requirements, data locations, data generation/access patterns) and the expected behavior of different services (e.g., properties, availabilities, SLA guarantees). This can be achieved by using historical information (e.g., resource metrics, application traces) to develop statistical Machine Learning models. The ML–based predictions can then be used to dynamically configure the computing fabric available to data-driven applications, for example, by allocating more services, pre-fetching data, etc.

6.4 Relevant Publications

This dissertation contains portions adapted from the following published papers with permissions from the copyright holder.

1. H. Kim, M. AbdelBaky, and M. Parashar. CometPortal: A portal for online risk analytics using CometCloud. In 17th International Conference on Computing Theory and Applications (ICCTA2009), 2009.

2. M. AbdelBaky, M. Parashar, K. Jordan, H. Kim, H. Jamjoom, Z.-Y. Shae, G. Pencheva, V. Sachdeva, J. Sexton, R. Tavakoli, et al. Enabling high-performance computing as a service. Computer, 45(10):72-80, 2012.

3. M. Parashar, M. AbdelBaky, I. Rodero, and A. Devarakonda. Cloud paradigms and practices for computational and data-enabled science and engineering. Computing in Science & Engineering, 15(4):10-18, 2013.

4. J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. Software defined federated cyber-infrastructure for science and engineering. In Proceedings of the 2014 ACM international workshop on Software-defined ecosystems, pages 9-12. ACM, 2014.

5. M. AbdelBaky, J. Diaz-Montes, M. Johnston, V. Sachdeva, R. L. Anderson, K. E. Jordan, and M. Parashar. Exploring HPC-based scientific software as a service using CometCloud. In Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on, pages 35-44. IEEE, 2014.

6. M. AbdelBaky, J. Diaz-Montes, M. Zou, and M. Parashar. A framework for realizing software-defined federations for scientific workflows. In Proceedings of the 2nd International Workshop on Software-Defined Ecosystems, pages 7-14. ACM, 2015.

7. J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. Cometcloud: Enabling software-defined federations for end-to-end application workflows. IEEE Internet Computing, 19(1):69-73, 2015.

8. M. Parashar, M. AbdelBaky, M. Zou, A. R. Zamani, and J. Diaz-Montes. Realizing the

potential of IoT using software-defined ecosystems. In Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on, pages 1149-1158. IEEE, 2015.

9. M. AbdelBaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder. Docker containers across multiple clouds and data centers. In 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pages 368371. IEEE, 2015.

10. J. Wang, M. AbdelBaky, J. Diaz-Montes, S. Purawat, M. Parashar, and I. Altintas. Kepler+CometCloud: dynamic scientific workflow execution on federated cloud resources. Procedia Computer Science, 80:700-711, 2016.

11. M. AbdelBaky, J. Diaz-Montes, M. Unuvar, M. Romanus, M. Steinder, and M. Parashar. Enabling distributed software-defined environments using dynamic infrastructure service composition. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2017 – to appear.

12. M. AbdelBaky, J. Diaz-Montes, and M. Parashar. Towards distributed software-defined environments. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2017 – to appear.

13. M. AbdelBaky, J. Diaz-Montes, and M. Parashar. Software-defined environments for science and engineering. International Journal of High Performance Computing Applications, 2017 – to appear.

14. M. Parashar, M. AbdelBaky, M. Zou, A. R. Zamani, E. Renart, and J. Diaz-Montes. Computing in the continuum: Combining pervasive devices and services to support data-driven applications. In IEEE International Conference on Distributed Computing Systems (ICDCS), 2017 – to appear.

References

[1] M. AbdelBaky, J. Diaz-Montes, M. Johnston, V. Sachdeva, R. L. Anderson, K. E. Jordan, and M. Parashar. Exploring hpc-based scientific software as a service using cometcloud. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*, pages 35–44. IEEE, 2014.

[2] M. AbdelBaky, J. Diaz-Montes, and M. Parashar. Software-defined environments for science and engineering. *International Journal of High Performance Computing Applications*, 2017 – to appear.

[3] M. AbdelBaky, J. Diaz-Montes, and M. Parashar. Towards distributed software-defined environments. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2017 – to appear.

