University of California
Santa Barbara

Using Workload Prediction and Federation to Increase Cloud Utilization

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Alexander E. Pucher

September 2016

The Dissertation of Alexander E. Pucher is approved.

Professor Chandra Krintz

Professor Amr El Abbadi

Professor Rich Wolski, Committee Chair

August 2016

Using Workload Prediction and Federation to Increase Cloud Utilization

Acknowledgements

I thank my amazing advisors Rich Wolski and Chandra Krintz who have guided me during these past years of my journey to the edge of the state of the art in distributed systems and cloud computing. Countless conversations and inputs encouraged me to follow my curiosity, expand on existing work, and take on a big challenge with its calculated risk. I also thank Divyakant Agrawal and Amr El Abbadi who took me into their fold during my early days at UC Santa Barbara and have been supportive ever since.

The process of doctoral studies is more than just an intellectual pursuit. I am grateful to have made great friends in the UC Santa Barbara community and all around California, friends with whom I spent many many hours and went on interesting and fun adventures. A special thank you goes to my lab mates who I have spent more time with than anyone else over the past three years. We shared the daily grind, waded together through the highs and lows of the PhD experience and still had a lot of fun.

Finally, I thank my family for their dedicated and sustained support all around the globe, wherever my studies and work took me – in intellectual, emotional, and worldly ways. I cherish the frequent conversations, and the visits and great hospitality at home.

Without all of you, I would not have reached this point. Thank you.

Curriculum Vitæ
Alexander E. Pucher

Education

2016	Ph.D. in Computer Science (Expected), University of California, Santa Barbara, United States.
2010	M.S. in Computer Science, Vienna University of Technology, Austria.
2009	B.S. in Computer Science, Vienna University of Technology, Austria.

Publications

Providing Lifetime Service-Level-Agreements for Cloud Spot Instances
A. Pucher, R. Wolski, C. Krintz
International Conference on Grid and Cloud Computing and Applications (GCA), 2015.

[Best Paper Award]
Using Trustworthy Simulation to Engineer Cloud Schedulers
A. Pucher, E. Gul, C. Krintz, and R. Wolski
IEEE International Conference on Cloud Engineering (IC2E), 2015.

Using Syntactic and Semantic Similarity of Web APIs to Estimate Porting Effort
H. Jayathilaka, A. Pucher, C. Krintz, and R. Wolski
International Journal of Services Computing (IJSC), 2014.

Cloud Platform Support for API Governance
C. Krintz, H. Jayathilaka, S. Dimopoulos, A. Pucher, R. Wolski, and T. Bultan
IC2E Workshop on the Future of PaaS, 2014.

Characterizing Tenant Behavior for Placement and Crisis Mitigation in Multi-Tenant DBMSs
A. J. Elmore, S. Das, A. Pucher, D. Agrawal, A. El Abbadi, and X. Yan
ACM SIGMOD International Conference on Management of Data, 2013.

Low-Latency Multi-Datacenter Databases Using Replicated Commit
H. Mahmoud, F. Nawab, A. Pucher, D. Agrawal, and A. El Abbadi
International Conference on Very Large Databases (VLDB), 2013.

TritonSort: A Balanced Large-Scale Sorting System
A. Rasmussen, G. Porter, M. Conley, H. Madhyastha, R. Mysore, A. Pucher, and A. Vahdat
USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2011.

Abstract

Using Workload Prediction and Federation to Increase Cloud Utilization

by

Alexander E. Pucher

The wide-spread adoption of cloud computing has changed how large-scale computing infrastructure is built and managed. Infrastructure-as-a-Service (IaaS) clouds consolidate different separate workloads onto a shared platform and provide a consistent quality of service by overprovisioning capacity. This additional capacity, however, remains idle for extended periods of time and represents a drag on system efficiency.

The smaller scale of private IaaS clouds compared to public clouds exacerbates over-provisioning inefficiencies as opportunities for workload consolidation in private clouds are limited. Federation and cycle harvesting capabilities from computational grids help to improve efficiency, but to date have seen only limited adoption in the cloud due to a fundamental mismatch between the usage models of grids and clouds. Computational grids provide high throughput of queued batch jobs on a best-effort basis and enforce user priorities through dynamic job preemption, while IaaS clouds provide immediate feedback to user requests and make ahead-of-time guarantees about resource availability.

We present a novel method to enable workload federation across IaaS clouds that overcomes this mismatch between grid and cloud usage models and improves system efficiency while also offering availability guarantees. We develop a new method for faster-than-realtime simulation of IaaS clouds to make predictions about system utilization and leverage this method to estimate the future availability of preemptible resources in the cloud. We then use these estimates to perform careful admission control and provide ahead-of-time bounds on the preemption probability of federated jobs executing on preemptible resources. Finally, we build an end-to-end prototype that addresses practical issues of workload federation and evaluate the prototype's efficacy using real-world traces from big data and compute-intensive production workloads.

Contents

Curriculum Vitae **v**

Abstract **vii**

1 Introduction **1**
- 1.1 Thesis Statement . . . 5
- 1.2 Challenges . . . 5
- 1.3 Methodology . . . 6
- 1.4 Contributions . . . 7
- 1.5 Permissions and Attributions . . . 8

2 Background **9**
- 2.1 Federation in Large-Scale Computing . . . 12
  - 2.1.1 Globus . . . 13
  - 2.1.2 Condor . . . 14
  - 2.1.3 Hadoop . . . 17
- 2.2 Federation in IaaS Clouds . . . 19

3 Validated Simulation For Engineering Cloud Schedulers **22**
- 3.1 Introduction . . . 22
- 3.2 Related Work . . . 26
- 3.3 Methodology . . . 29
  - 3.3.1 Model Construction . . . 31
  - 3.3.2 Discrete Event Simulation . . . 32
  - 3.3.3 Adding Perturbations . . . 33
  - 3.3.4 Scheduler Operation . . . 34
- 3.4 Results . . . 35
  - 3.4.1 Simulation Registration . . . 36
  - 3.4.2 Power-Aware Scheduler at Scale . . . 41
  - 3.4.3 Capacity Planning . . . 44
  - 3.4.4 Capacity Planning for Scale-Out Workloads . . . 46

3.5 Conclusion 49

4 Estimating Job Preemption Probability in IaaS Clouds 51
4.1 Introduction 51
4.2 Related Work 54
4.3 Methodology 57
4.3.1 Scheduling Model 59
4.3.2 Preemption Policy 60
4.3.3 Predicting Preemption 60
4.3.4 Evaluation Metrics 61
4.4 Results 62
4.4.1 Conditional Distributions and Sample Size 64
4.4.2 Prediction with Production Traces 66
4.4.3 SLA-Aware Co-Scheduling of Production Traces 70
4.4.4 SLA-Aware Co-Scheduling with Platform Scaling 72
4.5 Conclusion 74

5 IaaS Cloud Federation Using Preemptible Resources 76
5.1 Introduction 76
5.2 Related Work 81
5.3 Methodology 83
5.3.1 Federation Architecture 85
5.3.2 Instance Lifetime Guarantees 87
5.3.3 Estimating Native Utilization 89
5.3.4 Estimating Federation Overheads 90
5.3.5 Admission Control 92
5.3.6 Evaluation Traces 96
5.3.7 Evaluation Metrics 99
5.4 Results 100
5.4.1 Federation Baseline 102
5.4.2 Federation with SLA Guarantees 104
5.4.3 Federation with Platform Scaling 105
5.4.4 Efficiency Gains at Scale 108
5.4.5 Sensitivity to Duration Estimates 110
5.4.6 Sensitivity to Seasonality 112
5.5 Conclusion 113

6 Conclusion 114

7 Future Work 117

Bibliography 120

Chapter 1

Introduction

The wide-spread adoption of cloud computing has lead to a change in how large-scale computing infrastructure is built and managed. A "cloud" abstracts away the details of the underlying hardware resources in large computer systems and exposes system capabilities as services. Clouds also provide a high-level application programming interface (API) that simplifies provisioning of resources and comes with guarantees about their expected quality of service (QoS). The degree of abstraction and the associated QoS guarantees differ between the three types of "cloud" paradigms used in practice – Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

IaaS clouds provide on-demand access to virtualized cluster infrastructure, such as "instances" – virtual machines with associated storage and network interface – and an "object store" – a scalable key-value data store accessible via a REST-base API. Platform-as-a-Service (PaaS) abstracts away the operating system and software stack and enables rapid development of network-accessible applications against a streamlined and highly scalable API. Software-as-a-Service (SaaS) refers to software solutions delivered with an always-online model. IaaS and PaaS clouds typically cater to developers, whereas SaaS applications are tailored to end user needs.

The services offered by clouds, together with their QoS guarantees, are summarized in the cloud's service level agreement (SLA) [1]. The service quality is measured over fixed periods of time using different metrics. The specific metrics, their measurement windows and corresponding targets are defined as service level objectives (SLOs). These SLOs form the foundation for cloud customers' reasoning about the expected platform performance.

The IaaS model in particular has been adopted for "public clouds" and "private clouds" [2]. Public clouds are operated by commercial providers and offer customers access to resources with a pay-as-you-go model. Public provider infrastructure typically relies on large data centers and proprietary software stacks. Private clouds are deployed by a wide variety of organizations internally on enterprise IT infrastructure. Private clouds exist at varying scales and commonly use open-source frameworks for implementation. The decision between "outsourcing" to a public cloud and "insourcing" to a private cloud is driven by numerous factors such as privacy, cost and customization requirements.

Clouds consolidate workloads from different users onto the same platform and aim to create the illusion of "infinite" resources available to any individual user. The consolidation of different workloads smoothens demand so peaks in the workload of one user can be compensated for by a low in another user's workload. In order to create the illusion of infinite resources, clouds "overprovision" available capacity to handle worst-case peak demand. This additional hardware and their auxillary support systems, however, remain idle during non-peak times and represent a drag on overall system utilization and resource efficiency.

The smaller scale of private IaaS clouds relative to public clouds creates additional challenges. While public clouds rely on a diverse user base and take advantage of economics of scale, private clouds are smaller and cater to a more specialized audience. With the smaller user base comes less variety in demand patterns and thus fewer opportunities to smooth out aggregate resource demand. Finally, multiple private clouds may co-exist within the same organization in order to consistently guarantee access to part of the available resources for each organizational unit. This fragmentation further reduces the pool of workload available for consolidation and exacerbates the issue of overprovisioning.

Efficiency problems in large-scale systems due to idle capacity have been addressed in the past. Computational grid frameworks share load across different organizations via “federation” – the controlled pooling of resources across multiple clusters owned by different entities – to maximize user productivity. This sharing of load may use explicitly scheduled capacity, as implemented by Globus [3], or opportunistically available capacity via “cycle harvesting”, as implemented by Condor [4]. In a federation setting, resources ultimately remain in control of the owner and are only made available temporarily to users of another organization. To enforce local control, these systems require the ability to “preempt” (terminate) federated workload to free up space for locally generated workload if the system reaches capacity. The integration of federation and cycle harvesting capabilities has lead to a substantial increase in the efficiency of computational grids.

IaaS clouds can benefit from adopting federation and cycle harvesting. The pooling of resources across multiple clouds via federation reduces the need for overprovisioning by providing more opportunities for workload consolidation. The preemption mechanism implemented in cycle harvesting schedulers guarantees local control over resources even when these resources are in active use by federated workload. Together, these techniques allow IaaS clouds to participate in a federation without compromising their autonomy, while still reducing the required degree of overprovisioning. While useful for IaaS clouds in general, we expect the benefits to be especially pronounced for private clouds deployed for smaller user bases with a limited ability to consolidate workload internally.

There is, however, a fundamental difference in the usage patterns and guarantees expected by users of grids and cloud. Grid users submit batch jobs to a queuing system and expect the grid to make a best effort attempt at completing their jobs over time under changing conditions. Cloud users in contrast expect interactive feedback about the availability of requested resources, predictable progress, and consistent quality of service as set forward in the cloud's SLA.

Grid workloads consist of queued batch jobs whose execution is constrained by pre-defined resource quotas and job priorities. Grids optimize the utilization of available resources by prioritizing, re-ordering or re-routing jobs within a federation. IaaS clouds instead provide resources "on demand" with immediate feedback to the user about either the successful provisioning of resources or the rejection the request. Cloud users further expect provisioned resources to remain available until freed explicitly, which simplifies management of and reasoning about cloud applications. In a federation setting across independent organizations, however, the clouds' ahead-of-time availability guarantees are at odds with an organization's potential need to preempt running jobs to enforce quotas and priorities.

There is substantial interest in overcoming these challenges and enable "cloud federation" in academic and commercial contexts [5, 6]. Most recently, the National Science Foundation (NSF) funded the "Aristotle Cloud Federation" to create a next-generation scalable data infrastructure. In the spirit of computational grids, Aristotle allows each participating campus to buy and maintain its own resources as needed, but also share excess capacity when available. In an industry context, "hybrid" clouds [7, 8] that merge public and private cloud resources are a common use-case, albeit with limitations regarding preemption and ahead-of-time guarantees.

Automating workload federation between different clouds can reduce substantially the need for overprovisioning while still maintaining organization boundaries for provisioning and budgeting. In private clouds these improvements translate to productivity gains, while in public clouds they can simplify opportunistic capacity use and enable safe workload consolidation across multiple providers.

1.1 Thesis Statement

Can the utilization of private IaaS clouds be increased by using ahead-of-time guarantees on the availability of preemptible resources to implement workload federation?

1.2 Challenges

Building an end-to-end cloud federation system is difficult. Existing literature proposes numerous different approaches to cloud federation, but the proposed designs are either partial solutions or not implemented and evaluated in a real system. A major barrier to end-to-end evaluation of cloud federation architectures is the extensive engineering effort required to implement prototypes on top of production-quality IaaS frameworks.

Experimentation with cloud infrastructure at scale is difficult. The modification of open-source IaaS cloud frameworks is complex as their components interact with many different parts of the underlying hardware and operating systems. Thus, the implementation of research prototypes for cloud federation architectures is slow, error prone, and potentially unrepresentative of production system behavior. Even when implemented successfully, a cloud federation architecture must then be tested at the scale of real-world clouds and, potentially, at even larger scales for future use-cases. This makes real world experimentation with numerous competing designs infeasible. Simulation can alleviate these issues, but predictions must be validated against real world measurements to be trustworthy.

Cloud SLAs demand quality of service guarantees for each job request to be made *ahead of time*. The reliability of a preemptible tier of instances depends on the dynamic state of the cloud and future incoming workload. If the future utilization of the system can be predicted sufficient accuracy, guarantees about the availability of preemptible resources can be made. Batch workloads are promising candidates for prediction as they have a known input size and bounded runtime and thus do not require open-ended guarantees about resource availability. The use of a bounded look-ahead window enables us to predict resource availability and provide ahead-of-time guarantees *despite* preserving a cloud's preemption capabilities.

1.3 Methodology

To address these challenges and increase the efficiency of IaaS clouds via reduced overprovisioning, we design and implement an end-to-end architecture for workload federation across independent IaaS clouds that enables opportunistic cycle harvesting while still providing availability SLAs with ahead-of-time guarantees. We first evaluate the efficacy of our solution in simulation and then implement and evaluate a real-world prototype of our IaaS cloud federation architecture on top of the open-source Eucalyptus framework.

We enable accurate prediction of the behavior of IaaS clouds via validated simulation. Existing work on cloud simulation (c.f. Section 3.2) offers numerous approaches, albeit with limited or absent validation of results against measurements from real systems. We develop a method for creating simulation models that are validated at observable scale and then carefully extrapolate their results to larger scales. Our approach is inspired by perturbation theory and enables experimentation with federation schemes without building numerous real world prototypes at scale.

We develop a method to provide ahead-of-time guarantees on the availability of preemptible instances in IaaS clouds. We repurpose our validated simulation method to make dynamic predictions about the future state of the cloud with a Monte-Carlo style approach. These predictions are based on the historic behavior of the cloud and its current internal state. We then use these predictions to implement admission control to enable batch workloads with bounded job duration to execute opportunistically on preemptible capacity with ahead-of-time guarantees on their preemption probability. This enables federation of batch workloads between independent IaaS clouds in a way that preserves clouds' control over their own resources via preemption, but still offers availability SLOs to federation users.

We implement our proposed architecture in a prototype using the Eucalyptus IaaS framework and evaluate its efficacy with real-world workload traces. These traces are replayed on live clouds in faster-than-realtime and at smaller-than-realworld scale. This evaluation ensures that unmodeled aspects of the system do not invalidate our earlier simulation results. The implementation is complete end-to-end and addresses practical issues of cloud federation such as setup overheads and data transfer between source and destination clouds. We use production traces of computationally-intensive and big data analytics workloads to form realistic expectations about the efficiency gains achievable through cloud federation.

1.4 Contributions

In summary, in this dissertation, we investigate a new approach to IaaS cloud federation and develop novel methodology to support its real world implementation. We take inspiration from existing work in grid federation, cycle harvesting and simulation. In particular, we contribute:

- an architecture for job federation between independent IaaS clouds. The proposed architecture enables independent IaaS cloud to share workload opportunistically while maintaining local control over resources.

- a method for creating validated simulation models of IaaS clouds. Resulting models can be validated against measurements from clouds at observable scale and then be extrapolated to larger scale, thus allowing trustworthy experimentation with cloud federation at unobservable scale.

- a method for providing ahead-of-time availability guarantees for preemptible resources in IaaS clouds. These statistical guarantees about resource availability are made at request time and remain valid for a user-specified time window.

- an end-to-end implementation of cloud federation on the open-source Eucalyptus IaaS framework. The prototype implements our cloud federation architecture, addresses practical issues of job federation, and does not modify the internal structure of Eucalyptus. We perform an evaluation of the correctness of the federation mechanism and the availability guarantees made via replay of real-world traces recorded from production clouds. We further investigate the impact of other practical concerns, such as inaccurate bounds on job duration and seasonality in the workload.

1.5 Permissions and Attributions

1. Chapter 3 is a partial reprint of "Using Trustworthy Simulation to Engineer Cloud Schedulers" by A. Pucher, E. Gul, C. Krintz, and R. Wolski, and has previously appeared in IC2E 2015.

2. Chapter 4 is a partial reprint of "Providing Lifetime Service-Level-Agreements for Cloud Spot Instances" by A. Pucher, C. Krintz, and R. Wolski, and has previously appeared in GCA 2015.

Chapter 2

Background

"If we fail to understand and apply previous research, we will at best rediscover well-charted shores. At worst, we will wreck ourselves on well-charted rocks."
– Thain et al., Distributed computing in practice: the Condor experience, 2005.

Cloud computing realizes the long-held vision of computation as a utility [9]. In this chapter, we discuss the background of different types of clouds and list major public providers and open-source frameworks for private IaaS clouds. We further survey existing literature for improving utilization in computational grids and large clusters via federation and opportunistic computing. We conclude our disposition by highlighting recent developments in federation techniques for IaaS clouds specifically.

The "cloud" abstracts away the details of the underlying infrastructure and instead describes its capabilities in the form of services. The individual services and their promised "quality of service" (QoS) is defined and published in the cloud's Service-Level-Agreement (SLA). The service quality is measured quantitatively using different high-level metrics. Their corresponding targets and measurement time widows are defined as Service-Level-Objectives (SLOs). These explicit SLOs, such as object storage availability and durability measured over the period of a month, simplifies users' reasoning about the system reliability and performance ahead of time. While the SLA defines "guarantees" it also provides means to compensate cloud users for the provider's failure to meet the SLOs. There are three primary types of clouds that are distinguished based on their level of abstraction:

Infrastructure-as-a-Service (IaaS) clouds offer full-privilege access to virtual machine "instances" and scalable storage from a web-accessible "object store". Instances come with attached storage and a network device and can be launched and terminated on demand, either manually by the user or programmatically for load-proportional scaling of resources. IaaS clouds typically provide additional Application Programming Interfaces (APIs) for managing cluster infrastructure such as virtual networks, SAN storage, messaging, and load balancing.

Platform-as-a-Service (PaaS) clouds abstract away the operating system and software stack and enable rapid development of applications against a streamlined and scalable API. The focus of PaaS typically lies on web-applications and assumes a three-tier architecture with load balancer, application servers and backend database. The cloud user implements the business logic of the application and uses high-level APIs to access data storage, task scheduling, and other functionality. The PaaS cloud dynamically performs load balancing and scaling of the associated underlying infrastructure.

Software-as-a-Service (SaaS) refers to software solutions tailored to end-users delivered with an always-online model. SaaS applications fulfill a domain-specific use case such as email, calendar, and contact management. The infrastructure automatically performs maintenance tasks such as dynamic scaling, backup and patching. The user accesses the application as a remote service and thus requires a constantly available Internet connection. Depending on the specific application, limited offline capabilities may be provided.

Major public IaaS providers include Amazon AWS [10], Google Cloud Platform [11], Microsoft Azure [12], Rackspace [13], and IBM SoftLayer [14]. All commercial IaaS providers offer access to resources on a pay-as-you go basis, and may offer additional cost structures for opportunistic or long-term customers. They typically rely on dedicated data centers to house their hardware and use proprietary software stacks to offer access to compute instances, object store, and related services. Commercial providers keep the details of their implementation as trade secrets and limit benchmarking of their services at scale, thus making academic research on public clouds difficult.

Amazon AWS offers services across the entire spectrum from IaaS, over PaaS, to SaaS. Their IaaS services include the EC2 platform for cloud instances and the S3 object store, both of which integrate with a number of additional services for networking, data analytics, and others. The EC2 and S3 APIs have become the quasi-standard for open-source IaaS cloud frameworks and client libraries, such as euca2ools [15] and boto [16] for Python. Google Cloud Platform offers IaaS and PaaS services: cloud instances via Compute Engine, an object store via Cloud Storage, and a number of specialized database and analytics capabilities. Google App Engine is a prominent PaaS platform for hosting scalable web applications. Its API has been adopted by open-source PaaS frameworks such as AppScale [17]. Microsoft Azure is a cloud offering that provides a variety of IaaS, PaaS, and SaaS services and specializes in tight integration with other Microsoft products. Rackspace Public Cloud is an IaaS platform with several additional services for networking, data analytics, and "bare metal" hosting of applications. IBM SoftLayer equally provides IaaS compute and storage services, as well as networking, data analytics and bare metal capabilities.

Private IaaS clouds build on open-source frameworks such as OpenStack [18], Eucalyptus [19], CloudStack [20], OpenNebula [21], and Nimbus [22]. They implement major services found in public IaaS clouds, such as compute, storage and load balancing but allow deployment on clusters on-premise. They are further (partially) API compatible with commercial providers, such as Amazon AWS, and enable the creation of "hybrid" clouds that combine local infrastructure and resources rented from public providers. In contrast to public clouds, their source code is openly accessible and can be benchmarked and customized. This property makes them highly flexible for deployment at different scales and offers an opportunity to perform domain-specific optimization. As a consequence, academic research in the cloud context typically relies on these open-source frameworks.

2.1 Federation in Large-Scale Computing

Computational grid frameworks enable load sharing across different organizations via federation. Globus [23] is a grid computing framework that integrates compute clusters and, to a degree, pools of workstations with a standardized interface. It supports pooling of resources across organizations via "federation". We discuss Globus in Subsection 2.1.1.

Cycle harvesting enables opportunistic computing with temporary spare capacity. Condor [24] is a scheduler for batch computating on pools of workstations and implements a run-while-idle model. Idle workstations may be used opportunistically to run computation in the background. Active tasks are preempted (terminated) when the workstation's primary user returns or the workstation goes offline. Condor supports transfer and restoration of state via checkpointing and implements a federation mechanism – "flocking" – that allows combining pools of workstations across multiple organizations. We discuss Condor in Subsection 2.1.2.

Big data analytics framworks, such as Apache Hadoop, can integrate heterogeneous hardward and perform parallel computation and batch processing at large scale. Apache Hadoop and its distributed filesystem HDFS specifically have been adapted to perform federated storage and analysis of very large data sets and achieve high performance by imposing a processing pipeline structure (e.g. map-reduce) that is amenable to parallelization. Hadoop further optimizes performance by leveraging data locality (i.e. moving code to the data) and speculative execution of straggling tasks. We discuss Apache Hadoop further in Subsection 2.1.3.

2.1.1 Globus

Globus [25] has become the de-facto standard for managing and connecting computational grids. A major contribution of Globus is the standardization of communication protocols between diverse systems – most prominently via the Open Grid Services Infrastructure (OGSI) and more recently the Web Service Resource Framework (WSRF) [26]. OGSI provides abstractions for creating and managing stateful grid services across diverse compute environments. OSGI is a specification language based on the XML-based Web Service Definition Language (WSDL). WSRF is an evolution of OGSI that provides an improved separation of concerns in the specification and takes advantage of features added to WSDL over time.

The Globus framework was originally developed as middleware to enable the management and development of "metacomputers" – virtual supercomputers that integrate resources from diverse, geographically distributed computer systems into a single environment for parallel applications with stringent scheduling requirements [3]. Globus implements the paradigm of "separation of policy and mechanism" by providing low-level abstractions and allowing higher level policy to control their usage. Primary functionality includes resource allocation and monitoring, authentication, process management, and communication and storage primitives.

Grids enable the "controlled and coordinated sharing" of resources across collaborating organizations [27]. Globus decomposes the Grid architecture into multiple layers: fabric, connectivity, resource, collective and application. The framework builds higher level abstractions on top of the “fabric” layer, which implements low-level process management, communication and storage functionality within a single physical system. Globus then enables cross-system (and organization) authentication, discovery and coordination of individual resources. Applications developed on top of Globus may take advantage of the existing components in the form of services with defined APIs, modify them, or add entirely new domain-specific services.

Besides interoperability between diverse environments and computer systems, security is a primary concern. Globus integrates with various services within organizations, such as single sign-on and key management, and puts emphasis on encrypting data in-flight. Further, resource usage by different users is monitored and metered for auditing and accounting purposes.

The effective use of distributed grid resources for applications with strict scheduling requirements needs accurate information about the systems in both, static and dynamic terms. Globus maintains a directory of resources and monitors their basic availability via regular heartbeats. With increasing adoption of Globus, additional services were integrated that provide sinformation and forecasts about dynamic system properties, such as network latency and bandwidth [28].

