

© 2018 Phuong V. Nguyen

DOSSIER: DISTRIBUTED OPERATING SYSTEM AND INFRASTRUCTURE FOR
SCIENTIFIC DATA MANAGEMENT

BY

PHUONG V. NGUYEN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Doctoral Committee:

Professor Klara Nahrstedt, Chair
Professor Roy H. Campbell
Professor Indranil Gupta
Dr. Deepak Turaga

ABSTRACT

As scientific advancement and discovery have become increasingly data-driven and interdisciplinary, there are urging needs for advanced cyberinfrastructure to support managing and processing scientific data generated from day-to-day research. However, the development of data-driven cyberinfrastructure for scientific research areas has often lagged behind the development of such tools in other engineering and IT-related fields. Such the development gap is due to various diversity challenges of scientific data management and processing. *First*, these are the challenges in terms of the diversity of scientific *data* and data processing *tasks*, as the cyberinfrastructure should be able to support managing and processing heterogeneous types of scientific data that have been captured from scientific instruments. *Second*, as the cyberinfrastructure must help to shorten time from digital capture of data to interpretation and insights, it is challenging for the infrastructure to deal with the diversity of *users* and *scientific workload*. *Third*, it is the diversity of *scientific instruments*. Since there is still a significant number of scientific instruments that run their scientific software tools on old operating systems (e.g., Windows XP, Windows NT, Windows 2000), the cyberinfrastructure must help to bridge the performance and security gap between old scientific instruments and its advanced cloud-based infrastructure.

In this thesis, we aim to address the above diversity challenges by taking a holistic approach in designing a distributed operating system and infrastructure for scientific data management, named DOSSIER. At the core of DOSSIER is an *adaptive control microservice infrastructure* that is designed to tackle the aforementioned challenges of data cyberinfrastructure for distributed scientific data management. Particularly, to handle *heterogeneous scientific data processing and analysis*, we start with redesigning the execution environment for scientific workflows, which traditionally follows a monolithic approach, using a novel microservice architecture and latest virtualization technology (i.e., container technology). The microservice design enables dynamic composition of workflows, and thus, is efficient in dealing with heterogeneous workflows. The new microservice architecture also allows us to express system resources in a more simple way, and thus, enables the design of a new adaptive resource management mechanism to handle large-scale and dynamic scientific workloads. We are the first to apply feedback control theory to design a self-adaptation mechanism for scientific workflow management system to help shorten the time from data acquisition to insights. To address the security and performance gap issues when connecting old scientific instruments to cloud-based cyberinfrastructure, we design an edge-cloud architecture that puts cloudlet servers directly connected to the scientific instruments and act as the security shield for the aging instruments. Cloudlets will also coordinate with cloud-based backend system to tackle

the performance issue by scheduling data transfer and offloading processing tasks to cloudlets to avoid traffic congestion and guarantee performance of data processing jobs across edge-cloud architecture.

By designing, developing, and testing DOSSIER in the real scientific environments, we demonstrate that an edge-cloud microservice architecture with learning-based adaptive control resource management is needed for timely distributed scientific data management.

*To my beloved parents, for their unlimited confidence in me.
To my wife Linh, for her endless love and support, and my daughter Minh, for unknowingly
inspiring daddy everyday.*

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Professor Klara Nahrstedt, for all the guidance and support throughout my Ph.D. study. I really appreciate and feel fortunate for the opportunity to join her research group and being involved in various projects under Professor Nahrstedt's leadership. Without her trust and support, I would not make it through.

I would also like to thank my thesis committee members, Professor Roy H. Campbell, Professor Indranil Gupta, and Dr. Deepak Turaga, for their insightful comments and suggestions to help me improve the thesis.

I would like to express my gratitude to all the members of the T2C2, 4CEED, and BRACELET projects, including team members from Micro and Nanotechnology Lab, Materials Research Lab, and Engineering IT, which I was a part of since the beginning. The opportunity to work in an inter-disciplinary, highly collaborative team that develops an impactful product for real users, and at the same time, to apply my research results into production have been the highlight of my Ph.D. study.

I would like to thank all of my mentors of the summer internship I did during my Ph.D. program: Dr. Debessay Fesehay Kassa, Mr. Siddhartha Jain, Dr. Vinod Muthusamy, Dr. Aleksander Slominski, Dr. Vatche Ishakian. Their guidance and support have made those summers productive and memorable ones, and I learned a lot of practical experiences working in both industry and research labs environments.

I would also like to thank both current and former members of MONET research group for all the help during my time with the group. Especially, I would like to thank Dr. Debessay Fesehay Kassa and Dr. Long Vu for introducing me to the group (without your introduction, none of this research would have been possible) and for helping me through the early days of my Ph.D. research.

I would like express my dearest appreciation to my extended family from Vietnam, especially my Mom and Dad, for all the love, support and confidence they have in me. I would like to thank my dearest wife Linh for her love and support through thick and thin, and my beloved daughter Minh for being my everyday inspiration. Spending valuable time with my family really helped me to find a good balance during the busy and sometimes stressful Ph.D. life.

Last but not least, I would like to thank the National Science Foundation for supporting research done in this thesis through awards numbered 1443013 and 1659293. The opinions, findings and conclusions or recommendations expressed in this thesis are those of the author and do not necessarily reflect the view of the National Science Foundation.

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	1
CHAPTER 2	BACKGROUND AND CHALLENGES	4
CHAPTER 3	RELATED WORK	9
3.1	Scientific Data Management	9
3.2	Scientific Workflow Execution and Monitoring	9
3.3	Cloud-based Workflow Management Systems	10
3.4	Scalability and Adaptability of Workflow Systems	11
3.5	Edge-Cloud Architecture for Data Cyberinfrastructure	12
CHAPTER 4	OVERVIEW OF DOSSIER	13
CHAPTER 5	MICROSERVICE ARCHITECTURE FOR SCIENTIFIC DATA MAN- AGEMENT	17
5.1	Overview of Microservice Architecture	17
5.2	Task Dependency Service	19
5.3	Microservice Resource Models and Assumptions	21
CHAPTER 6	ADAPTIVE CONTROL FRAMEWORK FOR MICROSERVICE IN- FRASTRUCTURE	24
6.1	Overview	24
6.2	Integration of Adaptive Control Framework with Microservice Infrastructure	24
6.3	Microservice Monitoring	26
6.4	Microservice Adaptation	27
CHAPTER 7	QUEUEING NETWORK-BASED RESOURCE ADAPTATION	29
7.1	Modeling Motivations	29
7.2	Modeling Performance of Microservice Execution Environment	29
7.3	Microservice Adaptation as Optimization Problem	32
7.4	Greedy Resource Allocation Solutions	34
7.5	Evaluation	36
7.6	Summary and Discussions	45
CHAPTER 8	MONAD - MODEL PREDICTIVE CONTROL RESOURCE ADAPTATION	46
8.1	System Identification	46
8.2	Controller Design	48
8.3	Evaluation	50
8.4	Summary and Discussions	56

CHAPTER 9	ADAPTIVE MICROSERVICE INFRASTRUCTURE VIA MODEL-BASED REINFORCEMENT LEARNING	58
9.1	Background: Reinforcement Learning	58
9.2	Model-based Reinforcement Learning Adaptation for Microservice Infrastructure	59
9.3	Evaluation	64
9.4	Summary and Discussions	68
CHAPTER 10	4CEED - REAL-TIME ACQUISITION AND ANALYSIS FRAMEWORK FOR MATERIALS-RELATED CYBER-PHYSICAL ENVIRONMENTS	72
10.1	Extending DOSSIER to Material-related Environment	72
10.2	Curation Service	72
CHAPTER 11	BRACELET - HIERARCHICAL EDGE-CLOUD MICROSERVICE INFRASTRUCTURE	76
11.1	BRACELET's Architecture	78
11.2	BRACELET's Resource Management	83
11.3	BRACELET's Security Design	88
11.4	Evaluation	89
CHAPTER 12	CONCLUSIONS	94
APPENDIX A	DETAILS ON PARAMETRIC DECOMPOSITION PROCEDURE	95
REFERENCES	97

CHAPTER 1: INTRODUCTION

Scientific research has become increasingly data-driven and interdisciplinary. One of the key enabling factors for such a trend is that scientific instruments are becoming more and more advanced and capable of capturing digital data in real-time. One example is the evolution of *microscopes*, from traditional optical microscopes that use light to interact with sample to generate images, which are limited in the spatial resolution, to more advanced electron microscopes (e.g., transmission electron microscopes or TEMs) that use high energy beam of electrons to pass through sample to produce high-resolution images. Such advanced electron microscopes are often connected to computers that help to enable accurate calibration of microscope and real-time capturing of digital images. As more and more digital data being captured, it opens opportunities for data-driven analysis and data sharing across related research disciplines.

The development of data-driven cyberinfrastructure for scientific research areas (e.g., material science, biology), however, has often lagged behind the development of such tools in other engineering and IT-related fields. In particular, related efforts in data-driven cyberinfrastructure mainly focus on *homogenous, well-organized data* and support data processing and analysis in an offline or batch manner (e.g., in areas such as astronomy, high-energy physics). On the other hand, much less effort has been on *long-tail scientific data* - data of small or medium size collected during day-to-day research (e.g., digital images of a scientific experiment, captured by digital microscopes in the lab), or “dark data” that consists of unpublished data of failed experiments. It has been shown that more than 50% of scientific findings do not appear in the published literature [1], as only data of successful experiments are often included in publications.

As scientific advancement and discovery have increasingly moved to a data-driven and interdisciplinary approach, such a development gap in data-driven cyberinfrastructure can cause significant delay in bridging the innovation across scientific disciplines. For example, in material science domain, National Academy’ studies [2] suggest that it typically takes 20 years to go from the discovery of new materials to fabrication of new and next-generation devices based on the new materials. There are several issues with the current state of data capture and storage in materials and semiconductor fabrication domains that contribute to the long cycle from discovery of new materials to fabrication of new devices. The most notable issues include manual digital data capturing and transferring (i.e., often done via “sneaker-net” techniques), and the lack of advanced data management and sharing (i.e., researchers often store data in their local hard drive or use generic storage services, such as Box or Google Drive, that do not provide any assistance in organizing and processing data). Hence, to shorten the discovery cycle, it will require a major

transformation in how we collect digital data about materials and how we make the digital data available to computational tools for developing new materials and fabricating new devices to the research community.

Developing an advanced data management and infrastructure to support scientific data-driven research poses several challenges. *First*, in terms of *data management and processing*, the infrastructure should be able to support managing and processing heterogeneous types of scientific data that have been captured from instruments. Since scientific data processing job is often modeled as workflows, the infrastructure thus needs to support executing heterogeneous types of workflows. *Second*, the cyberinfrastructure needs to be *scalable and adaptive* to deal with varying and often bursty workload, in order to help shorten the time from digital capture to interpretation and insights. In addition, instead of being deployed on public cloud infrastructure with elastic and unlimited resources, cyberinfrastructure is often deployed within educational or research institutions with limited resources. Therefore, the cyberinfrastructure should be designed to support flexible resource management strategies to meet different objectives (e.g., a certain average job response time is guaranteed) and satisfy different constraints (e.g., total resource cost is under a limited budget) set by users. *Third*, in terms of *data acquisition*, due to a slow update cycle of scientific software, a large number of scientific instruments is still connected to computers running old and unsupported OS (e.g., Windows XP) whose software cannot be patched with important security protections. Hence, connecting these instruments to the cyberinfrastructure network can cause concerns about security and the performance gap between old software system and the new networked infrastructure.

In this thesis, we aim to address the above challenges by taking a holistic approach in designing a distributed operating system and infrastructure for scientific data management, named DOSSIER¹. At the core of DOSSIER is an *adaptive control microservice infrastructure* that is designed to tackle the aforementioned challenges of data cyberinfrastructure for distributed scientific data management. Particularly, to handle *heterogeneous scientific data processing and analysis*, we start with redesigning the execution environment for scientific workflows, which traditionally follows a monolithic approach, using a novel microservice architecture and latest virtualization technology (i.e., container technology). The microservice design enables more flexible and dynamic composition of workflows, and thus, is efficient in dealing with heterogeneous workflows. The new microservice architecture also allows us to express system resources in a more simple way, and thus, enables the design of a new adaptive resource management mechanism to handle large-scale and dynamic scientific workloads. We are the first to apply feedback control theory to

¹DOSSIER stands for **D**istributed **O**perating **S**ystem and **I**nfrastructure for **S**ciEntific **D**ata **M**anagement

design a self-adaptation mechanism for scientific workflow management system to help shorten the time from data acquisition to insights. To address the security and performance gap issues when connecting old scientific instruments to cloud-based cyberinfrastructure, we design an edge-cloud architecture that puts cloudlet servers directly connected to the scientific instruments and act as the security shield for insecure instrument computers. Cloudlets will also coordinate with cloud-based backend system to tackle the performance issue by scheduling data transfer and offloading processing tasks to cloudlets to avoid traffic congestion and guarantee performance of data processing jobs across edge-cloud architecture.

In summary, our thesis statement is that: *Edge-cloud microservice architecture with learning-based adaptive control resource management is needed for timely distributed scientific data management.*

The remaining of this thesis is organized as follows. In Chapter 2, we provide background information of scientific data environment, the material-related environment, through which we motivate, develop and validate our algorithms and approaches, and discuss in details the key challenges for building a data cyberinfrastructure for that environment. In Chapter 3, we summarize related work on scientific data management and cyberinfrastructure. In Chapter 4, we present an overview of DOSSIER and its high-level architecture. In the following chapters, we describe in details different components of DOSSIER, from its microservice execution architecture (Chapter 5) and adaptive control framework (Chapter 6), to its realizations using queueing network (Chapter 7), model predictive control (Chapter 8), and reinforcement learning (Chapter 9). Then, in Chapter 10, we present 4CEED - an application of DOSSIER in material-related environment. After that, we describe the design and implementation of BRACELET - our hierarchical edge-cloud microservice infrastructure to tackle the distributed data acquisition and processing challenges (Chapter 11). We conclude the thesis, summarize lessons learned and future directions in Chapter 12.

CHAPTER 2: BACKGROUND AND CHALLENGES

To better understand the target environment of this thesis - the material-related scientific experiments, we provide some background information on the materials, semiconductor experiments, and analytical instruments used in materials science research. In addition, we present some insights from our user study to shed light on the scientific user requirements and expectations.

Figure 2.1 shows a typical experiment flow in material-related research. In the first step, researchers create physical experimental samples, either in their labs or in shared fabrication facilities. These physical samples can range from microelectronic devices, to biological samples, to nanoparticles. Once physical samples are created, they must be prepared for analysis (Step 2). For example, with analysis using Scanning Electron Microscopes (SEM), the preparation usually involves cutting the sample into a size which can be placed under the microscope and attaching it to a SEM sample holder. The result of such preparation is called an analytical sample. The actual analysis of an analytical sample happens using analytic tools/instruments (Step 3), including SEM and other electron microscopes, as well as x-ray, ion, and optical scattering experiments.

The results of the analysis in Step 3 are the digital footprints of the physical experimental samples. These digital data can vary in format, depending on the type of analytic instrument used. For example, the output of SEM microscopes (Figure 2.2) consists of: (i) digital images of analytical sample that are stored in standard image format (e.g., .TIF, .GIF., or .JPEG), (ii) instrument specific information and meta-data (e.g., temperature, pressure, accelerating voltage, detector used, etc.) that are stored in a text file, and (iii) unstructured notes by researchers about the experimental or analytical results. On the other hand, output data from the TEM microscopes is in proprietary data format (i.e., DM3) that contains both image data and instrument specific meta-data. In such the case of proprietary data format, it might require another step to convert the results of analytic tools to standard formats (Step 4). The researchers must then transport the converted files to their personal workstation (Step 5 - which often uses a “sneakernet” of USB thumb drives) for follow-up interpretation (Step 6). If the interpretation result is negative, further modifications might be needed for the procedure to create physical experimental sample (Step 7), which causes repetitions of the process until the desired interpretation, insights and criteria are satisfied.

While new analytic techniques have allowed for a surge of nanomaterials research publications and related innovative products, the time between discovery of new materials and their application in, for example, semiconductor fabrication processes is at a relative stagnation, taking several years between an incepted material design and its commercial usage. This slow process can be attributed largely in part to communication of research, or rather the lack-there-of, specifically pertaining to

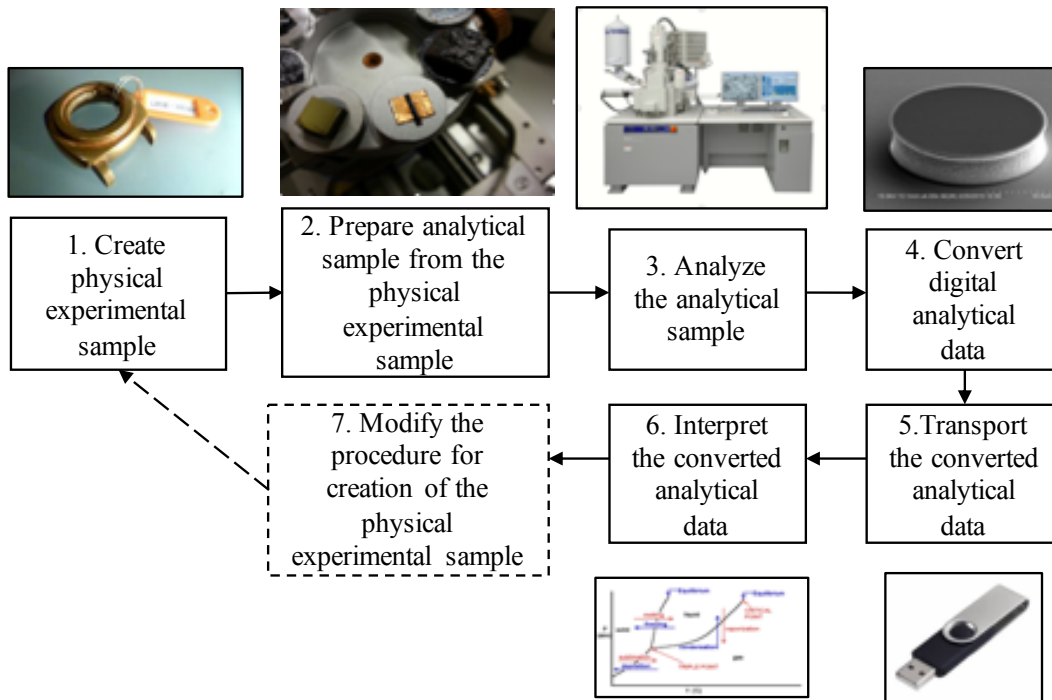


Figure 2.1: Typical experiment flow in material science.

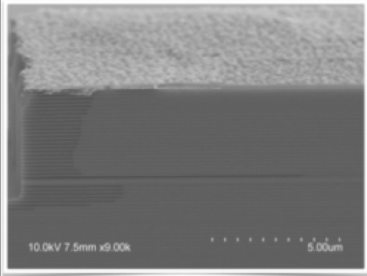
 <p>Result image of 07302013-Oxidation experiment</p>	<p>Experimental setting:</p> <p>Time 13min Temp 425 C</p> <p>(Structured meta data)</p>
	<p>Notes:</p> <p>Oxidation depth is about 12um. Oxidation layer composed of Al(0.98)GaAs with thickness of 30 nm. Furnace in 2111 MNT L, 2" diameter quartz.</p> <p>(Free text)</p>

Figure 2.2: An example of SEM output.

nanomaterial analysis tools. Most often negative results from these nanomaterial analysis tools are not published, the transportation of the collected data is often insecure, and the resulting data files are often proprietary, causing inherent loss of data through file conversions in order to work up the data for publication quality figures.

In order to accelerate the experimental process, it is necessary to have an expedient mean to capture, transport, and process the digital data (i.e., output of Step 3) in timely and in trusted manner before archiving them. Furthermore, it is necessary to have access to extensive data analysis and visualization for more efficient interpretation of the results. Such a distributed timely and trusted data-driven framework would greatly reduce the time, security and data loss risks of the manual efforts involved in the Step 4, 5, and 6 of the experimental scientific process. In addition, a networked platform that provides authorized access to archived experimental data (e.g., dark unpublished data) would help to close the communication gap between researchers and prevent unnecessary repetitions of the experimental process, caused by the lack of information in the literature.

There are a number of *diversity challenges* for building data-driven framework and cyberinfrastructure to support timely scientific data management in pre-publication phase..

First, the diversity of scientific *data* and data processing *tasks* represents a major challenge. The cyberinfrastructure should be able to support managing and processing heterogeneous types of scientific data that have been captured from instruments. This is challenging, since data generated from scientific experiments is often multi-modal (as shown in Figure 2.2) and the process of studying a single type of material sample often involves a number of heterogeneous types of experiments (Figure 2.3 shows a series of different types of experiments that are often executed during the semi-conductor experimental process). In addition, because multiple, co-dependent tasks, called task workflows, process scientific data, the cyber-infrastructure must support executing heterogeneous types of workflows.

Second, while the data infrastructure must help to shorten time from digital capture to interpretation and insights, it is challenging for the data infrastructure to deal with the diversity of *users* and *their scientific workload*. With current manual data acquisition and transfer, it often takes hours from when data is captured in the lab until users can process the data and gain insights from the experiments. The opportunity to shorten that time gap is to provide the ability to upload data right away during the ongoing lab session. Our recent user survey shows that 66% of users feel they have enough time during the microscope lab session to upload the data if such a data-upload service and/or tool exists. However, even if such a uploading tool exists, the unpredictability and the dynamism of workloads, uploaded from hundreds of scientific instruments during lab sessions

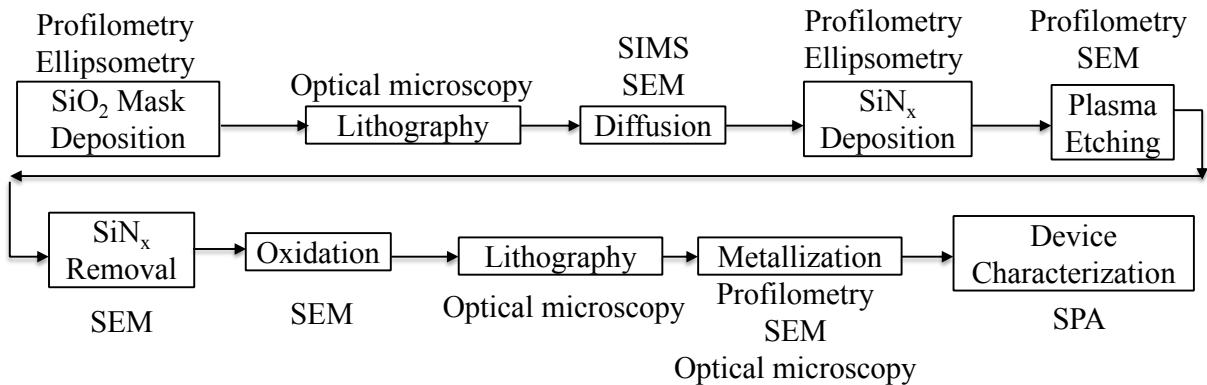
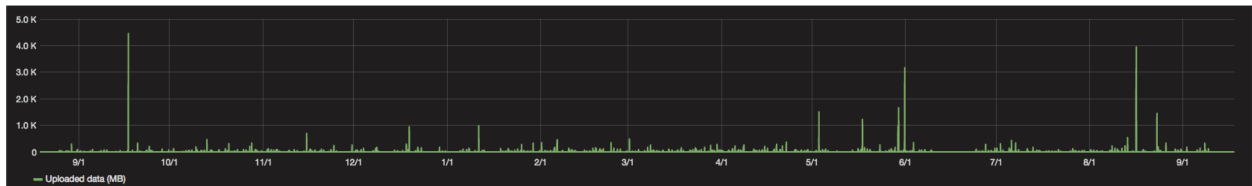
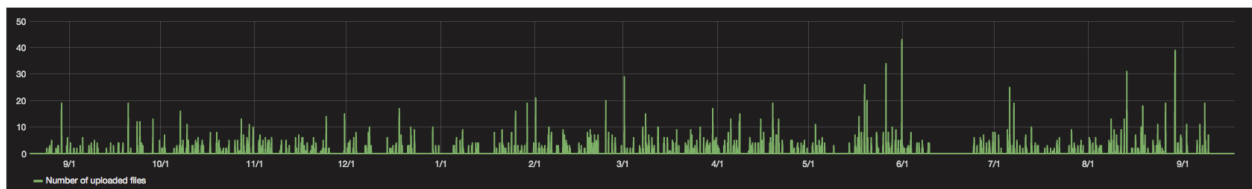


Figure 2.3: Heterogeneous types of experiments often done on a material sample.



(a) Amount of data uploaded

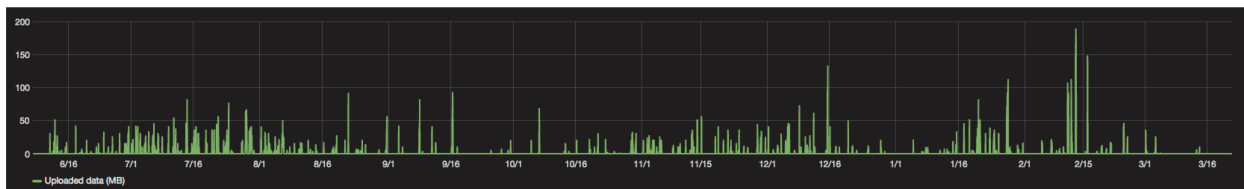


(b) Number of files uploaded

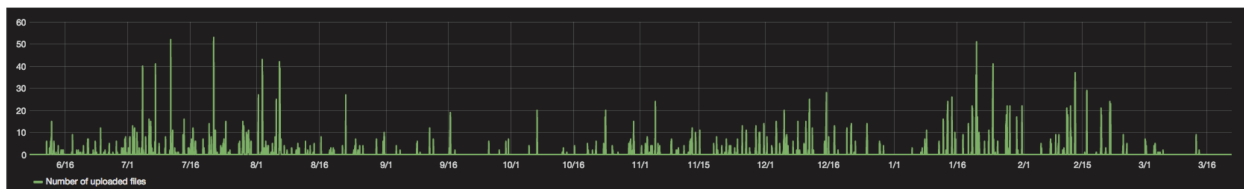
Figure 2.4: Data usage on MRL’s JOEL instrument from Sep 2015 to Sep 2016.

might pose a serious challenge for the scalability and adaptability of data infrastructure.

To better understand workload characteristics, we study the actual usage of experimental instruments in material research environment. In particular, we select two of the most popularly used instruments in MRL, namely JOEL and HeliosFIB, and collect information about experimental results created on those instruments, such as creation time, size of output file, etc, over a one-year time period. This information gives us vital information about the actual usage of these instruments and the typical workload generated from those instruments. The results on instrument usage are shown in Figure 2.4 and Figure 2.5. We can see that, on both instruments, the workloads are highly variable and often bursty. In general, there is a correlation between the number of files and the amount of data uploaded (especially in HeliosFIB case). However, for JOEL instrument, the variability in the amount of uploaded data seems to be more extreme, due to the fact that the



(a) Amount of data uploaded



(b) Number of files uploaded

Figure 2.5: Data usage on MRL’s HeliosFIB instrument from June 2015 to March 2016.

files produced by JOEL are generally large files and can vary in sizes. These results suggest that the data cyber-infrastructure that supports capturing and processing digital data generated from instruments needs to be scalable and highly adaptive to handle variable and bursty workloads.

Third, the diversity of *instruments* is a challenge. The data infrastructure must help to bridge the performance and security gap between old scientific instruments and their advanced cloud-based infrastructure. There is still a significant number of scientific instruments that run their scientific software tools on old operating systems (e.g., Windows XP, Windows NT, Windows 2000). Since these OSes cannot operate at the network speed of a powerful cloud and are not patched with the latest security patches, the instruments are taken offline and cannot connect to the cloud infrastructure. This is because if these instruments were put on the network, they would be destroyed by viruses and might represent major security threats and performance bottlenecks to the very expensive instruments and the overall network infrastructure. Furthermore, this situation will not go away, since instrument companies do not upgrade their instrument software at the same frequency with which the computing companies upgrade their OSes¹. Even more recent OSs, such as Windows 7, will become obsolete in the near future, and scientific instruments running on Windows 7 will eventually join the group of offline instruments. As a result, the current networked solution for scientific instruments is not evolvable and represents a major barrier to accelerating the pace of discovery and deployment of advanced cyber-infrastructure.

In this thesis, we design DOSSIER to tackle the above diversity challenges (i.e., diversity of data, tasks, users, workloads, and devices) to support timely distributed scientific data management.

¹It is often that the instrument companies (e.g., GE, Siemens) stop augmenting/updating their scientific softwares when OSes are upgraded to newer versions or when new OS patches come up. Hence, to make use of the instruments, scientific users have to run the instruments on older OSes.

CHAPTER 3: RELATED WORK

Before introducing our proposal solution, DOSSIER, for timely distributed scientific data management, we discuss various categories of related work on scientific data management, scientific workflow systems, distributed cyberinfrastructure, and briefly explain how our approach differs from the related work.

3.1 SCIENTIFIC DATA MANAGEMENT

The related efforts in scientific data management have been focusing on making existing datasets more accessible and shareable (e.g., DataUp [3], SkyServer [4]), toward long-terms preservation (e.g., SEAD [5]). Other efforts focus on providing easy access and collaboration to distributed cyberinfrastructure that incorporate cloud and grid technologies, such as HubZero [6], NanoHub [7] (for nanotechnology simulations), BrownDob [8], Data Conservancy Instance [9] (for cloud-based data curation). With DOSSIER, our focus shifts to capturing, accurately curating, scientific digital data in a timely and trusted manner *before* fully archiving and publishing them for wide access and sharing. Thus, our effort is complementary to those other efforts, and we could effectively leverage results of existing solutions (e.g., data preservation tools for long-term storage of data, data curation tools for curating data after it has been captured and stored). Although we will demonstrate our design of data management system for materials-related environments, it can be extended to use in other domains where the nested data model and workflow-based data processing mechanism apply.

3.2 SCIENTIFIC WORKFLOW EXECUTION AND MONITORING

Scientific workflow management systems (WfMS) [10][11] have traditionally employed a monolithic approach in workflow implementation and execution. In particular, each workflow is implemented as a tightly coupled set of tasks and has its own workflow execution plan that specifies how to run the workflow on a distributed computation infrastructure. For example, Pegasus [12] statically translates a workflow graph into an execution plan (e.g., selecting sites for tasks to run and cluster tasks based on various criteria) and the plan could not be changed once the execution runs. In other systems, such as Taverna [13], Triana [14], Kepler [15], all data movements and task submissions to grid infrastructure need to be explicitly specified and organized in the execution plan. In Shock/AWE [16][17], once being executed, the execution plan is often coordinated by

a centralized server (and often, with a single task queue) that is in charge of task invocation and synchronization. Such a static, infrastructure-dependent execution plan creation, and centralized coordination mechanism make the existing systems less efficient in dealing with large-scale and heterogeneous workloads.

In DOSSIER, we leverage the latest advances in cloud computing and virtualization technology to abstract away the infrastructure complexity (e.g., task allocation on actual servers/VMs is handled by cluster management system, such as YARN [18], Docker Swarm, Kubernetes [19]) and focus on the design of workflow execution model. In particular, by modeling tasks as micro-services, we are able to separate workflow’s task dependencies from task implementation, and thus allow more flexible and dynamic composition and execution of workflows.

Real-time monitoring is important to control workflow execution [11], and it is still an open issue. The common approaches for workflow monitoring are still based on analyzing execution log data [20], or provenance data [21][22], and often require extra implementation effort to collect such data. In DOSSIER, we leverage our micro service-based architecture and the publish/subscribe middleware to perform seamless performance monitoring of workflows and tasks.

3.3 CLOUD-BASED WORKFLOW MANAGEMENT SYSTEMS

With the increasing popularity of cloud infrastructure, there have been efforts [23][24] to deploy WfMS on the cloud to take advantage of the elasticity and service level agreement of cloud resources. Cloud-based deployment of WfMSs opens the opportunity to offer WfMS using the *scientific workflow-as-a-service* (SWfaaS) model [25][26] that can support a broader range of users with different requirements. While related works that leverage grid and cloud-based infrastructure often use resource models based on VMs and virtual resources, with the recent advances in virtualization technology, especially container technology, there have been efforts [27][16][28] to use container as the execution environment for executing workflow tasks. These efforts mainly focus on containerizing workflow’s tasks to overcome the dependency issues during workflow deployment and to improve the reproducibility of workflow implementation via containerized tasks. However, little effort has been done on the composability of workflows.

The microservice execution model and the use of container technology as the implementation standard for a task in DOSSIER allow us to offer workflow-as-a-service capability. Users can either execute a workflow that consists of tasks implemented in any language and packaged into containers; or easily compose and execute new workflows using reusable tasks. The micro-service execution model also allows to abstract the resource allocation as the allocation of consumers

over task’s micro-services. This simple allocation model enables efficient resource scheduling and adaptation strategies (to discuss shortly) to provide guarantees on various constraints set by users (e.g., deadline, cost), which are often supported by a software-as-a-service system.

Our DOSSIER system operates on top of a generic cloud resource management system that offers basic resource management capabilities. Although we use Kubernetes [19] in our implementation, DOSSIER can be used with other systems, such as YARN [18] and Mesos [29] that help allocate available computational resources to applications.

3.4 SCALABILITY AND ADAPTABILITY OF WORKFLOW SYSTEMS

To provide various guarantees when executing scientific workflows, there have been related works on scheduling and allocation of tasks under cost and deadline constraints of individual workflow [30][31], or workflow ensembles [32][33][34]. Techniques developed in these works mostly focus on optimizing execution order of tasks and the allocation of tasks on distributed resources and require advanced knowledge of workflow’s structure (e.g., critical paths, partitioning tasks in workflows)[35][36]. In addition, there have also been related work that leverage the elasticity of cloud resources to dynamically provision resources allocated for executing a workflow (e.g., CPU, memory of VMs), in order to meet certain deadline constraint [25][37][34].