[4] M. AbdelBaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder. Docker containers across multiple clouds and data centers. In *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pages 368–371. IEEE, 2015.

[5] M. AbdelBaky, J. Diaz-Montes, M. Unuvar, M. Romanus, M. Steinder, and M. Parashar. Enabling distributed software-defined environments using dynamic infrastructure service composition. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2017 – to appear.

[6] M. AbdelBaky, J. Diaz-Montes, M. Zou, and M. Parashar. A framework for realizing software-defined federations for scientific workflows. In *Proceedings of the 2nd International Workshop on Software-Defined Ecosystems*, pages 7–14. ACM, 2015.

[7] M. AbdelBaky, M. Parashar, K. Jordan, H. Kim, H. Jamjoom, Z.-Y. Shae, G. Pencheva, V. Sachdeva, J. Sexton, R. Tavakoli, et al. Enabling high-performance computing as a service. *Computer*, 45(10):72–80, 2012.

[8] S. Abrishami, M. Naghibzadeh, and D. H. Epema. Cost-driven scheduling of grid workflows using partial critical paths. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1400–1414, 2012.

[9] S. Abrishami, M. Naghibzadeh, and D. H. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158–169, 2013.

[10] R. J. Al-Ali, K. Amin, G. Von Laszewski, O. F. Rana, D. W. Walker, M. Hategan, and N. Zaluzec. Analysis and provision of qos for distributed grid applications. *Journal of Grid Computing*, 2(2):163–182, 2004.

[11] G. Allen and D. Katz. Computational science, infrastructure and interdisciplinary research on university campuses: Experiences and lessons from the center for computation & technology. Technical report, Technical Report CCT-TR-2010-1, Louisiana State University, 2010.

[12] M. Allison, S. Turner, and A. A. Allen. Towards interpreting models to orchestrate iaas multi-cloud infrastructures. In *Computer Science & Education (ICCSE), 2015 10th International Conference on*, pages 80–85. IEEE, 2015.

[13] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. Web services. In *Web Services*, pages 123–149. Springer, 2004.

[14] I. Altintas, J. Block, R. De Callafon, D. Crawl, C. Cowart, A. Gupta, M. Nguyen, H.-W. Braun, J. Schulze, M. Gollner, et al. Towards an integrated cyberinfrastructure for scalable data-driven monitoring, dynamic prediction and resilience of wildfires. *Procedia Computer Science*, 51:1633–1642, 2015.

[15] A. Amato and S. Venticinque. Multi-objective decision support for brokering of cloud sla. In *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pages 1241–1246. IEEE, 2013.

[16] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

[17] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[18] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[19] W. C. Arnold, D. Arroyo, W. Segmuller, M. Spreitzer, M. Steinder, and A. N. Tantawi. Workload orchestration and optimization for software defined environments. *IBM Journal of Research and Development*, 58(2/3):11–1, 2014.

[20] A. Bala and D. Chana. Article: A survey of various workflow scheduling algorithms in cloud environment. *IJCA Proceedings on 2nd National Conference on Information and Communication Technology*, NCICT(4):26–30, November 2011. Full text available.

[21] I. Baldin, J. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills. Exogeni: A multi-domain infrastructure-as-a-service testbed. In *The GENI Book*, pages 279–315. Springer, 2016.

[22] F. Berman, G. Fox, and A. J. Hey. *Grid computing: making the global infrastructure a reality*, volume 2. John Wiley and sons, 2003.

[23] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[24] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan. Bi-criteria Workflow Tasks Allocation and Scheduling in Cloud Computing Environments. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 638–645, June 2012.

[25] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12, 2013.

[26] T. Bicer, D. Chiu, and G. Agrawal. Time and cost sensitive data-intensive computing on hybrid clouds. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 636–643. IEEE Computer Society, 2012.

[27] S. Bilgaiyan, S. Sagnika, and M. Das. Workflow scheduling in cloud computing environment using Cat Swarm Optimization. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 680–685, Feb. 2014.

[28] L. F. Bittencourt, C. R. Senna, and E. R. Madeira. Enabling execution of service workflows in grid/cloud hybrid systems. In *IEEE/IFIP Network Operations and Management Symp. Wksps.*, pages 343–349, 2010.