2.1.2 Condor

Condor [4] is a high-throughput batch system for compute-intensive jobs executing on pools of non-dedicated workstations. It implements the “cycle harvesting” paradigm – the opportunistic execution of jobs on idle capacity – with a best-effort job completion “guarantee”. Condor implements a series of capabilities to take advantage of intermittent spare capacity effectively, such as checkpointing, job migration, remote system calls, and resource classification.

Open Science Grid (OSG) [29] is a consortium of independent science communities that opportunistically share spare compute capacity across distributed compute facilities via Condor. Workload federation across these sites is used extensively and has successfully supported projects of vast scale from weather prediction to protein analysis. OSG has also been a major driver in the maturing of Condors open-source software stack.

Condor jobs consist of a number of tasks that execute within a specialized runtime environment. Each job comes with an associated type that determines the treatment of the job and its tasks by the Condor scheduler. "Master-worker" batch jobs execute a flexible number of tasks without particular ordering and enable the concurrent execution of tasks on multiple workstations. This job type is malleable and allows the scheduler to selectively preempt and restart tasks on different workstations. Condor jobs that require co-scheduling of tasks, such as MPI applications, are referred to as "parallel" jobs. Parallel jobs depart from the cycle harvesting paradigm and require dedicated nodes. That is, the scheduler does not allow the preemption or migration of tasks even if the workstation is no longer found to be idle. Finally, "DAGMan" jobs execute a user-defined dependency graph of related jobs and automatically propagate results and errors between jobs.

The current system [30] supports multiple runtime environments, referred to as "universes". The universe determines the API available to the programmer. Common universes are the "standard","vanilla" and "VM" universes. The standard universe provides the ability to checkpoint and migrate the job between machines and allows the invocation of remote system calls to the machine used to submit the job in the first place. While this offers convenience to the developer it creates overhead and external dependencies. As an alternative, the vanilla universe avoids this overhead, but places the burden for transfer of state between tasks and the aggregation of results on the developer (it is the default for jobs in the OSG). Finally, the VM universe is implemented on top of virtual machines with arbitrary disk images and implements checkpointing via disk snapshots.

The Condor scheduler implements a priority scheme for resource allocation across jobs and users. The job priority affects the execution order of jobs in each user's individual queue of pending jobs. Resources are allocated between jobs of different users proportional to the user priority. The user's relative priorities are adjusted over time to account for actual resource usage and achieve fair resource sharing. That is, Condor does not starve jobs of low priority users, but allocates a smaller share of resources proportional to the ratio of individual user priorities. While Condor attempts to maximize the throughput of jobs, it does not make explicit guarantees about when or whether a submitted request will execute.

Condor's success in harvesting idle resources at the University of Wisconsin led to a proliferation of Condor pools at different sites. Apart from solving technological challenges this success is attributed to Condor's flexibility and its motto "leave the owner in control, regardless of the cost." [31] The developers retained this motto when adding cross-organization federation capabilities to Condor via "flocking".

The first generation of workload federation in Condor – "gateway flocking" [24] – allows jobs to be submitted across organization boundaries and is fully transparent to the user. Gateway nodes in each pool forward cluster status information and requests, such that resources can be acquired across multiple pools. The gateway nodes in each cluster are responsible for only admitting external requests if their requirements align with local policy. Thus, gateway nodes protect the independence of each organization.

The second generation of federation – "direct flocking" [32] – removes the transparency provided by gateways in favor of fine-grained access control and usage accounting on a per-user basis. Instead of negotiating federation on an organizational level, the individual user obtains access to multiple Condor pools and negotiates for resource directly. While direct flocking increases the burden on the user, it adds flexibility over gateway flocking and simplifies the overall system architecture by removing the dependency on gateway nodes.

As an adaptation from Globus, the Condor ecosystem spawned “Condor-G” [33] to integrate with other batch systems and implement standardized communication protocols. These protocols enable wide-area job management, authentication, and secure data transfer. Condor-G introduces the “gliding-in” of jobs as a way to run Condor jobs across disparate batch systems by creating “personal” Condor pools for a user from resources located in different systems.

2.1.3 Hadoop

Apache Hadoop [34] is an open-source batch system modeled after Google MapReduce [35] for the distributed processing and ad-hoc analysis of large quantities of unstructured data. Hadoop is the quasi-standard for big data analytics and has spawned a large ecosystem of related open-source projects, such as Apache YARN [36], Apache Spark [37], and Apache Hive [38]. The Hadoop Distributed File System (HDFS) [39] serves as the foundation for fault-tolerant data storage and high throughput file access. HDFS is organized in a hierarchical Master-Slave architecture with centralized name nodes keeping track of meta data and replica location and multiple data nodes redundantly storing file contents in fixed-size chunks.

Hadoop uses a batch processing architecture and implements the Map-Reduce paradigm. A Hadoop job is submitted to a queue and then processed in three phases: “map”, “shuffle”, and “reduce”. The work in each phase is split into multiple “tasks” that can be processed in parallel by worker nodes. The map phase reads input data in fixed-size chunks from HDFS and applies a user-defined transformation that emits a number of key-value tuples. The shuffle step then transfers related tuples (intermediate data) to pre-defined target nodes, the reducers. The reduce phase then applies a user-defined aggregation function to the tuples at each reducer separately and writes the results back to HDFS. The Hadoop scheduler keeps track of task progress and transparently masks faults by restarting failed or slow tasks, thus enabling rapid development of highly parallel applications.

Research on Hadoop federation proceeds in three primary areas: the placement of and access to HDFS data and replicas across different sites, the optimal scheduling and placement of tasks, and Hadoop integration with other compute frameworks.

G-Hadoop [40] is an extension to Hadoop for federated execution across multiple high performance computing (HPC) clusters belonging to different organizations. G-Hadoop moves map tasks to the cluster holding the (potentially large) input data and minimizes redundant data transfers by leveraging the cluster's SAN for intermediate data storage rather than node-local storage and HDFS. HDFS on the Grid [41] integrates Hadoop on top of Globus grids and addresses the challenge of data storage in the face of resource failure and preemption. Multicluster HDFS [42] explores the behavior of HDFS spanning multiple clusters with different configurations.

Cardosa et al. [43] investigate the workload-specific performance of different data placement schemes and job scheduling strategies for Hadoop in geographically distributed Hadoop clusters. They distinguish between Local MapReduce (LMR), Global MapReduce (GMR) and Distributed MapReduce (DMR). LMR creates a local replica of the input data before performing any computation and is preferred for jobs that do not aggregate (or even expand) the input data. DMR performs a two stage computation by first creating a local aggregate per cluster and then merging the intermediate results from all clusters in a second step. Finally, GMR executes tasks across all participating clusters, taking into account data locality on a best-effort basis. Fed-MR [44] expands on the DMR model, with a top-level Hadoop cluster federating jobs to region-specific clusters and then collecting and aggregating results. Fed-MR further considers practical issues of federation such as data placement constraints and the generation and automated merging of intermediate results. G-MR [45] efficiently processes sequences of Hadoop jobs across geo-distributed clusters by constructing a graph of data and transformation dependencies. Analysis of this dependency graphs allows for an effective choice of partial aggregation and replication strategies for inputs, intermediate data and outputs.

MapReduce On Opportunistic eNvironments (MOON) [46] modifies Hadoop to perform well on preemptible resources such as those found in Condor workstation pools. MOON uses a combination of stable and volatile nodes to store and process data by maintaining a core replica of critical data on dedicated, always available nodes. Data replication on preemptible resources is controlled automatically based on a user-defined availability SLO and MOON improves HDFS to handle the temporary unavailability of resources. The system further uses an aggressive task replication strategy to improve job throughput under preemption pressure.

Distributed Hadoop MapReduce on the Grid (HOG) [47] fully integrates Hadoop into the Open Science Grid (OSG) with federated and preemptible Condor resources. HOG tackles the challenges of high task failure rates due to preemption and data loss in a highly distributed and unstable setting. It provisions Hadoop worker nodes and HDFS data nodes through Condor's "gliding-in" and uses aggressive monitoring and replication strategies to handle correlated preemption or site-wide failures, albeit for practical reasons it still maintains some persistent state on a non-preemptible master node.

2.2 Federation in IaaS Clouds

CloudLab [5] is a platform for cloud research across multiple federated facilities at the University of Utah, Clemson University, the University of Wisconsin Madison, the

University of Massachusetts Amherst, and several industry partners. Similar to the OSG it provides a unified interface to access federated compute resources for research, with an emphasis on bare metal access for experimentation with open-source cloud frameworks at scale. The different facilities participating in CloudLab focus on different aspects of data center infrastructure, such as high storage density or high performance networking.

Another federated testbed for cloud research is OpenCirrus [6], with multiple locations at universities and industry research labs. The various sites have a different focus according to their operators' interests, such as networking, high-performance computing, and large scale data analytics. Similar to other federation efforts, the authors emphasize its unified user management, high tolerance to load spikes and ability to run workloads and experiments at scale.

Private IaaS frameworks have limited load sharing abilities via "hybrid cloud" capabilities. Most open-source IaaS frameworks implement mechanisms to spawn resources in public clouds, such as Amazon AWS, to increase the capacity of private clouds. Eucalyptus supports hybrid cloud capabilities through its client-side "euca2ools" utility [15], which provides a single API for accessing Eucalyptus deployments as well as Amazon AWS [7]. OpenStack supports federated identity management and allows one cloud's users to authenticate to another cooperating cloud without managing separate credentials [8].

As an alternative, an additional PaaS abstraction layer can be used to implement workload federation across multiple IaaS clouds, albeit with the restriction on applications developed on top of the specific PaaS framework. RightScale [48] offers a commercial PaaS framework for deploying applications across cloud boundaries. The open-source AppScale framework [49] has been used to deploy HPC applications across multiple AWS and Eucalyptus clouds.

Public IaaS providers implement opportunistic computing with a preemptible tier of instances. Amazon AWS originally introduced “spot instances”, a preemptible tier of instances whose price is determined by “spot market” where users bid on available spare capacity. Google Compute Cloud later also introduced a class of lower cost instances which may be preempted at any any time. Related work is explored in detail in Section 4.2.

To enable federation, a number of changes to the architecture of IaaS clouds have been proposed in the literature. Previous work makes a distinction between “inter-cloud” federation – the transfer of self-contained jobs from one cloud to another – and “cross-cloud” federation – the mashup of services such as storage and compute from multiple providers. We elucidate further in Section 5.2.

Chapter 3

Validated Simulation For Engineering Cloud Schedulers

3.1 Introduction

In this chapter, we develop a method for validated simulation of IaaS clouds that addresses some of the challenges inherent in experimentation with large scale systems. If we are to evaluate an architecture for cloud federation, it must be tested at scale. Replicating existing large scale cloud infrastructure, such as Amazon AWS, for research purposes is infeasible. There are budgetary constraints to installing infrastructure for research purposes, but also competitive considerations – commercial cloud providers hold the details of their implementation as trade secrets.

Simulation can overcome the scaling issue, but a simulation approach has limitations of its own. When driving a complex engineering effort, such as the development of a cloud federation architecture, we are concerned about the accuracy of the simulation model. Existing literature about cloud simulation provides methods for exploratory research at scale, but their results are typically not validated against measurements from real systems.

For cloud computing, simulation systems to date focus on *ab initio* techniques in which various low-level cloud components (machines, networks, storage devices, etc.) are simulated and these component simulations are then composed into a full system simulation. This "bottom up" approach is both flexible and easily extensible, and yields insights that stem from comparative ranking (*e.g.* "this" configuration is better than "that" one). The value of this approach cannot be underestimated, however, the scale and reliability requirements for clouds present challenges for *ab initio* methods with respect to accuracy that must be addressed before they can be considered "trustworthy" from an engineering perspective. This approach tends to produce results at scales that are difficult to test empirically. Finally, the composite model may become so complex that the error interactions between component models become untamable, even if components are validated individually against empirical measurement.

We propose a novel method for creating models of IaaS clouds with quantifiable accuracy for *validated simulation*. Our work explores an approach rooted in *perturbation theory* [50] that focuses on validation of simulated results against empirical measurement – at the cost of flexibility and extensibility – as a way of addressing the engineering needs for specific cloud systems. Specifically, we build a parsimonious "top down" model of the end-to-end system that derives from the implementation specifics of the system. We then add "noise" (taken from statistically sampled empirical measurements of the system) to "perturb" this model. For validation, we analyze the perturbed model's outputs statistically over multiple runs – as a Monte-Carlo [51] style simulation – and compare them to the distributions of end-to-end measurements taken from repeated runs of a real system.

We apply validated simulation to the development of cloud schedulers that support workload federation. For cloud schedulers in particular, this approach proves fruitful because the models are quite parsimonious (reducing the possibility of error propagation) and the system measurements are easily gathered at scales that are feasible for repeated measurement. Once validated, the model can be scaled up in any dimension characterized by independent performance response. For example, if the performance of the physical machines hosting user-allocated virtual machines is independent (due to the isolation properties of the cloud platform) then the physical machine count can be scaled without introducing additional error.

We emphasize that our work is intended to complement *ab initio* approaches in that it targets the development of a specific component (a cloud scheduler for example), that the component must be amenable to a perturbation-based approach to modeling, and that scaling is trustworthy only in the dimensions of independence. Further, our approach is intended to produce accuracy only in the parameters that are necessary for a particular component's operation *as an isolated feature.* That is, the method is appropriate for clouds because component operation is already isolated through internal modularity techniques for realiability reasons. Our method relies on this isolation property and access to the source code so that the relevant model parameters can be identified.

Even with these restrictions it is possible to use our simulation technique to explore performance and scaling properties in a manner similar to previous approaches. A key additional benefit of our method is that the results are validated at scales that can be tested and that there is evidence that their accuracy is preserved at different scales and configurations. Another practical benefit of using a top-down approach to simulation is the parsimonity of the model, which leads to fast execution. This maximizes the benefits of faster-than-realtime simulation and allows many different designs and scenarios to be tested within short time frames.

Before we adopt validated simulation to test federation architectures, we must first evaluate the feasibility of applying validated simulation in the cloud engineering context. For this purpose, we perform a case study that adds an invasive feature to an open-source IaaS framework. We implement a power-aware scheduler from the literature [52], whose properties are already well-known. We first predict its behavior for a set of workloads and cloud configurations with validated simulation. We then compare these predictions with measurements from a functionally equivalent implementation in a real system. Thus, this case study serves as evaluation of the efficacy of our perturbation-inspired approach to building validated simulation models. We also describe and detail the use of the validated simulation model to related problems, such as enterprise IT capacity-planning with production cloud traces, as a way of demonstrating its general utility besides supporting the development of our cloud federation architecture.

In summary, this chapter makes the following contributions:

- We outline a simulation approach that is designed to support production-quality engineering of cloud platforms by applying a new approach – perturbation-based modeling – to cloud simulation.
- We demonstrate the use of this methodology in the implementation of a new power optimizing scheduler for a production-quality private cloud platform.
- We evaluate the simulation's accuracy for both, reproducing observable cloud behavior and making predictions about the behavior of a scaled-up (and yet unobserved) variant of a cloud.
- We evaluate validated simulation's general utility for capacity planning with synthetic and production traces in private IaaS clouds.

We first describe the steps of our methodology for top-down model development, discrete event simulation, model fitting, and validation. As an example of this process, we show how we use it to implement a new power-optimizing scheduler. The empirical evaluation of our approach includes the registration of our simulator against a production-quality Eucalyptus private cloud, an evaluation of the scheduler in simulation and real-world implementation, and an investigation into capacity planning leveraging the simulator.

3.2 Related Work

Empirical evaluation of distributed systems technologies has a long tradition in Computer Science. Recently, Gustedt et al. [53] has classified methodologies and recommends best practices for performing experimental validation of large scale systems using real-scale experiments, emulation, benchmarking, and simulation. The authors discuss the importance of ab initio (high-level, imprecise, easily composable, and extensible simulation for use in comparative analysis and exploration) and validated simulation (simulation that produces behavior that matches that of a real system with low error).

Research in data center power-efficiency [54, 55, 56] predates the call for power-proportional computing [57], but has gained substantial traction and public interest since [58]. Recent work in datacenter power efficiency is surveyed by [59] and illustrates the significant potential of energy- and cost-savings via pro-active power-management. The power-aware scheduler we implement in this paper, was originally proposed in [52]. It uses the QBETS [60] predictor to estimate the number of hot spares needed to maintain a configurable responsiveness SLA.

Grid research has spawned multiple simulators. This includes SimGrid [61] and GridSim [62]. The former provides validation for some simulation components, the latter is an ab initio approach. Such systems are challenging to use for cloud systems since they lack support for on-demand resource allocation, elasticity, and other cloud features.

To facilitate cloud research on a broad scale, the community developed a series of domain-specific simulators. In particular, CloudSim [63] and NetworkCloudSim [64] allow simulation of large-scale clouds using an ab initio approach. CloudAnalyst [65] extends CloudSim to facilitate simulation of globally distributed applications such as social networks. These simulators model system components and workloads from the bottom up and compose them into large-scale configurations. Their approach is very flexible and extensible, but does not provide the accuracy guarantees necessary for evaluating production-quality cloud components. Alternatives allow for real-scale (in-situ) experimentation [6, 66], but their use is limited for practical reasons (e.g. time, available cluster size, budgetary constraints).

GreenCloud [67] is a simulator that focuses on exploring the energy consumption of different datacenter network architectures. It builds upon the NS2 [68] network simulator, and estimates the efficiency of hibernation and power-stepping strategies for servers and network components. Workloads and hardware are modeled with differing compute and communication capacities and, similar to our approach, the authors consider SLA requirements. GreenCloud inherits network level accuracy from NS2, but does not consider accuracy of per-node resource allocation or empirical validation of predictions.

EMUSIM [69] uses emulation of Bag-of-Task applications to extract performance properties and simulate their behavior at larger scale more accurately. An evaluation step ensures that emulation and simulation agree at observable scales. We similarly obtain empirical measurements at small scale and scale them up in simulation. In contrast, our approach focuses on cloud infrastructure components (not individual applications) and makes predictions about the utilization and resource-use of the cloud as a whole.

RC2Sim [70] is an integrated simulation and emulation environment for testing of production-quality cloud management code. It provides a compatible web API and emulates distributed operations, such as file transfers and remote shell access, on a single physical machine. This prior work focuses on functional testing of code rather than, as we do in this paper, on accurate simulation of resource-usage and execution time.

DCSim [71] simulates IaaS clouds with a specific focus on dynamic power- and SLA-optimization. The authors use tiered scale-out-type workloads and evaluate the advantage of VM migration and replication strategies over static provisioning. Similar to our work, they consider node power states and transitions, but do not perform an empirical evaluation of the simulation results.

GDCSim [72] addresses the thermal aspects of power-management in data centers by integrating existing models. Specifically, It investigates the interaction of workloads and resource management policies with heat dissipation and fluid dynamics of different physical data center layouts. Empirical validation of predictions is left for future work.

DCSim [73] uses detailed models and hardware specifications to simulate the impact of networking infrastructure on web applications. The authors augment their simulation model with workload characteristics obtained from real-world measurements and make accurate, empirically validated, predictions about latencies for a set of benchmarks. The work is similar to ours in terms of allowing trustworthy capacity planning, but targets 3-tier web applications instead of generic IaaS cloud infrastructures.

iCanCloud [74] uses hardware models to offer a limited POSIX-inspired API to emulated the execution of distributed applications on a simulated cloud platform. The authors also emphasize it's graphical user-interface as distinction to other simulators. As an "ab-initio" simulator, its performance is evaluated for a specific application use-case against an analytical model, measurements from Amazon EC2, and an CloudSim implementation. The work is different from our approach as it focuses on evaluating the performance of a specific application at scale and generates predictions based on detailed hardware models that are composed into a full cloud simulation.

PICS [75] is a cloud simulator that focuses on producing performance and cost predictions for batch workloads executing on public clouds. PICS implements a discrete event simulation to replay workload traces from synthetic or realworld sources with a focus on scheduling VMs, storage, and network. The authors perform a rigorous evaluation of simulation results against measurements taken from Amazon EC2 for a mix of workloads in terms of VM count, VM utilization, and realized cost over time. They further perform a sensitivity analysis regarding the calibration of performance parameters to a specific cloud as performed by the user. This work is similar to ours in its focus on job scheduling and aggregate cloud performance metrics, as well as its extensive evaluation of predictions against measurements from cloud systems executing the same workload. Differences exist in simulation registration and the granularity of the scheduling simulation. Our approach offers a structured process to calibrate and validate the simulation model with an existing cloud, whereas PICS relies on the user to configure relevant parameters. PICS investigates the behavior of scale-out workloads by simulating task-level granularity scheduling, while our simulation assumes fixed-size (non-malleable) jobs.

We take a “top-down” approach to cloud simulation, inspired by perturbation theory, rather than the “ab initio” approach explored by extant literature. Starting from a whitebox inspection of the cloud under investigation we derive a parsimonious model that we then “perturb” with “noise” (measurements taken from registration runs on the real system) until the desired level of accuracy is reached. As this noise is probabilistic, we perform Monte-Carlo style simulation to produce results and associated confidence bounds. Most importantly, the simulation’s accuracy can be validated end-to-end with measurements taken from benchmarks (or historic behavior) of the specific, modeled system. Thus, we achieve validated accuracy at the cost of model flexibility.

3.3 Methodology

We outline a process for conducting cloud scheduler research for private clouds in terms of a specific example in which we seek an implementation of a new power-optimizing scheduler for a private cloud. The scheduler uses on-line machine learning methodologies to predict (in real time) when machines should be powered on and off to avoid delays associated with machine spin up. Before an expensive engineering effort can be launched to implement such a scheduler or a skeptical IT professional can be convinced to introduce a new methodology, the reliability, performance, and efficacy properties of a new scheduler must be verified. Our goal with this process is to facilitate accurate, faster-than-realtime, end-to-end testing via validated simulation.

The goal of the methodology is to use a “top down” approach to simulation that models only those parameters that are necessary to capture the behavior of the component of interest with sufficient accuracy. Identifying the parameters of this model requires an understanding of the fault isolation properties of the platform which, in our example use case, comes from source code inspection. The fault isolation properties establish the independence of our model parameters which is required for trustworthy scaling of our simulations.

The approach is to:

1. start with the most parsimonious model of end-to-end behavior that is possible,
2. perturb the model using statistical sampling technique to represent unmodeled behavior,
3. test the model by comparing its outputs generated in simulation to measurements taken from the “real world” system,
4. if the model is insufficiently accurate, add terms, adjust the perturbation, and repeat.

Thus every addition of a variable to our model of the cloud should be justified by a necessary increase in accuracy. Variables that only contribute marginally to the aggregate result are omitted and modeled in aggregate as “perturbing” error terms. The level of accuracy that is acceptable is ultimately decided by the consumers of the simulation. In an engineering context the error terms may serve as inputs to a risk analysis, where variability is acceptably low when the difference in risk that greater accuracy would engender is deemed insignificant by those taking the risk.

3.3.1 Model Construction

The first step in our approach is a white-box inspection of the documentation and source code. With information about the control and data flow in hand, we are able to identify critical inputs, cloud components and their interactions, and relevant output metrics.

In this case study, we are interested in evaluating a new cloud scheduler, which requires user requests, the physical platform configuration, and the allocation algorithm as inputs. The cloud model consists only of a set of independent nodes with fixed resource capacities that hold a number of instances with fixed requirements. Interactions between this model and the scheduler implementation take place if and only if a request arrives or the life time of an instance expires.

The outputs of the cloud scheduler that we can observe are (a) request acceptance rate and (b) the allocation of “virtual machines” (VMs) to nodes over time. We quantify this behavior by computing the aggregate CPU time for each physical node devoted to work assigned to it by the scheduler. Comparing node CPU time, both in simulation and actual measurements, succinctly captures the end-to-end behavior of the system under test for the cloud scheduler component.

Note that a new scheduler may require additional modeling terms beyond those that capture the existing system’s behavior. In our case, we wish to implement and test a power-aware scheduler that predictively and pro-actively powers on and off nodes based on recent load history [52]. To enable this, we must extend the model to represent periodic polling of load (the periods are called "epochs" in the scheduler algorithm) and power states of the nodes (*awake*, *waking*, and *asleep*) that the scheduler can manipulate via messages to the nodes. We extend the output set of this scheduler to include the aggregate power-up delay it generates and the amount of time each node spends in the *awake* or *waking* state. We perturb the model by representing the delay necessary to power a node up as empirically determined distributions (so as to avoid their simulation overhead).

3.3.2 Discrete Event Simulation

To simulate the system in faster-than-real time, the next step is to develop a discrete-event simulation that captures only the changes in the states specified in the model. In our example, scheduler events are triggered by

- the arrival of a new VM request from the input trace
- the acceptance and launch of a new VM assigned to a node
- the termination and cleanup of a VM as reported by the node running it
- the expiration of a timer marking epoch boundaries,
- the expiration of a timer marking the end of a node power-up sequence

The simulation of the scheduler (either the existing or the new power-optimizing scheduler) from these events takes a trace of VM activity, which we represent as start-time and and duration pairs for a set of VMs.

Note that this event list demonstrates the parsimony in our approach. Through inspection, it is clear that Eucalyptus breaks the VM start and termination sequence into a series of separate "phases" for the purpose of error handling and fault tolerance. We represent these in our simulator by the perturbation of VM start-up and termination delays. Notice also that we can omit the node power-down time as its addition does not change the results in a way that we could detect.

At a high level, the simulation works as follows. User requests consist of request time (arrival), instance lifetime (duration) and instance type (size). The platform configuration contains physical node IDs and capacity (cores, memory, disk). The scheduler assigns requested instances to nodes, and removes them as they expire based on a policy (the algorithm implemented). Additionally, the scheduler is notified when node power states or epoch times change in order to perform power-budget accounting.

3.3.3 Adding Perturbations

To fit and evaluate the simulation model we extract performance information from a live cloud. We require two types of measurements: those we use to introduce perturbations (e.g. VM start-up, termination, etc. and those that we use to validate the simulations (*i.e.* the aggregated outputs of the scheduler).