In DOSSIER, using the microservice execution model, we are able to simplify the way we control the scalability of the system by controlling the number of consumers per task in a workflow. In addition, we are able to derive black-box performance model (without knowledge of workflow structures) of the system using the allocation of consumers as control inputs. From the performance model, we design self-adaptation mechanism based on feedback control theory, so that DOSSIER can self-adapt to the dynamism of workloads to guarantee performance constraints provided by users.

Feedback control-theoretic approaches, which are traditionally used in mechanical and electrical systems (e.g., in robotics), have been adopted in several types of software system [38], such as web server [39], distributed visual tracking system [40], adaptive real-time systems [41], and network congestion avoidance [42]. As far as we are concerned, we are the first to apply feedback control mechanism to enable self-adaptive and QoS-aware support in scientific workflow systems. Cloud-based workflow systems pose complex interactions between different workflow types and tasks (via task dependencies), as well as various performance and resource cost constraints, and thus a simple controller model such as PID (Proportional-Integral-Derivative) is not suitable. Hence, we use *multilayer neural networks*, which have theoretically-proven approximation power and

have been applied successfully in the identification and control of dynamic systems, to capture the performance model of the system. In addition, we employ the *model predictive control* [43] methodology and treat the controller design problem as an optimization problem, which allows us to incorporate various system constraints, using the receding horizon technique [44].

3.5 EDGE-CLOUD ARCHITECTURE FOR DATA CYBERINFRASTRUCTURE

Related work on *cyberinfrastructure* [6, 9, 45] has mainly focused on cloud-based, two-tier architecture and lacks of support for vulnerable scientific instruments running on out-of-date operating systems. In this thesis, we design *the first* microservice-based hierarchical edge-cloud architecture (Chapter 11) for cyberinfrastructure that seamlessly extends cloud-based infrastructure to the edges to help connect and protect otherwise disconnected and vulnerable instruments.

In terms of *computation offloading* in the edge-cloud architecture, there has been related work in mobile computing domain [46] that aims to minimize execution time or preserve energy on the mobile endpoints. Tong et al. [47] propose a workload placement algorithm to decide which mobile programs are placed (i.e., placement) on which edge-cloud servers, and how much computational capacity is needed (i.e., scaling). Tan et al. [48] propose a general model for deciding when and where to offload a job from a mobile user. Wang et al. [49] investigate the assignment and the scheduling of tasks over multiple cloudlets. Most of related work, however, only deals with workloads of independent tasks (sometimes with known task profiles). In DOSSIER, we are dealing with dependent tasks from multiple workflow types and we propose a novel resource placement and scaling across edge cloud infrastructure using a micro-service performance model.

In terms of *resource scaling of a cloud application*, there exists related work [50–53] on using predictive models (especially using machine learning techniques) to accurately predict performance and resource demand of applications to make informed decisions on resource scaling and reconfiguration. The main difference between those approaches and our approach is that we model the system performance at the granularity of *individual microservices* in order to support *both* resource scaling and computation placement decisions.

CHAPTER 4: OVERVIEW OF DOSSIER

As motivated in Chapter 1, this thesis focuses on designing a novel distributed operating system for data cyberinfrastructure, named DOSSIER, that addresses the challenges in scientific data management and processing, system scalability and adaptability, and data acquisition when dealing with heterogeneous and dynamic scientific workloads. With DOSSIER, we take a holistic approach and tackle the challenges in all layers of the cyberinfrastructure stack: from runtime system, monitoring, to adaptation and applications.

An architectural overview of our DOSSIER is presented in Figure 4.1. At the lowest level is the set of distributed computing and storage resources, as well as scientific instruments that are parts of the cyberinfrastructure. The distributed resources are managed by a basic resource management layer, whose main objectives are to provide an abstraction of the resources and to support basic resource allocation capabilities (e.g., to allocate a task to a computing node that has available resource). On top of the resource abstraction layer is a runtime system layer that provides execution environment for scientific applications (accessed via API component), such as scientific data processing and analysis tasks. The monitoring component monitors the performance of runtime components, storage usage, progress of executing workflows. The collected monitoring data can be used by the resource scheduling and adaptation component to dynamically and adaptively schedule the system resources to meet certain performance guarantees and resource constraints from applications. Application layer provides users access to various features, such as data management, curation, and scientific workflow composition. Cross-layer security component handles user authentication and access control across different layers of DOSSIER. In the following, we briefly introduce each component.

Similar to traditional operating systems that manage entities including memory, devices, files, and processes, DOSSIER is a distributed operating system that manages a set of core entities: *tasks*, *workflows*, *instruments*, and *data files* (in addition to datasets and nested collections for file organization, similar to folder in traditional OS). In particular, DOSSIER keeps track of progress, requests, and resources associated with tasks and workflows. Instruments are registered when they are connected to DOSSIER, so that DOSSIER is aware of the type, network, and workload characteristics of the instruments (which are vital information during data acquisition phase with edge-based cloudlets). DOSSIER also keeps track of all information about owner, creation date/time, permissions, etc. of the files, datasets, and collections.

In terms of the *runtime system*, we present a novel microservice execution environment for heterogeneous scientific workflows. In particular, instead of using a traditional monolithic, centralized

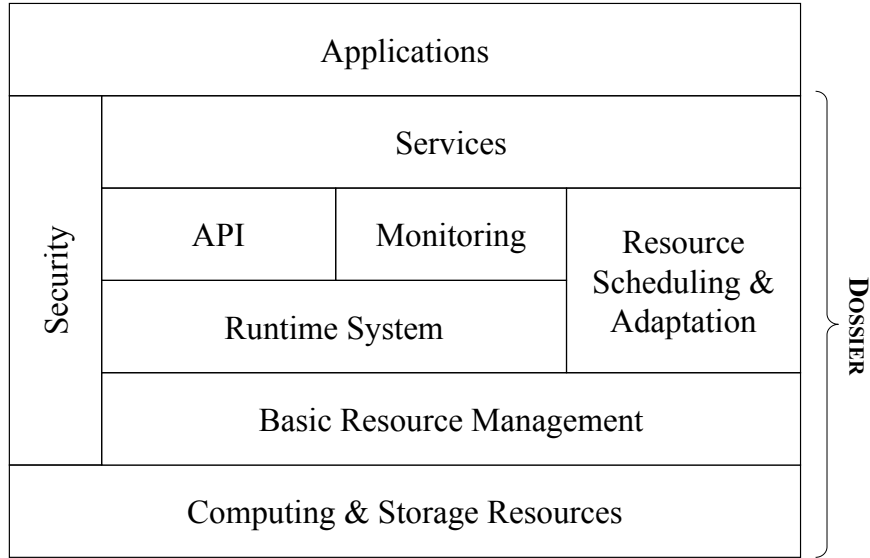


Figure 4.1: Architectural overview of DOSSIER.

task coordination approach, we design workflow execution mechanism based on a microservice architecture, where each task is modeled as a microservice with its own request queue and computing capability. With this design, we can separate complex task dependencies from the implementation of individual tasks, and thus, enable more scalable execution of heterogeneous workflows. We present the design of the microservice architecture in details in Chapter 5.

As motivated in Chapter 2, one of the key challenges for the data cyberinfrastructure is to help shorten time from data capture to insights. To address this challenge, we design a *adaptive control framework* for the microservice infrastructure to help the infrastructure to adapt with the dynamism of scientific workloads. At a glance, the framework consists of three main components: microservice monitoring (i.e., to monitor the actual performance of microservices), microservice performance model (i.e., to present formulation of the performance of microservices and provide near-future predictions of performance), and microservice resource adaptation (i.e., to leverage performance predictions to appropriately allocate system resources to microservices). We present a high-level overview of the adaptive control framework and how it is integrated with the microservice execution environment in Chapter 6.

From Chapter 7 to Chapter 9, we present different realizations of the adaptive control framework presented in Chapter 6. In Chapter 7, we present a white-box queueing network-based adaptation mechanism for microservice infrastructure. Specifically, we use a white-box queueing network model to model the performance of the microservice infrastructure and present multiple

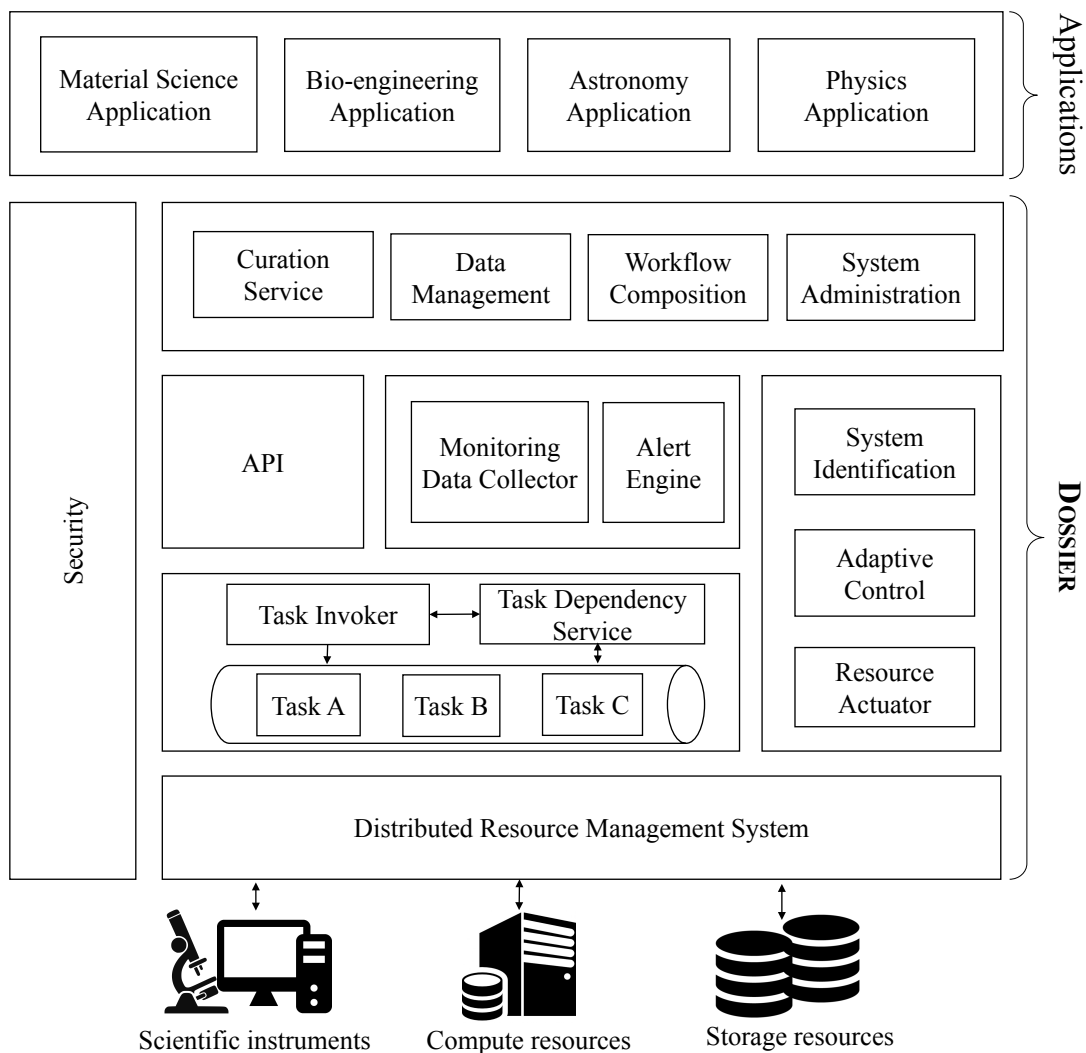


Figure 4.2: Detailed architecture of DOSSIER.

optimization-based microservice resource adaptation strategies that leverage that white-box model for near-future performance predictions.

In Chapter 8, we present MONAD, a novel self-adaptive mechanism for microservice infrastructure using control theory. In particular, we design a feedback control-based resource adaptation approach that is based on black-box neural network-based system model (and thus does not require any advanced knowledge of workflow structures like in white-box techniques) and employ a model predictive control-based resource adaptation mechanism that incorporates performance guarantees into adaptation objective to find optimal resource allocation strategies that satisfy resource budget constraints.

In Chapter 9, we present our novel model-based reinforcement learning approach for microservice resource adaptation. While the progress in (deep) reinforcement learning has shown great benefit in a variety of application domains (e.g., training AI agents to outperform human players in a lot of computerized games), we are one of the first to study the application of reinforcement learning, the model-based technique in particular, in the context of distributed system and networking domain. We will show in Chapter 9 how different concepts of reinforcement learning, such as episode, rewards, environment, policy, can be mapped to our targeted environment of distributed microservice infrastructure, and how we can train a resource management agent to operate by itself to perform resource adaptation.

In terms of *services*, DOSSIER's open architecture enable a number of service building blocks can be incorporated to, including curation service, data management, workflow composition, and system administration to name a few. These services can be used to build domain-specific scientific data management applications. In Chapter 10, we validate the practicality of these services in materials-related application domain by developing 4CEED - a real-time data acquisition and analysis framework for material-related environment. In particular, 4CEED provides a streamlined curation service that helps users to perform nimble and adaptive data collection from material research instruments by wrapping of data with extensive meta-data in real-time and in a trusted manner. A number of data processing tasks are developed specifically for various types of material data to process data uploaded from instruments. In addition, 4CEED leverages DOSSIER' service building blocks to provide advanced data management, curation, and sharing of the collected data after they have been processed by the back-end service.

In Chapter 11, we present a novel edge-cloud architecture that extends the cloud-based microservice infrastructure to the edges to help connect and secure old scientific instruments, as well as to help cloud-based infrastructure (with limited resources) to better handle dynamic scientific workloads uploaded from a large number of instruments. We will show how our edge-cloud architecture enables seamless extension of DOSSIER's microservice architecture to the edges and allows monitoring and adaptation of microservices across cloud and edges.

In terms of *security*, although this is not the main focus of the thesis as we focus more on the *performance* aspect of distributed scientific data management (i.e., how to provide timely responses to acquisition, processing and curation requests of scientific data), we employ state-of-the-art and existing security techniques and best practices in DOSSIER, such as user authentication for accessing application services, permission control on user data, encryption for data transfer, to ensure that user data is secure and is only accessible to the ones that has permission.

CHAPTER 5: MICROSERVICE ARCHITECTURE FOR SCIENTIFIC DATA MANAGEMENT

Traditional workflow management systems (or WfMSs for short) often employ a monolithic approach in workflow implementation and execution. In particular, each workflow is implemented as a tightly coupled set of tasks and has its own workflow execution plan that specifies how to run the workflow on a distributed computation infrastructure. For example (cf. surveys in [10] and [11] for more details), *Pegasus* statically translates a workflow graph into an execution plan (e.g., including selecting sites for tasks to run and cluster tasks based on various criteria) and the plan cannot be changed once it is executed. In other systems, such as *Taverna*, *Triana*, or *Kepler*, all data movements and task submissions to grid infrastructure need to be explicitly specified and organized in the execution plan. In *Shock/AWE* [17], once being executed, the execution plan is often coordinated by a centralized server (and often, with a single task queue) that is in charge of task invocation and synchronization. This, the static, infrastructure-dependent execution plan creation, and centralized coordination mechanism make the existing systems less efficient in dealing with large-scale workloads of heterogeneous workflows.

In DOSSIER, we leverage the latest advances in cloud computing and virtualization technology to abstract away the infrastructure complexity (e.g., task allocation on actual servers/VMs is handled by a cluster management system, such as YARN or Kubernetes) and focus on the design of the workflow execution model. We exploit the fact that the tasks in scientific workflows are only data-dependent and different workflows often share common tasks. In addition, data processing tasks are often quite simple and not algorithmic-heavy (e.g., extracting meta-data from raw file, generating preview from images, indexing meta-data, etc. - see Chapter 2 for more examples). Therefore, instead of the traditional monolithic approach, we model scientific data processing tasks as *microservices*, and separate workflow's task dependencies from the implementations of individual tasks. This microservice approach enables more flexible and scalable workflow composition and resource scheduling (e.g., resource scaling can be done at the task level, instead of the whole workflow).

5.1 OVERVIEW OF MICROSERVICE ARCHITECTURE

An architectural overview of the DOSSIER's microservice-based execution environment is presented in Figure 5.1.

In DOSSIER, the execution abstraction model is in terms of tasks (i.e., each task corresponds to a

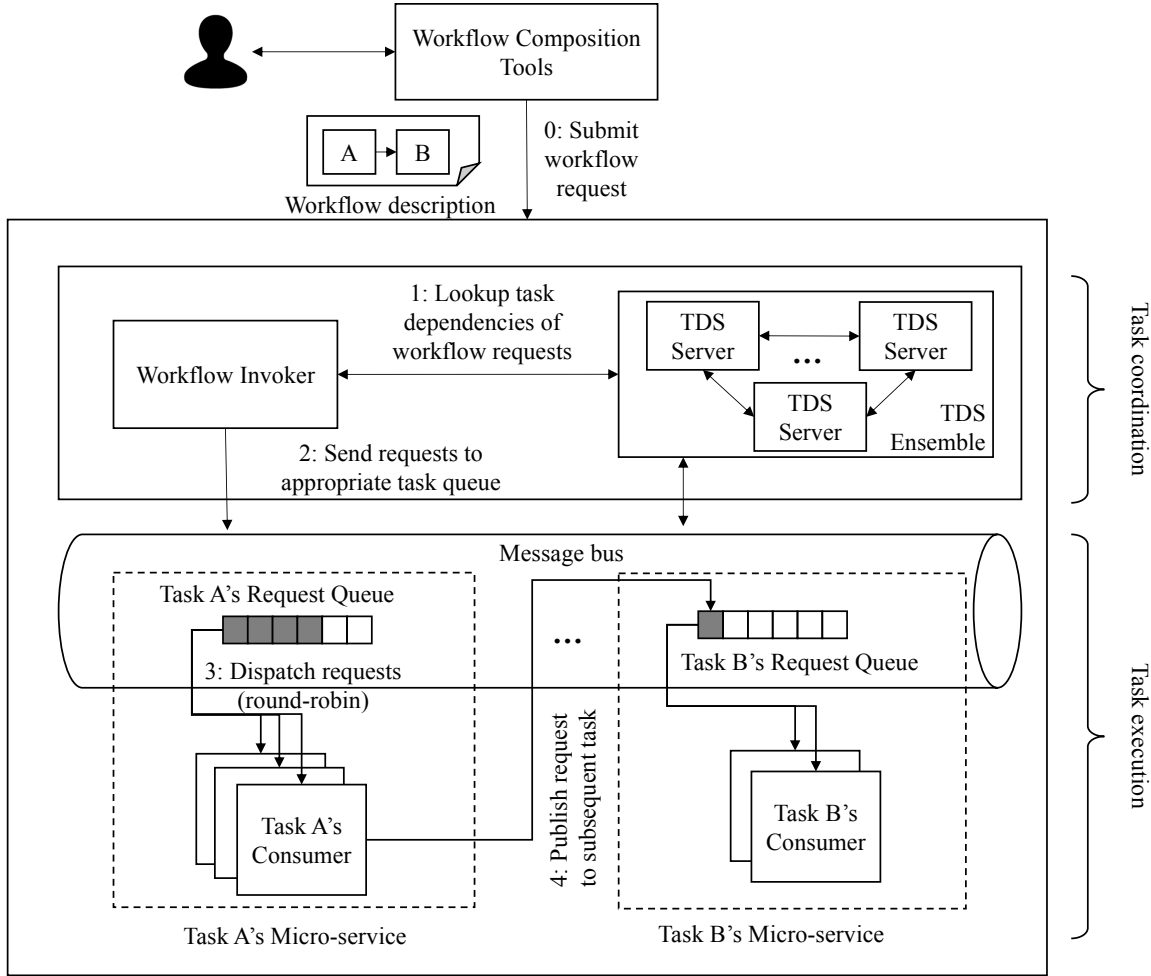


Figure 5.1: Design of workflow execution layer

data processing or data analysis task) and workflows (i.e., a Directed Acyclic Graph of tasks, which corresponds to a data processing job). Each task is modeled as a microservice that consists of a first-in-first-out request queue¹ and a set of consumers subscribing to the queue to handle requests. Task dependencies are maintained by a separate *task dependency service*² (or TDS). Figure 5.2 shows an example of a task dependency table of workflows type 1 & 2 maintained by TDS.

User interacts with the execution environment by submitting a workflow-based data processing request that includes input data, a *workflow type* (in case user requests for an existing workflow type in the system) or a *workflow description* (in case user composes his/her own workflow). A workflow description is presented in form of a task graph's edge list. Each workflow type has a corresponding workflow description that is already stored in TDS. Workflow input data (and all

¹In DOSSIER, the task requests are non-preemptive, and we do not perform admission control on incoming requests. We leave these extensions for future work.

²Workflow's task dependencies are checked by TDS to make sure there is no cycle in the workflow.

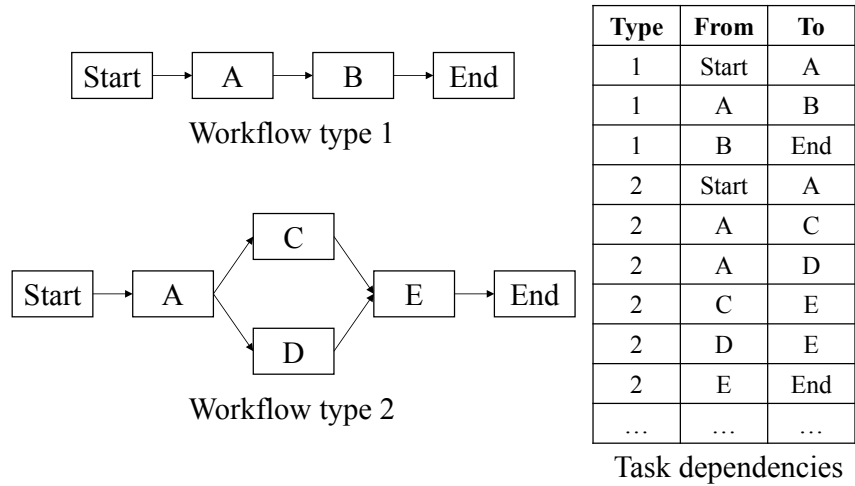


Figure 5.2: Example of workflow types and the corresponding task dependencies

intermediate results) are stored in a shared storage system that can be accessed by all tasks.

Whenever a workflow request arrives,³ the *task invoker* asks the TDS which task of the workflow should be processed first. Upon receiving response from TDS, given a request of workflow type 1, for example, the task invoker will send the request to task *A*'s request queue (i.e., the first task of workflow type 1) so that it can be processed by one of *A*'s consumers. Besides being a *subscriber* to its task request queue, each task consumer also acts as a *publisher* for other types of tasks following the workflow's task dependency graph. After a task consumer finishes processing a request, it will ask TDS about the subsequent task(s) of the workflow to "publish" the request to those tasks. For example, with a request of workflow type 1, after being processed by a task *A* consumer, the consumer will publish the request to task *B*'s request queue. The processing of the request ends when task *B*'s consumer is informed by the TDS that *B* is the last task of the workflow type 1.

5.2 TASK DEPENDENCY SERVICE

5.2.1 Task Synchronization

In addition to answering task dependency look-ups, TDS is also responsible for synchronization between tasks that run in parallel. For example, in the workflow type 2 (Figure 5.2), task *E* must wait for both tasks *C* and *D*, which can run in parallel, to finish before *E* can be processed.

³Only authorized users can send workflow requests to the system, and task invoker will check if the user has appropriate permissions to access the data required by the workflow request.

Because of the publish/subscribe mechanism, when C and D finish their work, they will publish the workflow request to task E 's queue. Let us assume that task C finishes before task D . The request published by C thus arrives first at task E 's request queue and is picked up by an E 's consumer. Since task E depends on the output of both C & D , the E 's consumer could not perform task E yet. Therefore, the E 's consumer creates a temporary *synchronization token* in TDS that holds status of E 's dependencies. For simplicity, we can consider the synchronization token as a *dependency counter* that is initialized as the number of dependencies a task has minus one (in our example, the token is initialized to be 1, since E has two dependencies). When task D finishes, it publishes the workflow request to E . Another E 's consumer that picks up the request published by D will check if a synchronization token for the request exists. Since a token has already been created, the consumer checks if the request is forwarded from the task's last dependency (i.e., current token value of 1). If not, the consumer decreases the token value by one and exits. In our example, since D is E 's last dependency, the consumer will proceed to perform actual processing of task E .

Algorithm 5.1 Task Synchronization Procedure

```

1: procedure synchronize(RID, TID)
2:   if |prev(TID)| == 1 then
3:     return True
4:   cur_token = get_token(RID, TID)
5:   if cur_token == null then
6:     create_token(RID, TID, |prev(TID)| - 1)
7:     return False
8:   else if cur_token == 1 then
9:     delete_token(RID, TID)
10:    return True
11:  else
12:    update_token(RID, TID, cur_token - 1)
13:  return False

```

We outline the synchronization procedure in Algorithm 5.1. The procedure `synchronize` is called at the beginning every time a task consumer processes a request. It takes two parameters, the request identifier `RID` and the consumer's task identifier `TID`, and it returns `True` if the synchronization is done (i.e., no more dependencies to wait for) and the task can be executed; and returns `False` otherwise. If there is only one dependency that task `TID` depends on (`prev` function returns the set of dependencies of a task), then there is no need for synchronization (Line 2-3). If this is the request forwarded from the first dependency of `TID`, a new synchronization token is created (Line 4-7). Function `create_token(RID, TID, |prev(TID)| - 1)` creates a new synchronization token for request `RID` at task `TID` and initialize it to `|prev(TID)| - 1`. If this is the request from the last dependency of `TID`, we delete the token from TDS and return `True` so that task `TID` can be processed for request `RID` (Line 8-10). Otherwise, we update the token on TDS (i.e., decrease its value

by `one - update_token(RID, TID, cur_token - 1)` and continue to wait for all dependencies to finish (Line 11-13).

5.2.2 Scalable Task Dependency Service

Since TDS is involved every time a task consumer is invoked (i.e., to perform synchronization), or during task dependency lookup, it is vital that the TDS is highly available and able to quickly respond to a large number of requests at the same time. To offer high availability and high performance, we designed TDS as an ensemble of multiple TDS servers and maintain a replica of task dependencies data on each server. For *read* requests (e.g., dependency lookup requests, token retrieval), any of the TDS servers can respond using its own local replica of task dependencies. Therefore, reads are quick and scalable. For *write* requests (i.e., for creating, updating synchronization tokens, and updating workflow’s task dependencies for applications such as dynamic workflow composition), to guarantee consistency across multiple TDS servers, we use a *quorum-based* write mechanism with leader election. Specifically, one server from the set of TDS servers is elected as the leader. When a write request is sent to a server, the server passes on the request to the leader. This leader then issues the same write request to all other TDS servers. The write request is deemed successful only if a strict majority of the servers, or a quorum, responds successfully to this write request.

5.3 MICROSERVICE RESOURCE MODELS AND ASSUMPTIONS

We finish off this section by describing various assumptions, performance metrics, guarantees, and microservice resource models used by DOSSIER.

5.3.1 Microservice Resource Model

Let us assume that the supported N workflow types compose of J types of tasks (i.e., each workflow type corresponds to a DAG of a subset, or all, of J types of tasks). We model each task type j ($1 \leq j \leq J$) as a microservice that handles requests of the task type j . Specifically, the microservice consists of a *request queue* that stores the task’s requests, and a set of uniform *task consumers*⁴ that subscribe to the request queue to perform actual processing of the task’s requests.

⁴Consumers of a task have uniform computational capacity, in terms of CPU and memory, and this low-level resource information is abstracted away by the cloud infrastructure. Hence, the WfMS only needs to control the number of consumers for each task and task consumers become the computational representation of resource.

A workflow request is processed by multiple micro-services that correspond to the tasks in the workflow. Micro-services communicate with each other via a publish/subscribe middleware⁵.

We denote the configuration of the numbers of consumers over tasks during time window (T_k, T_{k+1}) as $\mathbf{m}(k) = (m_1(k), m_2(k), \dots, m_J(k))$, where $m_j(k)$ is the number of consumers of task type j during the k -th time window. Since the more consumers subscribe to a task's request queue, the more requests can be processed in parallel (and the less time requests must wait in the queue), $\mathbf{m}(k)$ influences task's and workflow's processing times. Hence, we use $\mathbf{m}(k)$ to represent *resource allocation decision*⁶ to be made by the system, so that it can adapt with the dynamism of incoming workload to satisfy various performance guarantees.

5.3.2 Performance Metrics

At workflow level, we are measuring the *average processing time* (or average delay) of each workflow type i ($1 \leq i \leq N$), as well as the average delay over all types of workflows. The *processing delay* of a workflow request is defined as the duration between its arrival time t and the time when the workflow's last task is finished. The average delay of workflow type i over the time window (T_k, T_{k+1}) , denoted as $d_i(k)$, is calculated by averaging delays of all requests of type i that arrive during (T_k, T_{k+1}) . We denote $\mathbf{d}(k)$ as the vector form of the set of all average delays of workflow types in the k -th time window: $\mathbf{d}(k) = (d_1(k), d_2(k), \dots, d_N(k))$. The average delay of requests over all types of workflows in the time window (T_k, T_{k+1}) is denoted as $\bar{\mathbf{d}}(k)$.

At task level, the processing delay of a workflow request when it is processed by a microservice is measured from the time the request arrives at task's request queue until the request departs the microservice after being processed by one of the task consumers. As a result, the processing delay includes *both* the waiting time in the queue and the actual processing time by task consumer. Since, according to the Little's law⁷, this processing delay is proportional to the number of requests in the microservice (i.e., including requests waiting in the queue and requests being processed by task consumers), or *the number of work-in-progress*, also named work-in-progress or WIP for short. The more work-in-progress a microservice has, the longer delay is to be expected. We denote $w_j^e(k)$ as work-in-progress of task j ($1 \leq j \leq J$) on at location⁸ e ($0 \leq e \leq E$) during

⁵We assume that all workflow data and intermediate results between tasks are stored in a shared storage system that can be accessed by all tasks, and the data transfer times are included in the task processing time.

⁶From now, we refer to *resource allocation decision* as $\mathbf{m}(k)$ - the allocation of consumers over different task's micro-services.

⁷Wikipedia: https://en.wikipedia.org/wiki/Little%27s_law

⁸Location notation is used in case the infrastructure is distributed across cloud and multiple edges (as in Chapter 11). We refer to cloud-based infrastructure in case e is omitted.

time window (T_k, T_{k+1}) . We use $\mathbf{w}(k)$ as the vector representation of work-in-progress over all microservices during k -th time window.

5.3.3 Performance Guarantees

In this thesis, we use *absolute delay guarantee* for the average processing delay of individual workflow types (i.e., $\mathbf{d}_i(k)$) and of all workflow types (i.e., $\bar{\mathbf{d}}(k)$). Specifically, a delay threshold \mathcal{T}_i ($1 \leq i \leq N$) is assigned to each type of workflow i , so that the average delay of workflow type i over any time window (T_k, T_{k+1}) is guaranteed to be under the threshold: $\mathbf{d}_i(k) < \mathcal{T}_i$. Similarly, a delay threshold \mathcal{T} is used as the performance guarantee for all types of workflows: $\bar{\mathbf{d}}(k) < \mathcal{T}$.

CHAPTER 6: ADAPTIVE CONTROL FRAMEWORK FOR MICROSERVICE INFRASTRUCTURE

6.1 OVERVIEW

As motivated from Chapter 2, in order to shorten time from data capture to interpretation and insights, it is important to design DOSSIER to be adaptive with the dynamism of scientific workloads. Built upon the microservice infrastructure (Chapter 5), we employ a control-theoretic approach to design an adaptive control framework whose purpose is to dynamically adapt system resources to meet certain performance guarantees under changing workloads and limited resource capacity.

The framework (Figure 6.1) consists of three main components: A *monitor* that captures real-time performance of microservices, a *performance model* that provides near-future predictions of microservices' performances, and a *controller* that leverages performance predictions to make decisions on microservice resource allocation to maintain performance guarantees under resource constraints. Specifically, each time a performance guarantee is violated (e.g., $\bar{\mathbf{d}}(k)$ exceeds \mathcal{T}), based on the feedback er that captures the deviation of the performance metrics from the reference performance \mathcal{T} , the controller, with the help of performance model's predictions (i.e., $\hat{\mathbf{d}}(k+1)$), will explore different possible microservices' resource allocation (i.e., $\mathbf{m}(k+1)$) and decide on the optimal allocation of resources in the next time interval (i.e., $\mathbf{m}^*(k+1)$).