[29] N. Bobroff, L. Fong, S. Kalayci, Y. Liu, J. Martinez, I. Rodero, S. Sadjadi, and D. Villegas. Enabling interoperability among meta-schedulers. In *CCGrid*, pages 306–315, 2008.

[30] G. Breiter, M. Behrendt, M. Gupta, et al. Software defined environments based on tosca in ibm cloud implementations. *IBM Journal of Research and Development*, 58(2):1–10, 2014.

[31] R. Buyya, R. Ranjan, and R. N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Intl. Conf. on Algorithms and Architectures for Parallel Processing*, ICA3PP'10, pages 13–31, 2010.

[32] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1069–1075. ACM, 2005.

[33] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, and G. Righetti. Cloud federations in contrail. In *European Conference on Parallel Processing*, pages 159–168. Springer, 2011.

[34] E. Caron, F. Desprez, A. Muresan, and F. Suter. Budget Constrained Resource Allocation for Non-deterministic Workflows on an IaaS Cloud. In Y. Xiang, I. Stojmenovic, B. O. Apduhan, G. Wang, K. Nakano, and A. Zomaya, editors, *Algorithms and Architectures for Parallel Processing*, number 7439 in Lecture Notes in Computer Science, pages 186–201. Springer Berlin Heidelberg, Sept. 2012. DOI: 10.1007/978-3-642-33078-0_14.

[35] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.

[36] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How to enhance cloud architectures to enable cross-federation. In *CLOUD*, pages 337–345, 2010.

[37] S. Chaisiri, B.-S. Lee, and D. Niyato. Optimal virtual machine placement across multiple cloud providers. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 103–110. IEEE, 2009.

[38] S. Chaisiri, B.-S. Lee, and D. Niyato. Optimization of resource provisioning cost in cloud computing. *IEEE Transactions on Services Computing*, 5(2):164–177, 2012.

[39] F. Chang, J. Ren, and R. Viswanathan. Optimal Resource Allocation in Clouds. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 418–425, July 2010.

[40] J. Chase and I. Baldin. A retrospective on orca: Open resource control architecture. In *The GENI Book*, pages 127–147. Springer, 2016.

[41] J. Chase, L. Grit, D. Irwin, V. Marupadi, P. Shivam, and A. Yumerefendi. Beyond virtual data centers: Toward an open resource control architecture. In *Selected Papers from the International Conference on the Virtual Computing Initiative (ACM Digital Library)*, 2007.

[42] W.-K. Cheng, B.-Y. Ooi, and H.-Y. Chan. Resource federation in grid using automated intelligent agent negotiation. *Future Gener. Comput. Syst.*, 26(8):1116–1126, 2010.

[43] S. W. Choi, J. S. Her, and S. D. Kim. Modeling qos attributes and metrics for evaluating services in soa considering consumers' perspective as the first class requirement. In *Asia-Pacific Service Computing Conference, The 2nd IEEE*, pages 398–405. IEEE, 2007.

[44] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[45] M. Coppola, P. Dazzi, A. Lazouski, F. Martinelli, P. Mori, J. Jensen, I. Johnson, and P. Kershaw. The contrail approach to cloud federations. In *Proceedings of the International Symposium on Grids and Clouds (ISGC'12)*, volume 2, page 1, 2012.

[46] L. C. Crosswell and J. M. Thornton. Elixir: a distributed infrastructure for european biological data. *Trends in biotechnology*, 30(5):241, 2012.

[47] M. D. De Assunçao, A. Di Costanzo, and R. Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 141–150. ACM, 2009.

[48] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: The Montage example. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–12, Nov. 2008.

[49] M. Dias de Assuncao, R. Buyya, and S. Venugopal. InterGrid: a case for internetworking islands of grids. *Concurrency Computat. Pract. Exper.*, 20(8):997–1024, 2008.

[50] J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. Software defined federated cyber-infrastructure for science and engineering. In *Proceedings of the 2014 ACM international workshop on Software-defined ecosystems*, pages 9–12. ACM, 2014.

[51] J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. Cometcloud: Enabling software-defined federations for end-to-end application workflows. *IEEE Internet Computing*, 19(1):69–73, 2015.