We use a combination of log analysis and instrumentation to collect these measurements. Log analysis is preferable since it avoids the possibility of disturbing system performance through the introduction of instrumentation. In the case where the existing logs do not carry the information with sufficient resolution to drive the simulation, we take care to modify the source code of the platform to introduce additional logging information in a way that is unlikely to change execution performance. For example, logging new events that require synchronization of otherwise asynchronous activities must be avoided.

The measured noise – the values for launch and teardown delays – are fuzzy. To get accurate simulation results later on we represent the noise as empirical distribution of individual measurements rather than averaging them. In our simulation we then sample these distributions to obtain varying noise values when processing each individual event. As a consequence, the aggregate results of the simulation are non-deterministic and we are required to average them over multiple runs in a Monte-Carlo [51] style simulation.

3.3.4 Scheduler Operation

Both, the existing scheduler and the power-optimizing scheduler must be implemented for the simulator. The accuracy risk (and one of the reasons necessitating validation) comes from the observation that the simulated and real implementations may differ. Ideally, both the discrete-event simulation and the implementation for the real system share the same source code. In our example, that sharing is possible, but we opted instead to rely on validation so that we might implement the schedulers in different programming languages. The production system schedulers are written in C and our discrete event simulation is written in Scala.

The existing production scheduler uses a "greedy' scheduling algorithm to maximize multi-tenancy. When a new VM is to be assigned to a node, the scheduler considers the node list in a fixed order and uses a first-fit assignment algorithm.

The power-optimizing scheduler uses load measurements taken over discrete epochs to predict how many powered-up machines will be needed in the "next" epoch to avoid a power-on event with a specified probability. While a node is being powered on, the VM start will be delayed by the remaining duration of the power-up sequence. This delay is experienced by the user directly. Thus the goal of the power-optimizing scheduler is to minimize power usage, subject to an SLO specified by the cloud administrator that limits the probability of any given user experiencing a power-up delay. As in the greedy scheduler, we order hosts by status (*awake*, *waking*, and *asleep*) and ID. We place an incoming VM on the first available *awake* host (followed by a *waking* host). If no powered-up host can be found, the request will be enqueued for a powered-down node, which is immediately sent a wake-on-lan message. A start delay is incurred whenever a VM is placed on a machine in *waking* or *asleep* state.

The power manager uses a fast, non-parametric quantile predictor and makes conservative estimates about the number of hot spares needed to fulfill the responsiveness (non-delay) SLO. It determines the target count of active nodes in fixed time steps - epochs - by comparing the current spare capacity of the cloud to the size of request bursts in the past. Depending on the result of this comparison additional nodes are then woken up or powered down.

Note that our implementation of cloud federation described in the next chapter relies on a similar quantile predictor to make admission decisions. As we will show, the simulation model produces sufficiently accurate results to feed the power manager's predictor. This compels us to use a similar simulation model for evaluating the efficacy of our cloud federation architecture.

3.4 Results

In this section, we evaluate our approach and its example implementation. We first overview our experimental setup and then present the results that we achieve by statistically registering our simulator with the actual target IaaS system it simulates. Using registered simulation, we then evaluate our power-aware scheduler and evaluate a number of different capacity planning scenarios, using a number of different traces (actual and synthetic) and cloud configurations.

For our empirical measurements we use a seven node commodity hardware cluster.

Table 3.1: Summary of Synthetic Workloads . Units are in Seconds.

Name	Total Duration	VM Count	Arrival	Duration
Exponential	36305	443	$\lambda = 0.0125$	$\lambda = 0.002$
LogNormal	35646	420	$\mu = 3.8$ $\sigma = 1.0$	$\mu = 4.5$ $\sigma = 1.0$

Each node runs on CentOS v6.5 and holds four cores, 8 GB ram, and a 500 GB hard drive and is connected to the network via two 1 Gbit ethernet links. We set up Eucalyptus v3.4.2 with a dedicated head and storage node and six nodes serving as instance hosts. Since we need control over the placement of instances and power management of nodes, we install from source and inject a small code modification that enables explicit node selection by our scheduler. We implement the power manager to interact with the cluster controller via its shared-memory interface. Eucalyptus is a production-quality system and as such includes a number of security features that are in place to prevent these kinds of outside modifications. For this reason, we temporarily disable message signature verification to make the injection of load traces less labor intensive to implement.

We use a number of synthetic workloads and production cluster traces [76] to evaluate the simulator and the power-optimizing scheduler. The synthetic workloads are generated from exponential and lognormal distributions for instance arrival times and durations (details in Table 3.1). We also have access to anonymized traces from Eucalyptus installations used in enterprise production (described later in Table 3.7). Using these traces it is possible to "replay" the VM load and scheduling activity that took place when they were gathered, either in simulation or on a working Eucalyptus system.

3.4.1 Simulation Registration

For registration we execute a benchmark trace on a single node which has been separated from the six node Eucalyptus IaaS cluster and measure various system overheads.

Table 3.2: Summary of Empirical Cluster Attributes collected

Attribute	Description
VM start delay	Instance start delay until boot sequence
VM teardown delay	Instance termination delay until resources freed
Node wakeup delay	Time required for node wake-on-lan

The registration trace contains 100 instance start- and stop-requests over a period of 10 hours. We use choose a constant interarrival time between requests to avoid an implicit look-ahead bias towards our synthetic exponential and log-normal test traces. We collect the empirical samples of instance startup, instance termination, and power-up delays. Separately, the latencies for hibernation are obtained by manual execution of a script power cycling the machine. We then configure the simulator to use these empirical latency distributions and prepare for testing the power manager with synthetic workloads. The specific attributes that we profile in this study are shown in Table 3.2.

Registering the simulation this way is not necessarily straight forward: Eucalyptus uses a polling model so that it can control message traffic internally. The cluster nodes do not report the completion of state changes to the head node until they are polled explicitly, which currently happens in 6 seconds intervals and can lead to artifacts in the observed distributions. Anecdotally, in our first registration trace we used fixed interarrival times of requests which was a multiple of the 6 seconds polling interval. This caused us to measure delays in multiples of 6 seconds plus an offset – depending on when we first started the registration run within the 6 second polling window – only. To reduce distortions in our observations due to synchronized launch and polling intervals we introduced a small random variation in start- and stop-times of instance requests that we draw from a uniform distribution (between 0 and 6 seconds).

We use synthetic workloads during the registration testing phase so that we can ensure

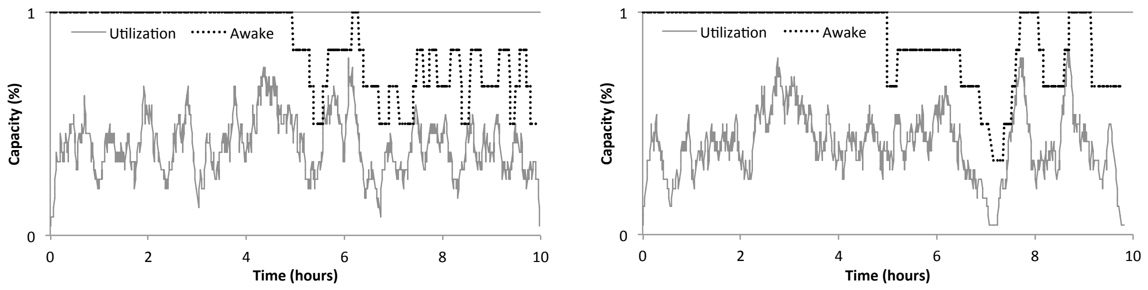


Figure 3.1: Timeries showing synthetic exponential (left) and lognormal (right) workload trace with power-optimizing scheduler activated at the 5 hour mark. The x-axis depicts time in one hour intervals, and the y-axis shows the fraction of the number of cores occupied. The dotted line shows the fraction of cores that belong to nodes that are powered-up.

Table 3.3: Utilization per Node (Exponential Trace)

	All	A	B	C	D	E	F
sim (mean)	0.4008	0.8727	0.7564	0.5195	0.2217	0.0346	0.0000
sim (sd)	0.0066	0.0033	0.0091	0.0085	0.0085	0.0047	0.0000
real (mean)	0.4033	0.8742	0.7551	0.5250	0.2311	0.0344	0.0000
real (sd)	0.0052	0.0053	0.0061	0.0062	0.0075	0.0024	0.0000

that the observed response of the system is meaningful on a feasible time frame. That is, a replay of the production traces described later in Table 3.7 in real time would span months if executed in real time. Alternatively, selective extractions of tractable "busy" periods might skew the sample and the attempt to "speed up" the trace (*i.e.* using a fitted probability model as described in [77]) could introduce additional error.

Thus, to test registration accuracy we choose two synthetic traces each having a duration of 10 hours, with a mean utilization of 1/3 of the 6 node cluster capacity. The

Table 3.4: Uptime per Node (Exponential Trace)

	All	A	B	C	D	E	F
sim (mean)	0.8711	1.0000	1.0000	1.0000	0.9128	0.7766	0.5375
sim (sd)	0.0127	0.0000	0.0000	0.0000	0.0166	0.0235	0.0119
real (mean)	0.8758	1.0000	1.0000	1.0000	0.9312	0.7704	0.5529
real (sd)	0.0094	0.0000	0.0000	0.0000	0.0115	0.0174	0.0098

Table 3.5: Utilization per Node (Lognormal trace)

	All	A	B	C	D	E	F
sim (mean)	0.3974	0.8555	0.7305	0.5140	0.1960	0.0665	0.0217
sim (sd)	0.0045	0.0024	0.0053	0.0082	0.0039	0.0001	0.0023
real (mean)	0.3985	0.8550	0.7223	0.5213	0.2025	0.0696	0.0202
real (sd)	0.0043	0.0022	0.0046	0.0059	0.0050	0.0037	0.0036

Table 3.6: Uptime per Node (Lognormal trace)

	All	A	B	C	D	E	F
sim (mean)	0.8565	0.9851	0.9851	0.9637	0.9161	0.7192	0.5696
sim (sd)	0.0025	0.0000	0.0000	0.0020	0.0000	0.0042	0.0038
real (mean)	0.8605	0.9851	0.9851	0.9652	0.9178	0.7292	0.5805
real (sd)	0.0048	0.0000	0.0000	0.0000	0.0001	0.0112	0.0033

first trace is generated from an exponential distribution for arrival times and instance durations, whereas the second trace uses a lognormal distribution for both. For the exponential distribution, these values of λ correspond to a mean inter arrival time of 80 seconds and a mean duration of 500 seconds. For the lognormal distribution, the mean inter arrival time is 81 seconds and the mean duration is 785 seconds. Note that there is a minimum lifetime of 360 seconds to allow for instance startup and all VM requests issued are single-core and uniform in memory and disk requirements.

In all test cases the power-optimizing scheduler is configured to guarantee a responsiveness SLA that at least 95% of all start requests will not be affected by a delay due to waking a node from hibernation. The epoch length is set to 300 seconds with a minimum history length of 60 epochs, which triggers activation of the power manager at the five hour mark in our benchmark traces.

Our results show agreement between simulation predictions and out-of-sample measurements. We repeat simulation and real world runs 12 times (a total of 120 hours) for each trace separately and compute the averages. For visualization, two exemplar runs from the benchmarks are shown in the graphs in Figure 3.1. The figures depict

Table 3.7: Summary of Private Cloud Dataset Characteristics

Data Set	Nodes	Cores/Node	Time Period	Description
DS2	7	12	Aug. 2012 to Apr. 2013	Medium sized company with 2,000 to 5,000 employees
DS3	7	8	Aug. 2012 to May 2013	Small company with 50 to 100 employees
DS5, DS6	31	32	Nov. 2013 to Dec. 2013	Large company with 50,000 to 100,000 employees

the activity of the power manager over time. The y-axis represents the number of cores used, normalized to maximum capacity. The x-axis represents time in one hour (3600 seconds) intervals. The solid line shows the number of cores occupied by instances in the cluster while the dotted line shows the number of cores available on awake nodes. The activation of the power manager can clearly be seen at the five hour mark. With changes in utilization, a fluctuation of the number of awake nodes can be observed. Due to the high frequency of these changes in our registration traces, we expect the impact of inaccuracies in the simulation to be exacerbated.

Tables 3.3 and 3.4 show average utilization and uptime (expressed as fractions, respectively) per node and their standard deviations using the synthetic exponential trace. The counterparts for the synthetic lognormal trace can be found in Tables 3.5 and 3.6. For this experiment, we are concerned with numerical accuracy and do not adjusted the power savings for the power manager's warmup period. We find a good match between simulation and real world observation, with the largest per-node difference of 2%.

Note that in our initial runs the registration of both, core utilization and up time between simulated and measured exponential runs did not seem to match as precisely as we had anticipated. In particular, the utilization and uptime of nodes seemed to differ to a greater extent than we had hoped. Investigating the cause of this inconsistency, we discovered an implementation bug in the power manager that we integrated into Eucalyptus. This discrepancy illustrates an ancillary benefit to trustworthy simulation. By working with a perturbative model we were able to anticipate the degree of accuracy we could expect and thus launch a targeted debugging effort when we did not achieve it.

3.4.2 Power-Aware Scheduler at Scale

The results described in the previous section show that the simulation of Eucalyptus with the power-optimizing scheduler match the observations of an actual Eucalyptus implementation of the scheduler to an error of less than 2% at scales that are feasible to test. In this section, we use the simulator to study the effects that the scheduler would have achieved in production settings had it been available and deployed.

We run the simulator using traces gathered from the logs generated by Eucalyptus when run in several production settings. The commercial enterprises who donated their Eucalyptus logs to the project asked not to be identified specifically. Table 3.7 summarizes the node and core counts for each commercial trace, its duration, and a description of the size of the business. We number the datasets as DS2, DS3, DS5, and DS6 as they are part of a larger collection of data sets. The anonymized traces from the collection are available to the research community from [76].

We do not replay these traces through a "live" installation of Eucalyptus because each of these traces spans several months in real time. Further, in order to observe the power-optimizations from a working system, it would have been necessary to recreate the specific deployments that generated each trace. Note, however, from Table 3.7, that the core and node counts in these production deployments are modest. Production enterprises are often partitioned into smaller units both to enhance fault isolation and to allow resource expenses to better track business unit organization.

Thus at these scales, the power-optimization results are likely to be nearly as accurate as those shown in Subsection 3.4.1. An inspection of the source code indicates that any additional overhead introduced by the additional nodes and cores would be covered by the perturbation terms in the model with one important caveat. At the time these traces were generated, the power-optimization scheduling algorithm did not exist nor had we begun its development. Thus we lack the specific hibernation and wake-on-lan response times that are necessary to parameterize the model. In the absence of this data, we use the empirical samples from our test cloud in its place. The result is an accurate simulation of what the efficacy would have been if the machine power-cycling performance response were the same as it is in our laboratory.

Two of the three production traces are nine months in length with highly variable resource demand. The third trace is one month in length and has a very regular workload. Further, we set the power-manager epoch time to 1000 seconds, as suggested by the original authors, and use the same 95% responsiveness SLA as before. Also, the history length for samples takes by the power-optimizing scheduler is set to 2000 samples.

To predict efficacy, we define power efficiency to be the total CPU time for all nodes normalized to its theoretical maximum possible, divided by the uptime of active nodes normalized to its always-on baseline (as shown in Equation 3.1).

$$efficiency = \frac{total\_cpu\_time/max\_cpu\_time}{total\_uptime/max\_uptime} \tag{3.1}$$

Recall from Subsection 3.4.1 that CPU time is the amount of CPU time used by a node to run the VMs assigned to it and uptime is the total duration that a node is in the powered-up or waking state. As such, this formulation of efficiency captures the degree to which the power used by the system is used to run VMs.

We run a Monte-Carlo simulation 30 times for each trace, once with power manager

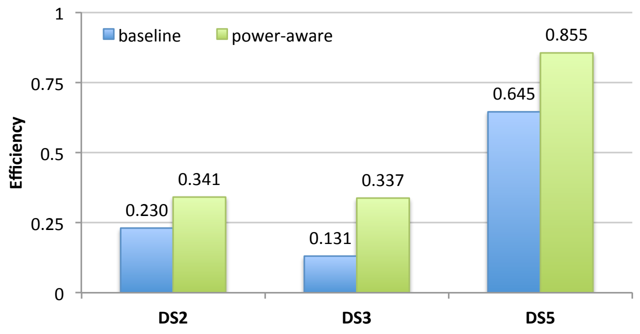


Figure 3.2: Comparison showing power-efficiency of the power-aware scheduler with bars representing baseline (left) and power-aware (right) efficiency for three production traces (DS2, DS3, and DS5).

enabled, once without. The results are shown in Figure 3.2. Error bars have been omitted due to the minimal deviation of the averages between runs. For the variable DS2 and DS3 the efficiency increases by a factor of 1.5 and 2.6 respectively, and for the constant DS5 by 1.3. While the relatively small improvement for the constant workload DS5 seems intuitive, the differences between DS2 and DS3 are not obvious at first. Close investigation shows that DS2 contains requests that demand access to the whole cluster after long periods of inactivity while DS3 has users demand small batches several times before issuing a large request. With this difference, the power manager becomes more conservative in its predictions for DS2 compared to DS3, which results in lower overall power-savings and efficiency.

Note that in our initial runs of the long-term traces we observed a large miss-percentage for the first two traces with our implementation of the power manager. The simulation and implementation agreed, but together they did not meet the SLO that the scheduling algorithm should have obtained. In communication with the authors of the algorithm, we found a discrepancy with our implementation when correcting for very long periods of inactivity on the cluster. We corrected our implementation, both in the simulator and for Eucalyptus itself, re-validated on our test bed and then executed the long-term traces again. This time the SLO was met without exception. Due to faster-than-realtime simulation the turnaround time for debugging, updating and re-evaluation correspond to a fraction of the time required for real-time testing alone.

By replaying real-world traces, the simulator helps to determine the impact of subtle differences in the workload before the production deployment of a new scheduler. This is especially true when synthetic traces do not exhibit all the properties of production workloads. This makes testing efforts more robust and provides insights for planning the deployment of a new IaaS resource manager.

3.4.3 Capacity Planning

Aside from software development and testing, trustworthy simulation can inform capacity and business planning. So far, we have solved one-dimensional, monotonic problems for testing a power manager's efficiency under a single SLO constraint. In contrast, stakeholders in enterprises have to consider multidimensional problems with consideration given to capital and operating expenses, ease of use, robustness of a system and transition policies, among others. Using the simulator to test different platform configurations against a recorded trace, we find Monte-Carlo simulation provides additional insights to inform trade-offs between cost and expected quality of service and simplifies the decision-making process.

In this experiment, we use a 1-month section of a production trace (depicted in Figure 3.3). Given this workload and our use of the power manager, we investigate how many nodes we can remove from the cluster while still meeting our chosen SLO (95% responsiveness and 99% start request acceptance). To enable this, we run the simulation with the power manager activated (PM) and without (base) and incrementally remove nodes from the base configuration until SLO violations occur. We use the same

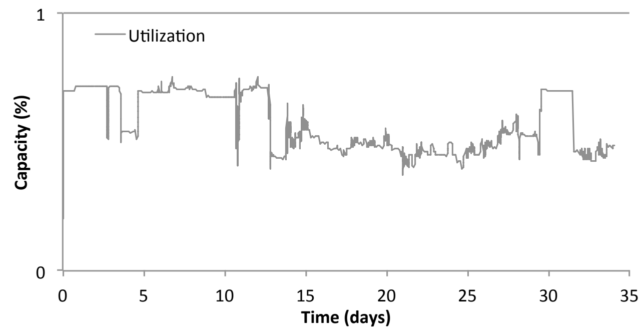


Figure 3.3: This production trace of an over-provisioned cluster with a fixed workload is the foundation of the down-sizing scenario.

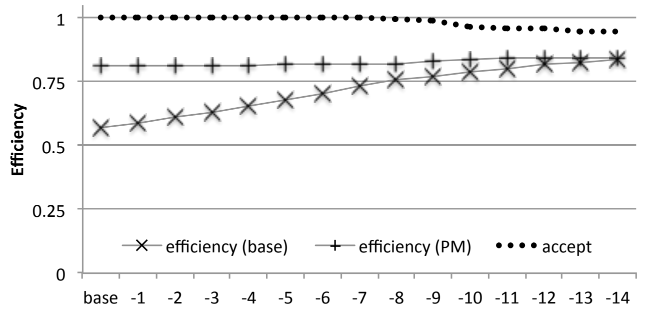


Figure 3.4: Power-efficiency increases while request acceptance rate decreases as node count goes down in the base case. The power-aware scheduler guarantees constantly high efficiency.

configuration of the power manager from previous experiment and register the system via the logged request delays of the real system. We conservatively assume 600 seconds for node power-up.

Figure 3.4 depicts the aggregate power-efficiency and request acceptance rate on the y-axis and the reduction in the number of nodes over the baseline system on the x-axis. We omit plotting misses due to wake-on-lan as the SLO is never violated in this experiment. There are two interesting insights revealed by the data. First, after a reduction by 8 nodes we cross the threshold of diminishing returns for the non-power-aware case, while the maximal reduction lies at 9 nodes before violating the acceptance SLO. However, even guaranteeing a 99.9% acceptance SLO would still allow for a reduction by 6 nodes (about

20% of the cluster). Second, for the power-aware case, we notice that the efficiency is almost constant and independent of the node count for the cluster's specific workload.

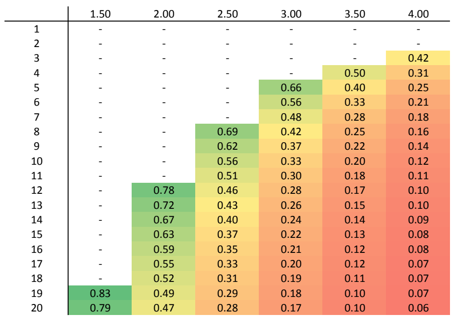
With this data about the non-parametric power-manager in hand, decision-makers can focus their attention on other aspects of the capacity planning problem. Furthermore, if different levels of quality of service guarantees are being considered, reliable estimates about their expected cost can be obtained via simulation.

3.4.4 Capacity Planning for Scale-Out Workloads

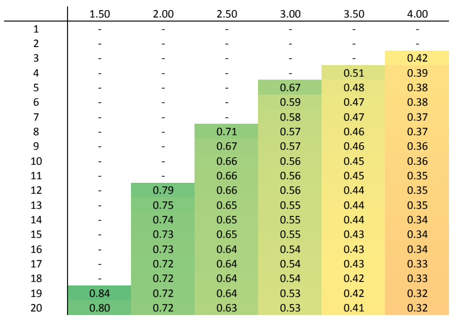
Our final use-case is capacity planning for scale. Simulation gives decision-makers the ability to make reliable forward-looking estimates about the hardware requirements for an expected workload without acquiring or renting all necessary resources (*i.e.* servers and infrastructure) for testing ahead of time.

In this set of scale-planning experiments, we run the simulator as a "parameter sweep" varying both the number of nodes in the simulation, and the intensity of the workload (by changing the mean arrival time) independently. Thus each simulation depicts the behavior of the cloud at a given size for a given workload intensity. The simulator assumes a scale-out workload, *e.g.* 3-tier web applications or MapReduce jobs, and computes the expected power-efficiency for a given platform size.

We configure a virtual cluster of nodes with properties similar to the cluster used as our real-world test bed, but double their core capacity. The workload is generated from a lognormal distribution similar to the one we use in our registration experiments (*c.f.* Table 3.1). We also set the SLO for not incurring a power-up delay to be 95% (and the scheduler achieves this SLO in each case). In addition, because the number of nodes at some point in the parameter sweep may be insufficient to run the offered load (the cloud is out of resources), we only report results for the cases where at least 99% of the



	1.50	2.00	2.50	3.00	3.50	4.00
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	-	-	-	-	0.42
4	-	-	-	-	0.50	0.31
5	-	-	-	0.66	0.40	0.25
6	-	-	-	0.56	0.33	0.21
7	-	-	-	0.48	0.28	0.18
8	-	-	0.69	0.42	0.25	0.16
9	-	-	0.62	0.37	0.22	0.14
10	-	-	0.56	0.33	0.20	0.12
11	-	-	0.51	0.30	0.18	0.11
12	-	0.78	0.46	0.28	0.17	0.10
13	-	0.72	0.43	0.26	0.15	0.10
14	-	0.67	0.40	0.24	0.14	0.09
15	-	0.63	0.37	0.22	0.13	0.08
16	-	0.59	0.35	0.21	0.12	0.08
17	-	0.55	0.33	0.20	0.12	0.07
18	-	0.52	0.31	0.19	0.11	0.07
19	0.83	0.49	0.29	0.18	0.10	0.07
20	0.79	0.47	0.28	0.17	0.10	0.06



	1.50	2.00	2.50	3.00	3.50	4.00
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	-	-	-	-	0.42
4	-	-	-	-	0.51	0.39
5	-	-	-	0.67	0.48	0.38
6	-	-	-	0.59	0.47	0.38
7	-	-	-	0.58	0.47	0.37
8	-	-	0.71	0.57	0.46	0.37
9	-	-	0.67	0.57	0.46	0.36
10	-	-	0.66	0.56	0.45	0.36
11	-	-	0.66	0.56	0.45	0.35
12	-	0.79	0.66	0.56	0.44	0.35
13	-	0.75	0.65	0.55	0.44	0.35
14	-	0.74	0.65	0.55	0.44	0.34
15	-	0.73	0.65	0.55	0.43	0.34
16	-	0.73	0.64	0.54	0.43	0.34
17	-	0.72	0.64	0.54	0.43	0.33
18	-	0.72	0.64	0.54	0.42	0.33
19	0.84	0.72	0.64	0.53	0.42	0.32
20	0.80	0.72	0.63	0.53	0.41	0.32

Figure 3.5: Parameter-sweep predicts changes in power-efficiency with increasing node count (y-axis) and workload intensity (Lognormal arrival, μ on x-axis, $\sigma = 1.0$) at the baseline (left) and with power-aware scheduler (right).

simulated VMs were able to run. The power-up delay for a node is again assumed to be 600 seconds.