6.2 INTEGRATION OF ADAPTIVE CONTROL FRAMEWORK WITH MICROSERVICE INFRASTRUCTURE

The integration of adaptive control framework with the existing microservice infrastructure can be presented in a layered architecture as shown in Figure 6.2. It consists of three main layers (from

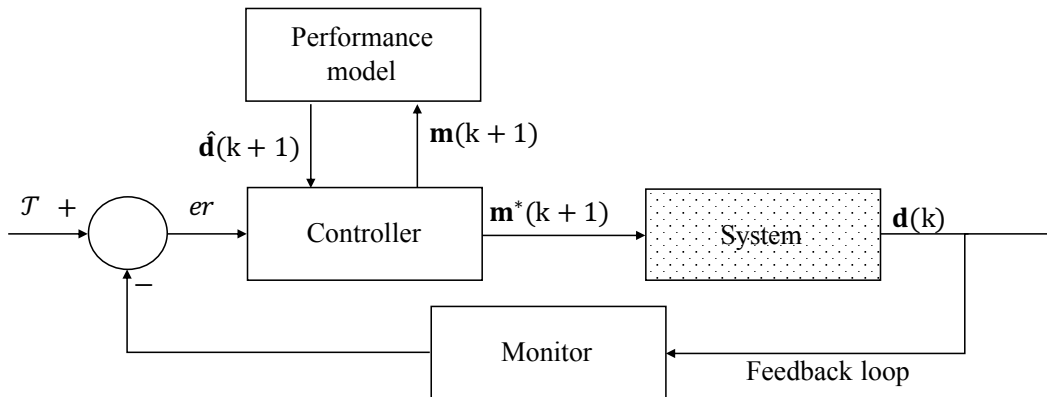


Figure 6.1: Adaptive control framework for microservice infrastructure

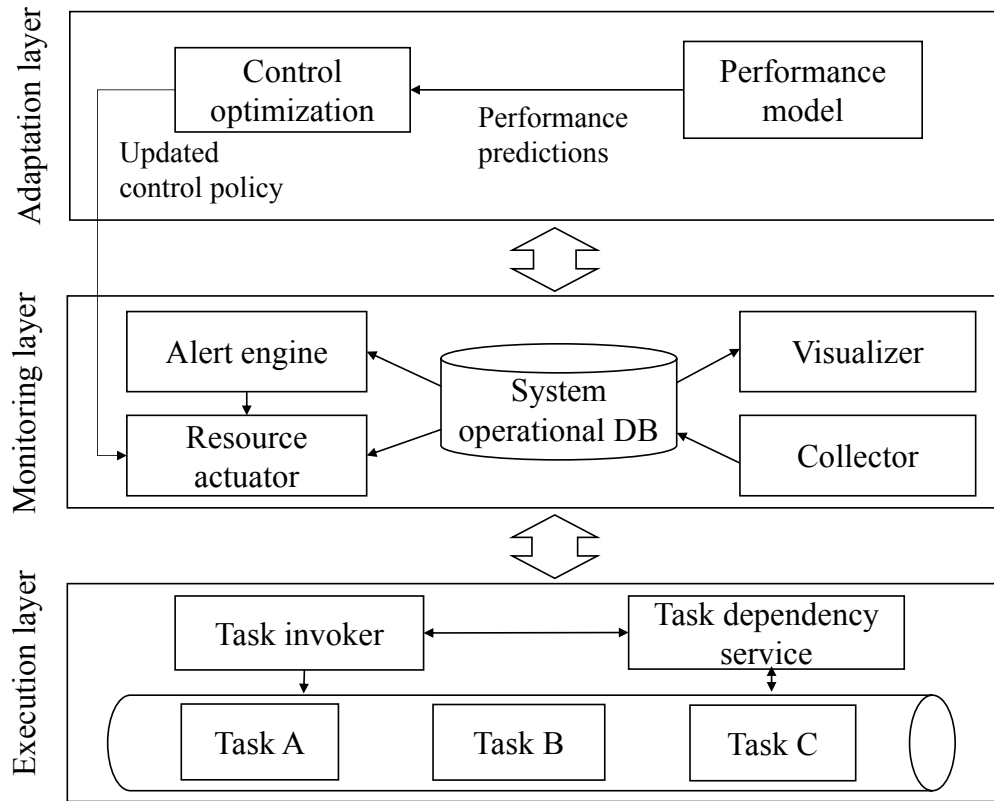


Figure 6.2: Integration of adaptive control framework with microservice infrastructure.

bottom-up): *microservice execution layer*, *monitoring layer*, and *adaptation layer*.

As presented in Chapter 5, the ***microservice execution layer*** is in charge of executing requested workflows. To deal with the heterogeneity of workflows, we designed the execution layer using a *microservice-based architecture*, in which tasks are modeled as *microservices* that interact with each other via event-based message passing mechanism.

The ***monitoring layer*** monitors the performance of workflows and task’s microservices (e.g., processing times, arrival rates of workflow requests of different types). The *collector* collects performance information and stores them in a time series database, which then can be presented to system administrators via a visualization interface (i.e., *visualizer*). The *alert engine* component periodically checks the performance information in the database and makes alerts if any performance guarantee is violated. To respond to alerts, *resource actuator* will consult with the adaptation layer (to be described) for resource allocation decisions and then, perform resource reallocation to adapt system performance.

The ***adaptation layer*** provides control decisions, which are in the form of resource allocations, so that the system can adapt to the dynamism of the incoming workload. The layer consists of a *performance model* component that provide near-future performance predictions, and a *control*

optimization component that leverages the performance predictions to optimize the allocation of resources under performance and resource constraints.

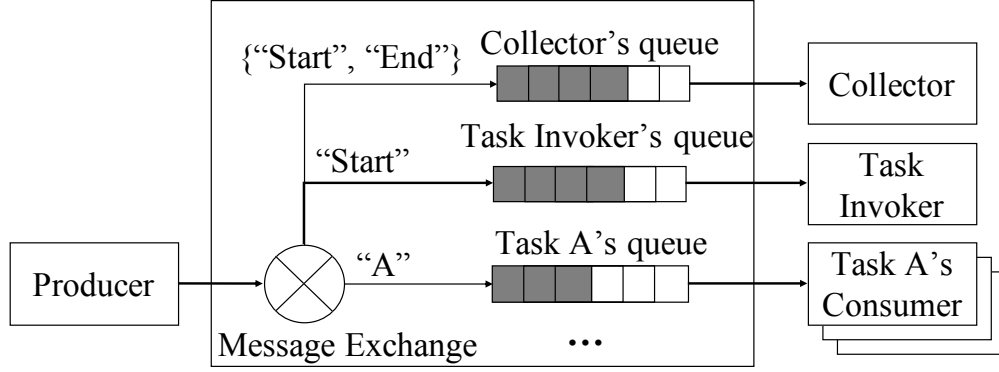
6.3 MICROSERVICE MONITORING

To support performance guarantees, it is important to monitor system states to respond to abnormal performance in a timely manner. In the following, we present the design of our monitoring layer that captures the system performance measures in a non-intrusive way.

At a glance, the monitoring layer consists of four main components: *collector*, *visualizer*, *alert engine*, and *resource actuator*. For each time window (T_k, T_{k+1}) , the collector collects information about system performance metrics (i.e., $\mathbf{d}(k)$), and the current allocation of resources (i.e., $\mathbf{m}(k)$). The collected information is time-stamped and stored in a time series database, called *system operational database*. Visualizer retrieves real-time performance data from the time series database and displays it to system administrators via an interactive Web interface. The alert engine periodically checks on key performance metrics from the database and triggers alert if any performance guarantee is violated (e.g., when $\bar{\mathbf{d}}(k)$ exceeds a processing time threshold \mathcal{T}). The triggered alert notifies resource actuator to consult adaptation layer to provide an updated allocation of resources (i.e., $\mathbf{m}(k+1)$), for the next time window (T_{k+1}, T_{k+2}) . Upon receiving $\mathbf{m}(k+1)$, the resource actuator will perform re-allocation of resources on the execution layer.

The main challenge for the monitoring layer is to be able to capture performance information in a non-intrusive way and with little or no modification to the implementation of applications. Often, the monitoring feature is implemented as part of the APIs and applications have to explicitly make calls to monitoring APIs to record their performance (e.g., calling monitoring APIs when the application starts and ends to record processing time). In DOSSIER, we leverage the publish/subscribe middleware used in the execution layer to design a monitoring service that *does not interfere with* to the performance and *does not require any modification* to the existing implementation of tasks and workflows.

Specifically, we leverage the subscription model of the Advanced Message Queuing Protocol (AMQP), the open standard that has been supported by most publish/subscribe middlewares, to perform non-intrusive performance monitoring. In the AMQP subscription model (cf. Figure 6.3), when messages arrive from publishers, they will be routed through an *exchange* to appropriate message queues. The routing decisions depend on the type of exchange used. In DOSSIER, we employ a *topic* type, in which the exchange routes messages to one or many queues (i.e., all queues receive the same copy of the message) based on matching between the message's *routing key* and



AMQP-based Publish/Subscribe Middleware

Figure 6.3: Leverage AMQP subscription model to perform non-intrusive monitoring

a *pattern* that was used to bind a queue to an exchange. As shown in the design of the execution layer (cf. Chapter 5), each task holds its own message queue (e.g., task A’s message queue is bound to the exchange to match messages with routing key “A”) and a set of consumers subscribed to its queue. We also use two pseudo tasks, i.e., “Start” and “End”, to respectively represent the invocation and the completion of each workflow (i.e., the task invoker essentially becomes the consumer of “Start” message queue). Using these two pseudo tasks, we introduce a separate message queue for the monitoring layer’s collector that is binded to all messages with routing keys “Start” or “End”. When collector processes a workflow request with routing key “Start”, it creates a new entry for the request in the time series database to mark its arrival. When collector processes a workflow request with routing key “End”, it updates the database entry of the started request to mark its completion, and records the processing time.

To capture statistics about arrival workload (i.e., the number of arrival requests of a workflow type over a time window), we simply perform an aggregation query over the time series database over that time windows to count the number entries having arrival time fall in between (T_k, T_{k+1}) . Then, performance metrics $\{d_i(k)\}$, $\bar{d}(k)$ can be calculated by averaging the processing times of requests in the time window (T_k, T_{k+1}) .

6.4 MICROSERVICE ADAPTATION

As described above, the two main components of the microservice adaptation layer are performance model and control optimization. In this thesis, we present and evaluate multiple solutions for microservice resource adaptation to tackle various diversity challenges mentioned in Chapter 2. Table 6.1 summarizes our solutions that are categorized by the control optimization technique used,

Thesis work	Controller	Performance model	Allocation decision	Diversity challenge
GRESMAN and 4CEED	Optimization-based heuristics	White-box	Scaling	Data, user, and task
MONAD	Model predictive control	Black-box	Scaling	Workload
BRACELET	Optimization-based heuristics	Black-box	Scaling & placement	Aging devices & workload
RL-MONAD	Reinforcement learning	Black-box	Scaling	Complex workload

Table 6.1: Realizations of the adaptive control framework.

types of performance model used to model performance of microservice infrastructure, types of allocation decision used for adaptation, and diversity challenges that the various solutions focus on. In the following chapters, we present in details these realizations of the microservice adaptation mechanism. In particular, in Chapter 7, we will present a white-box approach to model the performance of microservice infrastructure (i.e., GRESMAN). In Chapter 8, we will present a black-box neural network-based performance model of the microservice infrastructure and model predictive control-based approach for control optimization (i.e., MONAD). In Chapter 11, we present an adaptive control mechanism for edge-cloud architecture to tackle the diversity challenges in devices (scientific instruments) and workload. Ultimately, in Chapter 9, we present our recent results on using model-based reinforcement learning to train an agent that performs resource allocation by its own using a learned model of the microservice execution environment.

CHAPTER 7: QUEUEING NETWORK-BASED RESOURCE ADAPTATION

7.1 MODELING MOTIVATIONS

From the system architecture description in Chapter 5, it is intuitive to model each microservice as a *queue* (i.e., represented by the topic's message queue) with multiple *workers* (i.e., the subscribing consumers). In addition, microservices in the system are connected to each other because job requests are forwarded across the topics following the dependencies between tasks in a job. Hence, we can model the system as a *network of queues*, where each microservice is an individual queue in the network. Besides, as job requests can be of different job types, i.e., they arrive and then leave the system at different times, the queuing network model of the system is categorized as *multiple-class* and *open*.

By modeling the microservice execution environment as a multiple-class open queuing network (OQN), we are able to apply known results in queuing theory [54][55][56] to obtain the solution for the system's performance metrics. However, since there are numerous models that have been developed for OQN, choosing an appropriate one is non-trivial. While other related work that utilizes queuing network for performance modeling often opts out for simplified models to obtain analytical solutions, the results are limited by strong (and sometimes unrealistic) assumptions about the system, such as deterministic or exponential distribution of arrival rates of job requests and processing rates.

In this thesis, we decide to build our model based on more realistic assumptions. Particularly, we consider job request arrival rates and processing times at each topic, and both follow general distributions, represented by parameter sets $\Lambda = \{(\lambda_i, ca_i^2)\} (1 \leq i \leq N)$ and $\Gamma = \{(\mu_j, cs_j^2)\} (1 \leq j \leq J)$, respectively. Under these assumptions, each microservice is appropriate to a *GI/G/m* queue and the microservice infrastructure can be modeled as a *Generalized Multiple-class Jackson OQN* [54][55]. In the remaining of this chapter, we show how to leverage this model to obtain solution for performance metrics of the system.

7.2 MODELING PERFORMANCE OF MICROSERVICE EXECUTION ENVIRONMENT

Before analyzing our model using generalized multiple-class Jackson OQN, let us consider the special case, when job arrival rates and task processing rates are exponentially distributed (i.e., $ca_i^2 = 1, \forall 1 \leq i \leq N$ and $cs_j^2 = 1, \forall 1 \leq j \leq J$). In this case, each topic in the pub/sub system corresponds to a *M/M/m* queue. Because of the exponential distributions, we can aggregate all

λ_i	Expected arrival rate of requests for job type i to the system.
λ_{ij}	Expected arrival rate of requests for job type i at topic j .
ca_i^2	Squared coefficient of variant (scv), or variability, of arrival rate of job type i to the system.
$\tilde{\lambda}_j$	Aggregated job arrival rate of all job types at topic j .
\tilde{ca}_j^2	Aggregated scv of all job types at topic j .
Λ	Set of parameters representing system's workload: $\Lambda = \{(\lambda_i, ca_i^2)\}(1 \leq i \leq N)$.
μ_j	Expected processing rate of a task at topic j .
cs_j^2	Squared coefficient of variant (scv), or variability, of processing rate of a task at topic j .
Γ	Set of parameters representing system's computing capacity: $\Gamma = \{(\mu_j, cs_j^2)\}(1 \leq j \leq J)$.
$w_j(m_j)$	Expected number of work-in-progress requests for topic j (a function of m_j).
ν_j	Value of a job request at topic j .
$F_j(m_j)$	Cost of allocating m_j consumers subscribing to topic j .
\mathcal{M}	Resource cost budget.
\mathcal{T}	Work-in-progress, or equivalently, response time constraint.

Table 7.1: Notations used to describe parameters of queueing network model.

job types as a single type (since the combination of exponential distribution is also exponential). In addition, we can obtain the analytical solution of the expected number of work-in-progress job requests of a topic j (i.e., w_j) as a function of $\mu_j, \tilde{\lambda}_j, m_j$ following Erlang-C formula [54] (where $\tilde{\lambda}_j$ is the aggregated job arrival rate at topic j of all job types: $\tilde{\lambda}_j = \sum_{i=1}^N \lambda_{ij}$ with λ_{ij} is the expected arrival rate of job request type- i at topic j):

$$w_j^{M/M/m}(\mu_j, \tilde{\lambda}_j, m_j) = \frac{\tilde{\lambda}_j (\frac{\tilde{\lambda}_j}{\mu_j})^{m_j} \pi(0)}{(1 - \frac{\tilde{\lambda}_j}{\mu_j m_j})^2 m_j!} + \frac{\tilde{\lambda}_j}{\mu_j} \quad (7.1)$$

with:

$$\pi(0) = \left\{ \sum_{t=0}^{m_j-1} \frac{(\frac{\tilde{\lambda}_j}{\mu_j})^t}{t!} + \frac{(\frac{\tilde{\lambda}_j}{\mu_j})^{m_j}}{(1 - \frac{\tilde{\lambda}_j}{\mu_j m_j}) m_j!} \right\}^{-1}$$

For generalized case, since the job arrival rates and task processing rates are generally distributed, it is not possible to obtain exact analysis of w_j as in the special case. Hence, in this thesis, we employ an approximation method, named *parametric decomposition* [55], to measure the steady-state behavior solution for w_j . Specifically, for each topic j (in general case, is modeled as a GI/G/m queue), we can derive the aggregated job arrive rate and scv of all job types $\tilde{\lambda}_j$ and \tilde{ca}_j^2 respectively using parametric decomposition procedure.

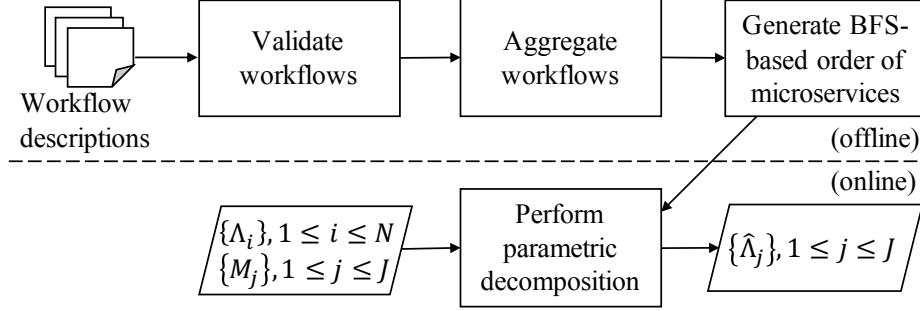


Figure 7.1: DECOMPOSE algorithm.

However, as the compute plane is modeled as a set of independent microservices, computing w_j requires decomposing $\{\Lambda_i\} (1 \leq i \leq N)$ to the aggregated arrival rate distribution of job requests at *each individual microservice*: $\{\hat{\Lambda}_j\} (1 \leq j \leq J)$. In this thesis, we present an algorithm, named DECOMPOSE, based on the parametric decomposition method, proposed by Vliet et al. [55]. The procedure is illustrated in Figure 7.1. It starts with loading and validating workflow descriptions, (which are stored in edge-list format that consists of a list of edges, specified by starting and ending vertex, of the workflow) of jobs that system supports to make sure that they are valid DAGs. After that, it merges all workflows into an aggregated graph, and then sorts the vertices in the aggregated graph (each vertex corresponds to a type of task or microservice) based on breadth-first search (BFS) ordering. Obtaining such an order of microservices can be done offline or periodically each time workflows are updated. Using this order, the procedure then performs parametric decomposition on each individual microservice (i.e., the BFS order is to ensure that a microservice is decomposed only after its precedent microservices have been decomposed). The parametric decomposition step is performed online as it takes real-time job arrival rates & processing time distributions to compute the aggregated arrival rate distribution at each microservice: $\{\hat{\Lambda}_j\} (1 \leq j \leq J)$.

With the aggregated rates and scvs, the expected number of work-in-progress job requests $w_j^{GI/G/m}$ is derived as an approximate function of $\tilde{\lambda}_j, \tilde{c}a_j^2, \mu_j, cs_j^2,$ and m_j . Among several good two-moment approximations of $w_j^{GI/G/m}$ that have been established for the GI/G/m queue [57], in this thesis, we use the common approximation formulation proposed in [58] that is based on an extension of the exact formula used in the M/M/m case:

$$\begin{aligned}
 & w_j^{GI/G/m}(\tilde{\lambda}_j, \tilde{c}a_j^2, \mu_j, cs_j^2, m_j) \\
 &= \frac{\tilde{\lambda}_j}{\mu_j} + \tilde{\lambda}_j \left(\frac{\tilde{c}a_j^2 + cs_j^2}{2} \right) (w_j^{M/M/m}(\mu_j, \tilde{\lambda}_j, m_j) - \frac{\tilde{\lambda}_j}{\mu_j})
 \end{aligned} \tag{7.2}$$

where $w_j^{M/M/m}(\mu_j, \tilde{\lambda}_j, m_j)$ is the expected number of job requests in progress of a M/M/m

queue as computed in Equation 7.1.

In Equation 7.2, we can consider $\tilde{\lambda}_j, \tilde{c}a_j^2, \mu_j, cs_j^2$ as given (i.e., either provided or calculated by parametric decomposition). Therefore, $w_j^{GI/G/m}$ becomes a function of m_j only, denoted as $w_j^{GI/G/m}(m_j)$.

Given the performance measure of individual topic $w_j^{GI/G/m}(m_j)$ obtained by Equation 7.2, the system performance measure (i.e., work-in-progress of the whole pub/sub system) can be calculated as a function of \mathbf{m} : $WIP(\mathbf{m}) = \sum_{j=1}^J \nu_j w_j^{GI/G/m}(m_j)$, where ν_j is the value of a job request at topic j . In the following, without any confusion, we use $w_j(m_j)$ to refer to $w_j^{GI/G/m}(m_j)$ for being concise.

7.3 MICROSERVICE ADAPTATION AS OPTIMIZATION PROBLEM

Let us consider a cloud-based pub/sub system that consists of J topics (i.e., supports processing J tasks) and accepts requests for N types of jobs, each job corresponds to a workflow of tasks supported by the pub/sub system (the summary of notations used in this chapter is presented in Table 7.1). For each type of job i , we assume that the arrival rate of requests follows a general distribution, denoted by expected rate λ_i and squared coefficient of variant (or *scv* for short) of the rate ca_i^2 . The set of parameters $\Lambda = \{(\lambda_i, ca_i^2)\} (1 \leq i \leq N)$ defines the system's workload.

In terms of computational parameters, for each topic j ($1 \leq j \leq J$), there are m_j (uniform) consumers subscribed to its message queue. For each consumer of a topic j , we assume that the time it takes to process the appropriate task follows a general distribution, denoted by expected processing rate μ_j and *scv* of the rate cs_j^2 . We assume that the processing rate parameters $\Gamma = \{(\mu_j, cs_j^2)\} (1 \leq j \leq J)$ depend on the implementation of consumers and task input data, and are given (e.g., by the collecting statistics of the processing time of completed tasks).

Since the workload and computational times could be considered as given, the numbers of consumers over topics $\mathbf{m} = (m_1, m_2, \dots, m_J)$ (which can be dynamically provisioned by exploiting the elasticity of the cloud infrastructure) are the main variables to measure performance of the elastic pub/sub system.

The system performance metrics include job's expected response time and cost of computational resources. Since response time is linearly related to the number of job requests being in the system (by Little's law), we use total *WIP* in the system as the performance metric. Particularly, *WIP* of the system is defined as $WIP(\mathbf{m}) = \sum_{j=1}^J \nu_j w_j(m_j)$, where ν_j and $w_j(m_j)$ are respectively the value of a work-in-progress of a topic¹ and the number of requests in progress per topic j

¹These values are used to penalize for work-in-progress of topics that have long average processing time. We

(if $\nu_j = 1, \forall j$, w equals the total number of jobs in the system). In terms of the resource cost, since in this thesis we consider allocating consumers over topics as the main resource allocation mechanism, the total resource cost depends on the number of provisioned consumers per topic and it is defined as $F(\mathbf{m}) = \sum_{j=1}^J F_j(m_j)$, where the function $F_j(m_j)$ (assumed to be given) is the cost of allocating m_j servers at station j . Since the performance of resource allocation algorithms depends on the shape of $F_j(m_j)$, we assume that $F_j(m_j) (\forall 1 \leq j \leq J)$ need to be a *non-decreasing convex function* of m_j . This assumption is reasonable since the resource cost increases as the number of consumers at a topic increases.

With the above notations and definitions, the resource management problem for cloud-based pub/sub system can be formulated as optimization problems. By using different objective functions for optimization problems, we allow users to flexibly choose between different resource provisioning strategies to suit their purposes.

For the first optimization problem, the objective is to minimize system's overall response time, or appropriately the w metric:

Problem Definition 7.1 (*Minimal Time Resource Allocation*) Given a cloud-based pub/sub system that supports N types of job and J different tasks (topics), a workload Λ , processing rates Γ , and a cost budget \mathcal{M} , find an optimal allocation \mathbf{m} of consumers to topics to minimize system's work-in-progress WIP :

$$\begin{aligned} \underset{\mathbf{m}}{\operatorname{argmin}} \quad & WIP(\mathbf{m}) = \sum_{j=1}^J \nu_j w_j(m_j) \\ \text{subject to} \quad & \sum_{j=1}^J F_j(m_j) = \mathcal{M} \end{aligned}$$

For the second optimization problem, the objective is to minimize the total resource cost of allocating consumers across topics:

Problem Definition 7.2 (*Minimal Cost Resource Allocation*) Given a cloud-based pub/sub system that supports N types of job and J different tasks, a workload Λ , processing rates Γ , and a WIP constraint \mathcal{T} , find an optimal allocation \mathbf{m} of consumers to topics to minimize system's total resource cost $F(\mathbf{m})$:

assume that these values are given.

$$\begin{aligned}
& \underset{\mathbf{m}}{\operatorname{argmin}} && F(\mathbf{m}) = \sum_{j=1}^J F_j(m_j) \\
& \text{subject to} && \sum_{j=1}^J \nu_j w_j(m_j) \leq \mathcal{T}
\end{aligned}$$

In order to solve the above problems, it is important to obtain the formulation for the performance metric WIP . In the next section, we will describe our approach to derive WIP of the elastic pub/sub system using queuing theory.

7.4 GREEDY RESOURCE ALLOCATION SOLUTIONS

With the formulation of system's performance metric WIP obtained from previous section, we now show how to efficiently solve the optimization problems described in previous section.

While we can view both optimization problems in Definition 7.1 and 7.2 as integer programming problems and apply standard solver to solve them, dynamic resource allocation for the system requires more efficient solutions. In this thesis, we propose greedy strategies to efficiently solve the optimization problems. In addition, by realizing the convex property of the objective functions, we are able to prove that the solutions by greedy algorithms are also the optimal solutions.

For the first optimization problem (Definition 7.1), by observing that $w_j(m_j)$ is a convex non-increasing function of m_j , $\forall 1 \leq j \leq J$ [59], we can solve the optimization problem in Definition 7.1 using a greedy strategy. Particularly, in Algorithm 7.1, each topic is initialized with one consumer, and then, the algorithm greedily finds the topic with the largest *benefit* if being allocated one more consumer. The benefit is defined to be proportional to the decrease of the number of work-in-progress job requests (i.e., $\nu_j[w_j(m_j^{i-1}) - w_j(m_j^{i-1} + 1)]$). The algorithm ends when it reaches the resources cost constraint \mathcal{M} .

Algorithm 7.1 Minimal Time Greedy Resource Allocation

- 1: **procedure** MINTIMEGREEDY
 - 2: Initial allocation \mathbf{m}^0 : $m_j^0 = 1, \forall 1 \leq j \leq J$
 - 3: $i = 1$ ▷ Initialize iteration count
 - 4: **while** $\sum_{j=1}^J F_j(m_j) < \mathcal{M}$ **do**
 - 5: Find $j^* = \operatorname{argmax}_{1 \leq j \leq J} \nu_j [w_j(m_j^{i-1}) - w_j(m_j^{i-1} + 1)]$
 - 6: $m_{j^*}^i = m_{j^*}^{i-1} + 1$ ▷ Add one consumer to most benefit topic
 - 7: $i = i + 1$
 - 8: **Return** \mathbf{m}^i
-

With the non-increasing convexity of $w_j(m_j)$, it can be proven that the solution of Algorithm 7.1 is also the optimal solution, based on Theorem 3 in [56].

For the second optimization problem (Definition 7.2), given the non-decreasing convexity of $F_j(m_j)$, $\forall 1 \leq j \leq J$ (as assumed) and the non-increasing convexity of $w_j(m_j)$, $\forall 1 \leq j \leq J$, we can again use the similar greedy strategy as in Algorithm 7.1 to find the optimal resource allocation solution. The minimal cost greedy resource allocation algorithm is presented in Algorithm 7.2.

Algorithm 7.2 Minimal Cost Greedy Resource Allocation

```

1: procedure MINCOSTGREEDY
2:   Initial allocation  $\mathbf{m}^0$ :  $m_j^0 = 1, \forall 1 \leq j \leq J$ 
3:    $i = 1$  ▷ Initialize iteration count
4:   while  $WIP(\mathbf{m}^i) \leq \mathcal{T}$  do
5:     Find  $j^* = \operatorname{argmax}_{1 \leq j \leq J} \frac{\nu_j[w_j(m_j^{i-1}) - w_j(m_j^{i-1} + 1)]}{F_j(m_j^{i-1} + 1) - F_j(m_j^{i-1})}$ 
6:      $m_{j^*}^i = m_{j^*}^{i-1} + 1$  ▷ Add one consumer to most benefit topic
7:      $i = i + 1$ 
8:   Return  $\mathbf{m}^i$ 

```

The main difference between Algorithm 7.2 and 7.1 is that, in Algorithm 7.2, the benefit of adding an additional consumer to a topic is defined to be inversely proportional to the increase in resource cost (i.e., $F_j(m_j^{i-1} + 1) - F_j(m_j^{i-1})$) and directly proportional to the decrease of the number of work-in-progress job requests (i.e., $\nu_j[w_j(m_j^{i-1}) - w_j(m_j^{i-1} + 1)]$) (Line 5). Based on Theorem 2 in [56], the solution by Algorithm 7.2 is proven to be “sufficiently close to the optimal solution”.

After decomposition, WIP , from a function of $\{\hat{\Lambda}_j\}$, $\{M_j\}$, and \mathbf{m} , becomes the function of \mathbf{m} only. Thus, the resource allocation problem becomes finding \mathbf{m} that minimizes system’s work-in-progress $WIP(\mathbf{m})$ while satisfying a cost constraint of the total resource cost $F(\mathbf{m})$.

To perform resource allocation **efficiently**, we present a *greedy elastic scaling algorithm*, denoted as GRESMAN, to dynamically find \mathbf{m} . In particular, the Algorithm 7.3 starts with the current configuration of consumers over microservices, and then, greedily finds the microservice with the largest *local benefit* if being allocated one additional consumer, denoted as $\Delta(m_j^i, m_j^i + 1)$. The notion of local benefit of a microservice is defined to be proportional to the decrease of the number of work-in-progress job requests of that microservice. The most beneficial microservice is added to a queue \mathcal{A} that maintains an ordered list of microservices being provisioned. The reason of having a queue \mathcal{A} is that order of allocation of consumers also affects system performance. As requests travel through the system following task dependencies, non-careful allocation order of consumers can cause bottleneck at a microservice if it is allocated with more consumer after its

precedent microservices. The algorithm ends when either system’s *WIP* is under a threshold \mathcal{T} (i.e., represent time constraint), or the total resource cost $F(\mathbf{m})$ reaches a certain budget \mathcal{C} .

Algorithm 7.3 Dynamic Greedy Elastic Scaling Algorithm (GRESMAN)

```

1: procedure GRESMAN
2:   Define  $\mathbf{m}^0$  as the current configuration of consumers
3:   Initialize allocation plan  $\mathcal{A} = []$ 
4:   Initialize iteration count  $i = 1$ 
5:    $\{\hat{\Lambda}_j\} = \text{DECOMPOSE}(\{\Lambda_j\}, \{M_j\}, \mathbf{m}^0)$  ▷ Initial decomposition
6:   while  $WIP(\mathbf{m}^0) > \mathcal{T}$  and  $F(\mathbf{m}) < \mathcal{C}$  do
7:      $\hat{j} = \text{argmax}_{1 \leq j \leq J} \Delta(m_j^i, m_j^i + 1)$  ▷ Find the most beneficial microservice
8:      $m_j^i = m_j^i + 1$  ▷ Add one consumer to that microservice
9:      $\mathcal{A}.\text{append}(\hat{j})$  ▷ Update allocation plan
10:     $\{\hat{\Lambda}_j\} = \text{DECOMPOSE}(\{\Lambda_j\}, \{M_j\}, \mathbf{m}^i)$  ▷ Update  $\{\hat{\Lambda}_j\}$ 
11:     $i = i + 1$  ▷ Update iteration count
12:   Return  $\mathcal{A}, \mathbf{m}^i$ 

```

In terms of complexity, the online component of the DECOMPOSE procedure requires to iterate over all vertices and edges in the aggregated graph. Therefore, the complexity of DECOMPOSE is $O(|V| + |E|)$, with V and E being the set of vertices (i.e., microservices and $|V| = J$) and edges (i.e., and interactions between microservices) in the aggregated graph. With GRESMAN, finding the most beneficial microservice (Line 7) and calculating *WIP* (Line 6 – executed once for each iteration) both require to iterate over all microservices. The number of iterations of the while loop depends on the convergence of *WIP* to under the time constraint \mathcal{T} , or the total resource cost (which equals the total number of consumers) reaches the budget constraint. For simplicity, if we assume that the cost constraint is met first, then the complexity of GRESMAN is $O(\mathcal{C} \cdot (|V| + |E|))$.

7.5 EVALUATION

7.5.1 Evaluation Settings

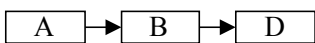
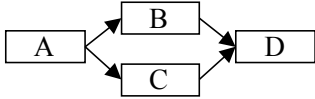
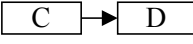
Implementation: We implemented our cloud-based elastic pub/sub system using RabbitMQ² as the message queue engine and Docker³ container technology as the implementation platform for consumers (for better isolation and server consolidation). Particularly, each consumer is implemented and encapsulated into a Docker image and subscribes to a RabbitMQ’s message queue

²RabbitMQ - <https://www.rabbitmq.com>

³Docker - <https://www.docker.com>

Task	Description	μ_j	cs_j^2
A	Unpacking digital microscope output files (e.g., DM3, HDF5)	4.2	0.33
B	Extracting and analyzing metadata from input file	3.7	0.5
C	Extracting and analyzing image from input file	6.7	0.4
D	Classifying the input file into appropriate experiment type & predicting if the experiment is successful or not	5.1	0.5

(a) Supported types of task.

Job type	Format	ca_i^2
1		0.33
2		0.5
3		0.25

(b) Supported types of job.

Figure 7.2: Tasks and jobs supported by the system.

of appropriate topic. We deploy the system on a cluster of three servers, each server is equipped with an Intel Xeon quad core processor (1.2Ghz for each core) and 16GB of RAM. We use Kubernetes⁴ as the Docker container orchestration engine for the cluster and each topic's consumer set is abstracted as a Kubernetes' ReplicationController. The resource manager (resource allocator in particular) interacts directly with Kubernetes to dynamically scale the size of ReplicationController (i.e., number of consumers) of each topic. All system components are implemented using Python programming language.