[52] J. Diaz-Montes, I. Rodero, M. Zou, and M. Parashar. Federating advanced cyber-infrastructures with autonomic capabilities. In *Cloud Computing for Data-Intensive Applications*, pages 201–227. Springer, 2014.

[53] J. Diaz-Montes, Y. Xie, I. Rodero, J. Zola, B. Ganapathysubramanian, and M. Parashar. Exploring the use of elastic resource federations for enabling large-scale scientific workflows. In *Proc. of Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS)*, pages 1–10, 2013.

[54] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou. Profiling applications for virtual machine placement in clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 660–667. IEEE, 2011.

[55] C. Dupont, T. Schulze, G. Giuliani, A. Somov, and F. Hermenier. An energy aware framework for virtual machine placement in cloud federated data centres. In *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*, pages 1–10. IEEE, 2012.

[56] E. E. Huedo, R. Montero, and I. Llorente. A recursive architecture for hierarchical grid resource management. *Future Generation Computer Systems*, 25:401–405, 2009.

[57] E. Elmroth and J. Tordsson. A standards-based grid resource brokering service supporting advance reservations, coallocation, and cross-grid interoperability. *Concurrency and Computation: Practice and Experience*, 21(18):2298–2335, 2009.

[58] C. Evangelinos and C. Hill. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. *ratio*, 2(2.40):2–34, 2008.

[59] F. Fakhfakh, H. H. Kacem, and A. H. Kacem. Workflow scheduling in cloud computing: A survey. In *Proceedings of the 2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, EDOCW '14, pages 372–378, Washington, DC, USA, 2014. IEEE Computer Society.

[60] H. M. Fard, R. Prodan, J. J. D. Barrionuevo, and T. Fahringer. A multi-objective approach for workflow scheduling in heterogeneous environments. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 300–309. IEEE Computer Society, 2012.

[61] A. J. Ferrer, F. HernáNdez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, et al. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66–77, 2012.

[62] R. Fichera, D. Washburn, and E. Chi. The software-defined data center is the future of infrastructure architecture. *Forrester Research*, 2012.

[63] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[64] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[65] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

[66] H. Franke, M. Hogstrom, D. Lindquist, B. McCredie, and S. Pappe. Preface: Software defined environments. *IBM Journal of Research and Development*, 58(2/3):0–1, 2014.

[67] M. E. Frincu and C. Craciun. Multi-objective meta-heuristics for scheduling applications with high availability requirements and cost constraints in multi-cloud environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 267–274. IEEE, 2011.

[68] G. Garzoglio, T. Levshina, M. Rynge, C. Sehgal, and M. Slyz. Supporting shared resource usage for a diverse user community: the osg experience and lessons learned. In *Journal of Physics: Conference Series*, volume 396, page 032046. IOP Publishing, 2012.

[69] Í. Goiri, F. Julià, J. O. Fitó, M. Macías, and J. Guitart. Supporting cpu-based guarantees in cloud slas via resource-level qos metrics. *Future Generation Computer Systems*, 28(8):1295–1302, 2012.

[70] L. Gong, X.-H. Sun, and E. F. Watson. Performance modeling and prediction of nondedicated network computing. *IEEE Transactions on Computers*, 51(9):1041–1055, 2002.

[71] I. Gorton, Y. Liu, and J. Yin. Exploring architecture options for a federated, cloud-based system biology knowledgebase. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE 2nd Intl. Conf. on*, pages 218–225, 2010.

[72] D. Gouscos, M. Kalikakis, and P. Georgiadis. An approach to modeling web service qos and provision price. In *Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on*, pages 121–130. IEEE, 2003.

[73] N. Grozev and R. Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, 44(3):369–390, 2014.

[74] N. Grozev and R. Buyya. Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments. *The Computer Journal*, 58(1):1–22, 2015.

[75] F. Hermenier, J. Lawall, and G. Muller. Btrplace: A flexible consolidation manager for highly available applications. *IEEE Transactions on dependable and Secure Computing*, 10(5):273–286, 2013.

[76] N. Huber, M. von Quast, M. Hauck, and S. Kounev. Evaluating and modeling virtualization performance overhead for cloud environments. In *CLOSER*, pages 563–573, 2011.

[77] T. T. Huu and J. Montagnat. Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 612–617. IEEE, 2010.