Figure 3.5 shows the mean power efficiency (as computed in Equation 3.1) for each combination of intensity and node count. Node counts on the vertical dimension of the figure correspond to the number of nodes the cloud has configured. The intensity value (horizontal dimension) show the value of μ used in each lognormal parameterization ($\sigma = 1.0$ in each case). Parameter-combinations that fail to achieve the target SLOs are marked "-" in the figure. Each entry covers a simulated time-frame of 60 days.

The left-hand table in Figure 3.5 shows power efficiency for different combinations of intensity and cloud size without the power-optimizing scheduler and the right-hand table shows the same with the scheduler activated. We use a heat map to color the efficiency numbers (green for high, red for low) for each combination.

For example, in the left-hand table, the entry for 19 nodes with $\mu = 1.5$ corresponds to a power efficiency of 0.83 (colored green). Thus a cloud with 19 nodes experiencing a workload with lognormal inter arrival times ($\mu = 1.5$, $sigma = 1.0$) and lognormal durations ($\mu = 3.8$, $\sigma = 1.0$ from Table 3.1) for 60 days would achieve a power efficiency of 0.83 without the power-optimizing scheduler. The same entry in the right-hand table shows that with 19 nodes and $\mu = 1.5$ the power-optimizing scheduler achieves an efficiency of 0.84.

As expected, these results indicate that as the inter arrival time goes down (smaller values of μ) the efficiencies converge to a high value both with and without the power-optimizing scheduler. These extreme cases correspond to the cloud being "full" almost all of the time leaving little efficiency to be gained by powering nodes on and off. At the other extreme, when inter arrival times are larger (large values of μ) the power-optimizing scheduler has more of an opportunity to save power. Indeed just looking at the heat map coloration of both tables shows the trend in efficiency in both dimensions. All green entries in the regular scheduler are green for the power optimizing scheduler (it does no harm). In addition, the power-optimizing is "greener" across all entries and never "red."

With faster-than-realtime simulation we can perform parameter sweeps across large ranges of configuration parameters, such as cluster size and workload intensity. Due to the embarrassingly-parallel nature of Monte-Carlo simulation, parameter-sweeps can be performed anywhere, from a personal laptop to a group of workstations, with a flexible trade-off between accuracy and wait time. Initial results are available within minutes, followed by increasing degrees of confidence and minimal convergence. The results for this experiment total at $450,000$ VM starts and $7,200$ days (about 20 years) of simulated time, and were generated on commodity laptop hardware within 8 hours. This demonstrates the practical ability of this approach to quickly estimate the impact of different workloads, additional hardware or new resource-allocation policies.

A core interest of cloud operators is the trade-off between risking service disruption due to increases in load or the introduction of new technologies, and unnecessary capital- and operation-expenses. In our example, the data reveals that the power manager can be used safely and without negative impacts on availability, independently of the platform size and workload intensity. At high levels of utilization its impact is marginalized, however, which can inform decisions based on the expected workload and cluster size.

Another insight that can be gained from the parameter-sweep is the amount of resources required to achieve a specific target utilization of the cluster (which equals cluster efficiency of the non-power-aware baseline). In our example, a mean utilization target of 50% demands 3 nodes for the light workload ($\mu = 4.0$) and moves up to 4, 7, 12, 19 and further with increasing workload intensity. Depending on the workload, the non-linear interactions between utilization and SLO constraints are hard to estimate analytically or with rules-of-thumb. Accurate simulation overcomes this limitation and helps allocate resources efficiently inside and around the cloud.

3.5 Conclusion

Simulation plays a key role in performing experimental exploration into large scale systems. As such, simulation has significant potential for facilitating research and experimentation with cloud computing infrastructures. Cloud research in general is important for advancing the state of the art in cloud performance, scale, energy efficiency, and fault tolerance, among other features. However, the wide spread use and commercial viability of cloud computing requires that simulated results be sufficiently trustworthy (accurate) to ensure adoption of research results in production settings and to justify the engineering effort required to achieve production levels of performance and reliability. Extant simulation systems typically trade off validation and accuracy for configurability and exploratory power, through the use of ab initio techniques that facilitate comparative evaluation of cloud components and application behavior.

In this chapter, we presented a new methodology for facilitating trust in the simulation of cloud components through the use of a tool employed in the physical sciences for simulation called perturbation theory. Using this methodology, we derive a parsimonious model from a real cloud infrastructure (in our case a Eucalyptus private cloud) for the cloud component under study (in our case a scheduler). We then perturb the model using statistical sampling to represent unmodeled behavior to facilitate simulation speed and scaling. We incrementally add parameters (component inputs) to the model (incorporating key unmodeled behavior) until we achieve an acceptable level of accuracy for the component, relative to the real system. It requires, however, that we have access to a production-quality cloud that we can interrogate and validate against.

The perturbation modeling approach achieves high accuracy for simulating an existing system. We also evaluate the predictive capabilities of validated simulation for modified clouds by implementing a power-aware scheduler first in simulation and then in a real cloud for reference measurement. We find strong agreement between prediction and measurement, encouraging us to apply validated simulation to a number of enterprise capacity-planning scenarios.

Chapter 4

Estimating Job Preemption Probability in IaaS Clouds

4.1 Introduction

A pre-requisite for independent organizations contemplating participation in a federation is the guarantee that they ultimately maintain control over their resources. When adapting “cycle harvesting” to the cloud context, the one cloud executing another cloud’s workload must be able to preempt (terminate) the foreign workload at any time to service locally generated workload should the need arise. This gives rise to a tiered service structure, with “high priority” instances from local users and “low priority” instances federated from remote clouds. In this priority scheme, incoming high priority requests can preempt low priority requests if the cloud runs out of spare capacity, but not vice versa. That is, federated workload executes opportunistically on intermittent spare capacity.

In the cloud context, the SLA requires that guarantees (SLOs) about the quality of service be made ahead of time, which is at odds with the potential need to preempt federated instances after admission. Making open-ended guarantees about the continued availability (non-preemption) of instances is equivalent to giving up control over the assigned resources. One way to solve this conflict is to make availability guarantees for a limited time period and provide ahead-of-time estimates about the preemption probability of low-priority instances for this limited window only.

We study such time-limited guarantees for batch workloads with bounded job durations. Common sources of batch jobs in clouds are big data frameworks, such as Apache Hadoop [34] and Apache Spark [37], and compute-intensive tasks, such as MPI. In IaaS clouds, a "job" can be represented as a fixed number of instances executing for a fixed duration. If we can obtain an estimate of the availability of preemptible capacity during the expected execution period of a job *at admission time*, we can estimate the job's preemption probability before launch and thus enable job federation on preemptible capacity while still providing an ahead-of-time availability SLO.

In practice, the preemption probability of a federated job depends on many factors, including the properties of the user's request, other active workload and the state of the cloud as a whole. Intuitively, jobs with long durations seem more likely to encounter preemption than jobs with short durations, and similarly, jobs requiring many instances in parallel more likely than jobs requiring few. Furthermore, as federated workload executes opportunistically on spare capacity, the current utilization level of the executing cloud may also affect this probability. A cloud with a large user base makes manual determination of these complex dependencies impractical, and thus must be able to make estimates autonomously and adapt to changes over time.

Assuming that the expected duration of a federated job is provided by the user, we can estimate its preemption probability if we know the amount of time until the executing cloud reaches capacity and triggers preemption. An estimate of this "time-to-preemption" can be obtained from the historic utilization trace of the cloud by taking advantage of a validated simulation model of the cloud (c.f. Chapter 3). Even with the precise job duration unknown to the user, an upper bound on job duration is sufficient to calculate an equivalent upper bound on a job's preemption probability.

In addition to supporting a federation scheme later on, an estimate of the time-to-preemption for preemptible instances can still provide benefits on a stand-alone cloud. In private clouds, preemptible instances allow enterprise users to exceed their respective quotas by taking advantage of otherwise unused capacity. Because a preemptible instance will be terminated if the capacity is needed to run a regular instance, users can run preemptible instances without a charge to their respective quotas. That is, the capacity for preemptible instances is "scavenged" and then reclaimed when it is needed for regular instances similar to "cycle harvesting" in Condor workstation pools.

Our approach uses on-line simulation to *predict* the time-to-preemption of instances based on the recent history of cloud activity. In particular, we use a Monte-Carlo style simulation (run every few minutes) to estimate the distribution of the possible lifetimes of preemptible instances from requests in the recent past. We use this non-parametric approach to compute the quantiles (i.e. percentiles) of the empirical distributions of instance lifetimes until preemption that are conditioned on the capacity currently available in the cloud. These quantiles then serve as a probabilistic lower bound on the time-to-preemption of future instances which the user can interpret as a statistical "guarantee" of preemptible instance lifetime. For example, the lower 0.05 quantile indicates the minimum instance lifetime that as user can expect with probability 0.95.

In summary, we examine the feasibility of improving the utilization of a single cloud by using predictions of the time-to-preemption of jobs (groups of instances) to perform careful admission control and provide an ahead-of-time SLO on the availability of preemptible resources. We accept two tiers of workloads – regular "high priority" and preemptible "low priority" jobs – *on a single cloud* to achieve greater resource utilization while still providing ahead-of-time availability guarantees for both tiers as a stepping stone to implementing a cross-cloud federation architecture (c.f. Chapter 5). In particular we

- demonstrate that it is possible to provide statistical guarantees on the availability of preemptible instances for a bounded time window using production private cloud workload traces, and
- detail the effectiveness of co-scheduling regular workloads with preemptible workloads with these guarantees to utilize otherwise unused resource capacity.

We evaluate our method via simulation, replaying synthetic and recorded production traces from Eucalyptus [78, 19] IaaS clusters deployed in production systems [76]. We use the synthetic workload traces to demonstrate the theoretical efficacy of our approach and the steps required to produce accurate time-to-preemption estimates via Monte-Carlo simulation. We then apply this method in a cloud scheduler that is capable of maintaining an configurable SLO on the maximum preemption probability of preemptible instances in an IaaS cloud even when facing irregular real world traces from commercial production environments.

4.2 Related Work

Preemptible instances in public IaaS clouds were first employed in 2009 as part of Amazon Web Services (AWS) [79]. These "spot instances" are typically available at a rate significantly lower than that of regular "on-demand" instances as they allow providers to opportunistically manage capacity. Spot instances do not, however, provide a guarantee (SLO) on their lifetime: spot instances can be preempted at any time, whereas on-demand instances provide a 99.95% SLO on their availability once started.

Amazon distributes available spot capacity via a spot market, where users submit bids for unused capacity on an hourly basis. Amazon keeps the specifics of their market making mechnism as a trade secret. It is commonly assumed that the latest price of spot instances depends on current supply and demand [80, 81], although the authors in [82] argue that spot instance prices are generated from a bounded random process with a dynamic hidden reserve price most of the time. The authors in [83] model pricing as a mixture of multiple Gaussian distributions and reveal the challenges with modeling analytically, empirically observed phenomena in the cloud. Zhao et al. [84] finds poor predictability of spot prices using ARIMA time-series methods.

Google recently introduced a preemptible tier of instances [85] for their public IaaS offering. In contrast to Amazon's "spot instances" the pricing of Google's preemptible instances is fixed (albeit still lower than regular) and comes with the caveat of guaranteed preemption every 24 hours.

The availability of preemptible resources in large-scale systems has been studied before. Effective performance of Condor depends on the availability of workstations and duration of jobs which determines the probability that a job will be preempted due to a user reactivating an idle workstation or rebooting it. Wolski et al. [86] explore the fitting of long-tail distributions to describe the properties of the workstations' availability distribution for dedicated and desktop Condor pools. They further investigate parametric models to represent IaaS workload behavior in [77]. Brevik et al. [87] develop methods to automatically estimate quantile bounds on the lifetime of machines and jobs on pools of workstations. A number of studies [88, 89, 90] characterize the workload of cluster traces published by Google [91]. Similarly, several studies [92, 93] investigate the properties of Hadoop workloads in large production systems at Carnegie Mellon University, Facebook and others.

Some studies investigate optimizations in IaaS clouds when instance lifetimes are known ahead of time. [94] optimizes power-consumption by carefully scheduling instances with known duration.

Due to their unreliability but low cost preemptible instances in IaaS clouds are typically used as opportunistic accelerators for batch workloads. Chohan et al. [80] uses spot instances to speed up Haddop jobs by leveraging built-in fault tolerance to task failures. SpotMPI [95] takes advantage of spot instances for executing MPI jobs and uses a dynamically adjusted checkpointing interval to minimize losses due to preemption. The authors of [96] investigate cost-optimization on spot instances via application-specific scheduling and checkpointing. Mattess et al. [97] compare bidding heuristics for executing batch workloads on spot instances. Menache et al. [98] develop an online learning algorithm trading off between on-demand and spot execution for batch workloads.

Previous research has also studied pricing models and user experience (Quality of Service) for services built entirely on preemptible instances. Andrzejak et al. [99] model the trade-offs between spot instance bids and realized execution time to achieve probabilistic deadline guarantees for long-running jobs with check-pointing. In [100] the authors investigate a hypothetical service provider running a QoS-sensitive web service purely on spot instances, with a focus on revenue maximization. Similarly, [81] investigates a service running purely over spot instances and finds that existing SLOs quoted by public clouds capture only part of the variables relevant to service performance and user experience. Sharma et al. [101] investigates the availability of a virtual cloud built entirely on preemptible instances. Yank [102] is a snapshot server that enables state persistence for preemptible VMs with advance termination warning. Mao [103] studies startup and teardown times for instances of different public IaaS providers and points out significant added overhead for preemptible instances in practice.

In our work, we also investigate production workloads with preemptible instances, but side-step manual analytical modeling via Monte-Carlo simulation to provide a powerful new type of SLO on job preemption probability for jobs with bounded lifetime. While revenue and user experience depend on the specifics of the end-application, guarantees on preemption probability simplify reasoning about the system as a whole and allow providers to use it as foundation for custom SLA models. It further serves as a stepping stone for distributed scheduling of jobs across a federation of clouds.

4.3 Methodology

The goals of our methodology are to define a method

- for predicting the minimum time a preemptible instance can remain active in the system before being preempted with configurable confidence bounds, and
- for using these predictions in scheduler-level admission control to ensure that all accepted preemptible instances meet their target duration with a fixed probability.

This latter requirement is consistent with current cloud abstractions in that requests are either accepted by the cloud (and thus subject to the SLOs advertised in the SLA) or rejected ahead of time because the pre-defined SLO on preemption probability cannot be met.

We base our estimates of this “time-to-preemption” on historical observations of previous instance behavior in the cloud. The major challenge lies in the fact that it is insufficient to track the realized lifetimes of preemptible instances in the past. First the realized lifetimes of preemptible instances do not help us determine the upper bound of the duration they could have executed for before preemption. That is, instances with short durations may run to completion without encountering preemption, even though they would have been preempted in the future, had they been longer. Second, unless the system already has an extensive history of (sufficiently long-running) preemptible instances we face a problem of small sample size. In order to make reliable estimates of the future time-to-preemption we need numerous observations of instances preemptions. To make this even more challenging, the preemption probability depends on various external factors, such as the utilization level of the cloud at instance launch, which require us to observe preemption in different circumstances.

To overcome these limitations, we rely on a validated simulation model (c.f. Chapter 3) to construct empirical distributions of the lifetimes until preemption as instances would have experienced them. We further condition the lifetimes on the available cloud capacity at instance launch and obtain a sufficient sample size via Monte-Carlo style simulation. We extract the quantiles from these lifetime distributions associated with the SLO offered by the IaaS cloud for preemptible instances (e.g. a 0.95 or 0.99 confidence bound on the likelihood that the instance will not be preempted) to predict minimum lifetime for each level of available capacity.

Note, that our availability guarantee – similar to a "survival" or "job completion" guarantee – is different from availability SLOs commonly advertized by commercial IaaS providers. For example, Amazon EC2's SLA [104] guarantees reachability of at least one instance per availability zone (past the first) for a fixed fraction (0.9995 or 0.99, minus exclusions) of a one month time frame. That is, multiple service interruptions are acceptable if the aggregate downtime does not exceeded the threshold. Our method provides a guarantee that the requested capacity will be available *continuously*, from request acceptance, for at least the time-to-preemption with a fixed probability (e.g. 0.99). That is, a job executing for (less than) the requested guarantee period has *at least* a 0.99 chance of doing so without interruption.

For admission control, we assume that preemptible requests are accompanied by a *user-specified* duration (maximum) when submitted. Our IaaS scheduler uses (a) the quantile estimates for the SLO generated by the simulation, (b) the instance size (also specified per request) and maximum duration from the user, and (c) the currently available capacity of the system, to decide whether to admit a preemptible request. The scheduler preempts instances if/when a regular instance request is made and the cloud has insufficient capacity to service the request.

4.3.1 Scheduling Model

Instance requests (to either start or stop an instance) are routed to a scheduler (as implemented by IaaS infrastructures such as Eucalyptus [78], Open Stack [18], and Cloud Stack [20]) which handles admission control and placement of instances on physical resources in a cluster of "nodes". IaaS clouds typically define "instance types" that describe the resources that an instance will consume (CPU cores, memory, ephemeral disk storage, etc.). In the Eucalyptus systems (production and research) that we investigate in this work, we observe that the memory footprint associated with each instance type is such that the instance placement decision by the scheduler can be made strictly on core count.

When an instance is admitted, the scheduler makes a placement decision by selecting a node on which the instance will run. In this study, we use simple first-fit placement in favor of more complex approaches to highlight the impact SLA-aware admission control. If a regular instance is requested, and the scheduler cannot find a node with available capacity, the scheduler selects one or more preemptible instances to preempt (terminate) so that the regular instance can be scheduled.

Further, our scheduler (like other Eucalyptus schedulers) assumes that the instance type definitions nest with respect to their core counts. For example, an empty 4-core node node is seen by the scheduler as having 1x 4-core slot, 2x 2-core slots, 4x 1-core slots, or 1x 2-core, 2x 1-core slots. The distinction between available cores and *available slots* is important when generating time-to-preemption estimates for different instance sizes and different cluster load levels.

4.3.2 Preemption Policy

The preemption policy affects the preemptible instance lifetime distributions generated by the simulation (but not the correctness of the method). Many policies are possible but each has an impact on user experience. In this thesis we chose a simple "Youngest-Job-First" (YJF) preemption policy. Choosing the "youngest" (i.e. the preemptible instance that has started most recently) to preempt among the candidate preemptible instances is an attempt to minimize the "regret" associated with an preemption in this online decision making problem [105]. That is, the amount of work that is lost because of an preemption is minimized.

4.3.3 Predicting Preemption

Past work has shown that cloud workloads can be highly variable and may not be easily described by single well-known distributions [106]. To address this problem we run a Monte-Carlo-style simulation on-line to generate the empirical distribution of observed instance lifetimes before preemption. However, we note that the time-to-preemption is affected by the capacity of the cloud that is occupied by regular (non-preemptible) workload and other preemptible instances. Intuitively, if the cloud is relatively "empty", a preemptible-instance that is introduced will likely live longer than if the cloud is close to "full" capacity. Thus, our Monte-Carlo simulation produces a *set* of empirical distributions, one conditioned on each level of possible occupancy.

The Monte-Carlo simulator generates a sample of "fictitious" preemptible instance requests with "infinite" requested duration using the recent cloud load history. It repeatedly chooses a random point in the history and simulates the arrival and (eventual) preemption of a preemptible instance, recording the occupancy level at he time the preemptible instance starts and its time-to-preemption. Running faster than real time, it generates a fixed number of such samples (e.g. 10000 samples) and divides them into empirical distributions based on occupancy level.

For example, a cloud with 100 cores has 101 possible occupancy levels: from 0 cores occupied to 100 cores occupied and each level of occupancy corresponds to a different distribution of preemptible instance lifetimes. We use quantiles of these distributions to quote the expected lifetime to the scheduler during the admission control phase based on the current occupancy level at the time the preemptible instance request is made. If the instance (based on its maximum duration specified by its user) is expected to be preempted with a higher probability than specified by the target probability (quoted as an SLO) for the cloud, it is rejected (not admitted). The cloud administrator is responsible for setting the SLO on preemption probability that is advertised to all cloud users.

4.3.4 Evaluation Metrics

To evaluate the system we use trace-based simulation with both synthetic and production traces taken from private Eucalyptus IaaS clouds. We replay each trace in its entirety and we log each individual state change in the simulated system. In each case, the simulator uses separate traces for preemptible instance and regular instance requests. We then generate summary statistics and evaluate our solution using two metrics:

- preemption fraction of preemptible instances

  $preempted = preemptions/admissions$

- admission fraction of preemptible instances

  $admitted = admissions/requests$

The enforcement of the target SLO probability has highest priority. After the SLO is fulfilled, a high number of completed preemptible instance requests is desirable to maximize utilization.

Table 4.1: Parameters of synthetic log-normal regular and preemptible instance workloads

	VM arrival	VM duration	VM cores	mean util.
regular	$\mu = 4, \sigma = 1$	$\mu = 6, \sigma = 1.5$	1	21.77
preemptible	$\mu = 4, \sigma = 1$	$\mu = 6, \sigma = 1.5$	1	21.95

4.4 Results

Our experiments are run in simulation, based on our previous work on validated simulation of private IaaS clouds. We use both, synthetic traces and anonymized production traces obtained from Eucalyptus IaaS cloud installations. For reproducibility we assume instant start and stop of instances in the traces and rely on a publicly available set of anonymized commercial production traces [76]. Our traces contain data about instance arrival times, duration and core counts.

Prediction with synthetic traces

To outline our approach and show its basic behavior we compare a scenario with an SLA-unaware scheduler and the SLA-aware scheduler using multiple different preemption SLO targets using 10-day synthetic traces (Parameters in Table 4.1). Our initial setup uses a single platform (IaaS cluster configuration) and synthetic regular and preemptible request traces. The platform contains 8 nodes with 4 cores each, for a total of 32 cores. As a rough estimate based on mean utilization the platform should be able to support the regular trace plus half the preemptible instance trace. We use a log-normal distribution to approximate the long-tailed empirical distribution of instance interarrival times and durations. Furthermore, we uniformly use a fixed core count of 1 per instance in the synthetic trace.

Note that there is a trade-off between the probabilistic guarantee given to the user and the fraction of preemptible instance requests that can be accepted by the scheduler. Greater "certainty" associated with a preemption SLO (in the form of a lower preemption

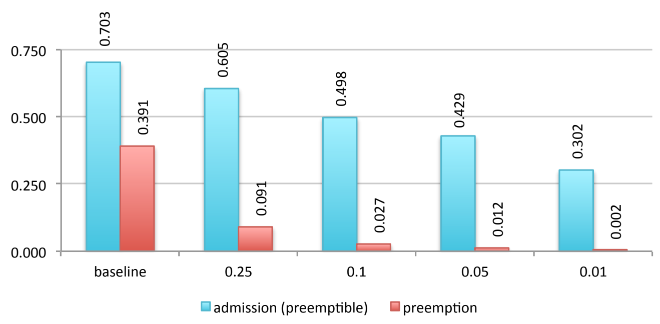


Figure 4.1: Fraction of admitted and preempted instances with synthetic log-normal traces.

probability) implies that fewer preemptible instance requests can be accepted (to decrease the possibility that an preemption will be necessary).

To illustrate this trade-off, we show the fraction of admitted preemptible instances, as well as the preemption fraction of preemptible instances in Figure 4.1. The x-axis shows different SLO probabilities, starting with the no-guarantees baseline on the left and then increasingly stringent SLOs of 0.25, 0.10, 0.05 and 0.01. The y-axis shows the fraction of admitted preemptible instance requests instances in gray and the fraction of preempted instances in black. The SLA-aware scheduler meets the SLO in all cases (the preemption fraction is less than the advertised guarantee level), at the cost of preemptively rejecting an higher fraction of preemptible instances for stricter SLOs. The measured quality of service (the fraction of preempted instances) are in fact stricter than the target SLO, showing that the predictions of time-to-preemption made by the simulation are conservative.

The most visible improvement is the step from the no-guarantees baseline to the 0.25 preemption SLO. While the baseline admits 70% of all requested preemptible instances, 39% of the admitted preemptible instances are preempted before completion. The 0.25 SLO in contrast admits 60% of all requested preemptible instances, but only 9% of the admitted preemptible instances are preempted. Subsequent decreases in the demanded

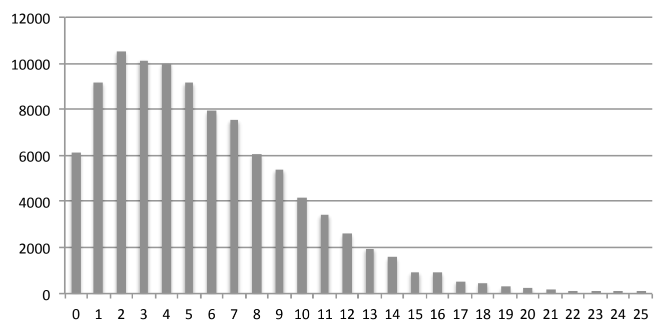


Figure 4.2: Number of time-to-preemption samples per available-slots bucket for a synthetic log-normal simulation run. Frequently encountered load levels (left) have many samples, corner cases (right) have few. The shape of the histogram depends on the historic workload.

maximum preemption fraction of preemptible instances decrease the number of admitted preemptible instances as well, but consistently (conservatively) achieve the preemption probability set forward in the SLA.

This experiment outlines the setup of our simulation driven approach to enforce fixed levels of preemption probability in a controlled environment. In the next section we discuss the simulation method in-depth.

4.4.1 Conditional Distributions and Sample Size

The scheduler computes conditional distributions for *all* possible core counts on a fixed duty schedule (every 6 hours of trace time in the previous experiments) based on the history of regular and preemptible instance behavior it has observed so far. This gives rise to the property that the sample sizes for "rarely" occurring conditions may be small. For example, if the cloud is moderately loaded, the number of examples where all but one of the cores is busy might occur infrequently or not at all.