Case study: We take the application of executing scientific computing workflows as the case study. Particularly, the system supports analyzing experimental data generated by digital microscopes (which are usually in formats of DM3, or HDF5 files). Four types of task are supported, which correspond to the steps needed to process input data (Figure 7.2(a)). Depending on the input data, the system can support three different types of job, each job consists of all or a subset of supported tasks (Figure 7.2(b)).

Parameter settings: The processing rates of tasks are given in Figure 7.2(a). The scv of job arrival rates are given in Figure 7.2(b), while the expected arrival rates of each job type (i.e., λ_i) are varied

⁴Kubernetes - <http://kubernetes.io>

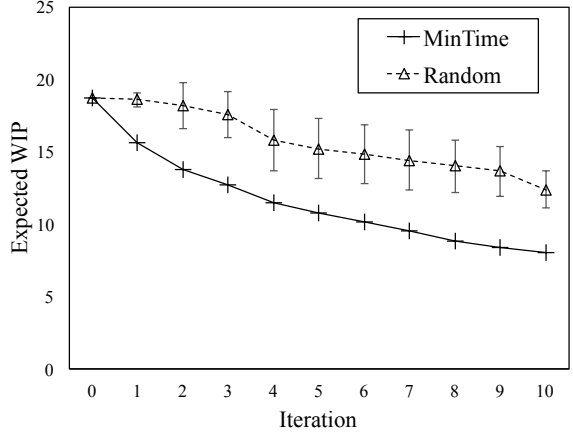


Figure 7.3: Numerical analysis comparison.

during the evaluation to represent changing workload. Note that the time unit we use for rates (i.e., processing time rate μ_j and job arrival rate λ_i) is *per minute*. To simplify the computation, we use a uniform resource cost function, i.e., $F_j(m_j) = m_j, \forall j$, and consider the job requests as equally important, i.e., $\nu_j = 1, \forall j$ ⁵.

In terms of comparing approach, we compare our resource management algorithms, named MinTime (Algorithm 7.1) and MinCost (Algorithm 7.2), with random resource allocation approach, named Random. In Random, for each iteration, a topic is randomly chosen to be allocated an additional consumer. To evaluate the performance of different algorithms, we initially allocate one consumer to each topic: $\mathbf{m} = (1, 1, 1, 1)$. Then, after each iteration (i.e., after a consumer is allocated to a topic), we measure the average response time of each type of job, as well as the average of all jobs. An algorithm is considered better if it achieves lower average response time after a given number of iterations (in case of minimal time allocation), or requires less iterations to reach a predefined response time threshold (in case of minimal cost allocation).

7.5.2 Numerical Analysis

First, we compare our algorithm, MinTime in particular, with Random using numerical analysis. Specifically, given a workload $\{\lambda_i\} = (3.0, 3.5, 3.0)$ and a cost constraint $\mathcal{M} = 10$ (since $F_j(m_j) = m_j$, \mathcal{M} is equivalent to the number of additional consumers allowed), we calculate the number of expected work-in-progress jobs in the system (i.e., $WIP(\mathbf{m})$) produced by each algorithm after each iteration (i.e., an iteration is equivalent to an additional consumer added). The

⁵Note that $F_j(m_j)$ and ν_j can be chosen in any form so that $WIP(\mathbf{m})$ and $F(\mathbf{m})$ maintain their non-increasing and non-decreasing convex properties.

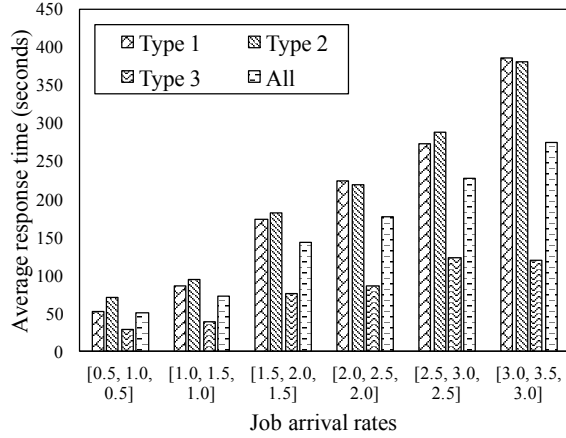


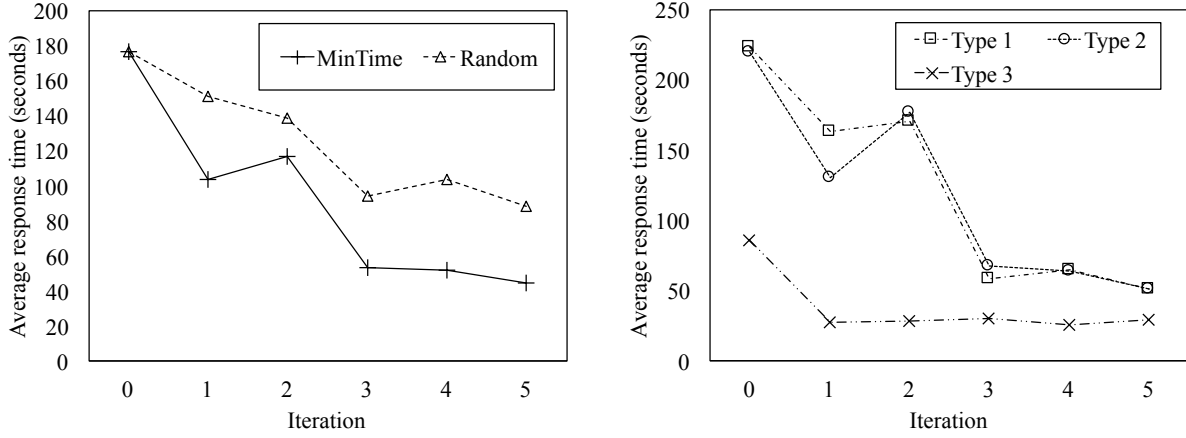
Figure 7.4: Average response time when incoming workload increase.

result in Figure 7.3 shows that MinTime outperforms Random by adding consumers to the most benefit topics, and thus helps reduce $WIP(m)$ at a faster rate. We observe similar result when comparing MinCost with Random.

We also notice that the result of Random can be different between different runs of Random algorithm (hence the error bars). Therefore, in the remaining of this section, we will use the Random’s best result after multiple runs.

7.5.3 Varying workload

We first evaluate the performance of the pub/sub system by varying the input workload. In this experiment, we fix the number of consumers for each topic to be 1 (i.e., $m = (1, 1, 1, 1)$) and increase the arrival rates of different job types. The results in Figure 7.4 show that, as expected, when the arrival rates increase, the average response time of the system (averaging over each individual job type as well as over all job types) increases. This result suggests that, in order to maintain average response time under a certain level (e.g., QoS constraint), we need to provision the system resources (i.e., number of consumers subscribing to topics). In addition, in Figure 7.4, we also observe that the increases in the average response time of different job types are different. For example, job type 3 seems to be less affected by the increase of the arrival rates, compared with job type 1 and type 2. This suggests that, when provisioning the number of consumers at each topic, one should consider the differences in the sensitivity of different job types to the changing workload. This insight is also consistent with our motivation in designing the greedy resource allocation strategies, in which, we give higher provisioning priority to topic whose provisioning gives largest benefit.



(a) Average response time over all types of job. (b) MinTime's performance with different types of job.

Figure 7.5: Minimize time resource allocation comparison.

7.5.4 Minimal Time Optimization Task

For the Minimal Time Resource Allocation task, given a workload $\{\lambda_i\} = (2.0, 2.5, 2.0)$ and a cost constraint $\mathcal{M} = 5$, we perform resource allocation using MinTime and Random. We measure the performance of each algorithm over iterations. Figure 7.5(a) shows that, as two algorithms reach the cost constraint (i.e., after 5 iterations), MinTime outperforms Random by achieving a lower average response time of all types of job. Although Random's allocation helps reduce the response time at some degree, it could not achieve optimal result due to its randomization in selecting topics for provisioning. In addition, MinTime also performs well with individual types of job. Figure 7.5(b) shows that the average response time of each type of job quickly drop after just a few consumers are added to the system.

7.5.5 Minimal Cost Optimization Task

For the Minimal Cost Resource Allocation task, given a workload $\{\lambda_i\} = (3.0, 3.5, 3.0)$ and a response time constraint of 50 seconds: $\mathcal{T} = 50$, we perform resource allocation using MinCost and Random until the system average response time of all types of job smaller than or equal \mathcal{T} . The result in Figure 7.6 shows that MinCost satisfies the response time constraint in just 5 iterations (i.e., 5 additional consumers are needed). On the other hand, even though Random helps reduces the response time, it struggles in bringing down the response time to below \mathcal{T} , even after 10 iterations.

The results in both optimization tasks help confirm the effectiveness of using greedy strategy in selecting the topics for resource provisioning that maximize the overall benefit.

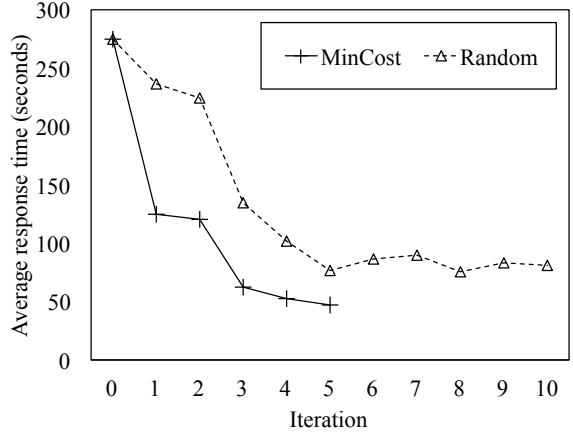


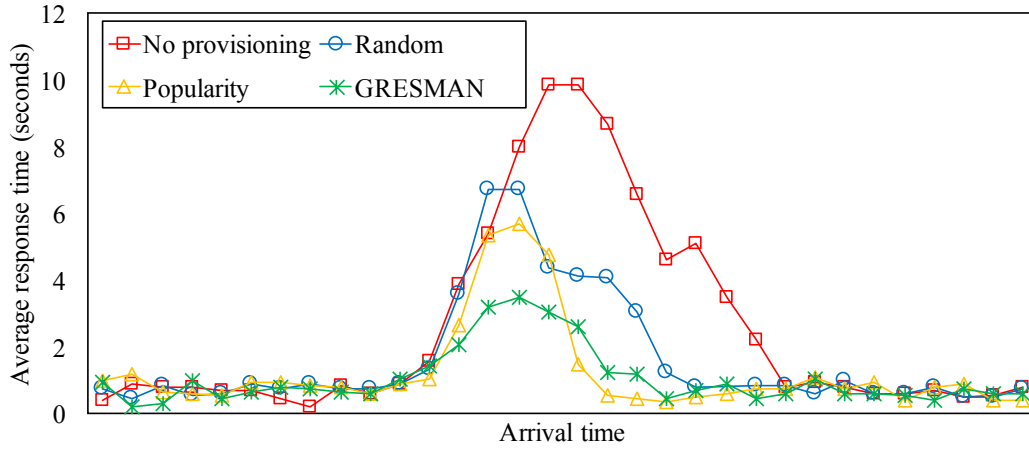
Figure 7.6: Minimize cost resource allocation comparison.

Effectiveness comparison of resource allocation strategies

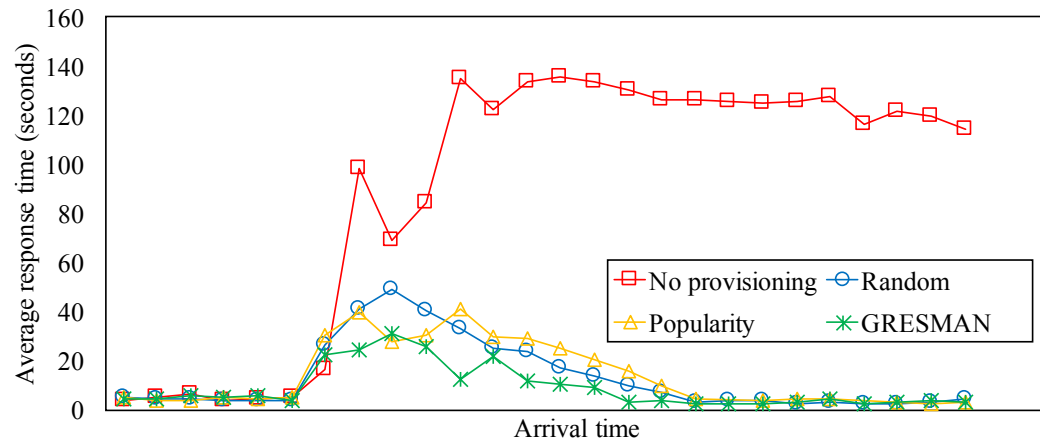
In this evaluation, we take the bursty workload situation described earlier and compare the effectiveness of our GRESMAN (Algorithm 7.3), with baseline resource allocation strategies: Random (which randomly assign consumers to microservices), and Popularity (which assign consumers to microservices proportional to the popularity of the task, or the number of workflows a task belongs to). The main metric for comparison is the average response time over all types of workflows.

In addition to the MDP workflows, we also use LIGO Inspiral Analysis workflows that analyze data from the coalescing of compact binary systems such as binary neutron stars and black holes. We initially allocate one consumer to each topic in MDP and LIGO workflows, except Inspiral & TrigBank tasks for their high popularity in LIGO’s workflows, and seek to minimize average job response time given a resource cost constraint of 10 and 60 additional consumers for MDP and LIGO workflows respectively. Resource allocation is kicked off when the resource manager observes that average job response times exceed a certain threshold (2 seconds for MDP, and 10 seconds for LIGO).

From the results in Figure 7.7, we can easily see the impact of the abnormal change in the number of arrival requests to the response time without provisioning in both workflow sets (the red line). The impact is more significant in LIGO case, since it is a more complex set of workflows with higher number of tasks and interactions between tasks. The results also show that, in both workflow set, our GRESMAN algorithm is more effective than the baselines in dealing with bursty workload situation, and the Popularity approach performs better than Random. That is because our approach can accurately capture the workload situation of each individual microservice and allocates additional consumers to the microservices that are most beneficial in bringing down the



(a) MDP workflows.



(b) LIGO workflows.

Figure 7.7: Effectiveness comparison of resource allocation approaches in bursty workload situation.

response time. On the other hand, random allocation does not consider the workload situation of individual microservice when allocating additional consumers, and Popularity approach relies on a simple heuristic based on the popularity of microservices to decide its allocation.

Scalability

To evaluate the scalability of the coordination service, we vary the arrival rates of job requests and measure the number of consumers that need to be provisioned (the allocation of consumers over microservices is generated by GRESMAN algorithm) so that the average response time of the system is kept under a certain threshold. We use both sets of workflows for this evaluation and set the response time threshold to 2 seconds for MDP and 10 seconds for LIGO workflows.

The result in Figure 7.8 shows that, as the arrival rate increases, the number of consumers that

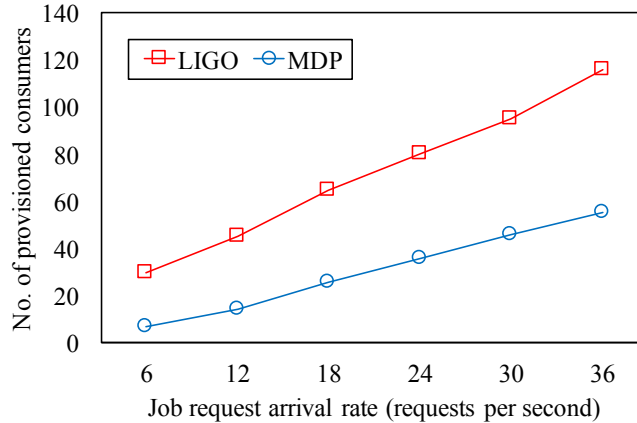
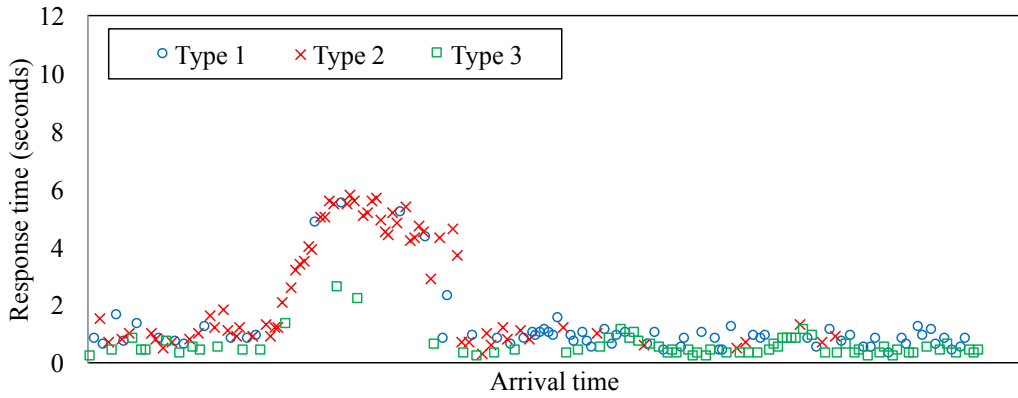


Figure 7.8: Scalability of coordination service by varying arrival rates of job requests.

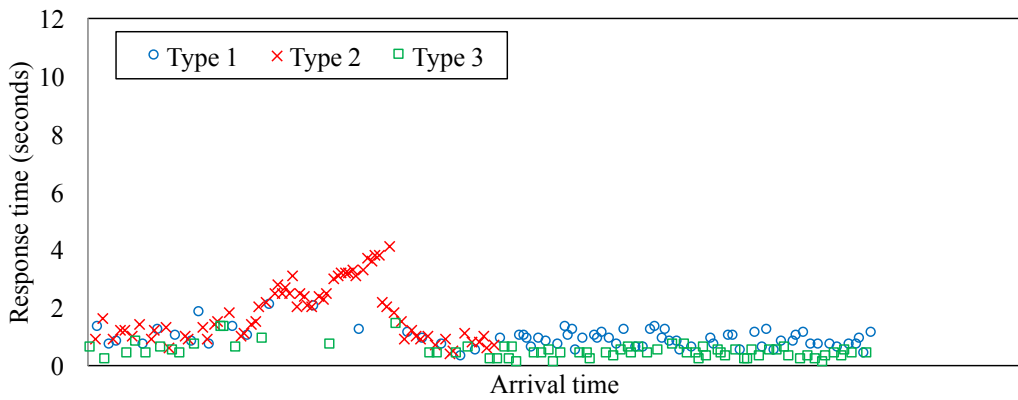
coordination service needs to provision must increase to maintain the quality of service (i.e., overall response time under a certain threshold). However, we also see that, in both sets of workflows, the required number of consumers is almost linear to the job request arrival rates. This is acceptable and allow us to further scale the coordination service to adapt to a high number of incoming requests. In our evaluation, with a modest setup of three-node cluster (described earlier), we can provision up to 150 consumers, each running as a Docker container, using Kubernetes without any performance issue.

Impact of resource order on provisioning overhead

As we have seen in Figure 7.7, it still takes some time for resource provisioning to take full effect and put the response time back to normal. One of the main reasons is due to the overhead of starting up provisioned consumers. This overhead is difficult to avoid without advance knowledge of arrival workload. Another reason, as we find out in our experiments, is due to the provisioning order of resources. Let us again take the bursty workload situation with MDP set in previous section as an example. In this case, our GRESMAN algorithm decides new optimal provisioning strategy as $\mathbf{m} = (m_A, m_B, m_C, m_D) = (2, 3, 2, 7)$. If we simply provision the microservices in the sequential order, e.g., first start with A , then B , C , and D , it might put D , which is the one needs additional consumers the most, under more stress, because other microservices have their consumers provisioned earlier and start sending more requests to D . To reduce the effect of the second overhead, we can carefully order the provisioning of topics. In particular, we leverage the allocation plan \mathcal{A} obtained from the GRESMAN algorithm (Algorithm 7.3), which specifies the provisioning order additional consumers to microservices that is most beneficial in bringing down the overall response time. Figure 7.9(b) shows the results of the carefully ordered provisioning,



(a) Effect of resource provisioning using GRESMAN on different job types.



(b) Carefully ordering provisioning of resources further improves provisioning effectiveness.

Figure 7.9: Impact of resource provisioning order in handling bursty workload with MDP workflows.

compared with the results by using sequential provisioning order (i.e., A, B, C, D) in Figure 7.9(a), when they both use the same optimal allocation $\mathbf{m} = (2, 3, 2, 7)$ generated by GRESMAN algorithm.

7.6 SUMMARY AND DISCUSSIONS

In this chapter, we present the first realization of the adaptive control framework introduced in Chapter 6. We model the system as a multiple-class open queuing network, a white-box approach, to derive system performance measures. Then, we formulate the resource management problem of the microservice workflow infrastructure, which can be considered as an elastic pub/sub system, as optimization problems using different objectives functions. Then, we introduce greedy algorithms to efficiently solve the optimization problems. Similar to other white-box approaches, this approach would work well in case that the distributions of incoming workflow requests follow Poisson distribution and the processing times of microservices follow general distribution. However, in case that the workflow workloads consist of a large number of tasks with complex interaction between tasks and in very dynamic workload situations, the cost of performing parametric decomposition, whenever the workload distributions change, will increase overhead and make the approach less practical.

CHAPTER 8: MONAD - MODEL PREDICTIVE CONTROL RESOURCE ADAPTATION

In this chapter, we present our work on MONAD that employs a control-theoretic approach in realizing the adaptation control framework (c.f. Chapter 6). Specifically, each time a performance guarantee is violated (e.g., $\bar{\mathbf{d}}_k$ exceeds \mathcal{T}), the adaptation layer is notified by the monitoring layer, and the controller, as part of the adaptation layer, will generate a new allocation of resources (i.e., \mathbf{m}_{k+1}) based on the feedback *er* that captures the deviation of the measured performance from the reference performance \mathcal{T} . There are typically two steps involved in developing a feedback control-based system: *system identification* and *controller design*. We present our solutions for each step in the remainder of this chapter.

8.1 SYSTEM IDENTIFICATION

In the system identification step, we develop a mathematical model of the system that we want to control using measurements of the system's input and output signals. Particularly, given a control input¹ (i.e., \mathbf{m}_k in our case) and the state of the system in the current time window (i.e., \mathbf{d}_k), the system model should be able to predict the performance of the system in the next time window (i.e., \mathbf{d}_{k+1}).

There are two common approaches to the system identification problem: *white-box* and *black-box* approach [60]. While the former assumes the understanding of how the system works internally (e.g., workflow structures) or the prior existence of a system model formulation; the latter assumes that no prior model or knowledge of internal system is available. Due to its realistic assumptions, the black-box approach is more desirable and it is our choice of approach in this thesis. In the following, we first present our design of the system model and then, the model training procedure.

8.1.1 System Model Design

There are many techniques to solve the system identification problem [60]. Capturing the performance model of complex dynamic systems, such as workflow systems, which are often non-linear and consist of multiple inputs, outputs with complex interactions, without knowledge of the system internals, requires techniques with good approximation power. In this thesis, we use *multilayer*

¹In the context of feedback control-based adaptation, we also use *control input* to refer to the allocation of resources.

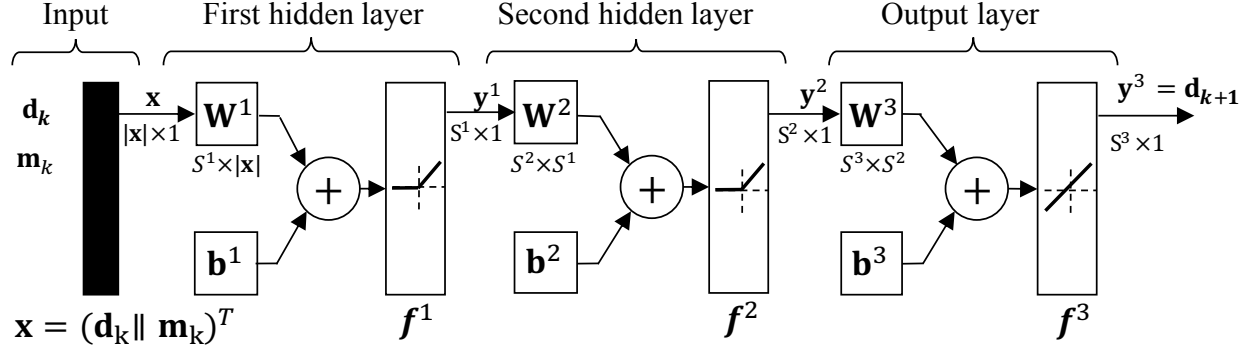


Figure 8.1: Neural network model of workflow system identification

neural networks, which have proven approximation power and have been applied successfully in the identification of dynamic and non-linear systems with multiple inputs and outputs.

Our neural network model of the workflow system is presented in Figure 8.1. The network consists of two hidden layers² and one output layer. The number of neurals in each layer is denoted as S^1 , S^2 , and S^3 respectively. Similarly, \mathbf{W}^i , \mathbf{b}^i ($i = 1, 2, 3$) represent the weight matrix and bias of each layer. Since most dynamic systems are non-linear, to introduce the non-linearity into the network model, we add a rectified linear unit (or ReLU) as the non-linear activation function (i.e., f^1 , f^2) on the output of the two hidden layers. The neural network model takes input \mathbf{x} as the combination of system states (i.e., \mathbf{d}_k) and control input (i.e., \mathbf{m}_k) in the current time window (T_k, T_{k+1}) , and predicts output as the system states \mathbf{d}_{k+1} of the next time window (i.e., (T_{k+1}, T_{k+2})). The neural network model can be presented as a matrix-based function of \mathbf{d}_k and \mathbf{m}_k :

$$\begin{aligned} \mathbf{d}_{k+1} &= \mathbf{f}(\mathbf{d}_k, \mathbf{m}_k) \\ &= \mathbf{W}^3 \mathbf{f}^2(\mathbf{W}^2 \mathbf{f}^1(\mathbf{W}^1(\mathbf{d}_k \parallel \mathbf{m}_k)^T + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3 \end{aligned} \quad (8.1)$$

8.1.2 Training System Identifier

The training procedure for system identifier's neural network model is presented in Figure 8.2. Specifically, the training process begins with the system model in an initial state and the control input is randomly generated. At the k -th time window (i.e., (T_k, T_{k+1})), the input of the neural network model is set as the combination of the current system states \mathbf{d}_k and system's control input \mathbf{m}_k . The neural network model can be trained using the backpropagation algorithm for

²The structure of the network and its parameters are decided empirically, as currently there is not yet a theoretical foundation for the design of neural networks.

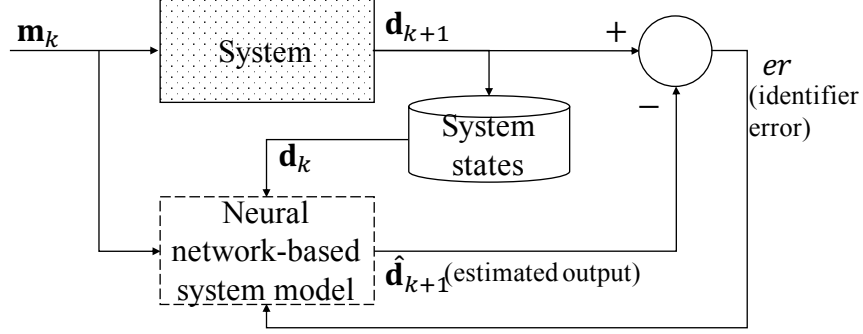


Figure 8.2: Training artificial neural network-based system identifier

feedforward neural networks. The predicted next states of the system $\hat{\mathbf{d}}_{k+1}$ are compared with the output of the real system \mathbf{d}_{k+1} (i.e., the reference values), and identifier error er is calculated for each of the neurons in the output layer. After that, the error values are propagated backwards through the network to update the model's parameters $\{\mathbf{W}^i, \mathbf{b}^i\} (i = 1, 2, 3)$.

8.2 CONTROLLER DESIGN

The controller design step uses the system model learned from the system identification step (i.e., function $\mathbf{f}(\mathbf{d}_k, \mathbf{m}_k)$) to design a controller that can produce control inputs to guide the system to follow a desired output. The three most common approaches for controller design, given a learned neural network system model are: model predictive control, NARMA-L2 control, and model reference control [61]. To incorporate various performance and cost constraints into controller design, we employ the model predictive control methodology and treat the controller design problem as an optimization problem using the receding horizon technique [62].

Specifically, at the k -th time window, when the performance constraint is violated (e.g., $\bar{\mathbf{d}}_k$ exceeds \mathcal{T}), the controller seeks to produce new control inputs \mathbf{m}_{k+1} to guide the system back to comply with the performance constraints. Using the receding horizon technique, we solve a control optimization problem over fixed T future intervals, from time window $(k + 1)$ -th to $(k + T)$ -th, to obtain a sequence of the next T control inputs $\mathbb{M} = \{\mathbf{m}_\tau\}, k + 1 \leq \tau \leq k + T$ that minimize the deviations from the predicted values to the reference trajectory (i.e., \mathcal{T}) over T time windows while satisfying the resource budget constraint (i.e., \mathcal{C} denotes the maximum number of consumers in the system). The control optimization problem to solve at time k -th is formally defined as follows:

$$\begin{aligned}
& \underset{\substack{\mathbb{M}=\{\mathbf{m}_\tau\} \\ k+1 \leq \tau \leq k+T}}{\operatorname{argmin}} & \sum_{\tau=k+1}^{k+T} l(\mathbf{d}_\tau, \mathbf{m}_\tau) \\
\text{subject to} & \mathbf{d}_{\tau+1} = \mathbf{f}(\mathbf{d}_\tau, \mathbf{m}_\tau), k \leq \tau \leq k+T-1 \\
& \sum_{j=1}^J m_\tau^j \leq \mathcal{C}, k+1 \leq \tau \leq k+T \\
& \mathbf{m}_\tau \in \mathbb{Z}_+^J, k+1 \leq \tau \leq k+T
\end{aligned} \tag{8.2}$$

where $l(\mathbf{d}_\tau, \mathbf{m}_\tau)$ is defined as:

$$l(\mathbf{d}_\tau, \mathbf{m}_\tau) = \sum_{i=1}^N \lambda_i \cdot (\mathcal{T}_i - \mathbf{d}_\tau^i)^2 + \sum_{j=1}^J \mu_j \cdot (\Delta m_\tau^j)^2 \tag{8.3}$$

In the above optimization problem, $l(\mathbf{d}_\tau, \mathbf{m}_\tau)$ is the instantaneous cost function at time τ ($k+1 \leq \tau \leq k+T$) that captures the deviation of system output \mathbf{d}_τ from reference performance $\{\mathcal{T}_i\}$, in which \mathcal{T}_i is the reference performance specific to workflow type i , while also accounting for the control increments (i.e., $\Delta m_\tau^j = m_\tau^j - m_{\tau-1}^j$) over different types of resources. In case we use performance guarantee \mathcal{T} on the average processing time across all types of workflows, we can replace the first term in Equation 8.3 (i.e., $\sum_{i=1}^N \lambda_i \cdot (\mathcal{T}_i - \mathbf{d}_\tau^i)^2$) by $(\mathcal{T} - \bar{\mathbf{d}}_\tau)^2$ to capture the deviation from the reference performance \mathcal{T} .

The model predictive control-based adaptation algorithm, denoted as MPCAdapt, is presented in Algorithm 8.1. The algorithm starts by initializing the control sequence \mathbb{M} using the same current control input \mathbf{m}_k for all T future time horizons (Line 4-5), and the set of constraints (Line 6-9). After that, we solve the optimization problem (i.e., minimize function) with objective function `mpc_obj_func` described in problem (8.2) (Line 11).

It is easy to observe that the optimization problem (8.2) is a *constrained non-linear integer programming* problem (i.e., because decision variables $\{\mathbf{m}_\tau\}$ are positive integers and the objective function is in quadratic form with non-linear component $\mathbf{f}(\mathbf{d}_\tau, \mathbf{m}_\tau)$), whose complexity is NP-hard. To solve the problem *efficiently*, we relax problem (8.2) into a *constrained non-linear optimization problem* (i.e., by relaxing the integrality constraint of control inputs) and use the *Sequential Least Squares Programming optimization algorithm* (or SLSQP), an iterative method, to solve the relaxed problem. The non-integer solution then can be used to approximate the integer solution for the original problem.

After finding the sequence of control input, denoted as \mathbb{M}^* , only the first ‘‘control move’’ $\mathbb{M}^*[0]$ is returned and is used as the control input for the next time window \mathbf{m}_{k+1} . As we move to the $(k+1)$ -th time window, we repeat the same control optimization process (i.e., MPCAdapt) for the

next T time windows (i.e., from $(k + 2)$ -th to $(k + T + 1)$ -th). The process ends when the system satisfies the performance constraints.

Algorithm 8.1 Model Predictive Control-based Adaptation

```

1: procedure MPCAdapt( $\mathbf{m}_k, T, \{\mathcal{T}_i\}, \mathcal{C}$ )
2:    $\mathbb{M} = \{\}$ 
3:    $\mathbb{C} = \{\}$ 
4:   for  $\iota$  in  $[0, T - 1]$  do # Initialize  $\mathbb{M}$ 
5:      $\mathbb{M}.append(\mathbf{m}_k)$ 
6:   for  $\iota$  from  $[0, T - 1]$  do # Add cost constraints
7:      $\mathbb{C} = \mathbb{C} \cup \{\mathcal{C} - \text{sum}(\mathbb{M}[\iota]) \geq 0\}$ 
8:   for  $\iota$  from  $[0, T - 1]$  do # Add positive resource constraints
9:      $\mathbb{C} = \mathbb{C} \cup \{\mathbb{M}[\iota][j] \geq 0, \forall j \in [0, J - 1]\}$ 
10:  # Solve the optimization problem as described in (8.2):
11:   $\mathbb{M}^* = \text{minimize}(\text{mpc\_obj\_func}_{\{\{\mathcal{T}_i\}, T\}}, \mathbb{M}, \mathbb{C})$ 
12:  # Only return the first control move:
13:  Return  $\mathbb{M}^*[0]$ 

```

Even though we are able to solve the optimization problem (8.2) in an online manner by relaxation, it is still inefficient to run MPCAdapt on a high dimensional input and over a large number of time windows. Therefore, we also introduce a *heuristic-based dynamic control algorithm* (Algorithm 8.2) to efficiently produce control inputs for the system to meet performance guarantees while satisfying resource constraints. Specifically, different from MPCAdapt, in HeuristicAdapt, we only consider the next time window, instead of looking ahead T time windows, and we iteratively (and greedily) allocate additional resources to the task that produces minimal instant cost (i.e., `instant_cost` function, based on Equation 8.3), instead of trying to solve a global optimization problem.