[78] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, 2011.

[79] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 159–168. IEEE, 2010.

[80] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl. Qos aggregation for web service composition using workflow patterns. In *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International*, pages 149–159. IEEE, 2004.

[81] G. Jung and H. Kim. Optimal Time-Cost Tradeoff of Parallel Service Workflow in Federated Heterogeneous Clouds. In *2013 IEEE 20th International Conference on Web Services (ICWS)*, pages 499–506, June 2013.

[82] S. Kalepu, S. Krishnaswamy, and S. W. Loke. Verity: a qos metric for selecting web services and providers. In *Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on*, pages 131–139. IEEE, 2003.

[83] J.-M. Kang, H. Bannazadeh, H. Rahimi, T. Lin, M. Faraji, and A. Leon-Garcia. Software-defined infrastructure and the future central office. In *Communications Workshops (ICC), 2013 IEEE Intl. Conf. on*, pages 225–229. IEEE, 2013.

[84] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes. Sky computing. *IEEE Internet Computing*, 13(5):43–51, 2009.

[85] A. Kertesz. Characterizing cloud federation approaches. In *Cloud Computing*, pages 277–296. Springer, 2014.

[86] A. Kertész and P. Kacsuk. Grid meta-broker architecture: Towards an interoperable grid resource brokering service. In *Euro-Par 2006*, pages 112–115. Springer, 2007.

[87] A. Kertész, G. Kecskemeti, M. Oriol, et al. Enhancing federated cloud management with an integrated service monitoring approach. *Journal of grid computing*, 11(4):699–720, 2013.

[88] H. Khazaei, J. Misic, and V. Misic. Performance of cloud centers with high degree of virtualization under batch task arrivals. *IEEE Transactions on Parallel and Distributed Systems*, 24(12):2429–2438, 2013.

[89] H. Khazaei, J. Misic, and V. B. Misic. Performance analysis of cloud computing centers using m/g/m/m+r queuing systems. *IEEE Transactions on Parallel and Distributed Systems*, 23(5):936–943, 2012.

[90] H. Khazaei, J. Misic, and V. B. Misic. A fine-grained performance model of cloud computing centers. *IEEE Transactions on parallel and distributed systems*, 24(11):2138–2147, 2013.

[91] H. Kim, M. AbdelBaky, and M. Parashar. Cometportal: A portal for online risk analytics using cometcloud. In *17th International Conference on Computing Theory and Applications (ICCTA2009)*, 2009.

[92] H. Kim, Y. El-Khamra, I. Rodero, S. Jha, and M. Parashar. Autonomic management of application workflows on hybrid computing infrastructure. *Scientific Programming*, 19(2-3):75–89, 2011.

[93] H. Kim and M. Parashar. Cometcloud: An autonomic cloud engine. *Cloud Computing: Principles and Paradigms*, pages 275–297, 2011.

[94] J.-K. Kim, D. A. Hensgen, T. Kidd, H. J. Siegel, D. S. John, C. Irvine, T. Levin, N. W. Porter, V. K. Prasanna, and R. F. Freund. A flexible multi-dimensional qos performance measure framework for distributed heterogeneous systems. *Cluster Computing*, 9(3):281–296, 2006.

[95] K. Kritikos, J. Domaschka, and A. Rossini. Srl: a scalability rule language for multi-cloud environments. In *IEEE CloudCom*, pages 1–9, 2014.

[96] K. Kurowski, J. Nabrzyski, and J. Pukacki. User preference driven multiobjective resource management in grid environments. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 114–121. IEEE, 2001.

[97] E. Laure and B. Jones. Enabling grids for e-science: The egee project. *Grid computing: infrastructure, service, and applications*, page 55, 2009.

[98] G. Lee, J. Kim, S. Hwang, C. Chang, H. Chang, M. Cho, B. Choi, K. Kim, K. Cho, S. Cho, et al. The kstar project: An advanced steady state superconducting tokamak experiment. *Nuclear Fusion*, 40(3Y):575, 2000.

[99] C.-S. Li, B. Brech, S. Crowder, D. Dias, H. Franke, M. Hogstrom, D. Lindquist, G. Pacifici, S. Pappe, B. Rajaraman, et al. Software defined environments: An introduction. *IBM Journal of Research and Development*, 58(2/3):1–1, 2014.