To provide an in-depth insight in the behavior of the Monte-Carlo simulation, we provide an exemplary intermediate results at the 9 day mark of our synthetic trace experiment for single-core instance slots. Figure 4.2 shows the number of samples generated

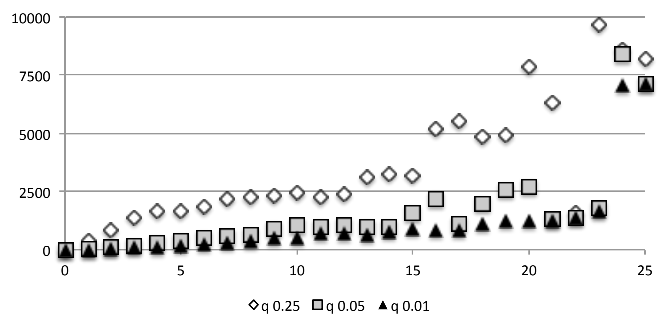


Figure 4.3: Quantiles of time-to-preemption per available-slots bucket for a synthetic log-normal simulation run. Predictions for common load levels (left) can be made with high confidence, predictions for infrequent ones (center) are rough estimates that become increasingly erratic for corner cases (right).

on the y-axis for each condition on the x-axis (available slot count). In our specific example, 2 to 4 open slots are encountered the most frequently, with about 10000 samples each. High open slot counts, which correspond to low cluster utilization, are increasingly uncommon. Based on the number of samples we expect predictions for common cases to be highly accurate, while infrequently occurring cases will be based on empirical distributions estimated from small samples.

Figure 4.3 shows the quantiles of the conditional distribution of times-to-preemption. The x-axis again shows the condition, while the y-axis indicates the time-to-preemption as estimated by a quantile. The estimates to the left correspond to the buckets with high sample count in Figure 4.2, whereas the estimates to the right decrease in sample count. The 0.25 quantile lies above the 0.10 quantile, followed by the 0.01 quantile. These quantiles estimate the minimum time a preemptible instance is expected to survive with the corresponding probability. For example, from the figure, 0.01 of the instances that are started when there are 15 free slots run for 900 seconds or less before being preempted. In the same column (for 15 free slots), 0.05 of the time-to-preemption samples are 1500 seconds or less, and 0.25 of them are 3200 seconds or less.

A combined look on the counts per bucket in Figure 4.2 and the corresponding quantiles in Figure 4.3 also provides an insight into the reliability of conditional estimates. Buckets 0 to 14 each have over 1000 samples each to determine quantiles from. This is generally enough for stringent preemption probabilities, such as 0.05 or 0.01. With increasing slot count (decreasing cluster utilization) a smoothly changing, and mostly increasing estimate of the time-to-preemption can be observed. Buckets 15 to 20 still have over 200 samples each, which is enough for rough estimates, but a look back at the quantiles shows that changes from bucket to bucket already become erratic. Estimates for 21 available slots and over appear extremely infrequently in our synthetic trace. Their samples are mostly artifacts from the initial warm-up period and as such, their estimated quantiles are not reliable (but also hardly used).

Since we are using a synthetic trace based on log-normal distributions for arrival time and instance duration, this specific example could be described analytically as well. However, for arbitrary traces, as found in production environments, this is challenging to impossible depending on the typical usage of the cluster. Monte-Carlo simulation offers a way to estimate arbitrary empirical distributions and can be tailored to achieve the desired degree of prediction accuracy.

4.4.2 Prediction with Production Traces

To study the utility of Monte-Carlo-based probability estimation in a more realistic setting, we use four different traces obtained from independent Eucalyptus IaaS production installations for our experiments. The origin of these traces is documented in [77, 106, 107]. and the traces themselves are available as part of a collection from [76]. Table 4.2 shows the mapping of data sets from the collection to experiments in this paper, together with a short description of their workload and platform properties.

Compared to synthetic traces there are a number of important differences. First,

Table 4.2: Mapping of recorded commercial production traces and their original hardware platforms from the data set collection [76] to experiments in this paper.

Name	Source	Organization	Workload	Nodes
A	DS2	Medium	bursts	7 x 8 cores
B	DS3	Medium	bursts	7 x 12 cores
C	DS5	Large	variable	31 x 32 cores
D	DS6	Large	constant	31 x 32 cores

instance starts show temporal auto-correlation. These "bursts" of instance starts are more extreme than ones observed in synthetic log-normal traces. Second, the behavior of users changes over time and causes change points which the empirical distribution derived via Monte-Carlo simulation only picks up over longer time frames. Third, instance sizes are no longer uniform in size as the traces contain instances with slot sizes between 1 and 30 cores.

To facilitate the experiments with real world traces, two modifications are made to the Monte-Carlo simulation. First, we expect that our randomization approach may not generate starting points needed for all conditional core-utilization levels needed, especially in the beginning of the experiment where data samples are scarce. To avoid rejecting preemptible instances unnecessarily due to a perceived lack of information, we linearly approximate quantiles of unobserved conditional distributions between observed "neighboring" distributions.

For example, if the empirical distributions conditioned over 20 slots and 18 slots are available, while there are no samples for 19 slots, the quantiles for 19 available slots are generated by linear approximation between the the matching quantiles of the neighbors. For example, the 0.01 quantile for 19 slots would then be calculated as $q(0.01|19) = (q(0.01|18) + q(0.01|20))/2$. In the case where multiple conditions are missing, we fit a line to the two endpoints in the range of missing values and use it to approximate the quantiles between. Additionally, the extreme points of zero and full utilization need

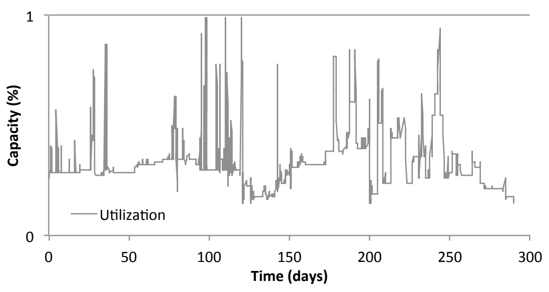


Figure 4.4: Production trace A as executed on its native platform shows constant load interleaved with bursts of large requests.

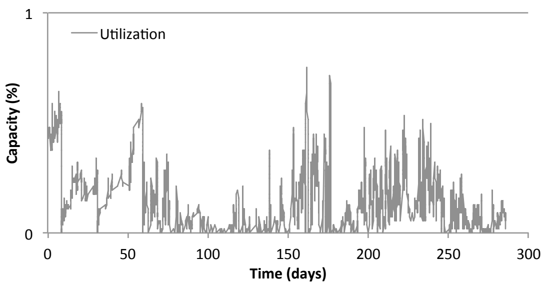


Figure 4.5: Production trace B as executed on its native platform shows highly variable load and bursts of large requests as well.

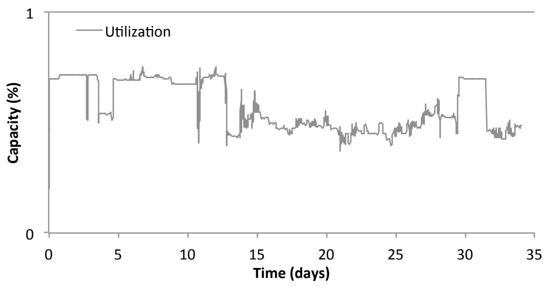


Figure 4.6: Production trace C as executed on its native platform shows a mixed pattern of load with constant plateaus and periods with higher variability.

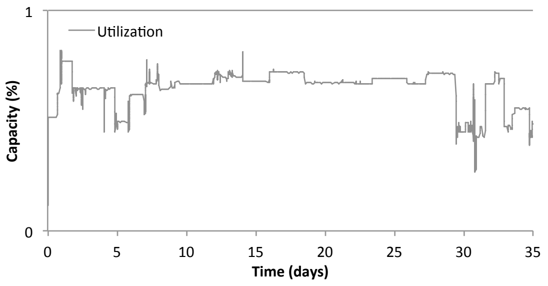


Figure 4.7: Production trace D as executed on its native platform shows a mostly constant load with a few spikes.

to be populated with useful data. We chose a zero value for expected lifetime before preemption if there are no slots available for a given capacity and conservatively use the quantiles for the lowest known cluster utilization as values for zero utilization as well.

Second, we start the real world traces after a delay of 24 hours as we do for the synthetic traces. Such a delay allows the scheduler to "warm up."

A visual inspection of the real-world traces shown in Figures 4.5 and 4.6 shows significant spikes at irregular intervals. If a spike in non-preemptible load appears in an environment already loaded with preemptible instances, we expect to see a high number of correlated preemptions, possibly leading to a violation of the SLO in the short-term.

If these correlated preemptions are not compensated for in the long-term by conservatively maintaining a capacity buffer, these short-term violations will sum up to an SLO violation over the course of the whole trace. We try to capture this auto-correlation by replaying the actual observed trace in our Monte-Carlo simulation rather than re-sampling the input distribution. That is, we choose random locations in the trace, but then replay the trace from those periods to include auto-correlation effects.

We take the same approach to handling change points in the production time series traces. In our experiments, the Monte-Carlo simulation that computes the empirical conditional distributions is re-run every 6 hours of trace time to capture changes that may have occurred in the underlying dynamics. Note that the choice of this interval is arbitrary and can be a configuration parameter in a production system. It should be large enough to accommodate a full rerun of the Monte-Carlo simulation – which merely takes 300 seconds (5 minutes) in our setup – and short enough to react to changes in overall workload and user behavior – which occurs over timeframes of days and weeks. In practice, the computation will temporarily consume instance capacity in the cloud, hence there is an economic aspect to the choice of parameter as well.

The third difference of real-world traces over to our synthetic ones are non-uniform instance core counts. This has two major implications: first, Monte-Carlo simulation must consider different instance sizes and second, placement decisions for regular instances made at any time may have consequences later in the trace. Because the scheduler attempts to find space for a regular instance and only preempts when there is insufficient capacity, the presence of preemptible instances can change where the scheduler places regular instances. As a result, because an instance cannot span nodes, it could be that the introduction of preemptible instances increases the "fragmentation" of the available core capacity and, hence, affects the ability to run regular instances. However, while preemptible instances *might* cause the scheduler to reject a regular instance it would have otherwise accepted (due to fragmentation effects) all of the regular instances that are accepted experience the same quality of service that they would have without preemptible instances present. This effect (detailed in Subsection 4.4.4) is small for the production workloads we study but grows as the cloud runs closer to capacity.

The conditional distribution of expected lifetimes therefore effectively becomes conditioned over instance capacity (taking into account fragmentation effects) in addition to available slot count. The conditioning over instance capacity does not increase the amount of data required for accurate estimates as we can re-run the same recorded trace with different virtual instance sizes. An increasingly diverse population of instance types therefore leads to a linear increase in computational effort for Monte-Carlo simulations, but not to a relative reduction of estimation accuracy. In practice, we do not expect this to be a severe problem due to the embarrassingly parallel nature of Monte-Carlo simulation.

4.4.3 SLA-Aware Co-Scheduling of Production Traces

Having addressed the issues associated with generating predictions for production traces the question remains whether these modifications allow effective admission control for varying types of production workloads. In this section we investigate the efficacy of our approach for co-scheduling different production workloads while maintaining an SLO for both, regular and preemptible instances.

We perform the evaluation with production traces in two parts and pair up our production traces based on similar platform sizes. The first combination uses highly variable workloads, “A” as regular trace and “B” as preemptible instance trace. The specifications of the physical cloud platform are taken from “A”, which contains 7 nodes with 12 cores each. We refer to this configuration as “A-B”. We use the inverse notation

Table 4.3: Results of co-scheduled workloads with production traces without SLA enforcement. In all cases the preemption fraction is greater than 0.01.

Baseline	A-B	B-A	C-D	D-C
admitted (regular)	0.977	1.000	1.000	0.997
admitted (preemptible)	1.000	0.850	0.943	0.963
preempted	0.013	0.024	0.016	0.013

Table 4.4: Results of co-scheduled workload with production workloads with SLA-aware scheduler, fulfilling the 0.01 preemption fraction SLO (equivalent to a 0.99 survival fraction)

SLA-aware	A-B	B-A	C-D	D-C
admitted (regular)	0.977	1.000	1.000	0.999
admitted (preemptible)	0.884	0.757	0.491	0.278
preemption	0.009	0.000	0.002	0.006

"B-A" to describe co-scheduling of "A" as preemptible instances in addition to "B" as regular trace and "B"'s physical platform, which contains 7 nodes with 8 cores each. In both cases, we set the SLO to 0.01 preemption fraction and we compare the results of the SLA-aware scheduler ("sla") with the SLA-unaware baseline scheduler ("base").

The second combination investigates the co-scheduling of the more constant workloads "C" and "D" with larger platforms of 31 nodes each. The experiments are defined analogously to the first part and we refer to them as "C-D" and "D-C".

An important side-note is that A contains a number of instances requiring 12 cores each, while the platform of B only provides a maximum of 8 cores per node. This practically lowers the load impact of A as preemptible trace over its impact as regular trace on its native platform, as high-core-count instances are rejected by the scheduler due to the physical limits of the platform.

The results are summarized in Table 4.3 for the baseline, while the results for the SLA-aware scheduler are presented in Table 4.4. The SLA-aware scheduler meets the threshold, while the baseline scheduler misses in all cases. The modifications discussed in the previous section allow the SLA-aware scheduler to successfully handle production traces. The results are, however, close due to low overall utilization of the underlying cluster hardware. In fact, the mean utilization of regular and preemptible traces combined is 26.62 cores. This compares to a platform capacity of 84 cores for A and 56 cores for B. This degree of under-utilization is typical for clouds over-provisioned to meet peak demand. Reducing the under-utilization is a prime goal of co-scheduling. In order to demonstrate the efficacy of our approach in more resource constrained scenarios, we perform a platform down-scaling experiment in simulation in the next section.

4.4.4 SLA-Aware Co-Scheduling with Platform Scaling

In this section we stress-test our approach to computing the conditional quantiles for preemptible instance lifetimes in increasingly resource constrained environments. We use setups “A-B” and “C-D” again, but vary the size of the underlying platform from N to $N - 3$ nodes for “A-B” and from N to $N - 15$ in steps of 5 nodes for “C-D”. This corresponds to a reduction in node count by about half while the request rate stays constant and SLO target on preemption remains at 0.01. The inverse experiments “B-A” and “D-C” show similar results and are skipped for brevity.

Figures 4.8 and 4.9 show the results for scaled-down platforms of A-B and C-D, respectively. This experiment demonstrates the robustness of the approach in fulfilling its target SLO. While the baseline scheduler does not meet the SLO in any single case, the SLA-aware approach works consistently with visible differences in behavior.

Thus, while regular instance requests cannot completely be fulfilled in increasingly constrained environments, the regular rejection fractions for the production traces are small. In each figure, the column labeled N represents a replay of the production workload using the number of nodes and cores that were present when the trace was gathered (i.e. the production scenario). In the cases where our methodology offers an SLO on

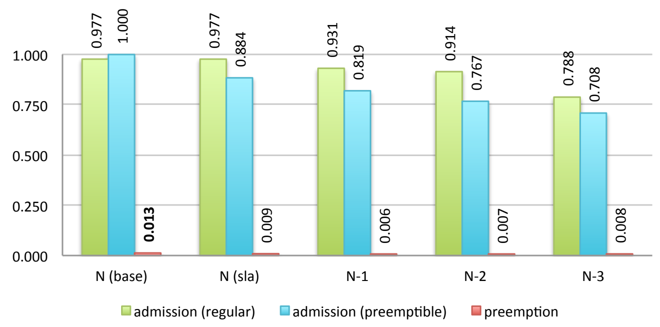


Figure 4.8: Admission and preemption fractions of regular and preemptible instances for A-B down-scaled. Non-SLA base, marked 'N (base)' for $N = 7$ nodes in the first column compared with 0.01 SLO with full and reduced node counts $N = [7, 6, 5, 4]$ in the other columns.

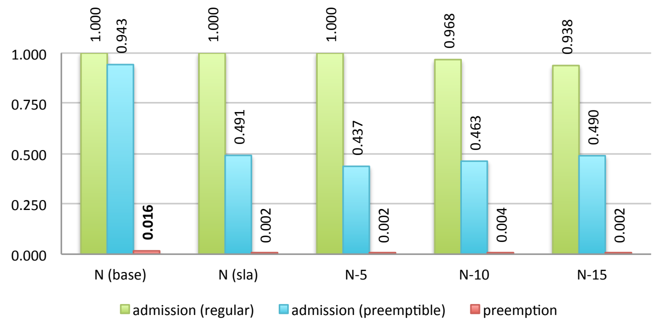


Figure 4.9: Admission and preemption fractions of regular and preemptible instances for C-D down-scaled. Non-SLA base, marked 'N (base)' for $N = 31$ nodes in the first column compared with 0.01 SLO with full and reduced node counts $N = [31, 26, 21, 16]$ in the other columns.

preemptible instance lifetime in the same environment, the fraction of admitted regular instances is equal to the baseline.

As a result, we conclude that the success of the predictions for the real-world production traces is not due to a lack of utilization (i.e. an abundance of extra capacity) in over provisioned production clouds. Shrinking these clouds does cause some of the observed production workload to be rejected, but the generated predictions of the time-to-preemption remain valid.

An additional observation is that for the down-scaling experiments the fraction of admitted preemptible instances may increase as the cluster size decreases (e.g. "C-D

$N - 3$"). An in-depth look at the simulation reveals that the rejected regular instances come in batches and with high core counts per instance. Their rejection due to capacity constraints leaves some additional capacity for preemptible instances. Furthermore, the inopportune placement of a preemptible instance does indeed lead to fragmentation and at times blocks the placement of large regular instances later on. While in our synthetic workloads the regular trace was completely unaffected by the preemptible trace, real-world traces are measurably (albeit minimally) impacted by the presence of preemptible instances.

4.5 Conclusion

In this chapter, we presented a novel approach to estimating the preemption probability of jobs with bounded duration. This allows us to provide ahead-of-time availability SLOs on preemptible instances by estimating their time-to-preemption given the current state of the cloud. We derive these state-dependent time-to-preemption estimates from simulations based on the cloud's historical utilization trace.

To overcome sample limitation, we leverage our previous work on validated simulation of IaaS clouds to perform a Monte-Carlo style simulation of preemptible instance behavior. We simulate how preemptible instances would have experienced execution under different load conditions in the past. By tabulating and extracting quantiles on the simulated lifetimes of preemptible instances, we are able to provide probabilistic bounds on the lifetime of such instances in the future.

We test the accuracy of our predictions by developing a cloud scheduler that uses time-to-preemption estimates to perform admission control of preemptible instances with bounded duration. For evaluation, we simulate the consolidation of different production workload traces onto a single cloud via co-scheduling of high-priority regular and low-priority preemptible instances. Admission control performs within expectations, increasing the utilization of the cloud while simultaneously maintaining a configurable upper bound on the preemption fraction of preemptible instances.

Besides for the improvement the utilization of a single cloud, ahead-of-time availability SLOs serve as a stepping stone for the implementation of cloud federation by enabling schedulers to provide immediate feedback about the expected quality of service for jobs executing on preemptible capacity.

Chapter 5

IaaS Cloud Federation Using Preemptible Resources

5.1 Introduction

This chapter builds on our work on validated simulation and the estimation of job preemption probabilities to investigate how a preemptible service class can serve as the basis for federating jobs across independent IaaS clouds. We use time-to-preemption estimates (c.f. Section 4.3) to provide probabilistic ahead-of-time availability guarantees while preserving the local cloud's ability to preempt federated workload at will.

Our goal is to provide additional SLA-covered capacity to clouds participating in a federation "for free". We cycle harvest idle capacity that would have gone to waste and, through preemption, do not require any kind of persistent resource commitments from clouds.

A complete solution to workload federation in the IaaS cloud context, however, must overcome additional practical challenges. First, the offloading of jobs between remote clouds incurs additional overheads for transfer input and output data not resident in the remote cloud. Additionally, startup delays for instance launch may be different from the original cloud as well. Second, execution times of the compute workload itself may vary across different platforms and thus, estimates for job execution duration by the user must be robust to estimation error. Third, cloud federation should increase overall efficiency with the number of clouds participating in a federation. A solution must further make joining a federation easy and minimize risks associated with required changes to existing infrastructure. Finally, federation capabilities add complexity to existing infrastructure that must be justified with a measurable increase in system efficiency under real-world conditions.

We develop and implement an architecture for workload federation of batch jobs across independent IaaS clouds that overcomes these challenges. Our prototype can be evaluated end-to-end on "live" cloud deployments with workload traces recorded from production systems. A "good" solution that ties together our previous work into a complete solution to the IaaS cloud federation problem must fulfill the following requirements:

- Local control must supersede remote control. Cloud infrastructure is costly and runs mission critical services. Any federation member must be able to preempt at will federated jobs running on local resources to retain its independence. The absence of local control can harm cloud providers and discourage them from joining a federation in the first place.

- Availability guarantees must be made ahead of time. The cloud's SLA allows users to reason about the expected quality of service before they deploy an application or send a request. While open-ended guarantees about resource availability are at odds with the potential need to preempt remote workload eventually, jobs with bounded duration should receive guarantees. Additionally, such guarantees must adapt to changes in system (and user) behavior over time.

- The solution must avoid a single point of failure. The dependency on centralized infrastructure threatens the reliability of the federation as a whole. Even without failure, a centralized point of contention limits severely the scalability of the architecture. The resulting distributed control over resources, especially for scheduling and resource allocation, creates the potential for contention via concurrent requests for the same resources which must be addressed.

- The solution must integrate with existing cloud infrastructure. A practical implementation must operate at the level of production quality IaaS frameworks. Modifications to the internal structure of these frameworks, even if theoretically possible in open-source, can impact the reliability of the entire system and requires extensive testing and quality assurance efforts that should be avoided.

- The solution must be complete end-to-end. In addition to implementing scheduling control and SLO enforcement, a complete solution must include facilities to monitor relevant cloud behavior, transfer data dependencies of federated jobs, and offer robustness to job duration estimation errors.

- Workload federation must improve the utilization of clouds in a federation under real-world conditions. The added complexity of federation capabilities must be justified via clear evidence of a benefit to each federation member. This requires end-to-end evaluation of the cloud federation system with workloads recorded from production systems.

Computational grids owe a major part of their success to the primary design principle of leaving the resource owner in control at all times. We adopt this design choice for our cloud federation architecture and strictly prioritize local workload over federated workload. We implement this prioritization by executing federated jobs exclusively on preemptible capacity. Furthermore, every cloud makes admission decisions locally while users are responsible for coordination across clouds, if desired.

We make availability guarantees about preemptible resources for jobs with bounded duration by leveraging our work on validated simulation and the prediction of job time-to-preemption. These guarantees are enforced at runtime through careful admission control, by comparing jobs' predicted time-to-preemption to a user-provided job duration estimate. As a consequence, our approach to workload federation dynamically adapts to the specific user request, the historic utilization pattern of each individual cloud, and the current state of the destination cloud.

Specifically, our method provides an upper bound on the preemption fraction (the fraction of preempted instances relative to accepted instance requests) for federated workload with a user-specified execution time bound (duration). Consequently 1.0 minus this preemption fraction serves as a proxy for the probabilistic "guarantee" that a user's job will execute for the requested duration once accepted by the cloud. The cloud will not accept a request (i.e. the request will "fail fast") if the total lifetime (the time a job is active and preemptible) cannot be assured probabilistically. Further, cloud administrators can set the target fraction (i.e. the maximum probability of a preemption) for the preemptible service class so that users can reason about the use of this service class based on its SLO.

Our architecture avoids a centralized controller or broker that represents as single point of failure. We bundle instances and data transfers into a single request on a per job basis. In our architecture, each IaaS cloud within a federation is considered to be a single entity (multiple availability zones would be considered separate clouds). Preemptible instances related to a single job are requested in a single, atomic transaction and come with a user-defined estimate about the maximum job duration and the minimal acceptable availability (the probability the job will execute for at least this duration). Finally, users seeking federated capacity use a randomization approach in determining the federation target to prevent crowding.

We implement our prototype on top on the Eucalyptus IaaS framework by introducing an additional abstraction layer to the server API and client-side tools. We take advantage of our work in on generating time-to-preemption estimates and extend it here to account for additional overheads. Our approach avoids any modifications to Eucalyptus' API or internal code, thus keeping in tact the framework's production-quality reliability properties. The management of resources is still performed by Eucalyptus and its tool chain, while users can take advantage of the additional federation capabilities through the new interface. While our prototype builds on Eucalyptus, the techniques layed out in this chapter are general and can be applied to (and across) other IaaS frameworks as well.

While we aim to maximize the transparency of the federation mechanism, it still introduces API extensions and requires some additional inputs from users who want to take advantage of federation capabilities. Our prediction mechanism requires an a-priori estimate of the maximum job duration (more specifically, the desired duration of SLO-coverage) if the user wants to use preemptible capacity with availability guarantees. We automate the remaining federation process in out tool chain and automatically adapt the user estimate to consider site-specific overheads. Additionally, our prototype transfers specified job input data to the federation target and transfers outputs (if any) back to the source cloud.

We evaluate our cloud federation prototype end-to-end via faster-than-realtime and smaller-than-realworld replay of several computationally intensive and data analytics workload traces recorded from production systems. We deploy the system across multiple geographically distributed sites and analyze its performance and reliability properties.

In summary, to make a preemptible service class practically useful as a cloud federation technique, our method

- prioritizes local control over remote control by federating jobs on preemptible capacity exclusively,
- provides an availability SLO on preemptible resources within a bounded time window,
- relies on de-centralized scheduling and resource allocation,
- does not interfere with the tested, internal structure of existing open-source IaaS frameworks, and
- is complete and testable end-to-end in live cloud deployments,

Our evaluation shows that our approach to IaaS cloud federation is able to maintain a probabilistic SLO on the preemption fraction of federated instances across a broad spectrum of workloads even under tight capacity constraints. We show that our approach scales with additional capacity, adapts to changes in cloud configurations, and is robust to seasonality in workload and inaccuracies in user-specified job durations.