8.3 EVALUATION

8.3.1 MONAD Deployment

In the following, we describe in detail our implementation of the MONAD system. We have deployed the MONAD system on a cluster of three servers, each server is equipped with an Intel Xeon quad core processor (1.2Ghz per core) and 16GB of RAM.

For the workflow execution layer, we implement TDS service using ZooKeeper and use a quorum of 3 TDS servers to maintain tasks dependencies data. We use RabbitMQ as the pub-

Algorithm 8.2 Heuristic Adaptation Algorithm

```
1: procedure HeuristicAdapt( $\mathbf{m}_k, \mathbf{d}_k, \{\mathcal{T}_i\}, \mathcal{C}$ )
2:   while  $\{\exists i \in [1, N] : \mathbf{d}_k^i > \mathcal{T}_i\}$  and  $\sum_{j=1}^J m_k^j \leq \mathcal{C}$  do
3:      $cur\_min = -1.0$ 
4:      $j\_min = -1$ 
5:     for  $j$  from 1 to  $J$  do
6:        $m_k^j = m_k^j + 1$ 
7:       if  $instant\_cost(\mathbf{d}_k, \mathbf{m}_k) < cur\_min$  then
8:          $cur\_min = instant\_cost(\mathbf{d}_k, \mathbf{m}_k)$ 
9:          $j\_min = j$ 
10:       $m_k^j = m_k^j - 1$ 
11:       $m_k^{j\_min} = m_k^{j\_min} + 1$ 
12:   Return  $\mathbf{m}_k$ 
```

lish/subscribe middleware and we implement each task’s micro-service as an ensemble of a RabbitMQ’s task request queue and a set of task consumers, each consumer is deployed as a Docker container. We use the round-robin dispatching mechanism for each task’s request queue so that multiple requests can be processed in parallel and each consumer receives a fair share of requests. In terms of fault tolerance, to make sure a request never gets lost (e.g., because of consumer crashes), we employ a message acknowledgment mechanism between request queue and its consumers: An acknowledgement is sent back from the consumer to tell the request queue that the consumer has finished processing a request.

We use Kubernetes as the container orchestration engine for the execution layer. With Kubernetes, we can abstract the set of consumers of each task’s micro-service as a Kubernetes’ Replication Controller, which helps ensure that, in the event of a container crash and server failure, a specified number of containers per task (or scaling factor) is always running at any time. Upon receiving a control input (i.e., \mathbf{m}_k) from the adaptation layer, the resource actuator simply instructs Kubernetes to change the scaling factor of each task’s replication controller.

For the monitoring layer, we use InfluxDB as the time series database to store the monitoring data, and Grafana as the visualization engine to provide real-time system performance status to administrators via an interactive interface. We use Kapacitor as the alert engine that monitors the time series database and invokes adaptation process whenever a performance guarantee is violated. Other components in monitoring layer are implemented using Python.

For the adaptation layer, we use Tensorflow to train system identifiers and use Python’s SciPy optimization package to solve the control optimization problem.

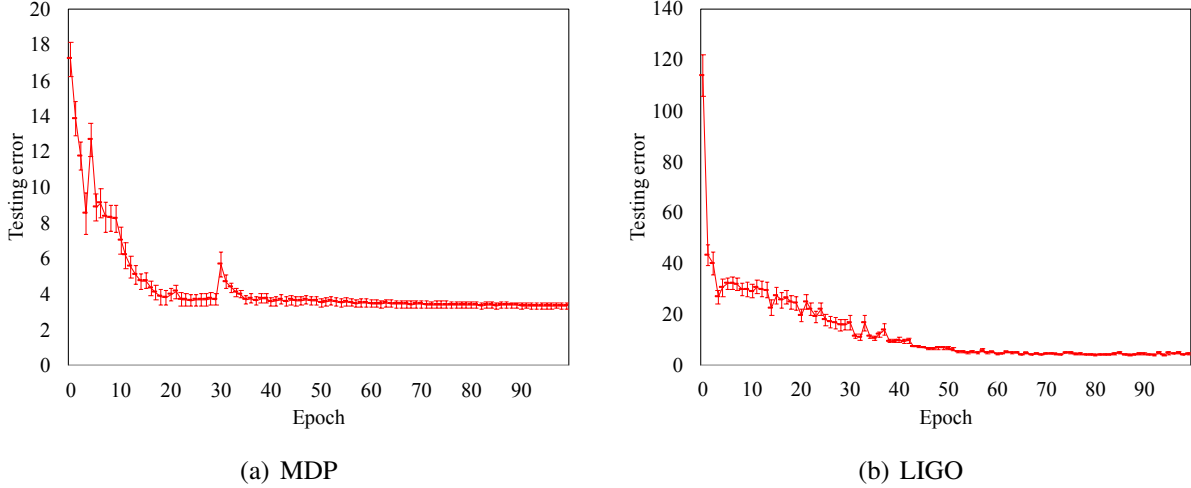


Figure 8.3: Testing performance of system identifiers

8.3.2 Evaluation Settings

Workflows: We use two workflow ensembles: the material science data processing workflows [45] (or MDP for short) that support processing experimental data generated by digital microscopes; and LIGO Inspiral Analysis workflows that analyze data from the coalescing of compact binary systems such as binary neutron stars and black holes. The MDP ensemble consists of 3 types of workflows and 4 types of tasks, and we use 3 complex workflows from LIGO: CAT, Full, and Injection, which consist of 7 types of tasks.

System Identification: To generate training and testing data for the system identification step, we randomly generate a workload by varying the arrival rates of requests of different workflow types and vary the allocation of resources over tasks. This way of generating data to train system identifiers allows us to capture a good variety of workload and resource allocation dynamics, so that our trained models can be better prepared for future workload situations. In addition, this allows us to collect the training data without the need of bootstrapping the system. The performance data (i.e., \mathbf{d}_k) and resource allocation (i.e., \mathbf{m}_k) are then captured by the monitoring layer and stored into the time series database for training. The data are aggregated over equal-length time windows. Via our experiments, we choose the window length (i.e., $T_{k+1} - T_k$) to be 10 seconds as it helps produce the best balance between the prediction accuracy and the data collection overhead (i.e., a too long time window might not capture the dynamism of workload, while a too short one might cause data aggregation overhead). Our collected dataset is aggregated from about 60K requests of MDP and about 15K requests of LIGO workflows. The data is then split using 80:20 ratio for training and testing.

In terms of the neural network’s model parameters, we set S^1 and S^2 equal to 32 neurals in each

hidden layer for MDP, 64 neurals for LIGO workflows (as LIGO has a higher input dimension). To train system identifiers for both sets of workflows, we set learning rate as 0.001, batch size as 100, and used 100 training epochs. Figure 8.3 shows the effectiveness of the trained system identifier using neural network model on two workflow ensembles when testing on unseen data.

Resource Adaptation: To evaluate the effectiveness of different adaptation algorithms in handling varying and bursty workload situations (i.e., workload with abnormally high arrival rate of requests), we emulate the bursty workload situation by abnormally increasing the arrival rates of requests of different types of workflows to up to 10 and 5 times compared with the normal rates on MDP and LIGO workflows (respectively), and measure the effectiveness of our resource allocation strategy. In this evaluation, we use the absolute delay guarantee for \bar{d}_k : $\bar{d}_k < \mathcal{T}$, and set \mathcal{T} equal 10 and 30 seconds for the MDP and LIGO workflows respectively. We set the resource constraint \mathcal{C} for the MDP and LIGO workflows to be 15 and 90 maximum number of consumers respectively.

For the MPCAdapt algorithm, we set $T = 5$, and use a uniform cost for resources $\mu_j = 1 \forall j$ and the same weight for all workflow types $\lambda_i = 1 \forall i$ ³.

8.3.3 Effectiveness of System Identification

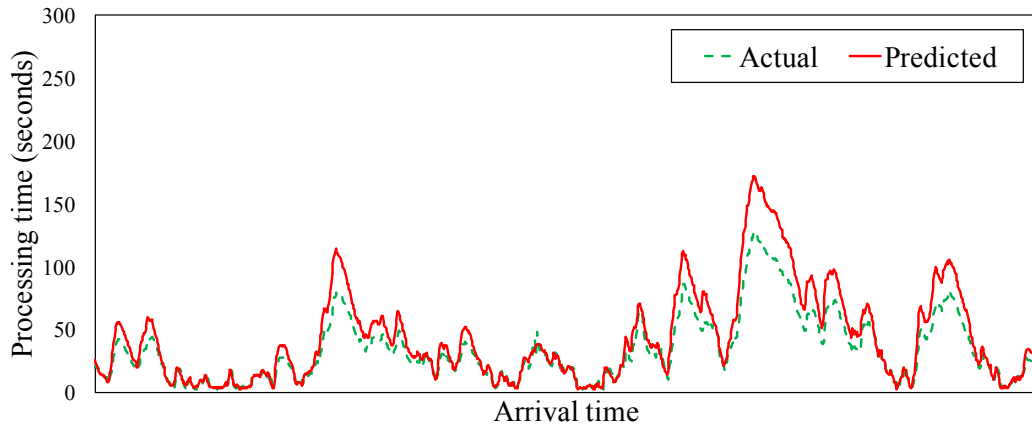
Figure 8.4 shows clearly the effectiveness of the trained neural network-based system identifier on accurately predicting the average processing delay over all workflow types on both MDP (cf. Figure 8.6(a)) and LIGO (cf. Figure 8.4(b)) workflow ensembles.

Further study of the results shows that our system identifier also performs well on predicting the processing time of *individual* workflow types, as shown in Figure 8.5 on two of MDP’s workflows. Although these MDP’s workflows pose different workload and performance characteristics, the system identifier can accurately predict the processing time of each type. These results help verify the effectiveness of using neural network-based model with multiple outputs, one for each workflow type, to capture the dynamic system model with complex interactions between tasks across different workflow types.

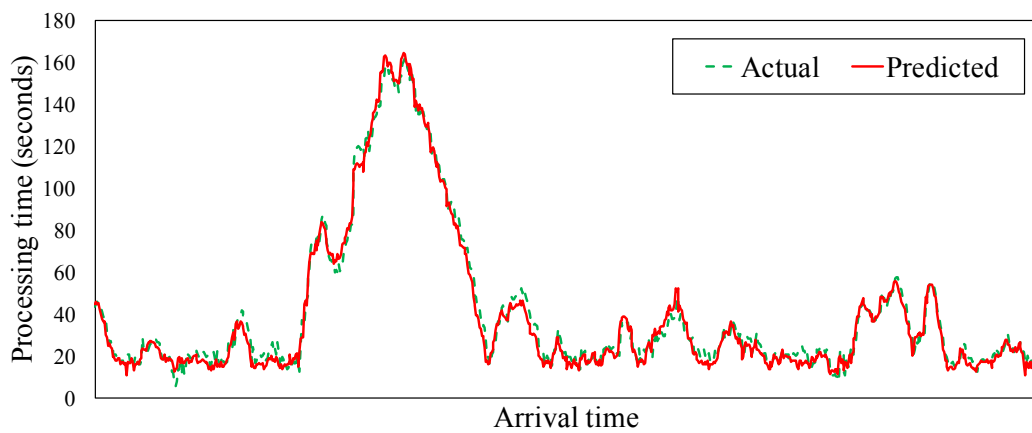
8.3.4 Effectiveness of Adaptation Algorithms

As we can see in Figure 8.6, the MPCAdapt algorithm outperforms HeuristicAdapt as it helps quickly neutralize the effect of abnormal and bursty workload on the performance of the system.

³The costs and weights can be easily set to more realistic values (if available) without affecting the performance of the algorithm.



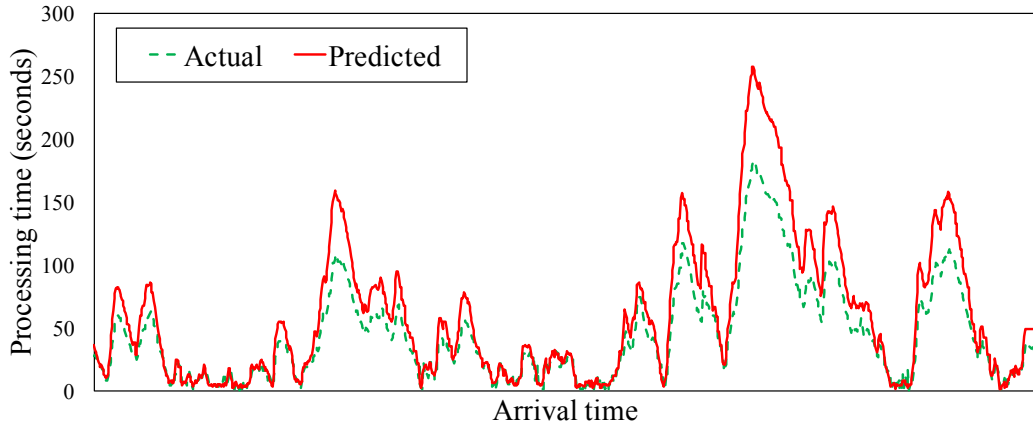
(a) MDP



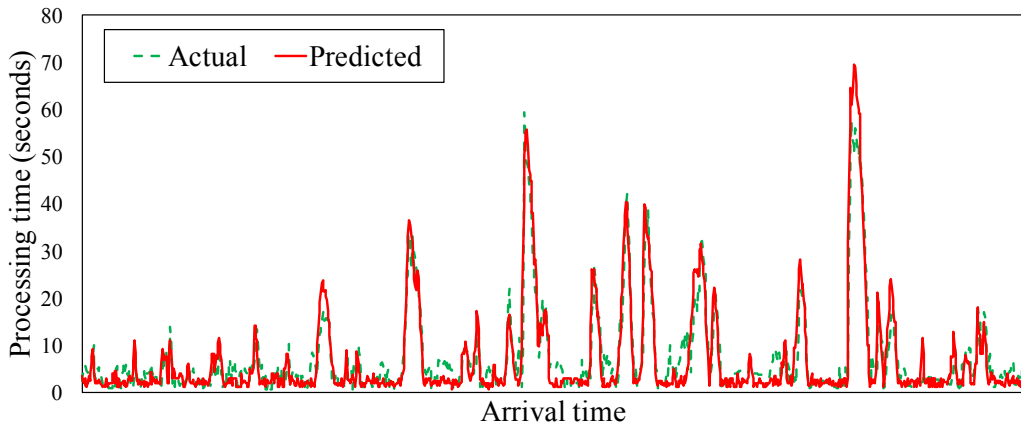
(b) LIGO

Figure 8.4: Effectiveness of system identification on predicting average performance on different workflow ensembles

This is because MPCAdapt can produce better resource adaptation strategies by solving the optimization problem over T future time windows, instead of using heuristic as in HeuristicAdapt which can lead to local optimum. In addition, by looking ahead a future time horizon and taking one control move at a time, the MPCAdapt algorithm can adjust its adaptation strategies to the external factors, such as changes in arrival workload. The HeuristicAdapt algorithm shows good promise⁴ as it offers acceptable performance while being more efficient to compute.

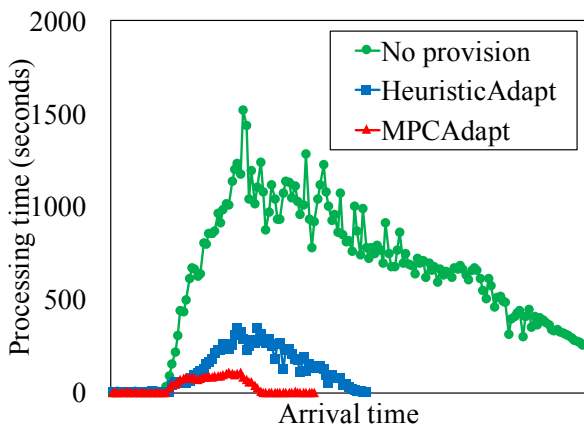


(a) MDP's DM3 file processing workflow

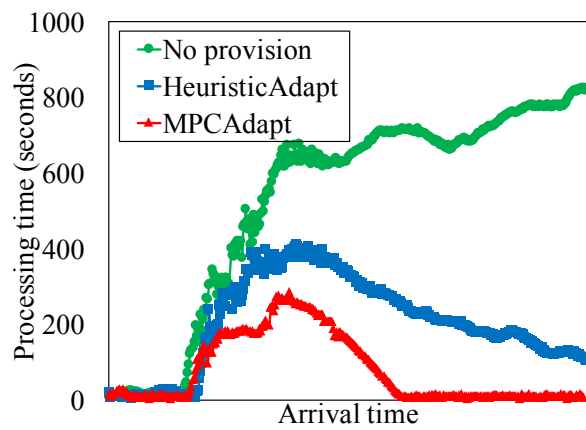


(b) MDP's experiment classification workflow

Figure 8.5: Effectiveness of system identification on predicting processing time of individual MDP workflows



(a) MDP



(b) LIGO

Figure 8.6: Effectiveness of adaptation algorithms on adapting system performance when dealing with bursty workload

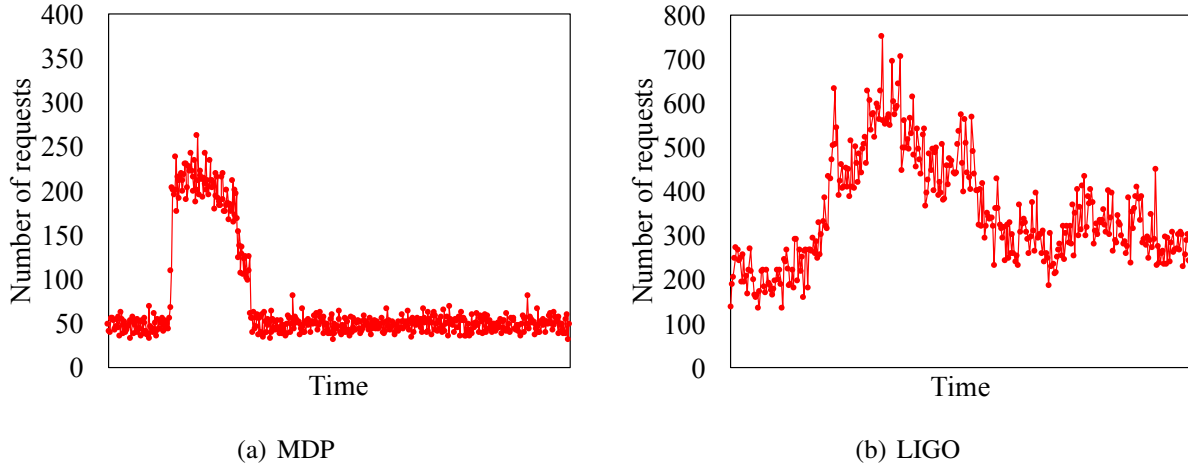


Figure 8.7: Incoming requests to TDS during the bursty workload (aggregated every 5 seconds)

8.3.5 Efficiency of TDS

To evaluate the efficiency of the TDS, we capture the number of requests (aggregated every 5 seconds) which arrive at TDS during the bursty workload experiment above (for both MDP and LIGO workflows). As expected, when additional consumers are allocated to tasks to process increasing workflow requests, the number of requests sent to TDS also increases (i.e., both dependency lookups and synchronization inquiries). Despite the increase in the number of requests to TDS, the maximum latency of responses by TDS is only 22ms for MDP and 37ms for LIGO workflows, which are insignificant compared with the workflow processing time. These results help verify the efficiency of TDS service in handling dynamic workload.

8.4 SUMMARY AND DISCUSSIONS

In summary, in this chapter, we presented the design and implementation of MONAD, a black box-based realization of the adaptive control framework for heterogeneous scientific workflows microservice infrastructure. MONAD uses artificial neural networks, a black-box model, to model performance of the microservice infrastructure and uses this model to assist resource allocation under model predictive control framework.

The main advantage of MONAD that makes it more practical, compared to white-box approach presented in previous chapter, is that MONAD does not require any advance knowledge of workflow structure, workload distribution, and task characterization. However, MONAD can only perform

⁴Although the results show that `HeuristicAdapt` is more efficient than `MPCAdapt`, we leave the evaluation on more complex workflow ensembles with higher number of dimensions (or tasks) for future work.

well if there exists training data of historical workloads to train the performance model of the system. In addition, similar to other supervised approaches, the amount of training data needed is proportional to the complexity of the workflow workload (i.e., represented by the number of tasks and workflow types that the infrastructure supports), and if there are changes in the workload configurations (e.g., new workflow types are supported), the performance model might need to be updated with new training data to capture those changes.

CHAPTER 9: ADAPTIVE MICROSERVICE INFRASTRUCTURE VIA MODEL-BASED REINFORCEMENT LEARNING

9.1 BACKGROUND: REINFORCEMENT LEARNING

We begin by providing some background about algorithms of Deep Reinforcement Learning. Firstly, we introduce the basic concepts about reinforcement learning, then we will move on to the algorithm we apply as our approach.

Reinforcement learning [63] is a type of algorithms which seek to find appropriate actions for an agent who interacts with an environment, often modeled as a Markov Decision Process (MDP). At each time step t of the MDP, the process is at a state $s_t \in \mathcal{S}$, where \mathcal{S} is a finite state set. An agent observes the state and makes an action $a_t \in \mathcal{A}$, where \mathcal{A} is a finite action set. A reinforcement learning algorithm does not assume any knowledge of a mathematical model of the MDP. It tries to learn an appropriate policy $\pi(a|s) = P(a_t|s_t)$ by maximizing an accumulation of its current immediate reward and all future immediate rewards $V_t = \sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in [0, 1]$ is the discount factor, and a_t follows policy $\pi(a|s)$. The reward aggregation is an action value function, and can be represented by $Q_\pi(s, a) = \mathbb{E}_\pi[V_t|s_t, a_t]$, denoting the expected return starting from state s_t , taking action a_t , and following policy $\pi(a|s)$ thereafter. A reinforcement learning algorithm tries to find the policy which can maximize the action value function. This policy is the best policy that an agent can take to achieve the best possible performance in the environment. By decoupling the immediate reward and discounted value of future return, we can write the action value function as

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[r_t + \gamma Q_\pi(s_{t+1}, a_{t+1})], \quad (9.1)$$

which is the Bellman Equation. The method of finding the optimal policy by learning and maximizing the Bellman Equation based on $(s_t, a_t, r_t, s_{t+1}, \gamma)$ tuples is called the Q-learning.

Instead of finding the optimal policy by finding an action which maximizes the action value function at each time step (as Q-learning does), an actor-critic algorithm applies a gradient on the action value function — the critic function — with regard to policy parameter θ as shown in Equation. 9.2, and uses this gradient to update parameters of the policy function — the actor function.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]. \quad (9.2)$$

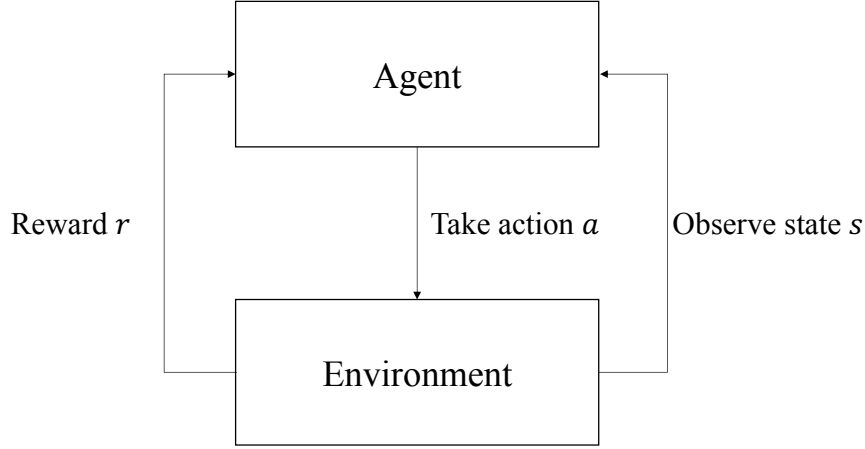


Figure 9.1: Interactions between agent and environment in reinforcement learning.

If we model action as deterministic decisions $a = \mu_\theta(s)$, Silver *et al.* proves that the gradient of action value function can be written as

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}]. \quad (9.3)$$

In this way we can perform the parameter update of deterministic policy function with

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu^k}(s, a)|_{a=\mu_\theta(s)}], \quad (9.4)$$

where α is learning rate, and θ^k is the set of parameters of policy function at iteration k . For example, if the policy function is modeled as a feedforward neural network, then θ^k consists of the weights and biases of neural network's layers.

9.2 MODEL-BASED REINFORCEMENT LEARNING ADAPTATION FOR MICROSERVICE INFRASTRUCTURE

Reinforcement learning, with its closed-loop feedback from the environment, is an ideal candidate for realization of our adaptive control framework for microservice infrastructure. In particular, the role of an agent in reinforcement learning is equivalent to that of a controller in the adaptive control framework: given current state and an objective (i.e., maximize long-term aggregated reward), make decision on the next action (e.g., allocation of resources across microservices).

In terms of the performance model, there are often two main directions regarding to the model of the environment in reinforcement learning: model-free and model-based approaches. With model-free approach, the agent relies on the feedback from actual environment and learns either a reward value function (i.e., Q-learning method) or a state-action policy function (i.e., policy gradient method). The main drawback of the model-free approach is its high sample complexity, as this approach uses actual interactions between agent and environment to improve and train its agent upon. With model-based approach, instead of using actual interactions with the real environment, we train a model of the environment and use this model's predictions to help with training the agent (either by using Q-learning or policy gradient method). The benefit of model-based approach is smaller sample complexity, compared with the model-free approach, by the use of environment model to draw synthetic feedback from the environment. This benefit makes model-based approach more suitable for systems and networking applications, since in these applications, waiting for actual feedbacks from real environment is often very time-consuming (e.g., the system performance and states are often monitored over periodic time intervals of tens of seconds or minutes). This is different from applications such as computerized games (e.g., Atari games) where the model-free approach shines, because in such applications, the agent can receive feedbacks from environment almost instantly due to the availability of computerized model of the games.

On the other hand, the disadvantage of the model-based approach is that we need to be able to train an accurate model of the environment, which is challenging in high-dimensional and complex environments, in order to achieve good performance on training the agent. There have been different approaches proposed to address this issue. For example, one approach is to train an *ensemble* of environment model to improve model's generalization and avoid being stuck in local optimal of an overfitted model. Another example is to design the environment model training and policy training in an iterative manner. The key idea of this method is that, by switching iteratively between *on-policy* (i.e., use the current policy to interact with real environment and collect more training data for environment model) and *off-policy* (i.e., turn off the current policy, retrain environment model with newly collected training data, and use the updated environment model to improve the current policy) modes, we can again avoid being stuck in a local optimal of environment model and gradually build more accurate model of environment (with help of newly collected training data), as well as improve the policy.

In this thesis, we propose to use model-based reinforcement learning for implementing our adaptive control framework. In the following, we first present some preliminaries on how state, action, and reward are define in the context of microservice infrastructure. Then, we will show how we construct and train a model for the environment (i.e., microservice execution environment in

our case), how we leverage the environment model to train a policy, and finally, how we integrate on-policy and off-policy training in a iterative framework.

9.2.1 Preliminaries: State, Action, and Reward

In the context of our microservice infrastructure, the *state* of environment $\mathbf{s}(k)$ corresponds to the average delays of different types of workflows in the k -th time window $\mathbf{d}(k)$ as defined in Chapter 5: $\mathbf{s}(k) = \mathbf{d}(k) = \{d_i(k)\}, 1 \leq i \leq N$.

Reward can be defined as the aggregated decrease in average delays of different workflow types observed from the environment after each time window:

$$r(k) = \sum_{i=1}^N d_i(k-1) - d_i(k) \quad (9.5)$$

An action $\mathbf{a}(k)$ that the agent makes corresponds to the decision of allocation of consumers across microservices $\mathbf{m}(k)$: $\mathbf{a}(k) = \mathbf{m}(k) = \{m_j(k)\}, 1 \leq j \leq J$. Since the total number of consumers over all microservices is bounded (i.e., $\sum_{j=1}^J m_j(k) \leq \mathcal{C}$), we also have to enforce such the constraint on the action made at each time window $\mathbf{a}(k)$.

9.2.2 Environment Model Learning

To train a model for the environment, we use a similar black-box neural network-based approach as shown in Chapter 8, and extend the neural network architecture to improve the generalization of the model, which is vital in model-based reinforcement learning scenario.

Our neural network-based environment model takes input \mathbf{x} as the combination of system states (i.e., $\mathbf{s}(k)$) and action (i.e., $\mathbf{a}(k)$) in the current time window (T_k, T_{k+1}) : $\mathbf{x} = (\mathbf{s}(k) \parallel \mathbf{a}(k))^T$, and predicts output as the system states \mathbf{s}_{k+1} of the next time window (i.e., (T_{k+1}, T_{k+2})). The neural network model consists of L layers, with the number of neurals in each layer is denoted as $S^{(l)}$ ($1 \leq l \leq L$). Correspondingly, $\mathbf{W}^{(l)}, \mathbf{b}^{(l)}$ ($1 \leq l \leq L$) represent the weight matrix and bias of layer i -th. Each layer l -th also includes a non-linear (except the last layer that uses the linear identity function) activation function $\mathbf{f}^{(l)}$ to introduce the non-linearity into the network model. In particular, we use rectified linear unit (or ReLU) as the non-linear activation function for the hidden layers and identity activation as the activation function for output layer. We denote $\mathbf{y}^{(l)}$ as the vector of outputs from layer l ($\mathbf{y}^{(0)} = \mathbf{x}$). To improve model's generalization, we add a *dropout* layer [64] after the output of each hidden layer. A dropout layer is represented by a

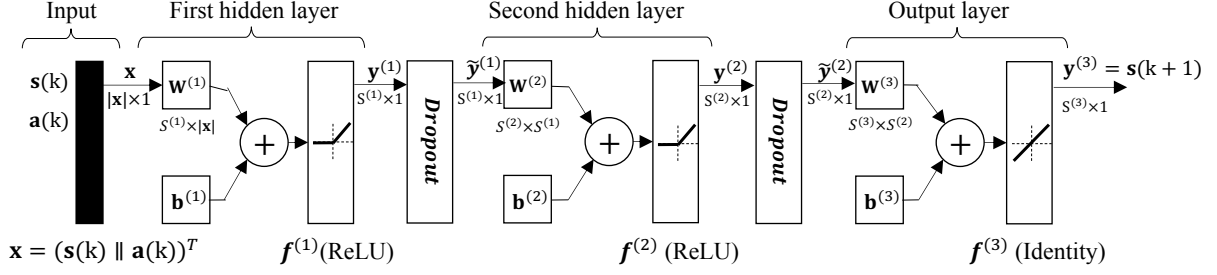


Figure 9.2: Neural network model of microservice environment.

Bernoulli distribution with parameter p , also known as *keep probability*, that represents how likely the updating gradients being kept during back-propagation training process. The neural network model can be described via a series of matrix-based calculations as follows ($0 \leq l \leq L - 1$):

$$\begin{aligned}
 r_j^{(l)} &\sim \text{Bernoulli}(p) (1 \leq j \leq S^{(l)}), \\
 \tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
 \mathbf{y}^{(l+1)} &= \mathbf{f}^{(l+1)}(\mathbf{W}^{(l+1)}\tilde{\mathbf{y}}^{(l)} + \mathbf{b}^{(l+1)})
 \end{aligned} \tag{9.6}$$

The output of the model is the predicted state of the environment in the next time window: $\mathbf{s}(k+1) = \mathbf{y}^{(L)}$. Figure 9.2 shows an actual neural network architecture of our environment model with two hidden layers and one dropout layer after each hidden layer.

If we denote Φ as the set of parameters of the environment models: $\Phi = \{\{\mathbf{W}^{(l)}\}, \{\mathbf{b}^{(l)}\}\}$ ($S^{(l)}, L, p$ are tuning parameters), then the environment model can be represented by a function $\hat{f}_\Phi: \mathbf{s}(k+1) = \hat{f}_\Phi(\mathbf{s}(k), \mathbf{a}(k))$. And the objective of environment model learning is to find a parameter Φ that minimize least square error of one-step prediction:

$$\min_{\Phi} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}(k), \mathbf{a}(k), \mathbf{s}(k+1)) \in \mathcal{D}} \|\mathbf{s}(k+1) - \hat{f}_\Phi(\mathbf{s}(k), \mathbf{a}(k))\| \tag{9.7}$$

where \mathcal{D} is the set of training data. In this case, we employ a common practice in designing neural network and let the network model predict the change in state (rather than the next state) given a state and an action as inputs. This helps to prevent the network model to memorize the previous state.

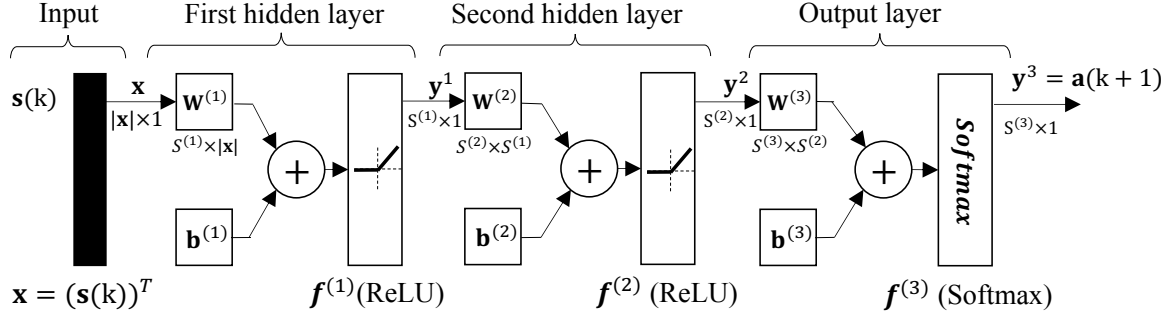


Figure 9.3: Actor network architecture.

