Wayne State University

Wayne State University Dissertations

January 2019

Automatic Resource Management And Performance Optimization In Clusters

Yudi Wei
Wayne State University, jadeskywei@gmail.com

Follow this and additional works at: https://digitalcommons.wayne.edu/oa_dissertations

Part of the Computer Engineering Commons

Recommended Citation
Wei, Yudi, "Automatic Resource Management And Performance Optimization In Clusters" (2019). *Wayne State University Dissertations*. 2231.
https://digitalcommons.wayne.edu/oa_dissertations/2231

This Open Access Dissertation is brought to you for free and open access by DigitalCommons@WayneState. It has been accepted for inclusion in Wayne State University Dissertations by an authorized administrator of DigitalCommons@WayneState.

AUTOMATIC RESOURCE MANAGEMENT AND PERFORMANCE OPTIMIZATION IN CLUSTERS

by

YUDI WEI

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2019

MAJOR: COMPUTER ENGINEERING

Approved By:

Advisor Date

©COPYRIGHT BY

YUDI WEI

2019

All Rights Reserved

ACKNOWLEDGMENTS

The study for PhD in the past years is full of challenges and adventures. The way to seek the truth is never easy. First of all, I can achieve nothing without the life-giving and all-inclusive spirit. I am grateful to two spiritual teachers and authors, Eckhart Tolle and Charles F. Haanel, for their spiritual books that show the way to recognize, understand and apply the power of the creative principle of thought.

I am thankful to many people who are kind and helpful. I would like to thank Dr. Feng Lin, Dr. Nathan Fisher, Dr. Cheng-Zhong Xu and Dr. Song Jiang for being my committee members and their suggestions to improve this work. Thanks also go to my friends, Xiangping Bu, Zhen Kong, Kun Wang, Yuehai Xu, Guoyao Xu and Xingbo Wu for their friendship and help. I would also like to thank all people who care about and pray for me, David Chang, Liang Wen, Chingshya Lee, Mike Chang, John Yan, etc..

Finally, I would like to thank my parents and every family member for their unconditional love and support for my study. I want to give special thanks to my brother, Bisheng Wei, for his encouragement and support all the time as always.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Background and Motivation

Cloud computing revolutionizes the way IT industry operates by moving to the cloud without owning the infrastructure. For example, latest report states that over 5 million organizations with 50 million users embrace Google Apps, which was launched in 2006. Amazon Web Services (AWS) host more than 11.6 million websites on 158k computers by EC2 and S3. It is still seeing unrelenting growth.

A fundamental technique of the cloud service model is consolidation of multiple applications on physical servers. A primary way for consolidation is multiplexing physical resource over virtualized servers. It realizes economies of scale on both supply and demand sides. On the supply side, the cloud reduces operation and maintenance cost and increases resource utilization. On the demand side, cloud clients are freed from the burden of IT planning and maintenance so as to be able to spend more time on critical tasks. Moreover, it eliminates an up-front venture capital by cloud users due to pay-as-you-go pricing model.

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., network, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [63]. In this model, on-demand self-service is a fundamental attribute, and metering and management techniques are required to realize this “low-touch, low-margin, low-commitment” self-service model.

Cloud resource usage has two ways of metering and tracking. The first is VM-level usage metering, such as storage space, bandwidth consumption, computing cycles. The second is performance-level metering, such as response time, throughput, availability (99.9% or 99.99% uptime), user-experienced QoS, etc. Performance guarantee is desirable since the capacity of certain amounts of resources may not satisfy the performance target considering the fluctuation of workload, and the uncertainty of resource capacity due to resource interference. Moreover, pricing metric should also be performance so that customer is exempted from paying the additional resource cost used to compensate for the interference effect [62]. Therefore, both of the guarantee metric and pricing metric should be performance. Performance is measured normally in two main metrics: response time and throughput. In many interactive benchmarks, such as TPC-W, dynamic arrival rate is often assumed to follow a Poisson distribution [4]. The defining characteristic is that its mean and variance are identical. Accordingly, throughput per time unit is similar with same resource allocations. Compared with throughput, users are more concerned with response time because it reflects the performance perceived by individual clients.

For the cloud providers, the profit comes from two factors: service level objectives (SLOs) it has committed in service level agreements (SLAs) and resource expenditures it pays for [62]. In order to maximize the profit, the cloud system should automatically minimize resource usage without compromising SLOs.