5.2 Related Work

Celesti et al. [108] make a distinction between two types of cloud federation architectures: "inter-cloud" federation that moves jobs between vertically integrated clouds and "cross-cloud" federation that allows service-granularity mashups between different clouds. Grozev and Buyya [109] survey a corpus of work in the "inter-cloud" context.

InterCloud [110] proposes an architecture for cloud federation using a network of brokers on an exchange that schedules jobs across different clouds based on QoS constraints. The design is evaluated in simulation trading-off cost and runtime when executing jobs across private and public clouds. The authors of [111] hypothesize a transition from today's "monolithic" cloud architectures to a "horizontal" federation model and design a step-by-step process for federating resources between clouds. The authors of [112] study a layered composition of cloud services for a specific use-case that enables inter-operation of multiple clouds executing a single applications. [113] is a representative for studies of cost-models trading off between executing workloads with local resources, out-sourcing to other providers and in-sourcing.

Some contemporary inter-cloud federation designs rely on centralized "brokers" to facilitate communication between providers and consumers and optimize various performance metrics. Azam et al. [114] use a broker to match customers and providers and predict resource usage for cost optimization. Similarly, [115, 116, 117, 118, 84, 119] use brokers to optimize cost by trading off between reserved and on-demand capacity based on expected future demand. Yao et al. [120] explore broker-based cost optimization for batch jobs with completion deadlines.

Several works implement software frameworks to connect real-world clouds and address challenges that an end-to-end implementation of cloud federation faces. Simarro et al. [121] develop a broker that schedules resources across different real-world providers based on dynamic pricing schemes. Superclouds [122] use nested paravirtualiztion in Xen to provide a consistent foundation for VM hosting across multiple providers. RESERVOIR [123] uses a peer-to-peer approach to negotiating resources in a federation of clouds at job-level granularity. Platforms such as Apcera [124] and the recently acquired ClusterK [125] facilitate the deployment of applications across multiple clouds and preemptible service classes via cost-aware scheduling and autonomous handling of instance preemption.

Our approach follows the "inter-cloud" federation design with job-level granularity. In contrast to existing work, our approach allows the executing cloud to remain in control of its own resources at all times (i.e. no open-ended guarantees), while still giving users a probabilistic guarantee on job completion probability. Guarantees are made ahead-of-time, based on a user-specified maximum job execution time. Thus, we primarily aim to ensure that guarantees on the job preemption fraction are met consistently for a variety of workloads, rather than to optimize a monetary "profit" function. We further use a de-centralized, user-focused approach to federation similar to Condor's "direct flocking". Finally, we evaluate our approach with a "live" implementation replaying scaled-down production workloads on real cloud systems.

5.3 Methodology

To represent the combination of computing and storage requirements in a workload, we define a *job* as a homogeneous group of instances and an associated data set in the cloud's object store. The goal, then, is to *predict* whether *each* job that is launched in the preemptible service class will

- transfer its inputs to the target execution site,
- execute for a duration specified with the job, and
- transfer any results back to its originating site

before locally generated work at the target site preempts it. The method must ensure that the fraction of incorrect predictions (i.e. the prediction error) is below a threshold set by the cloud administrators for the service class.

We define the preemption fraction of federated jobs as $\frac{P}{A}$, where A is the number of jobs submitted and admitted to a remote cloud for federation, and P is the number of jobs in A that are preempted (terminated) by the remote cloud before completion. Thus, the quantity $1.0 - P$ serves as the "success probability" associated with a the execution of a federated jobs. Note that in contrast to existing systems implementing preemption [24, 126], our model allows a user to reason about the how long *each* instance will execute before it can be preempted (with a specific probability estimate). Thus, the user's trade-off between preemptible and non-preemptible service tiers is quantifiable ahead-of-time.

We further assume that all clouds in a federation may request to federate work to other clouds and agree to accept federated work themselves. That is, there are no "free riders". Finally, we assume that each cloud serves as the "primary" cloud for some set of users and that cloud receiving a "native" job request from its own users will attempt to satisfy it locally, before forwarding to it a remote cloud for federated execution. Any job accepted from a "foreign" remote cloud (under the constraints of the admission control mechanism) can be preempted by a new local request for instances if there is insufficient capacity available to satisfy the locally generated request.

To enable ahead-of-time certainty for federated jobs, we introduce admission control that employs a predictive model for deciding whether to accept a request for a preemptible job (a group of instances and data). Admission control is tasked with accepting or rejecting incoming native and foreign job requests based on available capacity without queuing. Native jobs must be accepted as long as (a) sufficient spare capacity is available or (b) local capacity can be made available by terminating federated jobs. Federated jobs are admitted only if sufficient spare capacity is available (a) to fit the requested job and (b) to guarantee probabilistically that the remaining spare capacity is sufficient to absorb future native requests without triggering preemption. We assume that each foreign job requires inputs from the originating cloud's object store and that outputs from the job must be returned to the originating cloud. Further, we assume that there is sufficient storage in the object store of the cloud accepting a federated job to hold that jobs' inputs and outputs temporarily. As a consequence, preemptions are only triggered due to instance capacity constraints, not storage shortfall.

Key to our approach is that federated job requests come with a user-specified minimum *duration* for the job's computational needs until completion. A cloud only accepts a federated job (admits a preemptible job) if its lifetime (the job's duration plus system-specific launch overheads) is shorter than the lower bound estimate of the job's time-to-preemption, subject to a confidence level defined in the cloud's SLO. SLOs in our system are defined upon job submission and are immutable for the entire execution of the job.

Our implementation of this model system uploads input data to the object store of the remote cloud prior to job execution. It downloads results (output data), if any, upon job completion and deletes the inputs and outputs once they have been successfully transferred back to the requesting site. Our model accounts for predicted data transfer and management times in its estimates of the actual amount of time a job spends in the system and remains preemptible – which we refer to as the job's total "lifetime". As such, users need only specify the minimum duration that suits their applications' computational needs, while the system accounts for overheads internally.

5.3.1 Federation Architecture

In our architecture, a cloud federation consists of one or more IaaS clouds (or cloud availability zones). Each cloud operates an additional Web-API that can be accessed by clients wishing to take advantage of federation capabilities. Clouds within the federation allow remote access to their object store to enable data transfer in and out for federated jobs. Additionally, all clouds are assumed to provide compatible instance images and sizes that fulfill a pre-agreed set of minimum requirements. These images and instance sizes are defined and named per federation and may translate to different (but compatible) implementations within each participating clouds. That is, a "federation" consists of a standardized API for job submission with SLO requirements and a collection of compatible API endpoints and mappings for instance types and images.

Users obtain credentials from each cloud individually and manage their own configuration, similar to “direct flocking” in Condor [32]. A federation’s “config file” is distributed to users and contains information about the endpoint URLs for each cloud, the image types, and the standardized sizes. For example, the cloud administrators within the federation may agree on providing a “Hadoop-2.6” image that provides Apache Hadoop version 2.6.x on CentOS 7. Equally, instance sizes for federation-enabled jobs must agree on minimum specifications. For example, a “large” instance may be required to have at least 8 cores, 4 GB of RAM, and 50 GB of local storage – the rough equivalent of the default “m2.4xlarge” in Eucalyptus 4.2. Users later augment this configuration with their cloud-specific credentials and can expand “their” federation with further endpoints and mappings, if so desired.

Depending on the users’ affiliation (or negotiation abilities) their credentials are assigned either “high-priority” or “low-priority” access on a per cloud basis. High-priority requests execute jobs on regular instances and have the ability to preempt running instances from low-priority users should capacity run out. Low-priority requests rely on preemptible instances and are subject to admission control based on the user-specified SLO target. Typically, users are expected to be associated with exactly one “primary” high-priority cloud within their own organization while using low-priority requests on all other clouds. Cloud administrators can safely give low-priority access to users from external organizations as their resources can be reclaimed transparently by local users via preemption.

Our federation mechanism safely “bursts” load to remote destinations when local capacity runs out, rather than performing load balancing or cost optimization between individual clouds. When a user submits a federated job, we first route the request to the user's primary cloud before reaching out to other clouds within the federation. If the cloud has sufficient capacity – or can make sufficient capacity available by preempting low-priority jobs – the job launches and executes on the primary cloud. If there is insufficient capacity (or the user voluntarily uses a low-priority request) we send the request to another randomly selected cloud in the federation. If the remote cloud can provide the requested capacity subject to the minimum availability constraints, it accepts and launches the job on preemptible capacity. If the remote cloud rejects the request, we forward the request to the next candidate cloud. Should all clouds reject the request, we inform the user who may then decide to modify the request or wait for capacity to free up. This randomized approach consistently guarantees the requested quality of service while avoiding the need for centralized coordination.

5.3.2 Instance Lifetime Guarantees

When a user makes a request for a preemptible job, given the current state of the potential target cloud, the admission control algorithm must predict the minimum time until *native* workload on that cloud will cause a resource shortfall *if* the requested job is admitted.

The system predicts bounds (described below) on possible future required capacity for native jobs and uses this information to create a schedule for possible increases in non-preemptible utilization in the near future, starting from the current utilization level. It augments this schedule with load decreases from the expected completion of foreign jobs, based on these jobs' remaining lifetimes. Admission control admits a new foreign job only if the predicted total utilization of the cloud does not exceed the available capacity for the entire projected lifetime of the requested job.

Note that the admission of additional preemptible jobs does not affect the time-to-preemption of active preemptible jobs, as we use a "Youngest-Job-First" preemption policy (c.f. Section 4.3.2). All preemptible jobs active *before* the admission of a new preemptible job are relatively higher priority and will only be preempted *after* the new job has been preempted. That is, as a federated job spends more time executing on a cloud and additional preemptible jobs are admitted, the original job becomes less likely to be preempted *in the event* preemption occurs on the cloud.

To account for job startup and teardown (instance starts and terminations, job input and output data migration), admission control continuously monitors and tracks the histories of hypervisor overheads and transfer bandwidths between clouds in a federation. Thus the methodology requires a "training period" during which gathers information to make its initial estimates.

In summary, to construct a schedule, we must

- estimate possible increases in the non-preemptible utilization in the near future,
- estimate the overhead associated with transferring a job to the cloud where it will be executed,
- combine these estimates into a joint estimate of the lower bound on the time available to each job before it could be preempted, where the estimate error is less than or equal to the SLO target probability, and
- solve algorithmically a dynamic capacity planning problem using these estimates starting from the present state of the cloud.

If no schedule can be generated, the federation request is rejected, but no terminations of existing jobs are triggered. Thus only the arrival of new native requests can preempt foreign jobs if the cloud is at full capacity. We detail this methodology in the following subsections.

5.3.3 Estimating Native Utilization

The intuition behind our admission control mechanism is that the job preemption probability depends on the current load level of the cloud as well as predictable changes of the load level in the near future. For example, when the cloud's load is near capacity, new native workload is more likely to cause a preemption of foreign workload than if the cloud is relatively under utilized. Notice that it is only the *arrival* of new native instances that can trigger the termination of preemptible instances – admission control ensures that new foreign instances will be rejected if there is no spare capacity to host them. Moreover, it is an increase in native load (and not a decrease) that causes preemption.

As such, the admission control algorithm considers a new federated job request, it requires a prediction of the time until spare capacity is exhausted *and* at least one additional native instance is requested, i.e. a preemption is triggered. For example, if the a federated request is for 10 instances, the admission control algorithm requires the time until the cloud only has 9 available slots (or fewer) remaining for run instances on. We refer to this time estimate as the "time-to-preemption" (c.f. Section 4.3).

However, for scalability reasons, rather than making this estimate on a per-request basis, our system continuously computes the time until there will be a capacity shortfall for different possible federated job sizes. To do so, we sample the history of total capacity utilization of native instances at regular intervals. From each sample, we trace forward to identify the point at which the aggregate utilization of non-preemptible instances (native instance starts without compensating native instance terminations) increases by a fixed-size step (e.g. one instance slot). We repeat this tracing procedure for each possible magnitude of load change (two slots, three slots, and so on) and tabulate the results as a set of empirical distributions. This database of predictions is constantly updated but queried asynchronously by the admission control system. When the admission control system considers a new request for a federated job, it computes the time-to-preemption by retrieving the distribution corresponding to the load increase that is equal to the current level of available spare capacity (including the new job) plus one (the hypothetical native instance triggering preemption) from the table and uses a quantile corresponding to the SLO from this distribution as the estimate.

For example, at the time of a request, there are 10 available slots to run native instances, and a federated request requires 2 instances, the admission control system will extract the size distribution of previous time windows corresponding to increase in native workload of $10 - 2 + 1 = 9$ instances. If the target bound on job preemption probability is 0.05, then it will use the 0.05 quantile from this empirical distribution as an estimate of the *least* amount time until the newly added federated job might be preempted.

To manage seasonality (which we observe in numerous production IaaS workload traces), we assume (based on our experience with these traces) that different time periods during the day experience varying arrival rates of native requests and thus a different time-to-preemption for federated jobs. Our framework spreads time-to-preemption samples from historical tracing across different time-of-day bins, e.g. 2-hour bins such as 8:00 to 10:00. When making predictions for a newly arriving request, we serve a distribution of only those time-to-preemption samples specific to the request's time-of-day bin. Bin size can be set by cloud administrators and we find that a window of 1 to 3 hours is sufficient for the workloads we have considered.

5.3.4 Estimating Federation Overheads

To arrive at an effective job "lifetime" estimate, the overhead associated with launching a job on a remote cloud must be considered. This overhead is added to the minimum duration specified in the federated job request to determine if the request can be satisfied.

Job execution on a remote cloud can only start when instances are fully setup up and all required input data has been transferred to the executing cloud's object store. Note that with our federation design instance setup and in-bound data transfer occur in parallel, as do instance teardown and out-bound transfer. As a consequence, job preemption is possible during a job's setup phase, but not during teardown, as the job does not retain any active instance capacity.

Federation overheads consist of two components: system-specific instance startup time and the time required to transfer inputs. We record the instance startup delays for each cloud (available from the cloud logs) as empirical distribution which facilitates our extraction of arbitrary quantiles. We can refine further the method to account for instance attributes (i.e. not all instances may incur the same startup delay); we do not do so in this paper but will consider doing so as part of future work. Second, we account for data transfer times between two IaaS clouds in the federation case. We monitor aggregate data transfer bandwidth between source and destination clouds over time and store them similarly as an empirical distribution.

When combining the user-specified job duration with overhead estimates and comparing their sum to the estimated time-to-preemption, the probability that the comparison is incorrect must be less than or equal to the SLO preemption probability. We treat the distribution of overheads for instance startup and teardown as being independent of the distribution of time-to-preemption. As such, we choose the quantile to use as a bound on the overheads and the quantile to use as a bound on the time-to-preemption so that their product is less than or equal to the SLO preemption probability.

We address seasonality in overheads by binning similar to our time-to-preemption distributions. Such time slicing however can introduce error in our estimates for bins that contain differing numbers of samples. This makes quantile estimates from empirical distributions unreliable, especially in the tails which are critical to make SLO guarantees.

We address this issue by using the Binomial distribution to construct 95% confidence intervals around the quantiles as done by QBETS, a queue-bound predictor for time series [60]. Moreover, we test the impact of such binning in the empirical evaluation of our prototype.

5.3.5 Admission Control

For every admission decision, admission control solves a capacity planning problem using bounds estimates of native load increases and expected foreign job completions (the latter coming from minimum duration specifications accompanying each foreign job). It generates a prediction of the available spare capacity over time (in the near future) that takes into account the remaining lifetimes of the current foreign job set and makes admission decisions based on whether new requested foreign instances will "fit" within the available capacity for their entire lifetime.

Intuitively, the admission control system runs a discrete-event simulation in which there are two types of events: the termination of a foreign job, and the increase in occupancy of the cloud by native workload. The simulation runs at the time a new foreign job is to be considered for admission. The system discretizes cloud into "slots" – units of cloud allocation – that each job occupies.

For example, often administrators configure each node in a private cloud to host, at most, as many virtual machines as it has cores, with each virtual machine occupying a single core. In such an example, the cloud would have as many slots as it has cores it can devote to virtual machines. Further, in this example, the number of cores used by a job defines the number of slots it requires.

Note, that in contrast to to our generic implementation for estimating the time-to-preemption in Chapter 4 we only allow a single instance type (slot size) in this implementation. The admission control algorithm below can be extended to support multiple types, but as a consequence requires explicit scheduling control over the underlying cloud framework. With multiple instance sizes, we need control over instance placement and preemption to make accurate predictions about the the consequences of an admission decision. This is related to the asymmetric nesting behavior of instances on nodes. For example, assuming a cloud with two nodes – one with 2 free cores and another with 4 free cores – admits a 1-core instance, the future availability of a 4-core slot then depends on the 1-core instance's placement. If the instance launches on the first node, a 4-core instance can still be placed on the second node. If the 1-core instance launches on the second node, no space remains for a 4-core instance. Thus, if a native 4-core request arrives, the probability of triggering a preemption on the cloud depends on the placement decision made before. Thus, the placement policy affects the preemption probability of instances and must be known to the simulation.

Listing 5.1: Admission control algorithm for enforcing an 0.05 upper bound SLO on the preemption fraction of federated jobs.

```
// Configuration and user request, example
$p_{sla} \leftarrow 0.05$
$capacity \leftarrow 32$
$requested \leftarrow \{('m1.medium', 3600, ...)\}$

// Choose any $p_{pre}$, such that $p_{pre} * p_{dur} \geq p_{sla}$
$p_{pre} \leftarrow sqrt(p_{sla})$
$p_{dur} \leftarrow p_{sla}/p_{pre}$

$fed \leftarrow get\_preemptible\_instances() \cup requested$
$native \leftarrow get\_nonpreemptible\_instances()$

$Q_{fed} \leftarrow \{est\_time\_remaining(s, 1 - p_{dur}) | s \in fed\}$

$remaining \leftarrow |fed|$
$for(t_{dur} \leftarrow sort(Q_{fed}))\{$
    $c_{spare} \leftarrow capacity - |native| - remaining$
    $t_{pre} \leftarrow est\_time\_to\_increase(c_{spare} + 1, p_{pre})$

    $if(t_{pre} < t_{dur})$
        $REJECT$

    $remaining \leftarrow remaining - 1$
$\}$

$ADMIT$
```

The number of slots in a cloud is determined by how the administrators choose to define its virtual machine types in terms of core count, memory occupancy, and local disk allocation. However these values are fixed when the cloud is configured and published to the user community. Our admission control system uses this information to determine how many slots a cloud can support.

When the system considers admitting a new foreign job, it first computes the number of slots that are occupied by the current set of foreign jobs running in the cloud, and the number of slots that are occupied by native workload. If there are enough free slots to host the new foreign job, it then, hypothetically, adds the job to the set of foreign jobs in the cloud (otherwise the job is rejected).

We next predict a lower bound on the time until the number of unused slots (+1) will be consumed by native work with probability 1.0 minus the SLO preemption probability (e.g. 0.95 for a preemption probability of 0.05). If the new foreign job's lifetime (i.e. its duration plus startup and teardown overheads) is greater than or equal to this bound, the job is rejected.

Otherwise the system sorts the foreign jobs by their termination times (which were computed when the jobs were admitted) and "rolls" time forward from one job completion to the next. Each time a foreign job terminates the admission control system again predicts the bound on the time until native workload will exhaust available slot capacity. If the remaining lifetime of the new job, at each of these termination points, is greater than the prediction, the new job is rejected. If there is no point in time at which the system predicts that native workload will exhaust capacity (and thereby cause a preemption) between the time the new foreign job is submitted and the end of its lifetime, the new foreign job is admitted. Because we estimate load changes and their time-to-preemption using lower bound quantiles (determined by the SLO), admission control decisions will be probabilistically correct if the estimation mechanism is robust.

Listing 5.1 shows the capacity planning algorithm as pseudocode for an SLO probability of 0.05, a capacity of 32 slots, and a requested *m1.medium* foreign job requiring a single slot over a 3600 seconds duration.

Lines 1 through 13 initialize the discrete event simulation by computing the current "state" of the cloud in terms of the free and available slot capacity. Lines 16 through 23 are the main simulation loop in which the state of the cloud is reconsidered when each federated foreign job terminates. The function *est_time_to_increase* invoked on line 18 returns the lower bound on the time-until-preemption with probability p_{pre}. Because the foreign job lifetime is determined by the job's duration and its startup and teardown overheads (each of which must be estimated probabilistically) p_{pre} is set to the square root of the SLO probability. Similarly, the probability associated with the startup and teardown overhead (p_{dur} in the listing) is also set to the square root of the SLO probability. We assume that the overhead distributions and the time-to-preemption distributions are independent so that the product of p_{pre} and p_{dur} equals the SLO probability (p_{sla} in the listing).

Note that the simulation makes a single pass through all federated jobs in sorted order. Since the system computes the bounds predictions asynchronously, the time to furnish an estimate to the admission control algorithm is constant. Thus the complexity of the admission control algorithm is $O(nlogn)$ where n is the number of admitted foreign jobs.

5.3.6 Evaluation Traces

Our evaluation uses three groups of anonymized, real-world traces recorded from production systems. Each represents two separate sub-traces with jobs recorded from two "big data" clusters. They span multiple months in real time and contain mixed

Table 5.1: Production traces. Types are batch (B), batch pipelined (P), batch seasonal (S), long-running service (L), and mixed (M).

segment	num reqs	num insts	total data (TB)	trace dur (wks)	average inst dur (hours)	work type
DS-A 1	2.3k	3.0k	1100	2	2.2	B, S
DS-A 2	1.5k	2.0k	50	2	2.9	B, S
DS-B 1	1.3k	4.2k	80	12	2.2	B, P
DS-B 2	1.5k	2.4k	180	12	9.3	M
DS-C 1	1.2k	2.8k	–	30	10.0	M
DS-C 2	0.5k	1.9k	–	30	45.5	B, L

workloads from multiple cloud users and software frameworks. For our replay we extract long-running jobs over 1 hour in real-time duration, scale them to fit our testbed and use separate parts of the traces for model training and evaluation. We summarize these traces in Table 5.1. We set up our experiments with two clouds which are running a native workload each and, in case of overload, attempt to offload rejected native jobs to the other cloud as preemptible jobs with a 0.05 upper bound SLO on preemption probability (i.e. a 95% success guarantee).

Data set A (DS-A) covers a 5-month period and multiple Hadoop clusters used by a medium-size Internet business. For our evaluation, we extract workloads from two large clusters with a strong seasonal workload patterns as visualized in Figures 5.1 and 5.2. The source clusters typically execute jobs with hundreds of concurrent tasks. We extract a 2-week period for training and a consecutive 2-week period for evaluation, using a speedup of $50x$ for a total runtime per experiment of approximately 7 hours.

DS-B occurred over a 7-month period and includes multiple big data frameworks running on Apache YARN [36] on multiple clusters of another medium-sized Internet business. Part of the workload arrives in bursts as the clusters are used for interactive data analysis, while another part of the workload stems from periodically scheduled tasks. We use a replay speedup of $50x$ for a total experiment runtime of 39 hours.

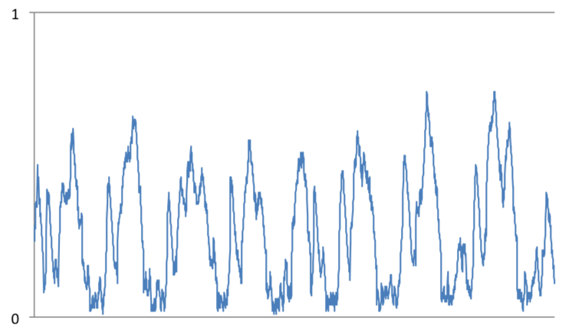


Figure 5.1: Partial visualization of segment DS-A cluster 1 showing strong seasonality in utilization over time.

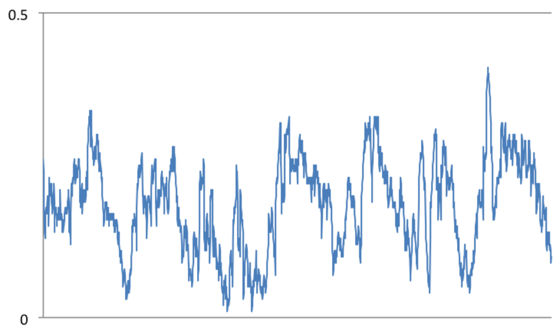


Figure 5.2: Partial visualization of segment DS-A cluster 2 showing erratic utilization with some seasonality.

DS-C covers a 9-month workload recorded from IaaS clouds used mainly for high performance computing tasks. The majority of requests arrive in irregular bursts, while part of the capacity is used for long-running services. This data set is extracted from a larger collection of publicly available private IaaS traces [76]. We replicate the original size of jobs in terms of core counts, but the jobs' data sizes were unavailable. We use a constant data size of 100 MB per job to simulate load on the storage backend and use a replay speedup of $100x$ for a total experiment runtime of 67 hours.

For DS-A and DS-B data sets, we scale down the size of the data inputs and outputs for each job by the same factor that we use for execution times. That is, we assume that the size of the data is proportional to the execution time for jobs executed by these big data frameworks. In our experiments, we transfer the data across the wide area between different cloud sites (e.g. UC Santa Barbara and CloudLab Utah) for federated jobs only. Thus, the total amount of data that we transfer over the Internet depends upon the speed up factor and the realized job federation fraction.

During our development, DS-C was available for testing and should be considered in-sample (train-train). Part of DS-A was available during development as well, while the evaluation uses a more recent trace segment that was unavailable during our development.

DS-B was available only after we completed the development of our prototype and thus represents a true out-of-sample test (train-test).

5.3.7 Evaluation Metrics

We consider three metrics to evaluate our method: the preemption fraction of federated jobs, the rejection fraction of requests, and the federation fraction of work. Preemption fraction measures the effectiveness of SLO enforcement:

$$preemption = \frac{|federated\_jobs\_preempted|}{|federated\_jobs\_accepted|}$$

This metric captures the fraction of pre-mature terminations of federated jobs (groups of instances) that have been accepted by the admission controller and launched in the system. It is intended to illustrate the degree to which the estimation methods and asynchronous capacity planning implementation is able to correctly implement the SLO.