9.2.3 Policy Learning

After training the environment model, the trained model can be used to generate synthetic samples of system dynamic. These synthetic samples can be used to train the policy.

In this thesis, we use actor-critic method for policy learning. With this method, we train simultaneously two neural network-based models of actor and critic. While the actor network tries to make decision on the next action $\mathbf{a}(k+1)$ given current state $\mathbf{s}(k)$ (i.e., learning the policy), the critic network is used to evaluate the value of action made by the actor network (i.e., learning value function).

While we can leverage vanilla neural network structure for the critic network, the design of the actor network requires more considerations to ensure that the output of the actor network satisfies the resource constraint (i.e., the total number of consumers across all microservices is bounded). To enforce such a constraint, we design action output of actor network as a categorical distribution, or a probability distribution over J different possible outcomes (each outcome corresponds to the allocation of a microservice), by applying a softmax activation function for the output layer (Figure 9.3). The categorical distribution can be then translated into numbers of consumers by multiplying with the total number of consumers \mathcal{C} :

$$m_j(k) = \lfloor \mathcal{C} * a_j(k) \rfloor, \forall 1 \leq j \leq J \quad (9.8)$$

In this thesis, we use deep deterministic policy gradient method (or DDPG) [65] to train the actor-critic policy networks. If we denote Θ as the parameter of the actor neural network, then the learned policy can be represented by a function $\hat{\pi}_\Theta: \mathbf{a}(k+1) = \hat{\pi}_\Theta(\mathbf{s}(k))$.

9.2.4 Iterative Model-based Reinforcement Learning

As mentioned at the beginning of this chapter, to overcome the limitation of the model-based reinforcement learning approach (i.e., the learned policy often exploits regions where scarce training data is available for the environment model), one method is to switch iteratively between *on-policy* (i.e., use the current policy to interact with real environment and collect more training data for environment model) and *off-policy* (i.e., turn off the current policy, retrain environment model with newly collected training data, and use the updated environment model to improve the current policy) modes. As a result, we can avoid being stuck in a local optimal point of the environment model and gradually build more accurate model of environment (with help of newly collected training data), as well as improve the policy.

The iterative training procedure is presented in Algorithm 9.1, in which the outer loop represents on-policy training on real environment, and the inner loop represents off-policy training using environment model.

Algorithm 9.1 Iterative Model-based Reinforcement Learning Procedure

- 1: Initialize $\hat{\pi}_\Theta$, \hat{f}_Φ , and \mathcal{D}
 - 2: **repeat**
 - 3: Collect sample from real environment using $\hat{\pi}_\Theta$ and add to \mathcal{D}
 - 4: Train environment model \hat{f}_Φ using \mathcal{D}
 - 5: **repeat**
 - 6: Collect synthetic samples from \hat{f}_Φ using $\hat{\pi}_\Theta$
 - 7: Update policy $\hat{\pi}_\Theta$ using DDPG algorithm
 - 8: **until** Performance of the policy stops improving
 - 9: **until** The policy performs well in real environment
-

9.3 EVALUATION

9.3.1 Evaluation Settings

We leverage microservice-based workflow infrastructure presented in Chapter 5 as the environment to evaluate the reinforcement learning-based resource adaptation. For environment model learning, we emulate workflow workload to the system and monitor the system performance in terms of work-in-progress of tasks and the corresponding number of allocated consumers. Specifically, we use the MDP workload (c.f. Chapter 7 and 8) that includes three types of workflows ($N = 3$) and four different types of tasks ($J = 4$). We initialize a random policy and collect

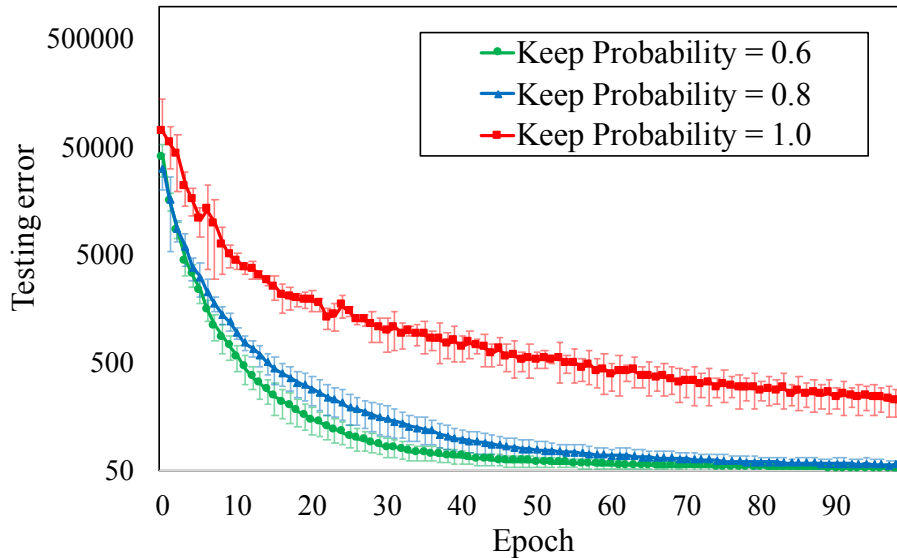


Figure 9.4: Testing error of environment model using different dropout rates.

training data from the real microservice execution environment. The collected information is used as the training dataset \mathcal{D} . The training dataset is then splitted with ratio 80% - 20% for training and testing respectively.

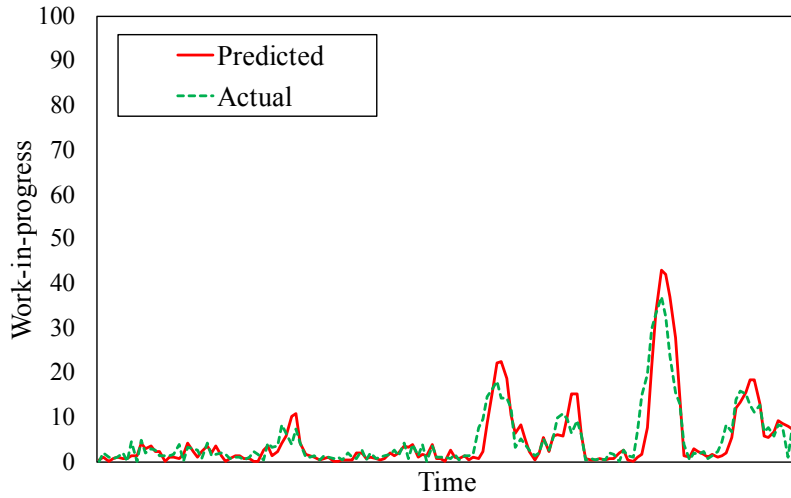
9.3.2 Effectiveness of Environment Model Learning

Figure 9.4 demonstrates the effect of using dropout layers in improving generalization of environment model when performing on testing data over training epoch (i.e., each epoch corresponds to one forward pass and one backward pass of all the training examples). Specifically, we can see that as we increase the dropout rate (i.e., decrease the keep probability p), the prediction results become less overfitted, which is demonstrated through smaller testing error. This is desirable since we want the model to generalize well when dealing with unseen data.

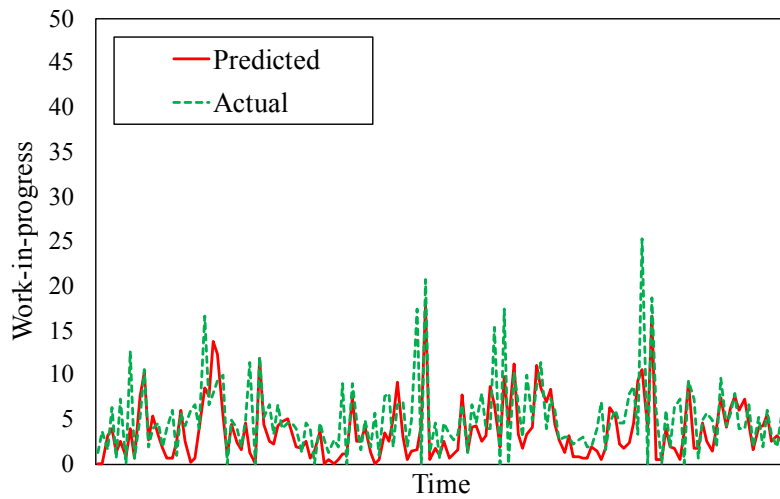
Figure 9.5 demonstrates qualitatively the effectiveness of using trained environment model to accurately predict near future performance, in terms of work-in-progress, of individual microservices (Figure 9.5-b and Figure 9.5-c), as well as average performance across all microservices (Figure 9.5-a) (each data point is 15 seconds apart on x-axis).

9.3.3 Effectiveness of Policy Learning

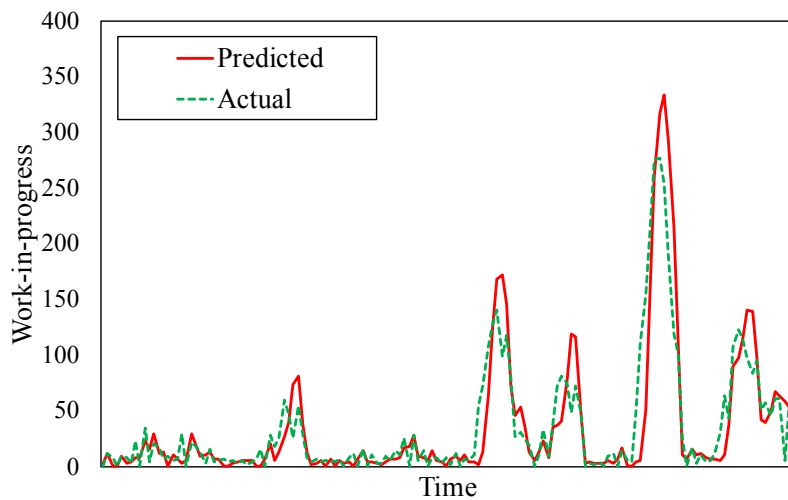
After obtaining a trained model of the environment, we perform policy training using synthetic state traces generated by the environment model. During policy training, we record the perfor-



(a) Average across all microservices

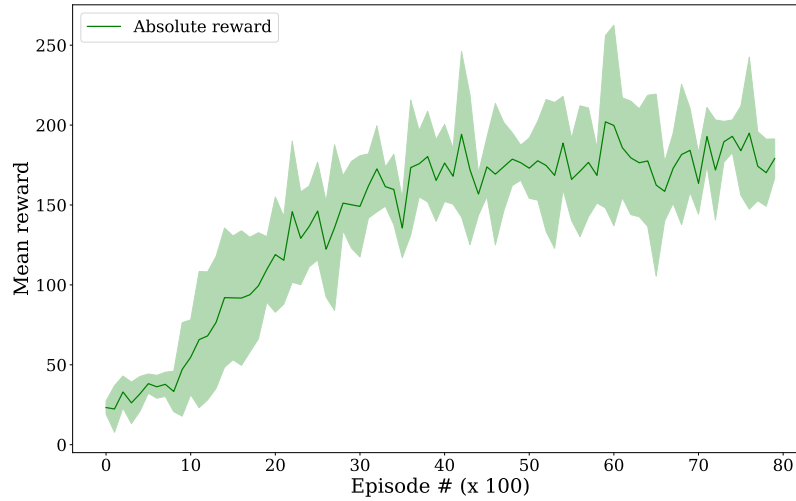


(b) Microservice A

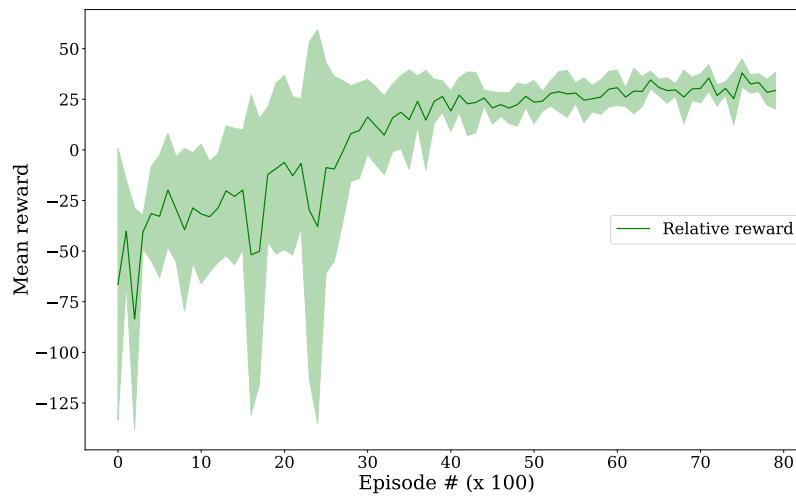


(c) Microservice D

Figure 9.5: Effectiveness of trained environment model when predicting average work-in-progress across microservices and work-in-progress of individual microservices.



(a) Absolute reward



(b) Relative reward

Figure 9.6: Effectiveness of training policy using trained environment model and different reward functions.

mance of the policy (by average aggregated reward) after every 50 episodes, each episode is defined as 20 consecutive time step with each time step corresponds to an action made by policy network. We evaluate with different reward functions as described previously, including absolute reward and relative reward, to see how effective each reward functions is in training policy network.

The result is shown in Figure 9.6. We can see that, with both reward functions, the performance of policy converge after around 500 episodes (for absolute reward) and 400 episodes (for relative reward). This convergence result demonstrates the effectiveness of using learned environment model to train policy network.

We also zoom into some sample episodes to see how policy network, trained with absolute re-

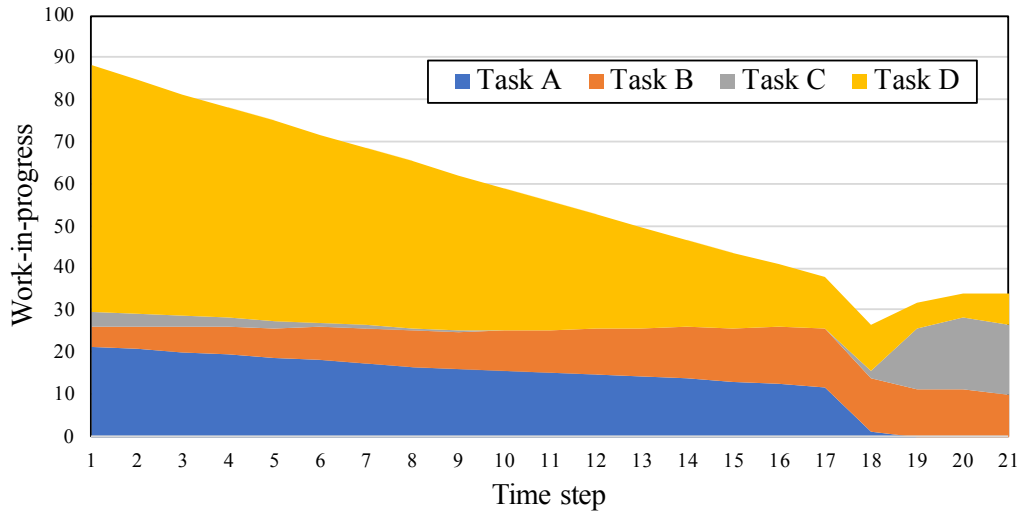
ward function, makes resource allocation decisions. Figure 9.7 and Figure 9.8 show two examples of two different episodes (each episode is limited to 20 time steps in our evaluation). In each example, Figure 9.7-a and Figure 9.8-a show how work-in-progress of microservices (i.e., system states) change overtime as the policy network makes resource allocation decisions, which are shown in Figure 9.7-b and Figure 9.8-b respectively. It demonstrates that the policy network is trained to allocate resources among microservices to gradually reduce the aggregated work-in-progress across microservices, which resonates with the use of absolute reward during training.

In addition, as shown in examples in Figure 9.7 and 9.8, the policy network is able to learn an efficient order of allocation among the microservices so that it can achieve the best reduction in aggregated work-in-progress across all microservices. Specifically, in both examples, the policy network tends to allocate resources to microservices following order of $D \rightarrow C \rightarrow B \rightarrow A$ (i.e., gradually prioritize to allocate consumers to D , C , B , and then A). Interestingly, this order is the reversed topological order of the tasks in the set of workflows that system supports, and thus, the allocation order makes sense as the requests are routed through the tasks in topological order.

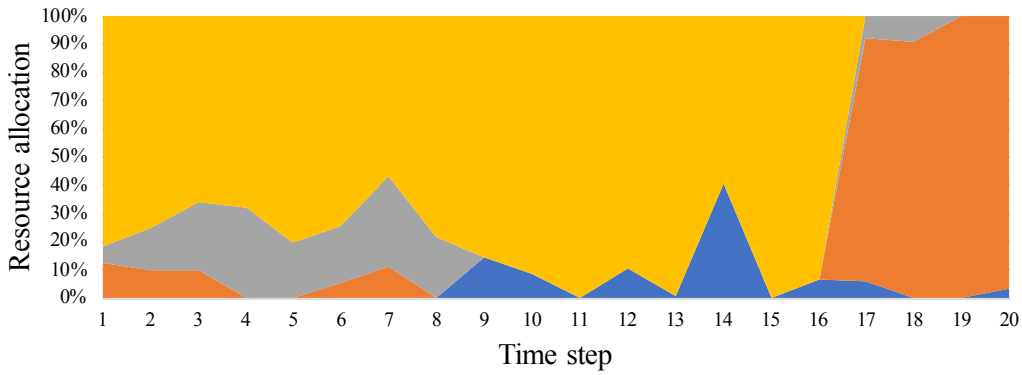
To evaluate the effectiveness of iterative policy learning, we use the initially learned policy to interact the real microservice environment and collect additional interaction data. This data is then used to update the environment model (i.e., microservice performance model) and then, update the initially learned policy. The result in Figure 9.9 show that the policy learning performance improves after an iteration: the newly learned policy is much faster to converge and it also achieves better convergence performance than the initially learned policy. This is because newly collected interaction data helps to environment model to make better predictions on the performance of microservice infrastructure and, as a result, also helps to train a better policy network.

9.4 SUMMARY AND DISCUSSIONS

In this chapter, we present another realization of the adaptive control framework using model-based reinforcement learning, as the next step from the model predictive control approach presented in Chapter 8 toward a *hand-off* approach for resource allocation of microservice infrastructure. We show how different notions in reinforcement learning, such as state, action, and reward, can be defined in the context of microservice infrastructure. Our preliminary evaluation results demonstrate the effectiveness of training accurate model of the microservice environment and effective policy networks using various reward functions. The results also show the effectiveness of iterative model learning and policy learning to help improve the performance of policy network over time. Similar to model predictive control approach presented in Chapter 8, the

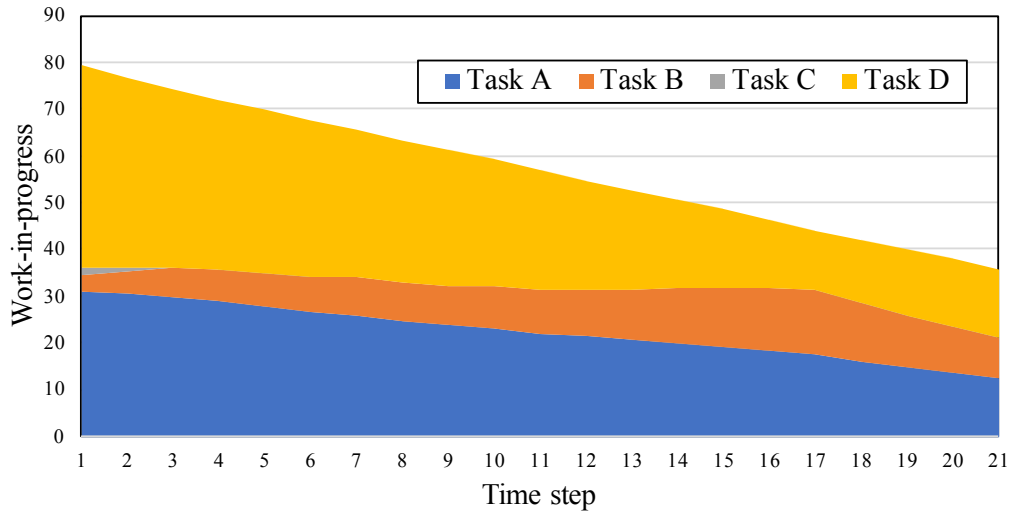


(a) Work-in-progress of microservices

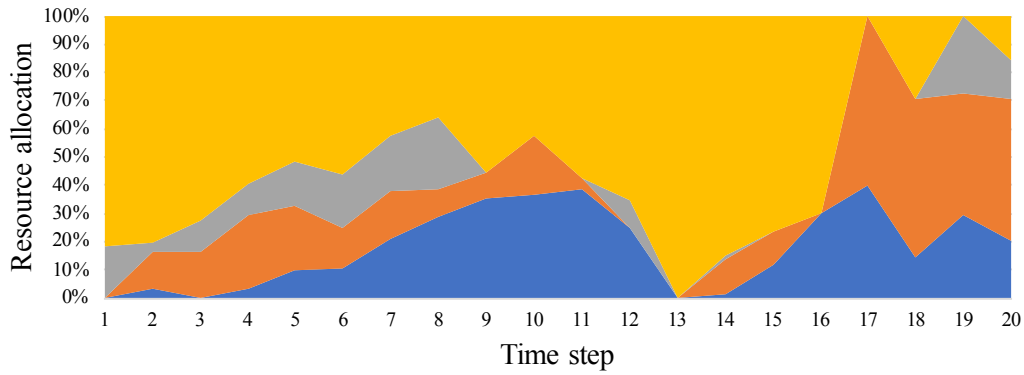


(b) Percentage of resource allocated to microservices

Figure 9.7: Sample 1 - Effectiveness of learned policy network in allocating resources to microservices to help reduce work-in-progress. Time steps are 15 seconds apart, and y-axis represents percentage of number of consumers allocated to tasks.



(a) Work-in-progress of microservices



(b) Percentage of resource allocated to microservices

Figure 9.8: Sample 2 - Effectiveness of learned policy network in allocating resources to microservices to help reduce work-in-progress. Time steps are 15 seconds apart, and y-axis represents percentage of number of consumers allocated to tasks.

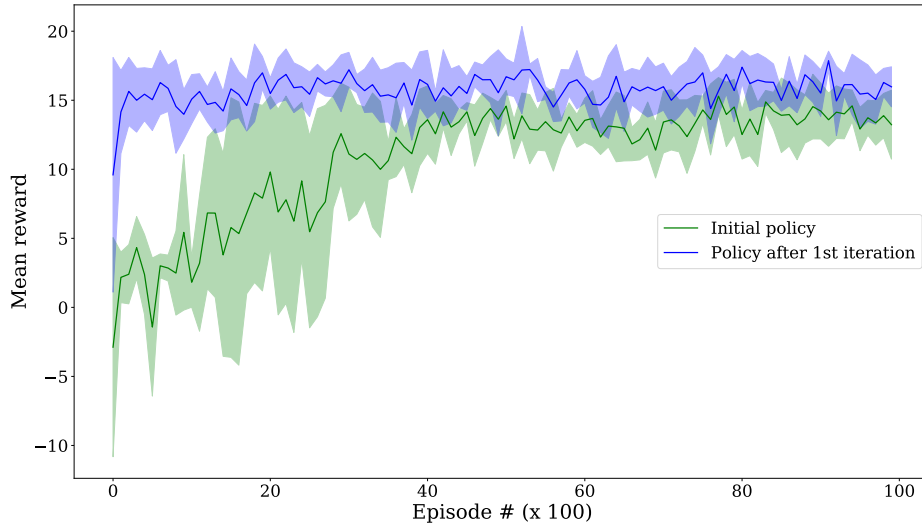


Figure 9.9: Effectiveness of iteratively collect more environment interactions by initial policy and retrain the environment model and policy.

model-based reinforcement learning approach also relies on a black-box model of the environment (both approaches use neural network to model the performance). However, in RL-MONAD, we use work-in-progress as the performance metric, instead of average workflow processing delays as in MONAD, as work-in-progress is fully observable after every time step while processing delays might not.

Although RL-MONAD is capable of learning resource allocation strategies and adapting with the changes of environment (thanks to its iterative learning process), its potential disadvantage is mainly in the overhead of training/updating performance model and policy network overtime (even though updating model and policy can be done in parallel while the system is still running with the old policy and model). In addition, through the experiments, we found out that RL-MONAD is often slow in response to sudden increase or decrease in the workload. This can be explained by the fact that RL-MONAD uses aggregated long-term rewards to train policy, while the workflow workload we experimented with only consists of tasks with relatively short processing times, which require faster response and shorter-term planning of resource allocation. Therefore, as a next step, we plan to perform further evaluations with more complex workflow workloads with longer average processing times to better measure the effectiveness of RL-MONAD when handling dynamic workload situations.

CHAPTER 10: 4CEED - REAL-TIME ACQUISITION AND ANALYSIS FRAMEWORK FOR MATERIALS-RELATED CYBER-PHYSICAL ENVIRONMENTS

10.1 EXTENDING DOSSIER TO MATERIAL-RELATED ENVIRONMENT

As briefly introduced in Chapter 4, to validate the effectiveness and practicality of using DOSSIER and its service building blocks in supporting data management applications of cyberinfrastructure, in this section, we present the architectural overview of the 4CEED framework for real-time capture, curation, coordination, collaboration, and distribution of scientific material-related data. The framework consists of two main services: *curation* (instrument and user tier) and *coordination* (cloud tier) services (Figure 10.1). The streamlined curation service that helps users to perform nimble and adaptive data collection from material research instruments by wrapping of data with extensive meta-data in timely and trusted manner. The coordination service is built based on DOSSIER and its service building blocks. In coordination service, a number of data processing tasks are developed specifically for various types of material data to process data uploaded from instruments.

The more detailed overview of 4CEED is presented in Figure 10.2. Since the coordination service is based on the DOSSIER infrastructure that has been described in Chapter 5 and 6, in the following, we focus on describing the design of 4CEED's curation service and how it takes advantage of the service building blocks provided by DOSSIER.

10.2 CURATION SERVICE

The data curation service consists of two main components: uploader (i.e., for uploading data from scientific instruments) and curator (i.e., for curating data after experimental sessions at instruments). Both uploader and curator applications are built based on DOSSIER's services. In particular, DOSSIER's data management service provides a novel *extensible data model* for heterogeneous and multi-modal scientific data (i.e., combination of multimedia data like images, structured and unstructured data, text, tags, etc.) (Figure 10.3). The data model is designed to be generic, so that it can support managing different types of scientific data from different application domains. The model is based on nested structures necessary to mimic hierarchical organization of scientific experiments. Specifically, the data model hierarchy includes three main concepts: *nested collections*, *datasets*, and *files*. At the lowest level, files represent experimental result data, such as images, text, or proprietary files. A dataset is a grouping of files and metadata capturing the

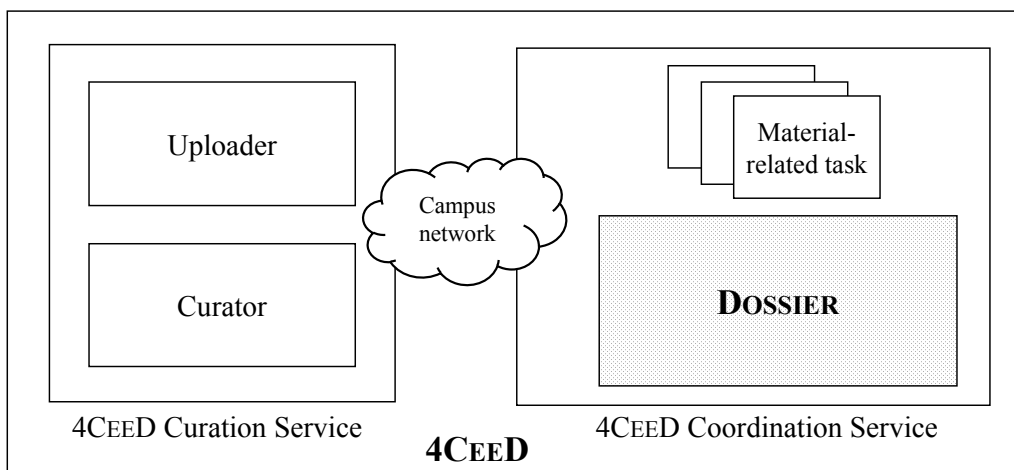


Figure 10.1: DOSSIER as part of 4CEED implementation.

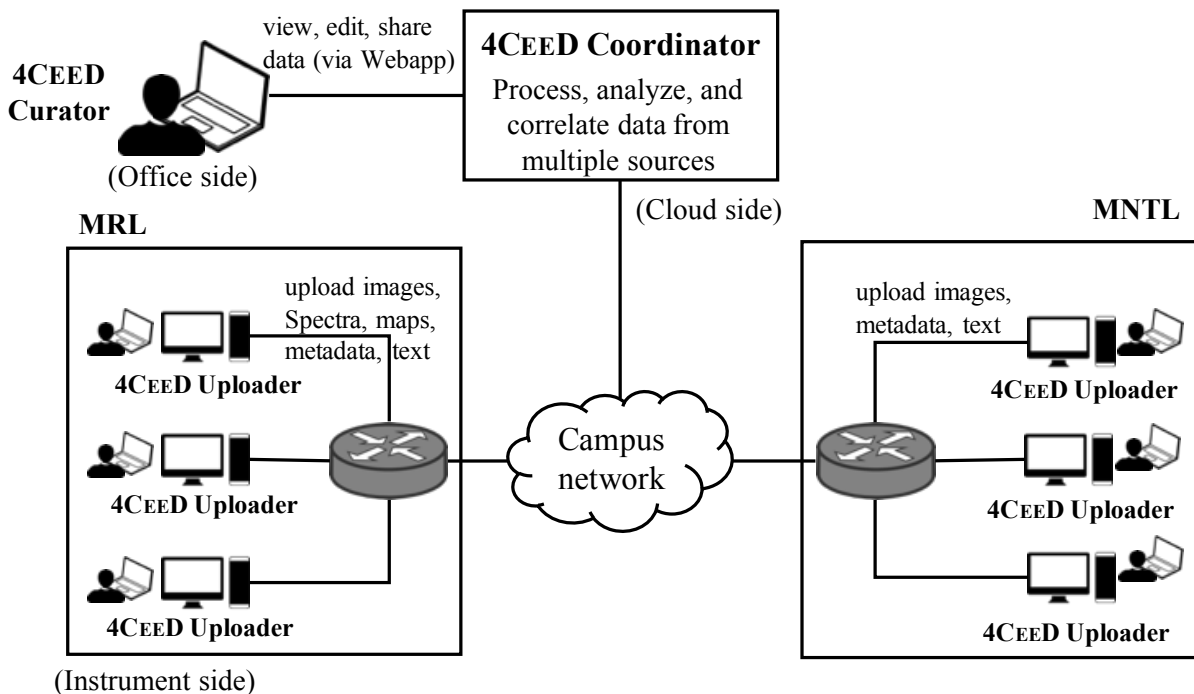


Figure 10.2: Overview of 4CeeD system.

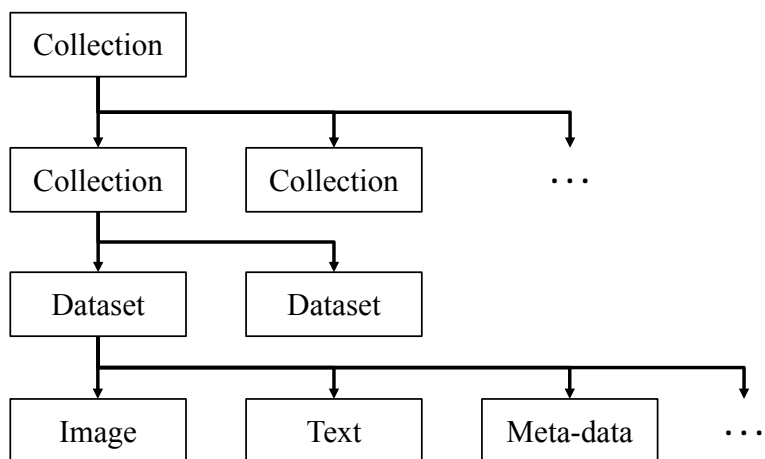


Figure 10.3: Data model of DOSSIER's data management service.

preparation information of the experimental sample. A collection is a way for users to organize their datasets (e.g., each collection represents experiment data for a day, or done by a specific instrument). The nested structure of collections provides users the flexibility to describe their own data organization. While the concepts of collections, datasets, and files provide a vertical organization of data, we use the concept of spaces for horizontal organization to support sharing of data. A space is a set of collections and datasets that are shared among multiple users. Different levels of user permissions (e.g., owner, editor, and viewer) can be configured for each space to enable flexible collaboration between users.

Using DOSSIER's data management service, 4CEED's *uploader* provides a new simple, user-in-the-loop interface for uploading raw data generated from materials-making and characterization instruments (e.g., microscopes) and device fabrication instruments during lab sessions. The user involvement in the data input step is because all materials and device fabrication instruments are supervised and controlled by experimenting users. Often, we want users to enter process-related data, notes regarding experimentation with new materials, reasoning on why a certain physical component was added or removed, and so forth. Specifically, the uploader provides an interface that consists of 3 simple dependent steps following the nested data model. In the first step, user creates or selects a collection or sub-collection from his/her own set of existing nested collections (stored on the cloud) visualized by a tree-based structure. After selecting (or creating) a collection, in step 2, user can create or select an existing dataset. Under dataset, users can manually enter meta-data associated with the experiment, or use provided meta-data templates (i.e., each template correspond to a collection of meta-data fields) for faster and more accurate recording of meta-data. In the third step, users can drag and drop multiple raw experimental files generated from the

instruments to the dataset selected/created in step 2 to submit to the cloud. Additional file-level meta-data can also be added in the third step.