[100] Z. Li and M. Parashar. A computational infrastructure for grid-based asynchronous parallel applications. In *Proceedings of the 16th international symposium on High performance distributed computing*, pages 229–230. ACM, 2007.

[101] T. Lin, J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia. Enabling sdn applications on software-defined infrastructure. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–7. IEEE, 2014.

[102] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111. IEEE, 1988.

[103] C. Liu and I. Foster. A constraint language approach to matchmaking. In *Intl. Workshop on Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications*, pages 7–14, 2004.

[104] Y. Liu, A. H. Ngu, and L. Z. Zeng. Qos computation and policing in dynamic web service selection. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73. ACM, 2004.

[105] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP journal*, 11(1):36–40, 1997.

[106] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web service level agreement (wsla) language specification. *IBM Corporation*, pages 815–824, 2003.

[107] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 22:1–22:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[108] M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12, Nov. 2011.

[109] M. Mao, J. Li, and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 41–48. IEEE, 2010.

[110] D. A. Menasce and E. Casalicchio. Qos in grid computing. *IEEE Internet Computing*, 8(4):85–87, 2004.

[111] H. Mohamed and D. Epema. KOALA: a co-allocating grid scheduler. *Concurrency Computat. Pract. Exper.*, 20:1851–1876, 2008.

[112] J. D. Montes, M. Zou, R. Singh, S. Tao, and M. Parashar. Data-driven workflows in multi-cloud marketplaces. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 168–175. IEEE, 2014.

[113] F. J. A. Morais, F. V. Brasileiro, R. V. Lopes, R. A. Santos, W. Satterfield, and L. Rosa. Autoflex: Service agnostic auto-scaling framework for iaas deployment models. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 42–49. IEEE, 2013.

[114] A. Mowshowitz. Virtual organization. *Communications of the ACM*, 40(9):30–37, 1997.

[115] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 69–82, 2013.

[116] A. Oprescu and T. Kielmann. Bag-of-Tasks Scheduling under Budget Constraints. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 351–359, Nov. 2010.

[117] S. Ostermann, R. Prodan, and T. Fahringer. Extending grids with cloud resource management for scientific computing. In *GRID*, pages 42–49, 2009.

[118] S. Ostermann, R. Prodan, and T. Fahringer. Dynamic cloud provisioning for scientific grid workflows. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 97–104. IEEE, 2010.

[119] M. Parashar, M. AbdelBaky, I. Rodero, and A. Devarakonda. Cloud paradigms and practices for computational and data-enabled science and engineering. *Computing in Science & Engineering*, 15(4):10–18, 2013.

[120] M. Parashar, M. AbdelBaky, M. Zou, A. R. Zamani, and J. Diaz-Montes. Realizing the potential of iot using software-defined ecosystems. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 1149–1158. IEEE, 2015.

[121] M. Parashar, M. AbdelBaky, M. Zou, A. R. Zamani, E. Renart, and J. Diaz-Montes. Computing in the continuum: Combining pervasive devices and services to support data-driven applications. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2017 – to appear.

[122] M. Parashar and C. A. Lee. Special issue on grid computing. *Proceedings of the IEEE*, 93(3):479–484, 2005.

[123] M. Parashar, V. Matossian, H. Klie, S. Thomas, M. Wheeler, T. Kurc, J. Saltz, and R. Versteeg. Towards dynamic data-driven management of the ruby gulch waste repository. In V. Alexandrov, G. van Albada, P. Sloot, and J. Dongarra, editors, *Computational Science – ICCS 2006*, volume 3993 of *Lecture Notes in Computer Science*, pages 384–392. Springer Berlin Heidelberg, 2006.

[124] D. Petcu. Multi-cloud: expectations and current approaches. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, pages 1–6. ACM, 2013.

[125] I. Petri, T. Beach, M. Zou, J. D. Montes, O. Rana, and M. Parashar. Exploring models and mechanisms for exchanging resources in a federated cloud. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 215–224. IEEE, 2014.