Rejection fraction measures the efficacy of federation in accepting incoming user requests:

$$rejection = \frac{|jobs\_rejected|}{|jobs\_requested|}$$

A rejected request has been rejected by the local cloud due to a lack of capacity and by the remote clouds in the federation due to the inability to guarantee the requested SLO preemption fraction. We only count jobs that are rejected first as native requests and subsequently as federated requests once.

This metric is intended to reflect the user experience with respect to rejected workload. When an unfederated cloud runs short of capacity, new user requests must be rejected. Federation should enable lower rejection rates by allowing some of the otherwise rejected workload to be run as federated workload thus improving user productivity.

The federation fraction represents the overall mutual gain from federation and measures the amount of work that results from access to remote capacity (completed jobs on remote, preemptible instances):

$$federation = \frac{work(federated\_jobs\_completed)}{work(jobs\_completed)}$$

The fraction uses the aggregate time spent by instances of successfully completed jobs. It divides the aggregate time of completed federated jobs by the aggregate time of all completed jobs (including federated jobs). This time includes the amount of time spent to set up instances, wait for data transfers, execute the actual computation tasks, and tear down the instances. Time spent by preempted jobs is not included.

5.4 Results

Our evaluation attempts to answer three primary questions about this new approach to cloud federation, which provides an ahead-of-time guarantee on the preemption probability of federated jobs executed on preemptible resources:

- Is it feasible to enforce an upper bound SLO on the preemption fraction of federated jobs with real-world workloads?
- Does the method scale with additional capacity to a federation of clouds while maintaining its guarantees?
- How sensitive is the method to inaccuracies and changes in inputs and how are guarantees affected?

To answer these questions, we emulate various federation settings via faster-than-realtime and smaller-than-production replay of workloads recorded from production clouds, using private cloud deployments equipped with our federation extensions. We choose this methodology over a purely simulated approach so that the experiments take into account the overheads (modeled and unmodeled) of working clouds. That is, while the workload is a replay of production workloads (sped up from their original durations), the systems are real.

In the experiments that follow, we implement a “burst capacity” model as described in Section 5.3.1. We assume that in response to a rejection from the primary cloud the user will forward the job immediately to another cloud and tacitly accept the preemption probability associated with federated execution which is set to 0.05 in all experiments. Note, that the experiments also require that the user also knows the maximum duration of the job ahead of time and provides this data with her request. We test the impact of inaccurate lifetime estimates on the SLO enforcement in Subsection 5.4.5.

Our traces include a mix of workloads from big data analytics applications (mostly execution traces from Apache Hadoop [34] ecosystems) and high-performance computing tasks, which we obtained from industry collaborators (c.f. Section 5.3.6). Our replay scales the duration, instance capacity and data size proportionally to fit our prototype cloud testbed. The emulation includes the launch and termination of actual cloud instances and the transfer of data to and from the cloud’s object store. It does not, however, execute original user code or access original files as this information has been purged from the traces for privacy reasons. Instead, we transfer similarly sized files with random data and “execute” jobs by launching instances and waiting for their recorded time to pass.

We have designed admission control as an independent component that can be used as part of any IaaS system that exports the measurement in information necessary to form the empirical distributions that the methodology requires. To test it, we implement our admission control in Python 2.7 and integrate it with the open-source Eucalyptus IaaS framework [78] version 4.1. Our extension complements existing API functionality on both the server and client side and extracts monitoring data from the cloud system logs. We install the system from repository packages on CentOS 6.7 and rely on Eucalyptus' default access control to manage user access rights. Due to the long-running nature of the experiments we concurrently use several different clusters located at CloudLab [5] APT Utah, CloudLab Clemson, and UC Santa Barbara with 4-8 physical hosts each.

5.4.1 Federation Baseline

Figure 5.3 shows six *baseline* experiments, replaying our three production data sets on two different cloud configurations each – *without* SLA-enabled admission control (i.e. without prediction or capacity planning). In these experiments, admission of federated jobs is based purely on available spare capacity (i.e., all foreign job requests are accepted if, at the time of their arrival, there is sufficient capacity to host them). Each group of columns on the x-axis represents one experiment, executing the sub-traces of a data set concurrently on two clouds linked via two-way federation. The category names describe the experimental setup for each data set, e.g. "DS-A", and the size of the clouds in the federation, e.g. "32-32" which indicates that each cloud employs 32 cores each.

We replay the three production data sets on two different cloud configurations. The first constrains resources for the workload while the second provides some additional total capacity. For DS-A we use 32-32 for the constrained scenario and 48-48 for the less constrained scenario. For DS-B we use 16-16 and 24-24, and for DS-C we use 24-24 and 32-32 for each respective scenario. On the y-axis we plot our three performance metrics: preemption fraction of federated requests(red), the rejection fraction of requests overall (blue), and the fraction of successfully federated (additional completed) work (green).

The figure shows how the relationship between workload and capacity impact each metric. DS-A shows a high preemption fraction (near 0.4) in the constrained scenario

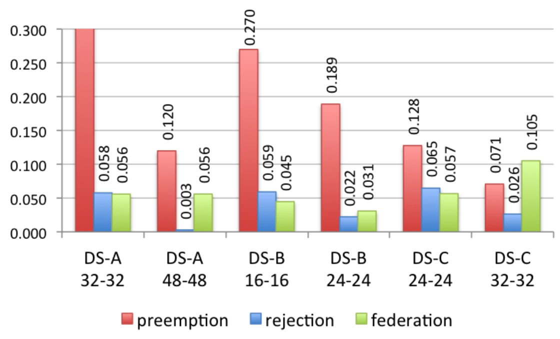


Figure 5.3: Federation between two capacity-constrained clouds without SLA enforcement for 3 production traces on different configurations. Federation enables more jobs at the price of frequent preemption.

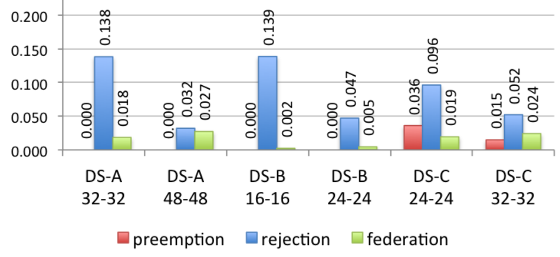


Figure 5.4: Federation between two capacity-constrained clouds with SLA enforcement enabled. Admission control rejects some jobs to provide a guarantee on preemption fraction, but still allows federation.

(32-32) while adding only 5% additional work as a result of federation (this work without federation would have been rejected). DS-A on the 48-48 configuration reduces preemption fraction by a factor of three, down to 0.12, and the rejection fraction to near zero. The fraction of successfully federated work remains the same as in the constrained scenario, as the additional capacity is used proportionally for native and foreign workload. DS-B has a preemption fraction of 0.27 in the constrained and 0.19 in the less constrained configuration. DS-C starts out with a 0.12 preemption fraction, which decreases to 0.07 with additional capacity.

The degree to which additional capacity affects federation activity and the preemption fraction depends on the workload. DS-A has a similar federated work fraction in both configurations. As such, extra capacity is devoted relatively equally to federated and native jobs. For DS-B, extra capacity hosts more native workload (the rejection fraction and the federation fraction both decrease). For DS-C, the added capacity is used primarily to host federated workload (the rejection fraction decreases but the federation fraction increases).

5.4.2 Federation with SLA Guarantees

Figure 5.4 shows the results for the same experiments as the previous section when we employ our SLA-enforcing admission control. Our method reduces the number of jobs admitted in order to meet its preemption fraction guarantees (using the 0.05 upper bound). As in Figure 5.3 for the baseline results, the x-axis shows the workload trace name and the size of the each cloud in the federation. Similarly, on the y-axis, we plot the preemption fraction (red) of federated jobs, the rejection fraction (blue) of requests, and the fraction of successfully completed federated work (green). The results show that our method maintains a preemption fraction below 0.05 for all experiments indicating that ahead-of-time guarantees on the preemption fraction of federated instances are possible. This result holds for differing levels of capacity constraints across production traces.

Compared to the baseline results, our method increases rejection fractions in all experiments which leads to less work completed overall. The native work (not shown) is unaffected, but the work performed by federated instances decreases. That is, our method trades off a small amount of the total federated work completed to provide an SLO on preemption fraction for federated jobs. These results also indicate that our method changes the way additional capacity is used for federation. Because our admission decisions for federated jobs depend on the absolute amount of available spare capacity at the time of the request, our system is less sensitive to workload characteristics and achieves higher federation fractions with larger cloud configurations.

We also observe that under SLO constraints, DS-B federates few jobs compared to DS-A and DS-C. The reason for this is that for both sub-traces (one per cloud in the federation), DS-A and DS-C contain large numbers of small and short-running jobs, with DS-C also containing a few long-running, service-like requests in sub-trace two. DS-B is different in that sub-trace one contains primarily short jobs with a large number of

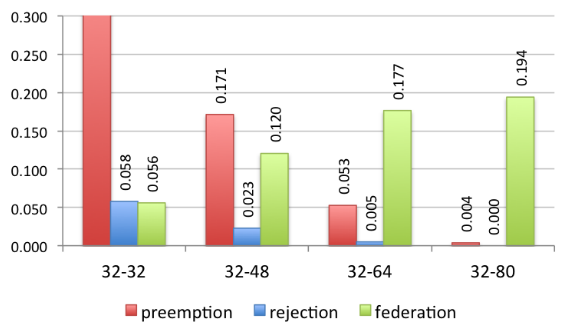


Figure 5.5: DS-A on multiple configurations without SLA enforcement. As capacity is added (left to right), preemption and rejection fractions decrease while the federation fraction increases.

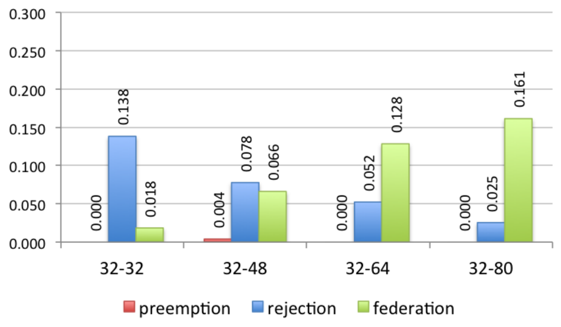


Figure 5.6: DS-A on multiple cloud configurations with SLA enforcement enabled. Preemption fractions remains below the 0.05 upper bound, while federation fractions increase with capacity.

instances per request and sub-trace two contains many long jobs with a small number of instances per request. This mismatch in workload characteristics between the two clouds in the federation, makes it more challenging to federate jobs in DS-B compared to DS-A and DS-C.

5.4.3 Federation with Platform Scaling

We next evaluate the method's ability to scale federation activity when adding capacity to the underlying clouds while keeping workload intensity constant. In particular, we investigate the capacity scaling behavior of our admission control for DS-A and DS-B. These data sets are of special interest as DS-A contains strong daily seasonality and DS-B when highly resource constrained appears ill-suited for federation. When a federation of clouds has sufficient capacity to accommodate all requests at any given time, SLA enforcement becomes unnecessary and may impose overhead. If our method scales gracefully with capacity, it will admit additional federation requests as the total capacity of the federation increases. Asymptotically, with increased capacity, our method should complete a similar amount of work as the no-guarantees baseline. In contrast to the baseline, users of the SLO-enabled federation experience a preemption fraction that meets the SLO targets, even as the workload of the clouds changes over time.

Figures 5.5 and 5.6 show the results of our capacity scaling experiments for DS-A for the baseline and SLA-enabled admission control, respectively. Figures 5.7 and 5.8 show the results for DS-B. The categories on the x-axis describe the size of the clouds in the federation, e.g. "32-48" identifies a pair of federated clouds, one with 32 cores and the other with 48 cores. The left graph in each pair shows federation statistics without SLA-enforcement, the right with SLA-enforcement enabled with an upper bound of 0.05 preemption fraction of federated instances. The y-axis is the preemption fraction (red), rejection fraction (blue) and federated work fraction (green).

Federation capacity increases from left to right in each graph. While the size of the cloud running sub-trace 1 is fixed at 32 cores for DS-A and 16 for DS-B, the size of the cloud running sub-trace 2 increases in steps of 16 cores, for a total federation capacity of 112 cores for DS-A and 80 cores for DS-B.

The baseline results for DS-A has a preemption fraction of near 0.4 for the 32-32 configuration. As the available capacity in cloud 2 increases, the preemption fraction and the rejection fraction decrease gradually. Inversely, the fraction of successfully federated work increases and levels off near 20% in the 32-80 configuration. For the largest configuration, the rejection fraction is at 0% indicating that every request sent to the system was accepted. Based on the rejection ratio, a capacity of 64 for cloud 2 provides sufficient capacity to accept 99% of all job requests sent to the clouds, although the preemption fraction of jobs is just above 0.05. If we manually enforce a 0.05 preemption SLO via static overprovisioning, we find that the optimal capacity of cloud 2 lies somewhere between 64 and 80 cores.

When we employ SLA-enabled admission control for this experiment (Figure 5.6) for DS-A, the preemption fraction of federated jobs drops to near zero across clouds. The

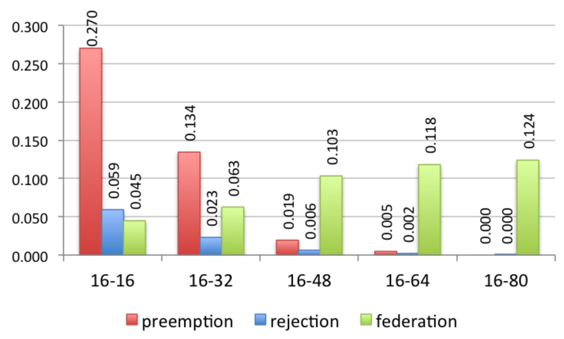


Figure 5.7: DS-B on multiple configurations without SLA enforcement. As capacity is added (left to right), preemption and rejection fractions decrease while the federation fraction increases.

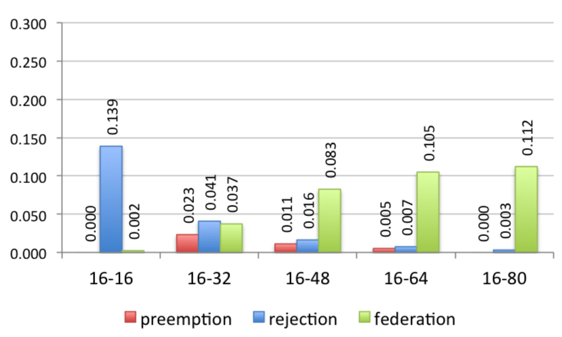


Figure 5.8: DS-B on multiple cloud configurations with SLA enforcement enabled. Preemption fraction remain below the 0.05 upper bound, while federation fractions increase with capacity.

admission controller achieves the target 0.05 upper bound on preemption fraction at the cost of higher rejection fractions, while increasing the federation fraction with added capacity. That is, rather than seeing a gradual reduction in preemption fractions and rejection fractions as capacity increases, the admission controller virtually removes the risk of preemption. Thus, using our approach, the cloud exhibits a direct relationship between cloud size and accepted work (as would be the case in an unfederated cloud setting).

Figure 5.7 represents the baseline for scaling cloud capacity with DS-B. The smallest configuration (16-16) results in a 0.27 preemption fraction. As the second cloud in the federation increases in size, the fractions decrease. Concurrently, the federation fraction increases from 4% to 12%. This trend in decreasing preemption fractions and rejection ratios, but increasing federation fraction is similar to those for DS-A. With a static, overprovisioning-based approach to SLA-enforcement we observe an optimal capacity of cloud 2 to be approximately 48 cores.

Figure 5.8 shows results of using SLA-enabled admission control for DS-B. The smallest configuration (16-16) results in a preemption fraction near 0 because very few jobs can be federated. As the federation capacity increases, the rejection fraction decreases while the federation fraction increases. Moreover, the system maintains a preemption fraction below 0.05 for all configurations. In contrast to DS-A where most configurations experience no preemptions, the admission method appears less conservative as the preemption fraction for scaling experiments with DS-B remains in the 0.01 to 0.02 range.

The comparison between baseline experiments for DS-A and DS-B offers an additional insight into the traces. The federation fractions across configurations indicate that DS-A has more potential for federation than DS-B. Both have a rejection fraction of near zero (indicating that most user requests are accepted), yet DS-A in its largest configuration has a 20% federation fraction while DS-B has 12%. Moreover, DS-A's higher federation activity leads to a disproportionately higher preemption fraction across capacities versus DS-B. DS-C behaves similarly to DS-A (and so we omit it for brevity).

These results show that admission control is able to scale gracefully with capacity and enables us to quantify the opportunity cost for automatically enforcing an SLO on the preemption fraction of federated jobs. Our admission control consistently maintains a preemption fraction below its 0.05 target and increases the amount of admitted federated work as capacity is added to the clouds. The differences in work completed between the baseline and the SLA-enabled cases are most visible in resource constrained settings. As capacity is added to the clouds, admission control accepts additional federated jobs, reaching federation fractions within 20% of the baseline.

5.4.4 Efficiency Gains at Scale

The utility of cloud federation capabilities should increase with a larger pool of workload available for consolidation. This effect should especially hold true for workloads with un- or anti-correlated user demand. If the efficiency of the federation as a whole

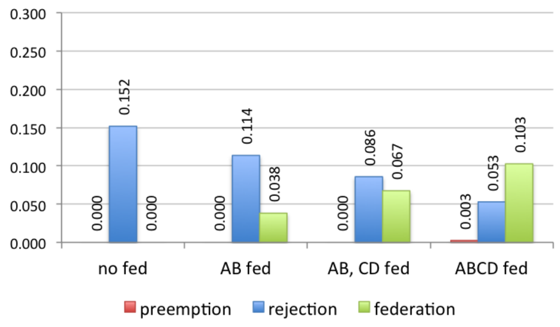


Figure 5.9: Efficiency gains of four clouds (A, B, C, D) with increasing federation size. Baseline (left), two separate clouds and a federation of two clouds (second), two federations of two clouds each (third), federation across all four clouds (right). As the number of federated clouds increases, the aggregate rejection fraction decreases while federated work increases.

increases with size, it will encourage participation of additional organizations as it grows.

We test this hypothesis of increased utility with scale by comparing the aggregate utilization of different federation setups with four individual clouds. Specifically, we use four clouds executing different workload traces and compare federation setups with 4 unconnected clouds, 2 unconnected and two federated clouds, two separate federations with two clouds each, and a single federation of all four clouds. If federation adds value with increasing total capacity and workload pool, the setup with four federated clouds should perform best while the unconnected setup should perform worst.

Our setup is similar to the scaling experiments with DS-A in Section 5.4.3 above. As cloud configurations, we chose four clouds with a configuration of "32-56-32-56", indicating two clouds with 32 cores each and two clouds with 56 cores each. We refer to these clouds as "A", "B", "C", and "D" respectively. To obtain the 4 different traces necessary to drive all four clouds, we use a new equal-length trace DS-Ax that was recorded immediately following the end of the orginal DS-A trace. We "time-shift" this new trace to simulate a workload executing concurrently with DS-A.

Figure 5.9 shows the increasing benefits of adding opportunities for workload consolidation by joining clouds together by federation. We plot four different federation setups as categories on the x-axis and again plot the preemption fraction, rejection fraction and federated work fraction on the y-axis. The leftmost setup represents the baseline, with all four clouds executing their workloads without federation. The next setup allows federation between clouds "A" and "B", while "C" and "D" remain separate. The third setup connects "A" and "B" as well as "C" and "D" within federations of 2 clouds each. The last setup on the right allows federation across all four clouds, maximizing opportunities for federation. In all setups the clouds execute their same, specific traces and we aggregate results across all four clouds for comparability.

We observe an incremental improvement in the aggregate amount work accepted by the clouds, as indicated by a decreasing rejection fraction from left to right. Equivalently, as the pool of workload and capacity increases by incrementally joining clouds together by federation, the amount of federated work increases. Admission control performs well in all scenarios and maintains a preemption fraction well below the 0.05 threshold. The most relevant result is the improvement from two federations of two clouds each to federation across all four clouds. Even with constant workload, as the federation grows in total capacity and user base, the smoothing of demand leads to even greater consolidation benefits.

5.4.5 Sensitivity to Duration Estimates

One fundamental requirement of our method for providing guarantees on the preemption fraction of federated jobs is the availability of a job duration *a-priori*, which flow directly into job lifetime estimates made by our system. A potential threat to the robustness of the method lies in obtaining inaccurate estimates of these durations. That is, we wish to investigate how sensitive the methodology is with respect to the degree to

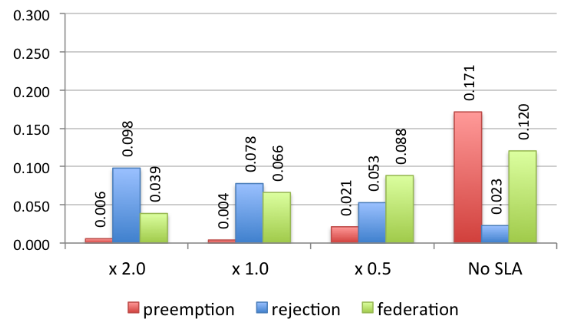


Figure 5.10: DS-A on 32-48 configuration with systematic estimation error of job durations. Overestimation, accurate estimation, underestimation and no-SLA baseline from left to right. Admission control consistently maintains the 0.05 upper bound on preemption fraction.

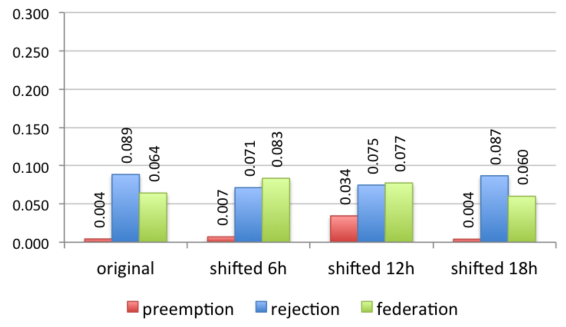


Figure 5.11: DS-A on 32-48 configuration with phase shift of daily seasonality between the traces on both clouds. While negative correlation between peak loads is beneficial (compare original to 12 hours shift), time remaining until the next peak matters as well (6 hours to 18 hours).

which requested minimum duration exceeds actual duration of computation for federated jobs.

Figure 5.10 shows DS-A executing on a 32-48 cloud configuration with admission control enabled and with different, systematic estimation errors of job durations. The left-most category executes trace data set DS-A and inflates the duration estimates by a factor of 2.0, effectively overestimating the duration of federated requests. The following categories show the results for accurate estimates (a factor of 1.0) and an underestimation of the runtime at a factor of 0.5. The last category repeats previous results for executing DS-A without admission control enabled for comparison.

The results indicate that admission control is still able to maintain its 0.05 bound on the preemption (red) fraction, although systematic underestimation has a negative impact on the realized preemption fraction. We also observe decreasing rejection fractions (blue) and increasing federation fractions (green) as runtime estimates become increasingly optimistic. While this experiment provides some evidence for robustness to variations in estimates, we expect that the tolerance to estimation errors in job runtime to vary with the specific workload.

5.4.6 Sensitivity to Seasonality

As another robustness test we investigate the impact of our seasonality correction on admission control and thus on the amount of federated work. Both sub-traces in DS-A show a recurring diurnal pattern. The first sub-trace changes the load by an order of magnitude from peak to trough, whereas the other sees less fluctuation. Notably, the load in the first cluster is positively correlated with the load in the second. Intuitively, this should be bad for federation as when available capacity in the local cloud runs out, the remote cloud also has less spare capacity than at other times. If the correlation was weaker, or even negative, more work should be federated. To test this hypothesis, we evaluate the performance of our system when introducing a "phase shift" between the two traces. We move the start times of jobs in the first cluster by several hours (in trace time) while leaving the second cluster to execute its original trace. Both clusters are still configured to enforce a 0.05 SLO on the preemption fraction of federated jobs.

Figure 5.11 shows the results of the 32-48 configuration of DS-A with the original trace setup on the left, followed by re-runs with the start times shifted in the first trace by 6, 12 and 18 hours. The results corroborate the hypothesis that negatively correlated workloads (12 hours phase shift) are better suited for federation than correlated peak loads (original). Considering that the majority of offloading occurs from cloud 1 to cloud 2, we observe that federated jobs are most likely to be accepted after the peak in cloud 2 has just passed (6 hours). As we get closer to the next seasonal surge in cloud 2 (12 and 18 hours), the federated work fraction (green) decreases and the rejection fraction (blue) goes up. Thus, correlation of utilization explains part of the federation performance, but the specifics of the workload matter as well. In all cases the the admission controller achieves its target SLO of less than 0.05 preemption fraction (red), although the realized preemption fraction changes with the time offset.

5.5 Conclusion

We present a novel approach to job federation across IaaS clouds that guarantees local control over resources by exclusively using preemptible instances for federated workload. To make this practical, we introduce a "predictable" tier of preemptible instances based on a method to provide an upper-bound SLO on the preemption fraction of federated jobs. This guarantee is learned statistically, is specific to each cloud participating in a federation, and dynamically adapts to changes in load and expected data transfer overheads.

The federation architecture is inspired by direct-flocking in Condor with decentralized control that avoids a single point of failure. Aiming at robustness, our implementation also integrates with existing open-source cloud frameworks and eschews invasive changes to these production-tested systems.

Our solution to job federation across IaaS clouds is complete and testable end-to-end as we implement a prototype on top of Eucalyptus IaaS to evaluate our approach using real-world replay of computationally intensive jobs and data analytics workloads recorded from production systems. Our prototype consistently maintains the upper-bound on the preemption probability of federated jobs, improves federation efficiency with scale, and is robust to seasonal patterns in load and systematic underestimation of job durations by the user.

Chapter 6

Conclusion

Cloud computing has changed the way large-scale computing infrastructure is built and managed. While private IaaS clouds have simplified provisioning of enterprise IT infrastructure within organizations, they suffer from inefficiencies due to the need for manual overprovisioning to meet predefined quality of service guarantees.