Besides virtualization, container is another popular way for multiplexing resources. The container is similar to the virtual machine because both encapsulate multi-resource demands and provide performance isolation. The former is a lightweight mechanism, and its resource demand cannot change, while the demand of the latter can change on the fly. In modern cluster systems, such as Mesos [46] and Yarn [82], container is provided to satisfy heterogenous demands of various tasks. It unlocks space for resource combination and optimization between tasks and servers. Moreover, it supports diverse programming models, such as Map-Reduce and MPI, so that different users can share resources in a cluster. Among these models, MapReduce [28] remains dominant parallel and distributed programming paradigm for applications to process big data in clusters [64] [58] [85]. It is one of the core technologies powering IT giants like Facebook. MapReduce jobs are transformed into a set of tasks The cluster scheduler uses a two-level architecture that consists of job scheduling and task scheduling, where tasks are basic objects of resource allocation.

In shared clusters, scheduling has three different objectives: system efficiency, fairness, and minimal job completion times (JCT). However, each may be in conflict with the other two under existing scheduling strategies or policies. While packing a set of tasks of complementary resource demands optimizes system efficiency by maximizing resource utilization of any resource type, fairness may be not preserved because these tasks are not from the job with minimal dominant share according to dominant resource fairness (DRF) policy, and JCT may be not minimal because they are not from the shortest job according to shortest remaining job first (SRJF). In addition, the objectives of fairness and minimal JCT may be not achieved together because the scheduling order under DRF is different from the one under SRJF. Particularly, SRJF may cause starvation or extreme unfairness for long jobs if short jobs keep coming.

In this dissertation, our work is to dynamically configure virtualized resources under the constraint of SLOs in terms of response time in virtual clusters, and optimize both system efficiency and JCTs while guaranteeing that no job finishes later than under DRF in physical clusters.

1.2 Challenges in Clusters

Dynamic resource provisioning is not trivial. Over-provisioning increases resource cost, hence resource waste and underutilization, or under-provisioning leads to SLA violations. As a result, data center administrators are faced with increasing challenges to meet SLOs in the presence of workload dynamics and cloud dynamics of unpredictable interactions across many applications.

In the physical clusters, there is also a lot of challenges in performance optimization from guarantee of JCTs , data locality to network utilization.

1.2.1 Challenges in Management of A Single Virtualized Resource

Workload dynamics: The intensity and the mix of typical application workloads vary over the lifetime of an application. As a result, the demands for individual resources also change over time. This implies that static resource allocation can't efficiently work.

Nonlinearity between SLOs and resource allocation: Nonlinearity makes it nontrivial to convert SLOs to corresponding resource share in the shared virtualized platform.

Unpredictable interference across co-hosted applications. Due to lack of perfect performance isolation, applications have unexpected impact on each other while contending for shared resource. Interference itself doesn't pose big challenge, instead the uncertainty of interference causes big trouble. For example, given steady workloads for all applications, the resource distribution satisfying SLOs currently may not meet them next time due to interference from contention. It feels like the impact of uncertain network delay on performance of distributed network system.

1.2.2 Challenges in Management of Multiple Virtualized Resources

Multiple-resource configuration provides more optimization space. In multi-resource virtualized clusters, resource optimization lies in whether it is able to find out a balanced configuration of multiple resources that meets application SLOs without over-supply or under-supply of resources.

However, workload dynamics poses great challenges to balancing resources. First, it is not easy to identify the over-provisioned or constrained resource type due to fluctuating workload. Second, a balanced configuration may be imbalanced later because time-varying workload results in time-varying resource requirements. The bottleneck may shift from one resource type to another within a VM, or from one VM to another one in the extension of multi-tier applications.

Moreover, capacity uncertainty caused by VMs interferences in the open shared environment [62] [65] [42] poses additional challenges to virtualized resources balancing. Even if given the same workload intensity and mix over a period, a balanced configuration also may be imbalanced at some point later within the period. The reason is that the capacity of one resource changes (increases or decreases), or the capacities of several resources change but with different extents. Even if with the same extent and configuration is still balanced, performance level is not the same since capacities vary.

Performance in SLO is measured normally in two main metrics: response time and throughput. In many interactive benchmarks, such as TPC-W, dynamic arrival rate is often assumed to follow a Poisson distribution. Accordingly, throughput per time unit is similar with same resource allocations. Compared with throughput, users are more concerned with response time because it reflects the performance perceived by individual clients. Unlike throughput, however, response time behaves nonlinearly with respect to resource allocations. The nonlinear relationship between resources and performance in terms of response time adds one more dimension to the task of balanced VM configuration [75].