Using DOSSIER's data curation service, 4CEED's *curator* provides a novel interface that allows users abilities to browse, view, edit their uploaded data at the office side, using the nested data model. Especially, as the raw uploaded data has been processed by the coordination service, users can see results of all data processing tasks done on the raw data. Examples of the tasks include extracting instrument-specific meta-data and image from DM3 file, generating previews for microscopy images, and classifying experimental data into appropriate types (e.g., diffusion, oxidation, etc.) or outcomes (i.e., success or failure). Each type of experimental data requires a different set of data processing tasks, which are expressed in form of a *workflow* or Directed Acyclic Graph (DAG) of tasks, to be applied. Each workflow or task graph corresponds to a type of a data processing job, which can be configured by the coordination service. In addition, within the curation service, we provide an "e-commerce style" search (i.e., similar to search feature on e-commerce sites like Amazon, Newegg) over shared data repository of experiments. Users can easily and efficiently search through a large amount of experimental data by combining traditional keyword-based search and structured data filtering (or faceted search). The structured data used in filtering can be instrument-specific meta-data, experiment-related settings, or the results of data processing tasks (e.g., outcome classification).

Both 4CEED's uploader and curator can be accessed as web-based applications (hence are platform-independent) and both require authentication to access, curate and share data. The communication between uploader, curator and the cloud-based system is via the HTTPS protocol to ensure security. 4CEED is open-sourced¹ and it can be deployed using different resource management deployment mechanisms, including: virtual machine (i.e., with minikube), single server (i.e., with docker-compose), cluster (i.e., using Kubernetes), and hosted solution (i.e., Google Cloud Platform).

¹4CEED's Github: <https://github.com/4ceed/4ceedframework/>

CHAPTER 11: BRACELET - HIERARCHICAL EDGE-CLOUD MICROSERVICE INFRASTRUCTURE

As motivated in Chapter 1, the main challenge in terms of data acquisition from scientific instruments is due to the performance and security gaps between scientific instruments and back-end cloud and network infrastructure. In particular, major scientific instruments and their software tools run old operating systems, such as Windows XP, Windows NT, Windows 2000, and Windows 3.11. Even though these OSES are capable of networking, they are set offline because they cannot operate at the network speed of a powerful cloud (e.g., if a fast cloud server communicates with a slow computer with older OS and network interfaces, connections often break quickly, and/or more data might be retransmitted because of delayed acknowledgements) and they are also not patched with the latest security patches (since companies such as Microsoft no longer support these older operating systems). In addition, the software updating cycle of scientific tools (e.g., instrument tools developed by companies such as Siemens and GE) is much slower, compared with that of IT tools. As a result, this issue will not go away and it prevents the instruments to be connected with the advanced networked cyberinfrastructure. Further, such a disconnection also hinders the ability to upload scientific data from the instruments to the cloud during experimental sessions (e.g., as in 4CEED).

To address this problem, we extend DOSSIER to include a networked edge component (or cloudlet), named BRACELET, between the scientific instruments and cloud as the middle tier of a three-tier hierarchy. As shown in Figure 11.1 where BRACELET is integrated into 4CEED architecture, BRACELET will be placed in each research lab to shape and protect traffic from instruments in the lab to the campus cluster hosting scientific data management system (i.e., 4CEED's coordination service). BRACELET will handle the mismatch in computational and network speeds (i.e., performance mismatch) and the mismatch in security, and enable enhanced computation over scientific data to offload some computation functions from older instruments (e.g., extraction of metadata from images and other data).

The BRACELET will run performance and security components (Figure 11.2) to process, protect, and upload data to cloud infrastructure and protect scientific instruments from threats coming from outside network. The BRACELET device design will have a dual and integrated architecture with two major components to enable uploading service from instruments to the cloud infrastructure, as follows.

The *performance component* will concentrate on the performance and reliability matching that needs to be done to connect older scientific instruments to the cloud infrastructure. Specifically,

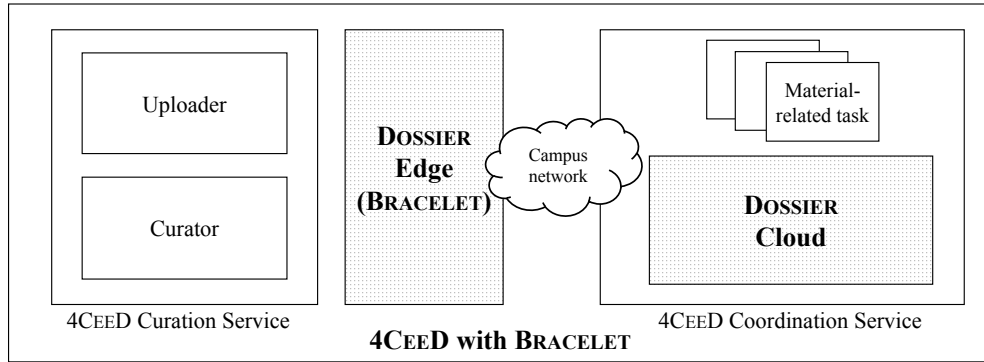


Figure 11.1: Integrating edge-based BRACELET to 4CEED’s data acquisition architecture.

BRACELET will receive scientific instruments’ data from the 4CEED uploader client, running network protocols that match the speed of slower instruments. BRACELET will be caching data, since users will upload data at different rates (because of running different types of instruments or having different operating systems and network interfaces). Then, BRACELET’s performance adaptation endpoint will coordinate with the cloud side to perform traffic shaping, multiplexing, aggregation, and scheduling of data traffic to the backend cloud system in a protected manner. In addition, BRACELET could also receive instructions from the cloud side (via performance adaptation endpoint) to perform certain data processing, such as metadata extraction from the instrument’s raw output data, to offload processing from slower and computation-limited instruments and hence speed up the end-to-end upload process. The runtime system is the same as that on the cloud (to create a uniform execution environment for computational offloading), and is monitored by a monitoring endpoint that aggregates and sends status information to the monitoring component on the cloud.

The *security component* will concentrate on the protection and security changes that need to be done to connect older scientific instruments to the cloud infrastructure and protect the instruments from external threats. Within BRACELET’s security component, we will explore the integration of the authentication and authorization protocols such as Shibboleth and OAuth to achieve authentication and authorization between BRACELET and cloud infrastructure; a registration protocol to create an admissible table of instruments that are allowed to upload data to the cloud infrastructure; white-listing of IP addresses of computers that control instruments; and a firewall to check which packets are allowed to pass further or to be admitted.

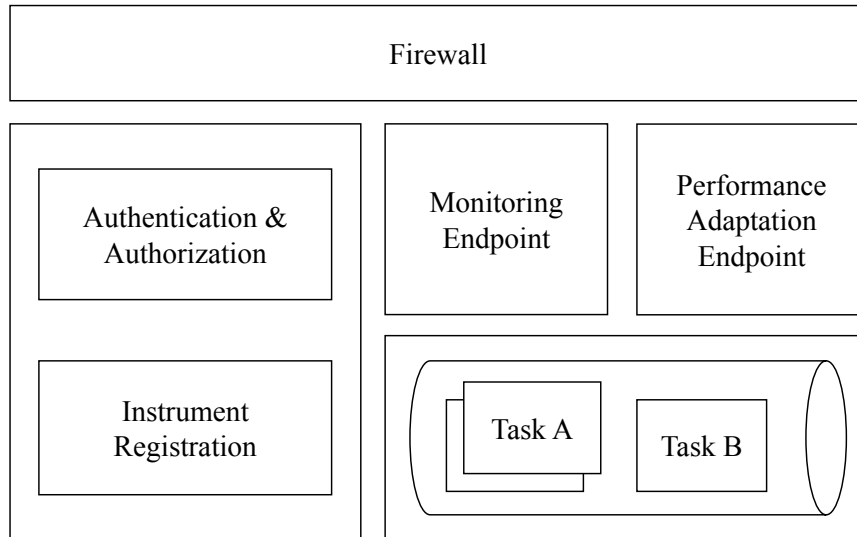


Figure 11.2: BRACELET's component architecture.

11.1 BRACELET'S ARCHITECTURE

An overview of BRACELET's 3-tier architecture is presented in Figure 11.3. In particular, the first tier, i.e., *instrument tier*, includes scientific instruments attached to computers running old operating systems that could not directly connect to the cloud (the new instruments that run with more advanced operating systems can connect directly to the cloud in the existing 2-tier architecture). On each instrument's computer, users use a uploader client to upload experiment data upstream. The second tier, i.e., the *edge* or *cloudlet tier*¹, includes edge-based devices, or cloudlets, that consist of two network interfaces: one connects to instruments' VLAN and another connects to the cloud via public network. Lastly, on the third tier, i.e., *cloud tier*, we deploy a cloud-based infrastructure that connects to the public network. The cloud-based tier supports data processing, curation, storage, correlation, and search of scientific experiment data uploaded from instruments via cloudlets.

11.1.1 BRACELET's Microservice Architecture

Figure 11.4 shows the detailed microservice architecture of BRACELET and its performance components. To enable seamless integration of cloudlets to the existing 2-tier cloud-based infrastructure, we design BRACELET by extending the cloud-based microservice architecture [45, 53] to

¹From now, we will use *edge*, *edge device* and *cloudlet* interchangeably.

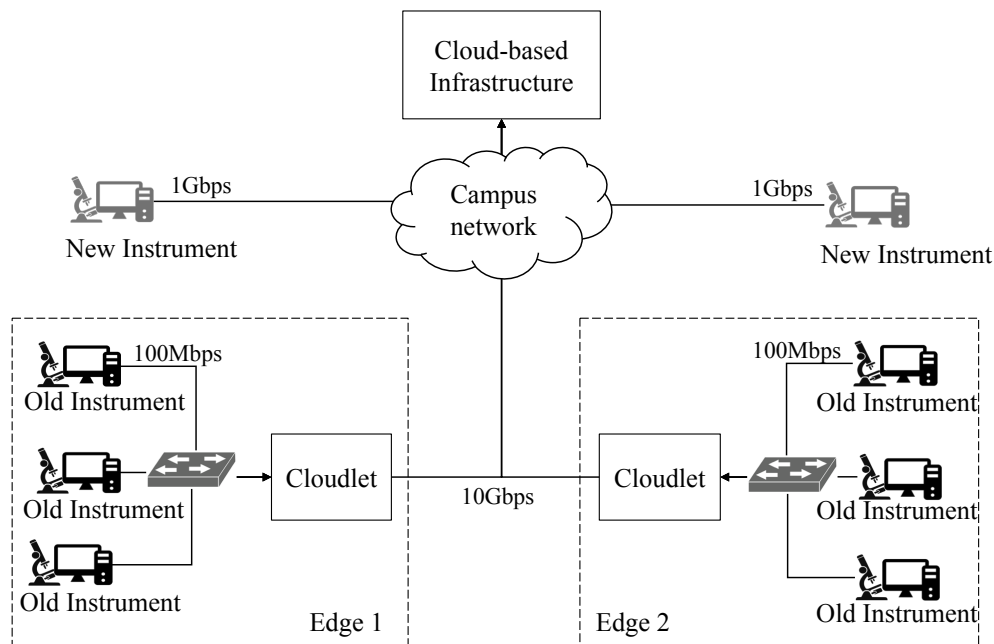


Figure 11.3: Overview of BRACELET system.

the edges. In particular, while the cloud-based infrastructure operates on the full 5-layer architecture, the cloudlets operate on three layers to enable computational offloading of tasks to the edges and seamless communication between edge- and cloud-based components. In the following, we describe all the layers in details.

Infrastructure Layer

Infrastructure layer provides a level of abstraction and virtualization of all computation and storage resources across cloud and edges. We leverage container technology for virtualization and use a container orchestration engine to manage the container allocation across edge-cloud infrastructure.

Execution Layer

We design execution layer using a microservice workflow execution model across cloud and edges. Experiment data uploaded from instruments will be handled by a specific type of *data processing workflow*, each workflow type corresponds to a directed acyclic graph (DAG) of a subset (or all) types of *data processing tasks* that system supports. We model each task as a

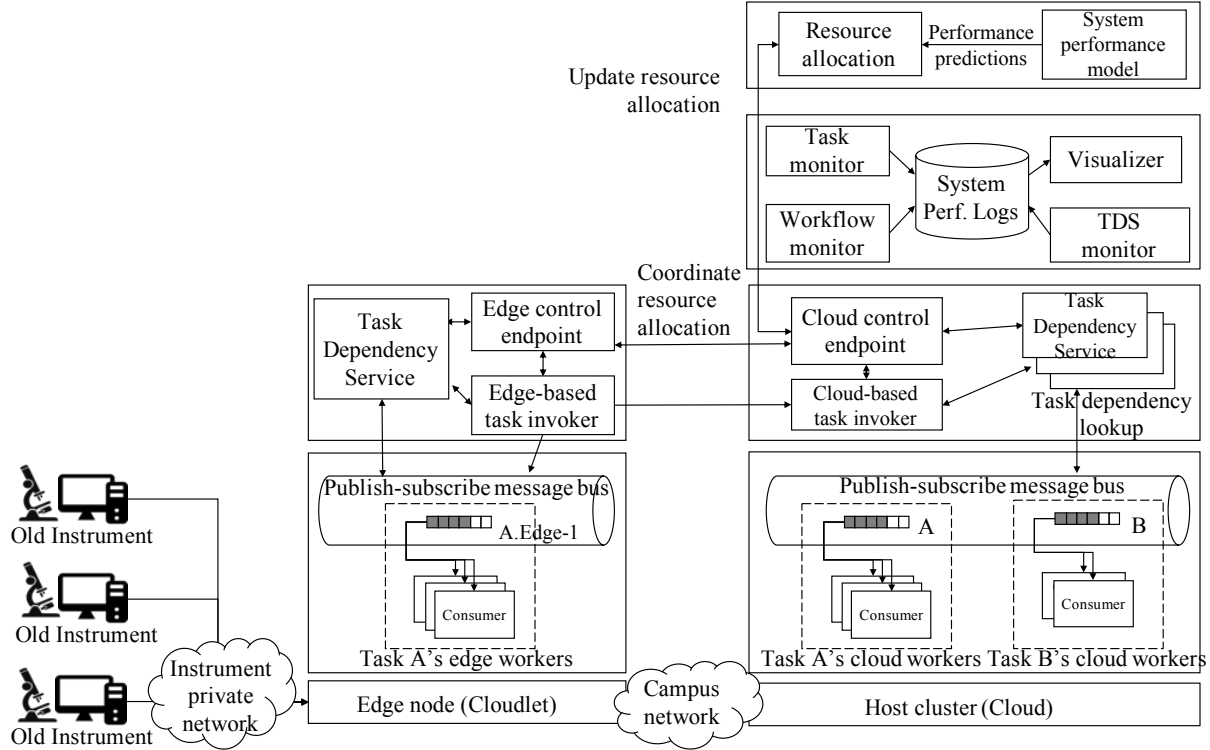


Figure 11.4: Detailed architecture of BRACELET system.

*microservice*² with its own *request queue* that stores the task's requests, and a set of *task consumers* that subscribe to the request queue to perform actual processing of the task's requests.

The communication between dependent tasks in a workflow follows the *publish-subscribe mechanism*. When a task request arrives at the queue, a task consumer subscribing to the queue will pick up the request to process it. After processing the request, the consumer asks the coordination layer (to be described shortly) about the subsequent tasks of the workflow and publish the request to the corresponding queues of the subsequent tasks. We assume that all workflow data and intermediate results between tasks are stored in a shared storage system that can be accessed by all micro-services across cloud and edges.

A microservice can be deployed on a cloudlet (or multiple cloudlets), on cloud, or on both cloud and cloudlets. The publish-subscribe message bus is available across cloud and edges to enable seamless communication between edge- and cloud-based micro-services.

²From now, we will use *task* and *microservice* interchangeably.

Job type	From	To
Wf1	Start	A
Wf1	A	B
Wf1	B	C
Wf1	C	End
Wf2	Start	C
Wf2	C	D
Wf2	D	End

Figure 11.5: Example of task dependency table.

Job type	From	To
...
Wf1.E1	Start	A.E1
Wf1.E1	A.E1	B.E1
Wf1.E1	B.E1	C
Wf1.E1	C	End
...

Figure 11.6: Updated task dependency table with an edge-based version of Wf1.

Coordination layer

On top of the execution layer is the coordination layer that consists of a *task dependency service*, or TDS, that maintains the dependencies between tasks of a workflow (i.e., task dependencies are essentially the directed edges of workflow’s task graph) and responds to task dependency lookups from the execution layer. Figure 11.5 shows an example of task dependencies maintained by TDS for two types of workflows (i.e., Wf1 and Wf2) and 4 types of tasks (i.e., A, B, C, and D - please note that the same task can be used by multiple workflow types).

The separation of task coordination from the execution of tasks enables more flexible and scalable workflow composition (i.e., we can support new workflow types by simply creating new set of task dependencies between the existing tasks). To offer high availability and high performance, we designed TDS as an ensemble of multiple TDS instances running on both cloud and edges and maintain a replica of task dependency data on each instance.

To coordinate resource allocation across cloud and edges, coordination layer maintains a *control endpoint* on each cloud and edge side. The *cloud control endpoint* is the centralized entity that receives new resource allocation from the adaptation layer (to be described shortly) and informs other *edge control endpoints* to implement new allocation.

Monitoring layer

Monitoring layer captures performance metrics (c.f. Chapter 5) of workflows, micro-services, and TDS. These metrics are stored in a performance logs database. Performance data is used by adaptation layer (to be described shortly) to make resource allocation decisions. Although monitoring services are running on the cloud, they still can seamlessly communicate with components running on edges to collect the performance metrics, thanks to the deployment of coordination and execution layers across cloud and edges.

Adaptation layer

Adaptation layer is the brain of BRACELET system. This layer consists of a *system performance model* that is trained on the performance logs collected by monitoring layer and provides near future performance predictions to help *resource allocation* module to dynamically allocate resources for micro-services across cloud and edges. We describe adaptation layer in details in Section 11.2.

11.1.2 Edge Cloud Microservice Execution Model

We end the architecture section by describing how micro-services are initially deployed and how workflows are executed seamlessly across cloud and edges by leveraging the dynamic configuration of workflow's task dependencies.

Since data can be uploaded to the cloud either via a cloudlet or directly from advanced instruments (which are able to connect directly to the cloud without cloudlet), all micro-services have to be deployed on the cloud, so that they can be ready to support processing all types of workflows. For each cloudlet, depending on the types of data that is uploaded from instruments to the cloudlet, micro-services of the corresponding data processing workflows have to be deployed on the cloudlet. Therefore, the initial deployment of micro-services on cloud and cloudlets can be decided in advance with knowledge of the types of uploaded data³. For example, if the system supports all types of workflows in Table 11.5, then micro-services of all tasks A, B, C, D are deployed on the cloud. If only data corresponding to workflow Wf1 is uploaded through an edge named E1, then initial deployment on E1 will include micro-services of the tasks in Wf1, namely A.E1, B.E1, and C.E1 (i.e., the edge-specific suffix is used to differentiate with cloud-based deployments of A, B, and C).

³This is a reasonable assumption since the type of uploaded data is specific to the type of instrument, which is known information.

With the above initial deployment, the execution of a workflow across cloud and edge can be conveniently handled by the cloud control endpoint via dynamic configuration of task dependencies on TDS. For example, to execute a workflow $Wf1$ across edge $E1$ and cloud (e.g., processing requests of task A and B on $E1$, and of task C on the cloud), the cloud control endpoint simply creates a new edge-based workflow type on TDS, namely $Wf1.E1$ (Table 11.6), that directs requests of task A and B to their $E1$ -based microservice deployments, namely $A.E1$ and $B.E1$ (task C is still handled by its cloud-based microservice deployment). After creating the new workflow type $Wf1.E1$, cloud control endpoint will inform edge control endpoint at $E1$ to use $Wf1.E1$ as the workflow type to process all requests for $Wf1$ of data being uploaded via $E1$ (instead of the initial cloud-only version of workflow $Wf1$ as shown in Table 11.5).

11.2 BRACELET'S RESOURCE MANAGEMENT

BRACELET's system performance is controlled by allocating resources to micro-services and by placing task computation across edges and cloud. In particular, for each microservice, the more consumers subscribe to a task's request queue, the more requests can be processed in parallel and the less time requests must wait in the queue. Therefore, $m(k)$ influences the work-in-progress $w(k)$ and workflow's processing times $d(k)$. In addition, the flexible execution model of workflows across edge and cloud (presented in Section 11.1.2) enables BRACELET's resource management to make timely decisions on whether to place the computation of a workflow's task on an edge- or cloud-based microservice to balance the workload across the infrastructure.

In this thesis, we present a novel approach to tackle both resource allocation and computation placement challenges of micro-services. In the following sections, we first present a microservice performance model that provides performance predictions of *individual micro-services*. These predictions not only can be used to estimate expected delays of different types of workflows (by aggregating delays of individual micro-services), but also can be used to explore different computation placement options of micro-services and choose the one that minimizes expected processing delays. After introducing the microservice performance model, we will show how to apply the model to solve the resource allocation and computation placement challenges.

11.2.1 Microservice Performance Model

Modeling performance of a system is basically to derive a function that takes system's resource configurations as inputs and produces prediction of system performance in the near future. In this

thesis, we use artificial neural network, a black-box and data-driven approach with proven approximation power and successful applications in modeling performance of non-linear and complex systems, to model the performance of micro-services.

Specifically, the neural network model takes input \mathbf{x} as the combination of microservice performance output (i.e., work-in-progress $\mathbf{w}(k)$) and microservice's resource configurations (i.e., number of consumers per microservice $\mathbf{m}(k)$) in the current time window (T_k, T_{k+1}) , and predicts microservice performance in the next time window (T_{k+1}, T_{k+2}) : $\mathbf{w}(k+1)$ ⁴. The neural network model consists of n layers, with the number of neurals in each layer is denoted as S^i ($1 \leq i \leq n$). Correspondingly, $\mathbf{W}^i, \mathbf{b}^i$ ($1 \leq i \leq n$) represent the weight matrix and bias of layer i -th. Each layer i -th also includes a non-linear (except the last layer that uses the linear identity function) activation function \mathbf{f}^i to introduce the non-linearity into the network model. The neural network model can be described as a function \mathbf{f} of $\mathbf{w}(k)$ and $\mathbf{m}(k)$ via a series of matrix calculations as follows:

$$\begin{aligned}
\mathbf{Z}^1 &= \mathbf{f}^1(\mathbf{W}^1(\mathbf{w}(k) \parallel \mathbf{m}(k))^T + \mathbf{b}^1) \\
\mathbf{Z}^2 &= \mathbf{f}^2(\mathbf{W}^2\mathbf{Z}^1) + \mathbf{b}^2 \\
&\dots \\
\mathbf{Z}^n &= \mathbf{f}^n(\mathbf{W}^n\mathbf{Z}^{n-1}) + \mathbf{b}^n \\
\mathbf{w}(k+1) &= \mathbf{f}(\mathbf{w}(k), \mathbf{m}(k)) = \mathbf{Z}^n
\end{aligned} \tag{11.1}$$

To train the microservice performance model, we define a loss function using standard root mean square error to capture the differences between values of work-in-progress predicted by the model (i.e., $w_j^e(k+1)$) and the values actually observed (i.e., $\hat{w}_j^e(k+1)$) over all micro-services on cloud and edges:

$$L(k+1) = \sqrt{\frac{1}{\mathbb{N}} \sum_{e,j} (w_j^e(k+1) - \hat{w}_j^e(k+1))^2} \tag{11.2}$$

where \mathbb{N} is the total number of microservices on cloud and edges. The model is trained using gradient descent optimizer and backpropagation is used as the gradient computing technique.

⁴Refer to Chapter 5 for a complete introduction on notations of microservice resource and performance.

11.2.2 Microservice Work-in-progress Optimization

We formulate the microservice resource allocation problem as an work-in-progress optimization problem (Problem (11.3)) whose objective is to minimize the work-in-progress across microservices on cloud and edges. Since work-in-progress is proportional to the average delay of each microservice as well as processing delay of workflows, such an objective also corresponds to minimizing the average delay across all workflow types. Specifically, at the end of k -th time window, we would like to solve an optimization problem to find the optimal number of consumers for microservices in the next time window (i.e., $\mathbf{m}(k+1)$) that minimizes the aggregated work-in-progress across all micro-services. The problem is subjected to resource constraints \mathcal{C}^e ($0 \leq e \leq E$) that represents the maximum number of task consumers can be allocated on cloud and on each edge.

$$\begin{aligned} \underset{\mathbf{m}(k+1)}{\text{argmin}} \quad & \sum_{e=0}^E \sum_{j=1}^J w_j^e(k+1) \\ \text{subject to} \quad & \sum_{j=1}^J m_j^e(k+1) \leq \mathcal{C}^e, \forall 0 \leq e \leq E \end{aligned} \tag{11.3}$$

For simplicity, if we assume that the objective function of (11.3) is a linear function of $\mathbf{m}(k+1)$, the optimization (11.3) is an integer linear programming problem, which is NP-hard. In fact, as we often see in a complex system that consists of a number of micro-services with complex dependency relationships to support various types of workflows, the formulation of performance metrics (i.e., $\mathbf{w}(k+1)$) by resource configuration (i.e., $\mathbf{m}(k+1)$) is often a non-linear and complex function. As a result, problem (11.3) could not be solved efficiently by well-known linear programming techniques.

In this thesis, we leverage the learned microservice performance model that captures the relationship between $\mathbf{w}(k+1)$ and $\mathbf{m}(k)$ (i.e., the performance model provides a function $\mathbf{w}(k+1) = \mathbf{f}(\mathbf{w}(k), \mathbf{m}(k))$) and present a greedy strategy (Algorithm 11.1) to efficiently solve the optimization problem (11.3). Specifically, given the current microservice allocation at time k (i.e., $\mathbf{m}(k)$), for each available task consumer that can be allocated (i.e., the while loop from Line 5-10), the algorithm greedily finds the microservice with the most *benefit* if it is allocated one additional consumer (Line 6). The *benefit* is defined to be the decrease in the work-in-progress of a microservice, i.e., $w_j^e(k) - \overline{w_j^e(k+1)}$, in which $\overline{w_j^e(k+1)}$ is the predicted work-in-progress of task j microservice on edge e if we allocate one additional consumer to it (i.e., $m_j^e(k+1) = m_j^e(k) + 1$).

Algorithm 11.1 Microservice Work-in-progress Optimization

```

1: procedure  $\mu$ SERVICEWIPOPT( $\mathbf{m}(k)$ )
2:   Initialize  $\mathbf{m}(k+1) = \mathbf{m}(k)$ 
3:    $\mathcal{C}_j = \{1, \dots, J\}$ 
4:    $\mathcal{C}_e = \{0, \dots, E\}$ 
5:   while  $\sum_{j=1}^J m_j^e(k+1) \leq \mathcal{C}^e (\forall 0 \leq e \leq E)$  do
6:     Find  $(j^*, e^*) = \operatorname{argmax}_{j \in \mathcal{C}_j, e \in \mathcal{C}_e} [w_j^e(k) - \overline{w_j^e(k+1)}]$ 
7:      $m_{j^*}^{e^*}(k+1) = m_{j^*}^{e^*}(k) + 1$ 
8:     if  $\sum_{j=1}^J m_j^{e^*}(k+1) = \mathcal{C}^{e^*}$  then
9:        $\mathcal{C}_e = \mathcal{C}_e \setminus e^*$ 
10:  Return  $\mathbf{m}(k+1)$ 

```

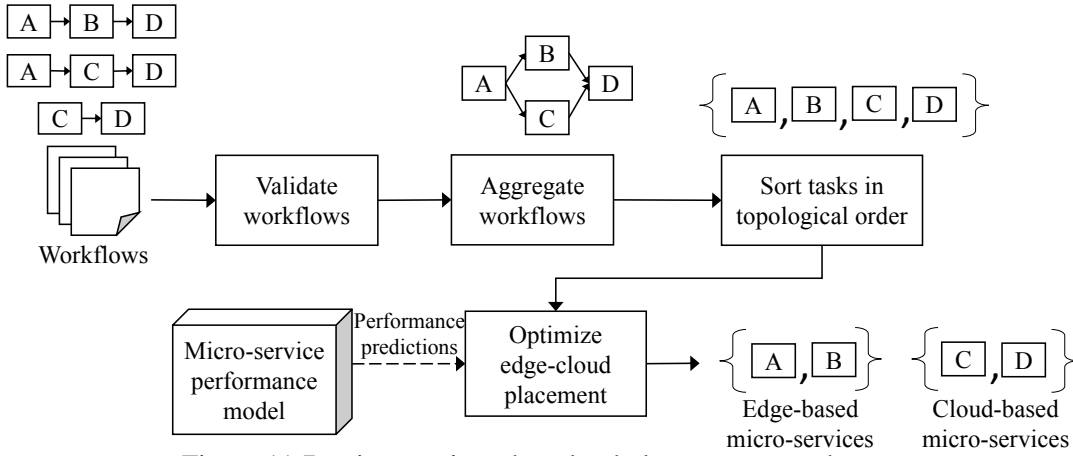


Figure 11.7: microservice edge-cloud placement procedure.

11.2.3 Microservice Computation Placement

As described in Section 11.1.2, the microservice execution model enables flexible placement of computation to edge- and cloud-based micro-services. In this section, we present our microservice placement strategy based on the system performance model shown in Section 11.2-A.

Our strategy is motivated from the following *invariant* of placing computation across cloud and edge: For task micro-services in a workflow (e.g., Wf1 in Figure 11.5), once a microservice (e.g., A) is placed on the cloud, all of its subsequent micro-services in the workflow (i.e., B and C) are also placed on the cloud (i.e., to avoid unnecessary and costly round-trip communications between cloud and edge).

The microservice placement procedure for each branch of an edge and cloud is presented in Figure 11.7. First, all the workflow types that correspond to data uploaded from the edge are validated to ensure that they are in DAG format. Then, all workflow types are aggregated into a single DAG graph. After that, all the tasks in the aggregated graph are sorted in topological order.

The next step is to find an *edge-cloud cut* to partition the set of tasks into two sets: one set whose computation is placed on the edge-based micro-services and another set whose computation is placed on the cloud-based micro-services. The placement procedure iterates over the tasks in the topological order obtained from previous step. At the j -th iteration ($1 \leq j \leq J$), the j -th task in the topological order is considered as the edge-cloud cut. It means that the computation of all tasks up to $(j - 1)$ -th in topological order is placed on the edge-based micro-services, and the computation of those from (j) -th is placed on the cloud-based micro-services.

To evaluate the placement of a task as the edge-cloud cut, we consider whether such placement helps to: (i) minimize the average delays of micro-services that involve in the placement (i.e., *min delay* criteria), and (ii) minimize the communication cost between edge and cloud (i.e., *min communication* criteria). We leverage the microservice performance model learned from Section 11.2-A to quantify these two criteria. In particular, at the current time k , the performance model is used to make predictions about work-in-progress of each microservice in the next time window $w(k + 1)$. By the Little Law, $w(k + 1)$ can be used to estimate the processing delay by summing up the work-in-progress requests of all task micro-services (i.e., quantifying min delay). In addition, the difference between work-in-progress in time k (i.e., $w(k)$) and in time $k + 1$ (i.e., $w(k + 1)$) can also be used to measure the number of requests transitioned between edge-based and cloud-based micro-services, which is proportional to the communication between edge and cloud (i.e., quantifying min communication). As we iterate through the tasks in the topological order, we report the task whose placement as the edge-cloud cut helps minimize average delays and communication cost and use it as the edge-cloud cut in placement decision.

11.2.4 BRACELET's Joint Resource Allocation Procedure

While both microservice placement and WIP optimization procedures rely on the performance model and both can be used to control system performance, placement procedure is more efficient than WIP optimization. In particular, the first three steps of the placement procedure can be done offline and the online step (i.e., optimize edge-cloud placement) only needs to iterate over all types of tasks in an edge-cloud branch. On the other hand, WIP optimization needs to iterate over all task micro-services (edge- and cloud-based ones) *for each* available consumer.

Thus, we combine the two procedures into a joint resource allocation procedure (Figure 11.8). Once started, the procedure keeps running as long as the system operates and periodically checks system performance metrics (via monitoring layer) to see if certain performance guarantee is violated. If there is violation, the procedure will first try to invoke the more efficient edge-cloud

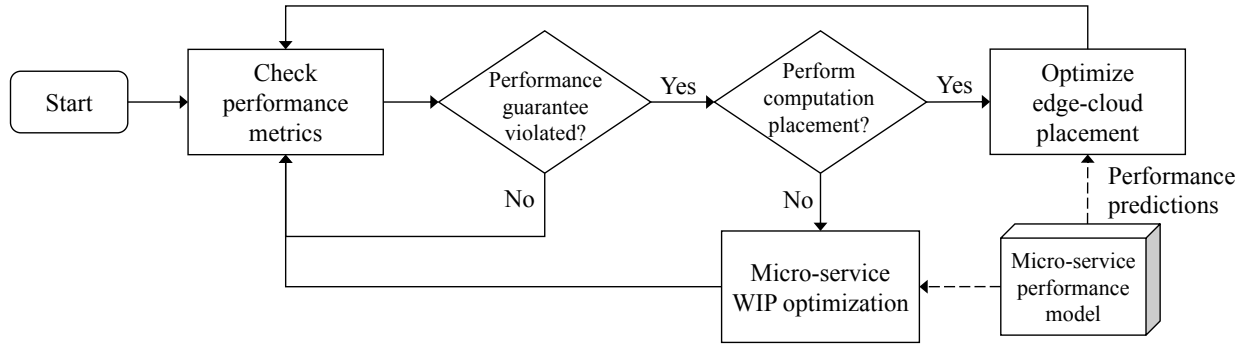


Figure 11.8: BRACELET’s joint resource allocation procedure.