In this thesis, we present a method for increasing the resource efficiency of IaaS clouds via workload federation on preemptible resources. We take inspiration from existing work on federation and cycle harvesting in computational grids and aim to produce a practical solution that performs in real-world settings. We develop a validated simulation model for private IaaS clouds that reduces the engineering effort required to build cloud prototypes and thus enable rapid development.

Our cloud federation architecture allows multiple independent organizations to share intermittent spare capacity opportunistically without making long-term commitments. This sharing reduces the required amount of overprovisioning, while retaining full control over resources within organizational boundaries. In this way, with minimal investment clouds can be run closer to capacity without compromising on quality of service guarantees.

Our architecture preserves the autonomy of federation members by allowing preemption of federated workload by the executing cloud while still adhering to the fundamental cloud requirement of providing ahead-of-time guarantees by enforcing an SLO on the preemption probability of federated jobs. For this to be possible we use an a-priori upper bound on job duration which is provided the user. To make this practical for real-world batch jobs, we expect this user estimate to be of limited accuracy, so we design our method to be robust to underestimation but also adapt dynamically to overestimation. We then make ahead-of-time guarantees on the preemption probability of requested jobs by comparing job duration bounds with their estimated time-to-preemption. This approach takes advantage of our validated simulation model and uses the cloud's historic utilization trace to generate bounds on the minimum time that similar jobs would have executed in the past.

For initial evaluation of our federation design, we rely on a validated simulation model we construct for multiple private IaaS clouds. Specifically, we take inspiration from perturbation theory and develop a parsimonious top-down model of these clouds to estimate relevant performance metrics based on historic utilization traces. We use a Monte-Carlo style simulation and validate the model predictions end-to-end against measurements taken from live cloud systems. We evaluate our approach to validated simulation in a scheduler case-study before applying it to cloud federation. We find that our method proves to be robust to moderate system modification and capacity scaling.

For end-to-end evaluation of our federation architecture we implement a full prototype of our cloud federation system on top of the Eucalyptus IaaS framework and evaluate it with faster-than-realtime and smaller-than-realworld replay of computationally-intensive and data analytics workload traces recorded from production systems. Our prototype considers various practical aspects of workload federation, such as data transfers between federating clouds and delays induced by instance startup, and automatically adjusts decisions based on current utilization levels and seasonality in workload patterns to maintain its availability SLOs in different environments. Our evaluation results corroborate our hypothesis that the our workload federation method can improve the utilization of IaaS clouds and that ahead-of-time guarantees on job preemption probability can be enforced in practice.

Chapter 7

Future Work

We envision numerous ways of expanding on research in real-world cloud federation. In this section, we lay out a number of potential directions.

The replacement of the user-provided lifetime estimate with an autonomous prediction based on job attributes has the potential to substantially improve usability and efficiency. In the current design we expect a job runtime estimate from the user, which is an unnecessary burden and error prone. As a consequence, we designed the admission control mechanism to make conservative decisions to correct for potential error in these estimates. We suggest to replacing (or augment) the user-estimate with automated prediction of job runtimes which can tighten prediction bounds and as a result allows a federation to accept additional jobs.

We expect to learn valuable lessons from deploying and using the system in production. Besides testing the robustness of the implementation and API, a question about the impact in user behavior arises. Time-to-preemption estimates are based on historic utilization traces and the introduction of federation capabilities will undoutably affect user behavior. While we address part of this issue by using only cloud-native utilization data (excluding federated requests), we expect the impact on user-behavior to be pervasive. Additionally, user behavior may evolve over time which practically makes the inclusion of change point detection a necessity.

Another issue is the extension of the prediction to account for limited storage capacity. The system currently assumes that input and output data, if transferable in time, will fit in the object store of the destination cloud. In practice the priorities between local and federated job must be enforced by preempting federated workload that causes a storage space shortage for native jobs.

The prediction model further requires extension to account for differences in cloud hardware and infrastructure if federation is performed across a large set of different clouds – comparable to the issue of resource discovery and classification in computational grids. Experience from existing literature suggests an approach similar to Condor ClassAds, but will require additional flexibility to account for malleability of batch jobs in the cloud.

In a similar vein, the elastic trade-off between size and turnaround time for malleable batch jobs could improve can further improve federation efficiency. In the current design the non-linear relationship between core count and duration for admission control leads to differing admission decisions for the same unit of work, based on the exact combination of these attributes.

Adopting federation for public clouds, we envision augmenting existing service tiers with a new type of "predictable" preemptible capacity. Two tiers of preemptible instances could be offered within a single cloud, one with and another without availability guarantees, differentiated by price points. Additionally, federation across multiple public cloud providers seems feasible with a standardized packaging of jobs and SLOs.

The stateless random load balancer for choosing federation targets represents a scalability bottleneck. If most clouds in the federation are full or unsuitable for user jobs, the randomization creates unnecessary communication and delays. A solution that reduces these overheads must, however, ensure that individual clouds are not crowded.

The analytical modeling of the potential gains from federation given a set of clouds and their workloads could motivate cloud operators to join a federation. Additionally, it could inform the efficient expansion of existing infrastructure, so resources are added to relevant locations only.

While this work on simulation and federation focuses on IaaS clouds, the methods may well be applied to other domains. For example, we expect that this method should adapt well to use with microservices and containers. The method could also be extended to make a distinction between job preemption and timely "evacuation" via migration if ahead notice of termination is available, e.g. as is the case with Amazon EC2 spot instances.

A hybrid approach to "top-down" (parsimonious) and "bottom-up" (compositive) modeling of large-scale systems promises to provide an effective trade-off between model simplicity and prediction accuracy, as shown by validated simulation. We believe this simulation method, combined with (partial) evaluation against empirical results, has the potential to provide academics and engineers with deeper quantitative insights into the impact of decisions than extant – purely compositive – simulation models.

Bibliography

[1] A. Keller and H. Ludwig, *The wsla framework: Specifying and monitoring service level agreements for web services*, *Journal of Network and Systems Management* **11** (2003), no. 1 57–81.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, *et. al.*, *Above the clouds: A berkeley view of cloud computing*, .

[3] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, *International Journal of High Performance Computing Applications* **11** (1997), no. 2 115–128.

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

[5] R. Ricci, E. Eide, and The CloudLab Team, *Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications*, *USENIX ;login:* **39** (Dec., 2014).

[6] A. I. Avetisyan, R. Campbell, I. Gupta, M. T. Heath, S. Y. Ko, G. R. Ganger, M. A. Kozuch, D. O'Hallaron, M. Kunze, T. T. Kwan, *et. al.*, *Open Cirrus: A global cloud computing testbed*, *Computer* **43** (2010), no. 4 35–43.

[7] "Eucalyptus 4.0.2 Hybrid Cloud Guide." `http://docs.hpcloud.com/pdf/static/Eucalyptus_4.0/hybrid-guide-4.0.2.pdf`, 2016. [Online; accessed 21-June-2016].

[8] "Configuring Keystone for Federation." `http://docs.openstack.org/developer/keystone/configure_federation.html`, 2016. [Online; accessed 21-June-2016].

[9] D. F. Parkhill, *Challenge of the computer utility*, .

[10] "Amazon Web Services home page." `http://aws.amazon.com/`.

[11] "Google Cloud Platform." `http://cloud.google.com/`. [Online; accessed 01-May-2016].

[12] "Microsoft Azure." [Online; accessed Jun-2016] "https://azure.microsoft.com/".

[13] "Rackspace Cloud." [Online; accessed Jun-2016] "http://www.rackspace.com/cloud/".

[14] "IBM SoftLayer." [Online; accessed Jun-2016] "http://www.softlayer.com/".

[15] "euca2ools repository." [Online; accessed Jun-2016] `https://github.com/eucalyptus/euca2ools`.

[16] "boto: A python interface to amazon web services." [Online; accessed Jun-2016] `https://github.com/boto/boto`.

[17] C. Krintz, *The appscale cloud platform: Enabling portable, scalable web application deployment*, in *Internet Computing*, IEEE, 2013.

[18] "OpenStack." [Online; accessed Aug-2014] "http://www.openstack.org/".

[19] Eucalyptus Systems Inc., *`http://www.eucalyptus.com`*, June, 2013.

[20] "CloudStack." [Online; accessed Aug-2014] "http://cloudstack.apache.org/".

[21] D. Milojičić, I. M. Llorente, and R. S. Montero, *Opennebula: A cloud management tool*, *IEEE Internet Computing* **15** (2011), no. 2 11–14.

[22] "Nimbus is cloud computing for science." [Online; accessed Jun-2016] `http://www.nimbusproject.org/`.

[23] I. Foster, *Globus toolkit version 4: Software for service-oriented systems*, in *Network and Parallel Computing* (H. Jin, D. Reed, and W. Jiang, eds.), vol. 3779 of *Lecture Notes in Computer Science*, pp. 2–13. Springer Berlin Heidelberg, 2005.

[24] D. H. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne, *A worldwide flock of condors: Load sharing among workstation clusters*, *Future Generation Computer Systems* **12** (1996), no. 1 53–65.

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

[26] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke, *Modeling and managing state in distributed systems: The role of ogsi and wsrf*, *Proceedings of the IEEE* **93** (2005), no. 3 604–612.

[27] I. Foster, C. Kesselman, and S. Tuecke, *The anatomy of the grid: Enabling scalable virtual organizations*, *International journal of high performance computing applications* **15** (2001), no. 3 200–222.

[28] R. Wolski, N. T. Spring, and J. Hayes, *The network weather service: a distributed resource performance forecasting service for metacomputing*, *Future Generation Computer Systems* **15** (1999), no. 5 757–768.

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

[30] C. Team, *Condor® version 7.7. 6 manual*, .

[31] D. Thain, T. Tannenbaum, and M. Livny, *Distributed computing in practice: the condor experience*, *Concurrency and computation: practice and experience* **17** (2005), no. 2-4 323–356.

[32] D. Thain, T. Tannenbaum, and M. Livny, *Condor and the grid*, *Grid computing: Making the global infrastructure a reality* (2003) 299–335.

[33] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, *Condor-g: A computation management agent for multi-institutional grids*, *Cluster Computing* **5** (2002), no. 3 237–246.

[34] "Hadoop MapReduce." "http://hadoop.apache.org/".

[35] J. Dean and S. Ghemawat, *Mapreduce: simplified data processing on large clusters*, *Communications of the ACM* **51** (2008), no. 1 107–113.

[36] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, *Apache hadoop yarn: Yet another resource negotiator*, in *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, (New York, NY, USA), pp. 5:1–5:16, ACM, 2013.

[37] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, *Spark: Cluster computing with working sets.*, *HotCloud* **10** (2010) 10–10.

[38] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, *Hive: a warehousing solution over a map-reduce framework*, *Proceedings of the VLDB Endowment* **2** (2009), no. 2 1626–1629.

[39] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, *The hadoop distributed file system*, in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pp. 1–10, IEEE, 2010.

[40] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, *G-hadoop: Mapreduce across distributed data centers for data-intensive computing*, *Future Generation Computer Systems* **29** (2013), no. 3 739–750.

[41] G. Attebury, A. Baranovski, K. Bloom, B. Bockelman, D. Kcira, J. Letts, T. Levshina, C. Lundestedt, T. Martin, W. Maier, *et. al.*, *Hadoop distributed file system for the grid*, in *2009 IEEE Nuclear Science Symposium Conference Record (NSS/MIC)*, pp. 1056–1061, IEEE, 2009.

[42] I. Tomašić, J. Ugovšek, A. Rashkovska, and R. Trobec, *Multicluster hadoop distributed file system*, in *MIPRO, 2012 Proceedings of the 35th International Convention*, pp. 301–305, IEEE, 2012.

[43] M. Cardosa, C. Wang, A. Nangia, A. Chandra, and J. Weissman, *Exploring mapreduce efficiency with highly-distributed data*, in *Proceedings of the second international workshop on MapReduce and its applications*, pp. 27–34, ACM, 2011.

[44] C.-Y. Wang, T.-L. Tai, S. Jui-Shing, C. Jyh-Biau, and S. Ce-Kuen, *Federated mapreduce to transparently run applications on multicluster environment*, in *2014 IEEE International Congress on Big Data*, pp. 296–303, IEEE, 2014.

[45] C. Jayalath, J. Stephen, and P. Eugster, *From the cloud to the atmosphere: running mapreduce across data centers*, *IEEE Transactions on Computers* **63** (2014), no. 1 74–87.

[46] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang, *Moon: Mapreduce on opportunistic environments*, in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 95–106, ACM, 2010.

[47] C. He, D. Weitzel, D. Swanson, and Y. Lu, *Hog: Distributed hadoop mapreduce on the grid*, in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pp. 1276–1283, IEEE, 2012.

[48] "RightSclae: Hybrid Cloud." `http://www.rightscale.com/solutions/problems-we-solve/hybrid-cloud`, 2016. [Online; accessed 21-June-2016].

[49] C. Bunch and C. Krintz, *Enabling automated hpc/database deployment via the appscale hybrid cloud platform*, in *Proceedings of the first annual workshop on High performance computing meets databases*, pp. 13–16, ACM, 2011.

[50] "Perturbation Theory." `http://en.wikipedia.org/wiki/Perturbation_theory`.

[51] "Monte Carlo Method." `http://en.wikipedia.org/wiki/Monte_Carlo_method`.

[52] R. Wolski and J. Brevik, *QPRED: Using Quantile Predictions to Improve Power Usage for Private Clouds*, Tech. Rep. UCSB-CS-2014-06, Computer Science Department of the University of California, Santa Barbara, Santa Barbara, CA 93106, September, 2014.

[53] J. Gustedt, E. Jeannot, and M. Quinson, *Experimental Validation in Large-Scale Systems: a Survey of Methodologies*, *Parallel Processing Letters* **19** (2009), no. 3 399–418.

[54] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, *Load balancing and unbalancing for power and performance in cluster-based systems*, in *Workshop on compilers and operating systems for low power*, vol. 180, pp. 182–195, Barcelona, Spain, 2001.

[55] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, *Managing energy and server resources in hosting centers*, in *ACM SIGOPS Operating Systems Review*, vol. 35, pp. 103–116, ACM, 2001.

[56] X. Fan, W.-D. Weber, and L. A. Barroso, *Power provisioning for a warehouse-sized computer*, *ACM SIGARCH Computer Architecture News* **35** (2007), no. 2 13–23.

[57] L. A. Barroso and U. Holzle, *The case for energy-proportional computing*, *Computer* **40** (2007), no. 12 33–37.

[58] R. Brown *et. al.*, *Report to congress on server and data center energy efficiency: Public law 109-431*, 2008.

[59] A. Beloglazov, J. Abawajy, and R. Buyya, *Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing*, *Future Generation Computer Systems* **28** (2012), no. 5 755–768.

[60] D. Nurmi, J. Brevik, and R. Wolski, *Qbets: queue bounds estimation from time series*, in *Job Scheduling Strategies for Parallel Processing*, pp. 76–101, Springer, 2007.

[61] H. Casanova, *Simgrid: A toolkit for the simulation of application scheduling*, in *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pp. 430–437, IEEE, 2001.

[62] R. Buyya and M. Murshed, *Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing*, *Concurrency and computation: practice and experience* **14** (2002), no. 13-15 1175–1220.

[63] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. D. Rose, and R. Buyya, *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, *Software: Practice and Experience* **41** (2011), no. 1 23–50.

[64] S. K. Garg and R. Buyya, *Networkcloudsim: Modelling parallel applications in cloud simulations*, in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pp. 105–113, IEEE, 2011.

[65] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, *Cloudanalyst: A CloudSim-based visual modeller for analysing cloud computing environments and applications*, in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pp. 446–452, IEEE, 2010.

[66] M. Silva, M. Hines, D. Gallo, L. Qi, R. K. Dong, and D. D. Silva, *CloudBench: Experiment Automation for Cloud Environments*, in *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pp. 302–311, March, 2013.

[67] D. Kliazovich, P. Bouvry, and S. U. Khan, *GreenCloud: a packet-level simulator of energy-aware cloud computing data centers*, *The Journal of Supercomputing* **62** (2012), no. 3 1263–1283.

[68] "NS-2 Network Simulator." `http://www.isi.edu/nsnam/ns/`.

[69] R. N. Calheiros, M. A. Netto, C. A. D. Rose, and R. Buyya, *EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications*, *Software: Practice and Experience* **43** (2013), no. 5 595–612.

[70] D. Citron and A. Zlotnick, *Testing large-scale cloud management*, *IBM Journal of Research and Development* **55** (2011), no. 6 6–1.

[71] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, *DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management*, in *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)*, pp. 385–392, Oct, 2012.

[72] S. K. S. Gupta, R. Gilbert, A. Banerjee, Z. Abbasi, T. Mukherjee, and G. Varsamopoulos, *GDCSim: A tool for analyzing Green Data Center design and resource management techniques*, in *Green Computing Conference and Workshops (IGCC), 2011 International*, pp. 1–8, July, 2011.

[73] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, *MDCSim: A multi-tier data center simulation, platform*, in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pp. 1–9, IEEE, 2009.

[74] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, *icancloud: A flexible and scalable cloud infrastructure simulator*, *Journal of Grid Computing* **10** (2012), no. 1 185–209.

[75] I. K. Kim, W. Wang, and M. Humphrey, *Pics: A public iaas cloud simulator*, in *2015 IEEE 8th International Conference on Cloud Computing*, pp. 211–220, June, 2015.

[76] R. Wolski and J. Brevik, ***http://www.cs.ucsb.edu/~rich/workload***, June, 2013.

[77] R. Wolski and J. Brevik, *Using Parametric Models to Represent Private Cloud Workloads*, *IEEE Transactions on Services Computing* **4** (October, 2014) 714–725.

[78] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, *The eucalyptus open-source cloud-computing system*, in *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pp. 124–131, IEEE, 2009.

[79] "Announcing Amazon EC2 Spot Instances." [Online; accessed Aug-2014] "http://aws.amazon.com/about-aws/whats-new/2009/12/14/announcing-amazon-ec2-spot-instances/".

[80] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, *See spot run: Using spot instances for mapreduce workflows*, in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, (Berkeley, CA, USA), pp. 7–7, USENIX Association, 2010.

[81] J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya, *Tradeoffs between profit and customer satisfaction for service provisioning in the cloud*, in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, (New York, NY, USA), pp. 229–238, ACM, 2011.

[82] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, *Deconstructing amazon ec2 spot instance pricing*, *ACM Transactions on Economics and Computation* **1** (2013), no. 3 16.

[83] B. Javadi, R. Thulasiram, and R. Buyya, *Statistical modeling of spot instance prices in public cloud environments*, in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pp. 219–228, Dec, 2011.

[84] H. Zhao, M. Pan, X. Liu, X. Li, and Y. Fang, *Exploring fine-grained resource rental planning in cloud computing*, *IEEE Transactions on Cloud Computing* **3** (July, 2015) 304–317.

[85] P. Nash, *Introducing Preemptible VMs, a new class of compute available at 70% off standard pricing*, May, 2015.

[86] R. Wolski, D. Nurmi, and J. Brevik, *An analysis of availability distributions in condor*, in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–6, IEEE, 2007.

[87] J. Brevik, D. Nurmi, and R. Wolski, *Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems*, in *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pp. 190–199, IEEE, 2004.

[88] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, *Towards characterizing cloud backend workloads: insights from google compute clusters*, *ACM SIGMETRICS Performance Evaluation Review* **37** (2010), no. 4 34–41.

[89] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, *Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis*, in *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, (New York, NY, USA), pp. 7:1–7:13, ACM, 2012.

[90] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, *Analysis and lessons from a publicly available google cluster trace*, *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95* (2010).

[91] "Google cluster traces." `http://code.google.com/p/googleclusterdata`.

[92] K. Ren, Y. Kwon, M. Balazinska, and B. Howe, *Hadoop's adolescence: an analysis of hadoop usage in scientific workloads*, *Proceedings of the VLDB Endowment* **6** (2013), no. 10 853–864.

[93] Y. Chen, S. Alspaugh, and R. Katz, *Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads*, *Proceedings of the VLDB Endowment* **5** (2012), no. 12 1802–1813.

[94] T. Knauth and C. Fetzer, *Spot-on for timed instances: Striking a balance between spot and on-demand instances*, in *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pp. 105–112, Nov, 2012.

[95] M. Taifi, J. Y. Shi, and A. Khreishah, *Spotmpi: a framework for auction-based hpc computing using amazon spot instances*, in *International Conference on Algorithms and Architectures for Parallel Processing*, pp. 109–120, Springer, 2011.

[96] S. Khatua and N. Mukherjee, *Application-centric resource provisioning for amazon ec2 spot instances*, in *Proceedings of the 19th International Conference on Parallel Processing*, Euro-Par'13, (Berlin, Heidelberg), pp. 267–278, Springer-Verlag, 2013.

[97] M. Mattess, C. Vecchiola, and R. Buyya, *Managing peak loads by leasing cloud infrastructure services from a spot market*, in *Proceedings of the 2010 IEEE 12th International Conference on High Performance Computing and Communications*, HPCC '10, (Washington, DC, USA), pp. 180–188, IEEE Computer Society, 2010.

[98] I. Menache, O. Shamir, and N. Jain, *On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud*, in *11th International Conference on Autonomic Computing (ICAC 14)*, (Philadelphia, PA), pp. 177–187, USENIX Association, June, 2014.

[99] A. Andrzejak, D. Kondo, and S. Yi, *Decision model for cloud computing under sla constraints*, in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pp. 257–266, Aug, 2010.

[100] M. Mazzucco and M. Dumas, *Achieving performance and availability guarantees with spot instances*, in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pp. 296–303, Sept, 2011.

[101] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, *Spotcheck: Designing a derivative iaas cloud on the spot market*, in *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, (New York, NY, USA), pp. 16:1–16:15, ACM, 2015.

[102] R. Singh, P. Sharma, D. Irwin, P. Shenoy, and K. K. Ramakrishnan, *Here today, gone tomorrow: Exploiting transient servers in datacenters*, *IEEE Internet Computing* **18** (July, 2014) 22–29.

[103] M. Mao and M. Humphrey, *A performance study on the vm startup time in the cloud*, in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp. 423–430, IEEE, 2012.

[104] "Amazon ec2 service level agreement." [Online; accessed Jun-2016] `https://aws.amazon.com/ec2/sla/`.

[105] Y. Mansour, *Regret minimization and job scheduling*, in *SOFSEM 2010: Theory and Practice of Computer Science* (J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorný, and B. Rumpe, eds.), vol. 5901 of *Lecture Notes in Computer Science*, pp. 71–76. Springer Berlin Heidelberg, 2010.

[106] R. Wolski and J. Brevik, *Using parametric models to represent private cloud workloads*, Tech. Rep. UCSB-CS-2013-05, University of California, Santa Barbara, August, 2013. `http://128.111.41.26/research/tech_reports/reports/2013-05.pdf`.

[107] A. Pucher, E. Gul, C. Krintz, and R. Wolski, *Using Trustworthy Simulation to Engineer Cloud Schedulers*, in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, March, 2015.

[108] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, *How to enhance cloud architectures to enable cross-federation*, in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 337–345, IEEE, 2010.

[109] N. Grozev and R. Buyya, *Inter-cloud architectures and application brokering: taxonomy and survey*, *Software: Practice and Experience* **44** (2014), no. 3 369–390.

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

[111] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, *Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures*, *Computer* **45** (2012), no. 12 65–72.

[112] D. Villegas, N. Bobroff, I. Rodero, J. Delgado, Y. Liu, A. Devarakonda, L. Fong, S. M. Sadjadi, and M. Parashar, *Cloud federation in a layered service model*, *Journal of Computer and System Sciences* **78** (2012), no. 5 1330–1344.

[113] I. Goiri, J. Guitart, and J. Torres, *Characterizing cloud federation for enhancing providers' profit*, in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 123–130, July, 2010.

[114] M. Aazam and E. N. Huh, *Broker as a service (baas) pricing and resource estimation model*, in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pp. 463–468, Dec, 2014.

[115] W. Wang, D. Niu, B. Li, and B. Liang, *Dynamic cloud resource reservation via cloud brokerage*, in *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems*, ICDCS '13, (Washington, DC, USA), pp. 400–409, IEEE Computer Society, 2013.

[116] K. Liu, J. Peng, W. Liu, P. Yao, and Z. Huang, *Dynamic resource reservation via broker federation in cloud service: A fine-grained heuristic-based approach*, in *2014 IEEE Global Communications Conference*, pp. 2338–2343, Dec, 2014.

[117] O. Rogers and D. Cliff, *A financial brokerage model for cloud computing*, *Journal of Cloud Computing: Advances, Systems and Applications* **1** (2012), no. 1 1–12.

[118] X. Jin, Y. K. Kwok, and Y. Yan, *Competitive cloud resource procurements via cloud brokerage*, in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 2, pp. 355–362, Dec, 2013.

[119] Y. Song, M. Zafer, and K.-W. Lee, *Optimal bidding in spot instance market*, in *INFOCOM, 2012 Proceedings IEEE*, pp. 190–198, March, 2012.

[120] M. Yao, P. Zhang, Y. Li, J. Hu, C. Lin, and X. Y. Li, *Cutting your cloud computing cost for deadline-constrained batch jobs*, in *Web Services (ICWS), 2014 IEEE International Conference on*, pp. 337–344, June, 2014.

[121] J. L. L. Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, *Dynamic placement of virtual machines for cost optimization in multi-cloud environments*, in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pp. 1–7, July, 2011.

[122] D. Williams, H. Jamjoom, and H. Weatherspoon, *Plug into the supercloud*, *IEEE Internet Computing* **17** (Mar., 2013) 28–34.

[123] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, *et. al.*, *The reservoir model and architecture for open federated cloud computing*, *IBM Journal of Research and Development* **53** (2009), no. 4 4–1.

[124] “Apcera.” [Online; accessed Jun-2016] "http://www.apcera.com”.

[125] “ClusterK.” [Online; accessed Apr-2015] "http://www.clusterk.com”.

[126] J. Barr, *Amazon EC2 Spot Instances – And Now How Much Would You Pay?*, Dec., 2009.