[126] I. Petri, J. Diaz-Montes, O. Rana, M. Punceva, I. Rodero, and M. Parashar. Modelling and implementing social community clouds. *IEEE Transactions on Services Computing*, PP(99):1–1, 2015.

[127] I. Petri, J. Diaz-Montes, M. Zou, T. Beach, O. Rana, and M. Parashar. Market models for federated clouds. *IEEE Transactions on Cloud Computing*, 3(3):398–410, July 2015.

[128] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, W. Frank, et al. The open science grid. In *Journal of Physics: Conference Series*, volume 78, page 012057. IOP Publishing, 2007.

[129] M. Rahman, X. Li, and H. Palit. Hybrid Heuristic for Scheduling Data Analytics Workflow Applications in Hybrid Cloud Environment. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 966–974, May 2011.

[130] N. Ranaldo and E. Zimeo. A framework for qos-based resource brokering in grid computing. In *Emerging Web Services Technology, Volume II*, pages 159–170. Springer, 2008.

[131] R. Ranjan, A. Harwood, R. Buyya, et al. Grid federation: An economy based, scalable distributed resource management system for large-scale resource coupling. *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia*, 2004.

[132] P. Riteau, M. Tsugawa, A. Matsunaga, et al. Large-scale cloud computing research: Sky computing on FutureGrid and Grid'5000. In *ERCIM News*, 2010.

[133] B. Rochwerger, D. Breitgand, E. Levy, et al. The reservoir model and architecture for open federated cloud computing. *IBM J. of Research and Development*, 53, 2009.

[134] I. Rodero, J. Corbalán, R. M. Badia, and J. Labarta. enanos grid resource broker. In *Advances in Grid Computing-EGC 2005*, pages 111–121. Springer, 2005.

[135] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[136] A. Roytman, A. Kansal, S. Govindan, J. Liu, and S. Nath. Pacman: performance aware virtual machine consolidation. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 83–94, 2013.

[137] B. Sabata, S. Chatterjee, M. Davis, J. J. Sydir, and T. F. Lawrence. Taxonomy for qos specifications. In *Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on*, pages 100–107. IEEE, 1997.

[138] A. Sardouk, M. Mansouri, L. Merghem-Boulahia, D. Gaiti, and R. Rahim-Amoud. Crisis management using mas-based wireless sensor networks. *Computer Networks*, 57(1):29 – 45, 2013.

[139] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[140] C. Schmidt and M. Parashar. Squid: Enabling search in dht-based systems. *Journal of Parallel and Distributed Computing*, 68(7):962–975, 2008.

[141] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 71–84. USENIX Association, 2005.

[142] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, 2003.

[143] N. Thio and S. Karunasekera. Automatic measurement of a qos metric for web service recommendation. In *Software Engineering Conference, 2005. Proceedings. 2005 Australian*, pages 202–211. IEEE, 2005.

[144] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, et al. Xsede: accelerating scientific discovery. *Computing in Science & Engineering*, 16(5):62–74, 2014.

[145] H.-L. Truong, R. Samborski, and T. Fahringer. Towards a framework for monitoring and analyzing qos metrics of grid services. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, pages 65–65. IEEE, 2006.

[146] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris. Automated, elastic resource provisioning for nosql clusters using tiramola. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 34–41. IEEE, 2013.

[147] H. N. Van, F. D. Tran, and J.-M. Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Computer and Information Technology, 2009. CIT'09. Ninth IEEE International Conference on*, volume 1, pages 357–362. IEEE, 2009.

[148] H. N. Van, F. D. Tran, and J.-M. Menaud. Performance and power management for cloud infrastructures. In *2010 IEEE 3rd international Conference on Cloud Computing*, pages 329–336. IEEE, 2010.

[149] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 228–235. IEEE, 2010.

[150] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. *Future Generation Computer Systems*, 29(4):973–985, 2013.

[151] P. Varalakshmi, A. Ramaswamy, A. Balasubramanian, and P. Vijaykumar. An Optimal Workflow Based Scheduling and Resource Allocation in Cloud. In A. Abraham, J. L. Mauri, J. F. Buford, J. Suzuki, and S. M. Thampi, editors, *Advances in Computing and Communications*, number 190 in Communications in Computer and Information Science, pages 411–420. Springer Berlin Heidelberg, July 2011. DOI: 10.1007/978-3-642-22709-7_41.