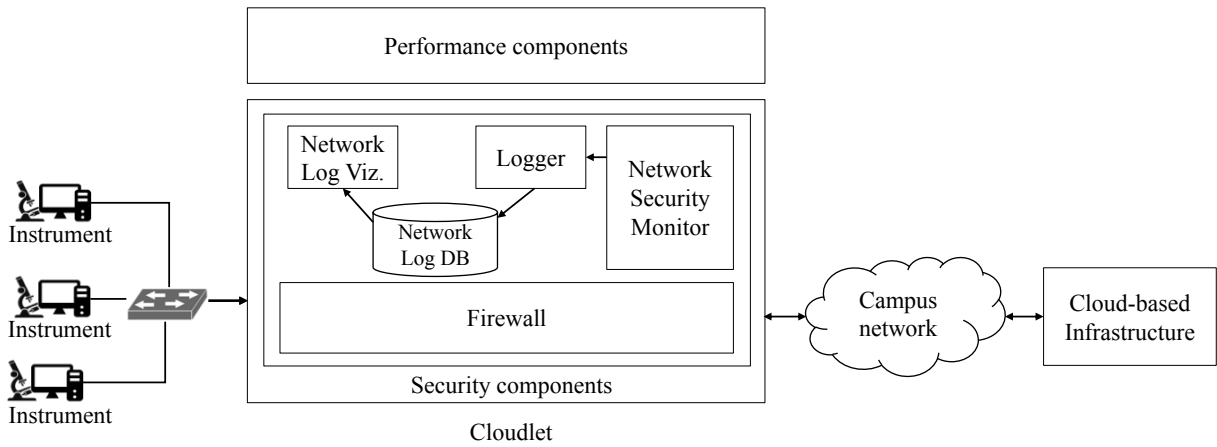


Figure 11.9: BRACELET’s security component design.

placement procedure to mitigate the violation. The procedure will keep trying with edge-cloud placement optimization for a predefined number of times before invoking the more expensive work-in-progress optimization, in case the performance violation could not be mitigated.

11.3 BRACELET’S SECURITY DESIGN

BRACELET’s security components are designed to help protect vulnerable scientific instruments once they are connected to the edge-cloud cyber-infrastructure. They consist of a software firewall that is configured with *whitelisting rules* to enable only data traffic from instruments to the cloud and certain control traffic from the cloud to the cloudlet. Furthermore, each cloudlet also includes a network security monitor component to listen to and capture meta-data of all network traffic in and out of the cloudlet. The security monitor component is also capable of applying customizable scripts to filter and analyze network traffic to detect and alert of potential attacks. All network monitoring logs are collected, parsed, and transformed by a *logger* component, and stored into a

network logs database. Real-time network traffics and statistics can be queried and visualized to BRACELET’s admin by the *network log visualization* component.

In addition to data driven monitoring and detection at cloudlet, all vulnerable scientific instruments are connected to a managed switch so that instrument’s MAC layer address is checked to ensure that the instrument can only talk to cloudlet and not to other peer instruments. At application level, users are required to login on each instrument to upload data, and the login sessions are additionally verified with *instrument reservation database* as part of the two-factor authentication process.

11.4 EVALUATION

11.4.1 Evaluation Settings

We implement BRACELET by extending the implementation of the existing cloud-based microservice infrastructure [45, 53] to the edges. The whole edge-cloud infrastructure cluster is managed by a Kubernetes container orchestration engine and each cloudlet is a remote node in the extended Kubernetes cluster⁵. Each cloudlet has its own locality tagging that is used to schedule micro-services on to the edge. A microservice consists of a RabbitMQ request queue and a set of Docker-based task consumers that are deployed as a ReplicationController set on Kubernetes. TDS service is based on Apache Zookeeper coordination system. We configure Zookeeper and RabbitMQ using ensemble and cluster mode respectively so that we have a Zookeeper and RabbitMQ endpoint on each edge and cloud side (i.e., to improve availability and enable seamless communication between micro-services). Monitoring layer’s implementation is similar to the one in [53], and we use Tensorflow to build microservice performance model used in the adaptation layer. We use Bro as the network security monitor at cloudlet and use ELK stack for logging, storing, and visualizing the collected network security logs.

In terms of the workload, we use the MDP workflow ensemble [45, 53] that supports processing experimental data generated by digital microscopes. MDP consists of three types of workflows (numbered 1, 2, and 3) and four types of tasks (named A, B, C, and D).

In order to obtain training data to train the performance model, we let the system runs through a *bootstrapping process* in which we randomly vary the arrival rates of incoming requests of different workflow types and randomly vary the allocation of consumers across microservices (i.e., $\mathbf{m}(k)$)

⁵The BRACELET system is deployed on a cloud-based cluster of two nodes, each node is equipped with an Intel Xeon quad core processor, 1.2Ghz per core, and 16GB of RAM, and two cloudlets, each cloudlet is equipped with Intel Core i7 CPU 3.4Ghz and 8GB of RAM.

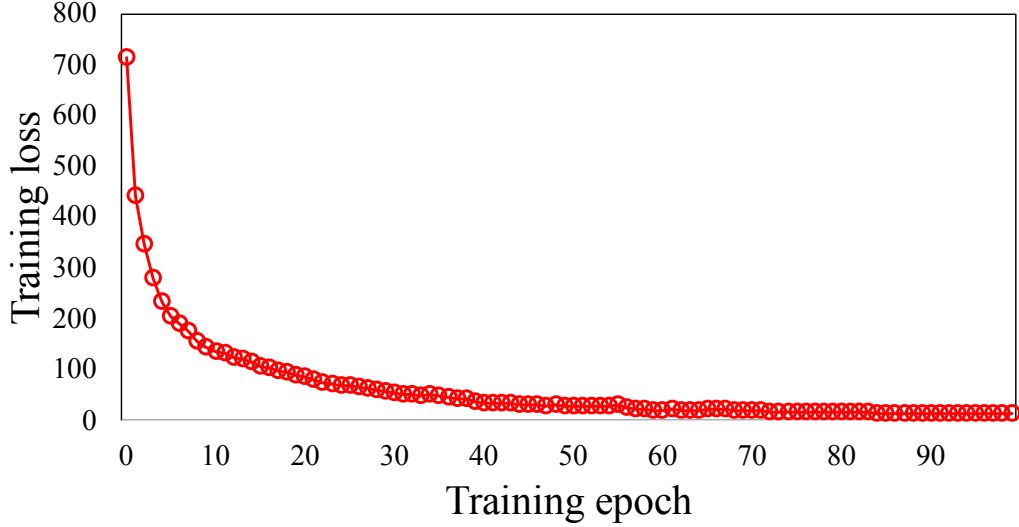


Figure 11.10: Training error of microservice performance model.

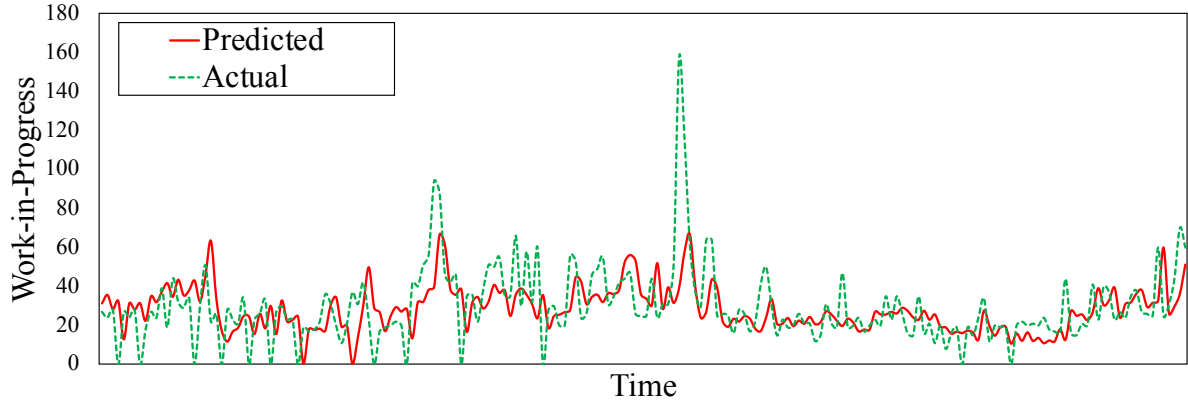
as well as the computation placement of tasks in a workflow across cloud and edges. We record the actual performance output $\hat{\mathbf{w}}(k+1) = \{\hat{w}_j^e(k+1)\}$ to use as the ground-truth data to train the model.

To evaluate BRACELET’s microservice placement and work-in-progress optimization strategies, we emulate up to 5x spikes of workflow requests to BRACELET (both cloud and edge sides) and measure how effective our approaches is, compared to related approaches. BRACELET’s resource allocation procedure is invoked if average delay $\bar{d}(k)$ exceeds performance guarantee of 20 seconds.

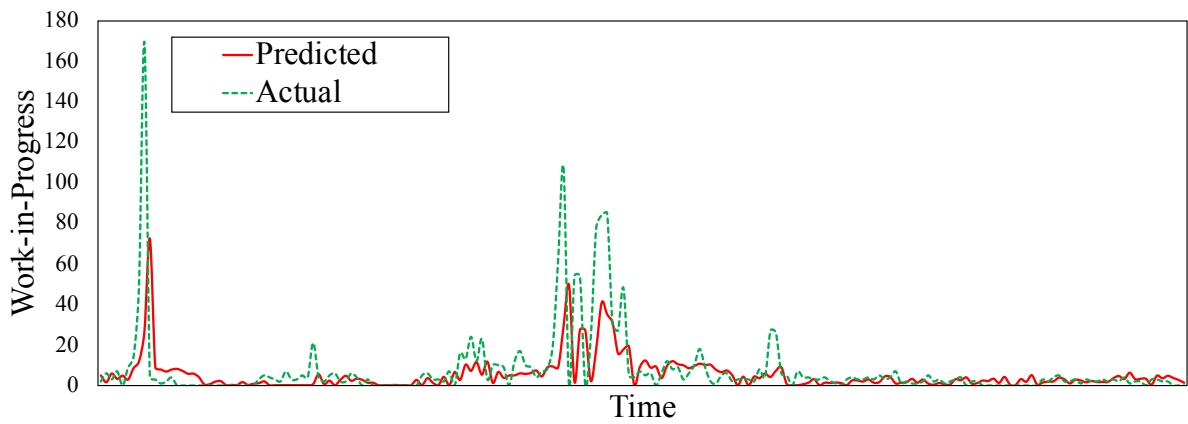
11.4.2 Evaluation of Microservice Performance Model

We design the neural network with two hidden layers and an output layer (i.e., total number of layers $n = 3$), with the number of hidden neurals S^1 and S^2 both equal 128, and f^1 and f^2 are ReLU function. For training, we set learning rate as 0.001, batch size as 100, and use 100 training epochs. Figure 11.10 shows the training loss $L(k+1)$ over training epochs. It verifies the effectiveness of using neural network to capture performance of micro-services.

Figure 11.11 evaluates the generalization of trained performance model when testing with unseen data. It shows that the model is able to predict quite accurately the near future performance (measured by work-in-progress) of micro-services on both cloud and edge. Especially, it can predict the spikes in work-in-progress very effectively, which is vital in making resource allocation decisions.



(a) Cloud-based microservice D



(b) Edge-based microservice C on cloudlet E1

Figure 11.11: Effectiveness of microservice performance prediction. Each time step on x-axis is 15 seconds apart, which corresponds to the length of time interval (T_k, T_{k+1}) .

11.4.3 Evaluation of Microservice Computation Placement

We fix the number of consumers of micro-services $m(k)$ and evaluate how BRACELET's microservice computation placement strategy can handle sudden spike in the incoming requests. We compare our placement strategy with other related approaches:

- *Bandwidth-optimized* [47]: Initially, all requests are handled by cloud-based micro-services. When performance guarantee is violated, the processing of requests that arrive from an edge is offloaded to the edge-based micro-services.
- *Delay-optimized*: This strategy is often employed in mobile cloud computing scenario. Initially, requests arriving from an edge are handled by edge-based micro-services. When performance violation occurs, the processing of requests is offloaded to the cloud-based micro-services.

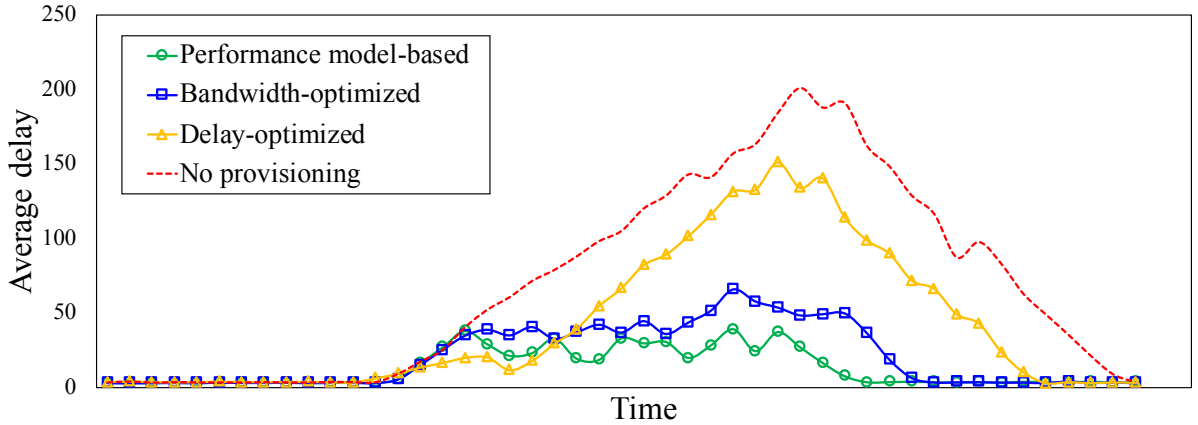


Figure 11.12: Effectiveness of BRACELET’s microservice placement strategy compared with others. Each time step on x-axis is 15 seconds apart, which corresponds to the length of time interval (T_k, T_{k+1}) .

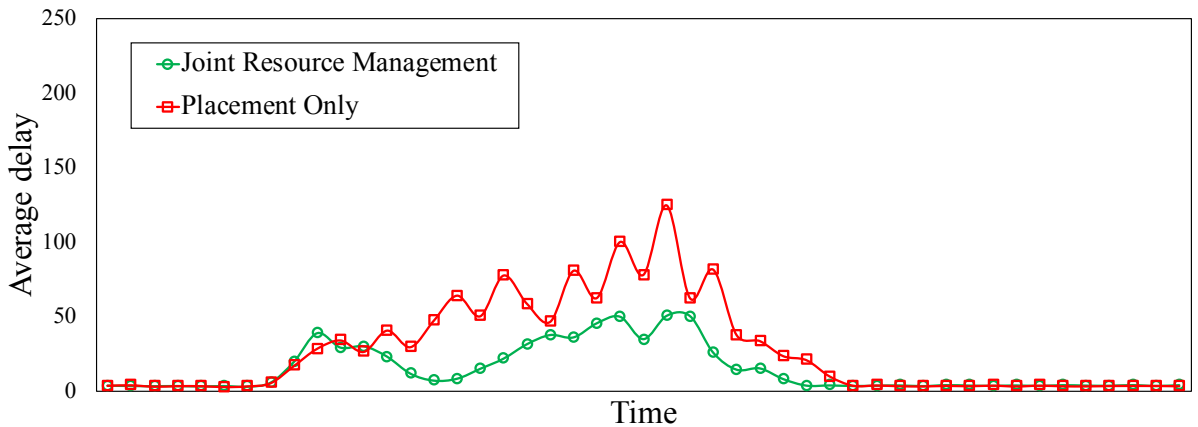


Figure 11.13: Effectiveness of BRACELET’s joint resource allocation procedure. Each time step on x-axis is 15 seconds apart.

The result in Figure 11.12 shows that our computation placement strategy outperforms other approaches by dynamically evaluating different placement options using the accurate performance model (i.e., the dynamism of placement decisions is captured by the ups and downs in average delay when using our strategy). Delay-optimized scheme performs poorly since it creates congestion on cloud-based micro-services when performance violation occurs.

11.4.4 Evaluation of Joint Resource Allocation Procedure

It is intuitive that, given additional consumers to allocate, the microservice WIP optimization procedure can help improve the result when using only microservice computation placement. We show that, even *without* any additional consumer to allocate (i.e., the extreme case), the joint procedure introduced in Section 11.2 can still achieve better result by smartly re-arranging consumers

among micro-services to the ones that are most in need, compared to when using only microservice computation placement.

In particular, given an initial allocation of consumers over micro-services, when performance guarantee is violated, the joint procedure will first try with changing computation placement. After a number of attempts (i.e., 3 retries in our evaluation), without any additional consumers, the joint procedure will try to re-arrange the current set of consumers and re-allocate them to micro-services that are most beneficial from such re-allocation using Algorithm 11.1. Result in Figure 11.13 shows that such the joint approach greatly helps improve the result of resource allocation even in the extreme case when there is no additional resource.

CHAPTER 12: CONCLUSIONS

In this thesis, we present DOSSIER - our holistic approach in designing a distributed operating system and infrastructure for scientific data management. As part of DOSSIER's novel architecture, we take a radical approach to design a microservice execution environment for scientific workflows that enables more flexible and dynamic composition of workflows, and thus, is efficient in dealing with heterogeneous workflows. At the core of DOSSIER is an *adaptive control microservice infrastructure* that is designed to tackle the diversity challenges of data cyberinfrastructure for distributed scientific data management, including data, task, user, workload, and device diversity. We present in this thesis a number of realizations of the adaptive control framework and discuss pros and cons, as well as, recommend situations where each solution can be applied.

To support developing new scientific data management applications, DOSSIER offers a variety of core functionalities via service model, such as data management service, data curation, workflow composition, etc. Using these services, users can easily develop new applications with domain-specific interfaces for data inputting, and new domain-specific data processing tasks to be deployed and run on DOSSIER's microservice infrastructure. Some results and lessons from the development of DOSSIER have been applied into production environment at the University of Illinois at Urbana-Champaign to serve users in material sciences. In the future, we look forward to extending DOSSIER's application to other related domains that share similar characteristics to material sciences, and investigating other extensions and improvements of DOSSIER, such as cross-institutional federated data cyberinfrastructure, combining scientific experiment data with sensory data for provenance and reproducibility of experiments, further enhancing search and data correlation capability of DOSSIER to accelerate scientific discovery, to name a few.

By designing, developing, and testing DOSSIER in the real environments, we demonstrate that an edge-cloud microservice architecture with learning-based adaptive control resource management is needed for timely distributed scientific data management.

APPENDIX A: DETAILS ON PARAMETRIC DECOMPOSITION PROCEDURE

In this section, we describe in details how to use parametric decomposition to derive aggregated job arrive rate and scv at each topic, when the elastic pub/sub system is modeled as a generalized Jackson OQN (i.e., each topic is modeled a GI/G/m queue). In this thesis, we employ a parametric decomposition procedure similar to the one described in [55].

The realistic assumption about the general distribution of job arrival and processing rates makes aggregating multiple types of job more difficult. In the special case, where the job arrival rates and processing rates are exponentially distributed and each topic is modeled as a M/M/m queue, we can aggregate multiple job types by just simply summing up the arrival rates of all job types at a topic (since the combination of exponential distribution is also exponential). In addition, since the rate of a job type departing a topic is the same as the arrival rate (Burke's theorem [66]), the aggregation at each topic is not affected by the flows of different job types across topics. On the other hand, in generalized case, as the rates are no longer exponential, the aggregation of multiple job types depend on the flows of jobs across topics.

To analyze the flows of different types of job requests across topics, parametric decomposition employs a *divide-and-conquer* approach (hence decomposition). Particularly, we treat each topic in isolation and divide the workflows, each corresponds to a type of job, into three basic building blocks: *merge*, *split*, and *follow through*.

Merge

The merge building block represents the merging of arrival requests of multiple types of job at a topic. The aggregated job request arrival rates at topic j is approximated as follow:

$$\tilde{\lambda}_j = \sum_{i=1}^N \lambda_i 1_{ij} \quad (\text{A.1})$$

where λ_i is the expected arrival rate of job type i and 1_{ij} is an indicator function: $1_{ij} = 1$ if job type i goes through topic j and $1_{ij} = 0$ otherwise.

And the aggregated scv of arrival rates over all types of job types is calculated as follow:

$$\tilde{ca}_j^2 = \sum_{i=1}^N \frac{\lambda_i}{\tilde{\lambda}_j} ca_{ij}^2 1_{ij} \quad (\text{A.2})$$

where ca_{ij}^2 is the scv of the arrival rate of job type i at topic j .

Follow through

As an aggregated stream of multiple types of job requests enters, is processed, and departs topic j , the scv of aggregated departure stream is approximated as follow [67]:

$$\tilde{cd}_j^2 = 1 + (1 - \rho_j^2)(\tilde{ca}_j^2 - 1) + \frac{\rho_j^2}{(m_j)^{\frac{1}{2}}}(cs_j^2 - 1) \quad (\text{A.3})$$

where ρ_j is the *traffic intensity* at topic j : $\rho_j = \frac{\tilde{\lambda}_j}{\mu_j m_j}$. Please also note that the departure stream is equivalent to the arrival stream to the follow-up topic.

Split

After being processed by a topic, aggregated stream of multiple types of job might be splitted into different paths as it departs the topic. The scv of individual departure stream of product type i from topic j , cd_{ij}^2 , can be approximated as follow [68]:

$$cd_{ij}^2 = \frac{\lambda_i}{\tilde{\lambda}_j} \tilde{cd}_j^2 + (1 - \frac{\lambda_i}{\tilde{\lambda}_j}) \times (\frac{\lambda_i}{\tilde{\lambda}_j} + (1 - \frac{\lambda_i}{\tilde{\lambda}_j}) ca_{ij}^2) \quad (\text{A.4})$$

REFERENCES

- [1] A. R. Ferguson, J. L. Nielson, M. H. Cragin, A. E. Bandrowski, and M. E. Martone, “Big data from small data: data-sharing in the ‘long tail’ of neuroscience,” *Nature neuroscience*, vol. 17, no. 11, p. 1442, 2014.
- [2] N. Science and T. C. (OSTP), “Materials genome initiative for global competitiveness,” 2011.
- [3] C. Strasser, J. Kunze, S. Abrams, and P. Cruse, “Dataup: A tool to help researchers describe and share tabular data,” *F1000Research*, vol. 3, 2014.
- [4] A. S. Szalay, J. Gray, A. R. Thakar, P. Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg, “The sdss skyserver: public access to the sloan digital sky server data,” in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, 2002, pp. 570–581.
- [5] B. Plale, R. H. McDonald, K. Chandrasekar, I. Kouper, S. Konkiel, M. L. Hedstrom, J. Myers, and P. Kumar, “Sead virtual archive: Building a federation of institutional repositories for long-term data preservation in sustainability science,” *International Journal of Digital Curation*, vol. 8, no. 2, pp. 172–180, 2013.
- [6] M. McLennan and R. Kennell, “Hubzero: a platform for dissemination and collaboration in computational science and engineering,” *Computing in Science & Engineering*, vol. 12, no. 2, 2010.
- [7] G. Klimeck, M. McLennan, S. P. Brophy, G. B. Adams III, and M. S. Lundstrom, “nanohub.org: Advancing education and research in nanotechnology,” *Computing in Science & Engineering*, vol. 10, no. 5, pp. 17–23, 2008.
- [8] S. Padhy, G. Jansen, J. Alameda, E. Black, L. Diesendruck, M. Dietze, P. Kumar, R. Kooper, J. Lee, R. Liu et al., “Brown dog: Leveraging everything towards autocuration,” in *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE, 2015, pp. 493–500.
- [9] M. S. Mayernik, G. S. Choudhury, T. DiLauro, E. Metsger, B. Pralle, M. Rippin, and R. Duerr, “The data conservancy instance: Infrastructure and organizational services for research data curation,” *D-Lib Magazine*, vol. 18, no. 9/10, 2012.
- [10] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, “A survey of data-intensive scientific workflow management,” *Journal of Grid Computing*, vol. 13, no. 4, pp. 457–493, 2015.
- [11] E. Deelman, D. Gannon, M. Shields, and I. Taylor, “Workflows and e-science: An overview of workflow system features and capabilities,” *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.
- [12] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good et al., “Pegasus: A framework for mapping complex scientific workflows onto distributed systems,” *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

- [13] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat et al., “Taverna: a tool for the composition and enactment of bioinformatics workflows,” *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.
- [14] I. Taylor, M. Shields, I. Wang, and A. Harrison, “Visual grid workflow in triana,” *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 153–169, 2005.
- [15] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, “Kepler: an extensible system for design and execution of scientific workflows,” in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*. IEEE, 2004, pp. 423–424.
- [16] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. D’Souza, S. Devoid, D. Murphy-Olson, N. Desai et al., “Skyport: container-based execution environment management for multi-cloud scientific workflows,” in *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*. IEEE Press, 2014, pp. 25–32.
- [17] W. Tang, J. Wilkening, N. Desai, W. Gerlach, A. Wilke, and F. Meyer, “A scalable data analysis platform for metagenomics,” in *Big Data, 2013 IEEE International Conference on*. IEEE, 2013, pp. 21–26.
- [18] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth et al., “Apache hadoop yarn: Yet another resource negotiator,” in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.
- [19] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, omega, and kubernetes,” *ACM Queue*, vol. 14, pp. 70–93, 2016. [Online]. Available: <http://queue.acm.org/detail.cfm?id=2898444>
- [20] D. Gunter, E. Deelman, T. Samak, C. H. Brooks, M. Goode, G. Juve, G. Mehta, P. Moraes, F. Silva, M. Swamy et al., “Online workflow management and performance analysis with stampede,” in *Proceedings of the 7th International Conference on Network and Services Management*. International Federation for Information Processing, 2011, pp. 152–161.
- [21] F. Horta, J. Dias, K. A. Ocana, D. de Oliveira, E. Ogasawara, and M. Mattoso, “Using provenance to visualize data from large-scale experiments,” in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion*. IEEE, 2012, pp. 1418–1419.
- [22] F. Costa, V. Silva, D. De Oliveira, K. Ocaña, E. Ogasawara, J. Dias, and M. Mattoso, “Capturing and querying workflow runtime provenance with prov: a practical approach,” in *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. ACM, 2013, pp. 282–289.
- [23] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, “On the use of cloud computing for scientific workflows,” in *eScience, 2008. eScience’08. IEEE Fourth International Conference on*. IEEE, 2008, pp. 640–645.
- [24] E. Deelman, “Grids and clouds: Making workflow applications work in heterogeneous distributed environments,” *International Journal of High Performance Computing Applications*, vol. 24, no. 3, pp. 284–298, 2010.

- [25] J. Wang, P. Korambath, I. Altintas, J. Davis, and D. Crawl, “Workflow as a service in the cloud: architecture and scheduling algorithms,” *Procedia Computer Science*, vol. 29, pp. 546–556, 2014.
- [26] Y. Zhao, Y. Li, W. Tian, and R. Xue, “Scientific-workflow-management-as-a-service in the cloud,” in *Cloud and Green Computing (CGC), 2012 Second International Conference on*. IEEE, 2012, pp. 97–104.
- [27] C. Zheng and D. Thain, “Integrating containers into workflows: A case study using makeflow, work queue, and docker,” in *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing*. ACM, 2015, pp. 31–38.
- [28] K. Liu, K. Aida, S. Yokoyama, and Y. Masatani, “Flexible container-based computing platform on cloud for scientific workflows,” in *Cloud Computing Research and Innovations (IC-CCRI), 2016 International Conference on*. IEEE, 2016, pp. 56–63.
- [29] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, “Mesos: A platform for fine-grained resource sharing in the data center.”
- [30] J. Yu, R. Buyya, and C. K. Tham, “Cost-based scheduling of scientific workflow applications on utility grids,” in *e-Science and Grid Computing, 2005. First International Conference on*. Ieee, 2005, pp. 8–pp.
- [31] S. Abrishami, M. Naghibzadeh, and D. H. Epema, “Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds,” *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.
- [32] H. Zhao and R. Sakellariou, “Scheduling multiple dags onto heterogeneous systems,” in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 14–pp.
- [33] R. Tolosana-Calasanz, J. Á. Bañares, C. Pham, and O. F. Rana, “Enforcing qos in scientific workflow systems enacted over cloud infrastructures,” *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1300–1315, 2012.
- [34] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, “Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds,” *Future Generation Computer Systems*, vol. 48, pp. 1–18, 2015.
- [35] E. Ogasawara, J. Dias, F. Porto, P. Valduriez, and M. Mattoso, “An algebraic approach for data-centric scientific workflows,” *Proc. of VLDB Endowment*, vol. 4, no. 12, pp. 1328–1339, 2011.
- [36] W. Chen and E. Deelman, “Partitioning and scheduling workflows across multiple sites with storage constraints,” *Parallel Processing and Applied Mathematics*, pp. 11–20, 2012.
- [37] S.-M. Park and M. Humphrey, “Predictable high-performance computing using feedback control and admission control,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 396–411, 2011.

- [38] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in software services," *IEEE Control Systems*, vol. 23, no. 3, pp. 74–90, 2003.
- [39] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 9, pp. 1014–1027, 2006.
- [40] B. Li and K. Nahrstedt, "A control-based middleware framework for quality-of-service adaptations," *IEEE journal on selected areas in communications*, vol. 17, no. 9, pp. 1632–1650, 1999.
- [41] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Real-Time Systems*, vol. 23, no. 1, pp. 85–126, 2002.
- [42] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM computer communication review*, vol. 18, no. 4. ACM, 1988, pp. 314–329.
- [43] M. T. Hagan, H. B. Demuth, and O. D. Jesús, "An introduction to the use of neural networks in control systems," *International Journal of Robust and Nonlinear Control*, vol. 12, no. 11, pp. 959–985, 2002.
- [44] S. DI, "Neural generalized predictive control: A newton-raphson implementation," 1997.
- [45] P. Nguyen, S. Konstanty, T. Nicholson, T. Obrien, A. Schwartz-Duval, T. Spila, K. Nahrstedt, R. H. Campbell, I. Gupta, M. Chan, K. McHenry, and N. Paquin, "4ceed: Real-time data acquisition and analysis framework for material-related cyber-physical environments," in *Cluster, Cloud and Grid Computing (CCGrid), 2017 17th IEEE/ACM International Symposium on*. IEEE, 2017.
- [46] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
- [47] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, 2016, pp. 1–9.
- [48] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, 2017, pp. 1–9.
- [49] L. Wang, L. Jiao, D. Kliazovich, and P. Bouvry, "Reconciling task assignment and scheduling in mobile edge clouds," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*, 2016, pp. 1–6.
- [50] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic distributed resource scaling for infrastructure-as-a-service." in *Autonomic Computing (ICAC), 2013 USENIX International Conference on*, vol. 13, 2013, pp. 69–82.

- [51] A. Matsunaga and J. A. Fortes, “On the use of machine learning to predict the time and resources consumed by applications,” in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 495–504.
- [52] R. C.-L. Chiang, J. Hwang, H. H. Huang, and T. Wood, “Matrix: Achieving predictable virtual machine performance in the clouds,” in *Autonomic Computing (ICAC), 2014 USENIX International Conference on*, 2014, pp. 45–56.
- [53] P. Nguyen and K. Nahrstedt, “Monad: Self-adaptive micro-service infrastructure for heterogeneous scientific workflows,” in *Autonomic Computing (ICAC), 2017 14th IEEE International Conference on*. IEEE, 2017.
- [54] G. R. Bitran and R. Morabito, “State-of-the-art survey: Open queueing networks: Optimization and performance evaluation models for discrete manufacturing systems,” *Production and Operations Management*, vol. 5, no. 2, pp. 163–193, 1996.
- [55] M. Van Vliet and A. H. R. Kan, “Machine allocation algorithms for job shop manufacturing,” *Journal of Intelligent Manufacturing*, vol. 2, no. 2, pp. 83–94, 1991.
- [56] O. J. Boxma, A. R. Kan, and M. van Vliet, “Machine allocation problems in manufacturing networks,” *European Journal of Operational Research*, vol. 45, no. 1, pp. 47–54, 1990.
- [57] H. C. Tijms, *Stochastic Modelling and Analysis: A Computational Approach*. New York, NY, USA: John Wiley & Sons, Inc., 1986.
- [58] W. Whitt, “Approximations for the gi/g/m queue,” *Production and Operations Management*, vol. 2, no. 2, pp. 114–161, 1993.
- [59] M. Dyer and L. Proll, “On the validity of marginal analysis for allocating servers in m/m/c queues,” *Management Science*, vol. 23, no. 9, pp. 1019–1022, 1977.
- [60] G. F. Franklin et al., *Feedback control of dynamic systems*. Addison-Wesley Reading, MA, 1994, vol. 3.
- [61] M. T. Hagan et al., “An introduction to the use of neural networks in control systems,” *International Journal of Robust and Nonlinear Control*, vol. 12, no. 11, pp. 959–985, 2002.
- [62] S. DI, “Neural generalized predictive control: A newton-raphson implementation,” 1997.
- [63] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.
- [64] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [65] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

- [66] P. J. Burke, "The output of a queuing system," *Operations research*, vol. 4, no. 6, pp. 699–704, 1956.
- [67] W. Whitt, "The queueing network analyzer," *Bell Labs Technical Journal*, vol. 62, no. 9, pp. 2779–2815, 1983.
- [68] G. R. Bitran and D. Tirupati, "Multiproduct queueing networks with deterministic routing: Decomposition approach and the notion of interference," *Management Science*, vol. 34, no. 1, pp. 75–100, 1988.