[152] C. Vazquez, E. Huedo, R. Montero, and I. Llorente. Dynamic provision of computing resources from grid infrastructures and cloud providers. In *Intl. Conf. on Grid and Pervasive Computing (GPC)*, pages 113–120, 2009.

[153] C. Vecchiola, X. Chu, and R. Buyya. Aneka: a software platform for .net-based cloud computing. *High Speed and Large Scale Scientific Computing*, 18:267–295, 2009.

[154] D. Villegas, N. Bobroff, I. Rodero, J. Delgado, Y. Liu, A. Devarakonda, L. Fong, S. M. Sadjadi, and M. Parashar. Cloud federation in a layered service model. *Journal of Computer and System Sciences*, 78(5):1330–1344, 2012.

[155] S. Wahle, T. Magedanz, S. Fox, and E. Power. Heterogeous resource description and management in generic resource federation frameworks. In *12th IFIP/IEEE Intl. Symp. on Integrated Network Management (IM 2011) and Workshops*, pages 1196–1199, 2011.

[156] C. Wang and J.-L. Pazat. A chemistry-inspired middleware for self-adaptive service orchestration and choreography. In *CCGRID*, pages 426–433. IEEE Computer Society, 2013.

[157] D. Wang, D. J. Foran, J. Ren, H. Zhong, I. Y. Kim, and X. Qi. Exploring automatic prostate histopathology image gleason grading via local structure modeling. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2649–2652. IEEE, 2015.

[158] J. Wang, M. AbdelBaky, J. Diaz-Montes, S. Purawat, M. Parashar, and I. Altintas. Kepler+ cometcloud: dynamic scientific workflow execution on federated cloud resources. *Procedia Computer Science*, 80:700–711, 2016.

[159] L. Wang, R. Duan, X. Li, S. Lu, T. Hung, R. Calheiros, and R. Buyya. An Iterative Optimization Framework for Adaptive Workflow Management in Computational Clouds. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1049–1056, July 2013.

[160] P. Wieder, J. Seidel, O. Wäldrich, et al. Using sla for resource management and scheduling-a survey. In *Grid Middleware and Services*, pages 335–347. Springer, 2008.

[161] L. A. Wolsey. *Integer programming*, volume 42. Wiley New York, 1998.

[162] P. Wright, Y. L. Sun, T. Harmer, A. Keenan, A. Stewart, and R. Perrott. A constraints-based resource discovery model for multi-provider cloud environments. *Journal of cloud computing: advances, systems and applications*, 1(1):6, 2012.

[163] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1):256–293, Mar. 2011.

[164] Z. Wu, Z. Ni, L. Gu, and X. Liu. A Revised Discrete Particle Swarm Optimization for Cloud Workflow Scheduling. In *2010 International Conference on Computational Intelligence and Security (CIS)*, pages 184–188, Dec. 2010.

[165] X. Yang, B. Nasser, M. Surridge, and S. Middleton. A business-oriented cloud federation model for real-time applications. *Future Generation Computer Systems*, 28(8):1158–1167, 2012.

[166] Y. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady. Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 91–98, July 2010.

[167] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200, 2005.

[168] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[169] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Computing Surveys (CSUR)*, 47(4):63, 2015.

[170] X. Zhang and Y. Yan. Modeling and characterizing parallel computing performance on heterogeneous networks of workstations. In *Parallel and Distributed Processing, 1995. Proceedings. Seventh IEEE Symposium on*, pages 25–34. IEEE, 1995.

[171] C. Zhao, S. Li, Y. Yang, J. Wang, Z. Dong, L. Liu, and L. Li. An autonomic performance-aware workflow job management for service-oriented computing. In *2010 Ninth International Conference on Grid and Cloud Computing*, pages 270–275. IEEE, 2010.

[172] H. Zhong, K. Tao, and X. Zhang. An Approach to Optimized Resource Scheduling Algorithm for Open-Source Cloud Systems. In *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual*, pages 124–129, July 2010.

[173] H.-J. Zimmermann. Fuzzy programming and linear programming with several objective functions. *Fuzzy sets and systems*, 1(1):45–55, 1978.