Multi-tier applications further complicate the problem due to chain reaction during resource contention. An imbalanced configuration for one application may lead to an imbalanced configuration for one tier of another application co-hosted on the same physical server by over-provisioning one or several resources which are needed for the tier of that application to achieve balanced configuration. This may lead to an imbalanced configuration of the other tier(s) of that application residing in a different physical machine by over-provisioning resources to achieve its QoS target. As a result, the error of an imbalanced configuration on one physical machine spreads to another physical machine, and even to the whole virtual cluster. However, no existing work coordinates tiers within an application to restrict error propagation. This motivated us to consider an application as a configuration unit to coordinate configuration among different resources and tiers during contention.

1.2.3 Challenges in Performance Optimization

The multi-resource version of max-min fairness, i.e., DRF, is based on instantaneous share at any time point. On one hand, it causes resource fragments, which are resources in a node that cannot be utilized by any task, thus degrading system performance. On the other hand, the envy-free property of fairness, i.e., no user should prefer the resource allocation of another user, may elongate JCT. The reason is that JCT are improved by trading resource allocation of long jobs with that of short jobs. Since users cannot experience the instantaneous share, they care more about whether a job or its last task finishes within its *fair sharing job completion time* (FJCT), the one resulting from DRF policy. Our observation is that by keeping in mind the FJCT of each job, the constraints placed by DRF on the optimization of both cluster efficiency and actual JCTs can be removed. Specifically by dynamically sorting jobs in the increasing order of their FJCTs (instead of instantaneous share), packing tasks of multiple short jobs by drawing credit from long jobs not only accelerates short jobs, but also guarantees that long jobs will not decelerate. Moreover, packing optimizes system performance by maximizing resource utilization in each dimension.

To make tradeoff between the three conflicting objectives, a scheduler called Tetris [39] was designed to employ dot product to pack tasks of jobs at a fixed-value (75th) percentile of dominant share. Whenever resource is available, it computes the packing score and job score (related to job length) for each task, and selects the one with the largest weighted score. While Tetris improves system efficiency, there are two main disadvantages. First, the percentile knob constrains improvement of JCT. When the dominant shares of short jobs go beyond the fixed-value percentile, they are not allowed to run further even though they will not decelerate long jobs. Therefore it wastes the opportunity to accelerate short jobs. Second, the knob cannot guarantee the FJCTs. Packing heuristics favor certain tasks. For example, dot product favors large resource demand. As a result, the jobs that have extremely small task demands may be even starved (subsection 5.2.1), even if the percentile rank is restricted to a small value.

The recent work [40] proposed altruistic scheduling (AS) in which long jobs yield resource allocation to short jobs. While it improves the average JCT, the improvement is achieved at the cost of long jobs, because it provides guarantee of FJCTs only in *offline scheduling*, rather than in online scheduling. The AS results also show that a significant ratio (16%) of jobs elongate with performance up to 0.62 times worse, or with slowdown up to 1.61

times. Moreover, AS serves as an outside layer or a plug-in of any cluster scheduler, such as DRF and Tetris, focusing on job acceleration with no consideration of cluster efficiency. Since AS depends on a scheduler, it is *complementary* to our independent scheduling work. Our objective is to optimize both system efficiency and JCTs, while theoretically guarantee FJCTs in *online scheduling*.

In multi-resource clusters, system efficiency depends on effective utilization of any resource type. A MapReduce job is divided into Map and Reduce tasks. A Map task can run on the same node as its input data (node locality), on the same rack (rack locality) or on a different rack (off-rack). Compared with a node-local task, a non-local one needs additional network resources and extra time for data transfer. Tradeoff should be made when the resource demand of a non-local task is larger than a local task. There were studies on achieving good data locality without impairing fairness in a slot-based cluster [49] [97]. A slot is a fixed amount of a single resource type, such as memory in Hadoop. In multi-resource clusters, the heterogeneity of multi-resource requirements by tasks may render them ineffective. To choose between a local and a non-local task, current work [39] imposed a fixed-value penalty to discount the packing score of a non-local task. Though the fixed-value improves the average performance, its fixity may cause sub-optimal performance due to lack of flexibility. Moreover, the value is chosen offline from a large space of different values, demanding reconfiguration and much time and effort whenever data workloads variate largely. Therefore, data locality poses great challenges to cluster scheduling.

Network utilization is crucial to performance in large-scale data-intensive clusters because network bisection bandwidth tends to become a bottleneck [29]. In Map-Reduce jobs, network is mainly used by reduce tasks for intermediate data transfer. The reason is that schedulers strive to avoid remote data transfer by placing map tasks as many as possible on the nodes storing their data. There is stage barrier to network utilization because reduce tasks cannot start if map tasks have not finished. Therefore, scheduling tasks of the same stage in most or all jobs may cause imbalanced network utilization and elongation of network-intensive jobs. Existing schedulers, including Tetris, is oblivious of this barrier. Though Tetris scheduled the last few tasks when the map stage is closed to completion, the effect is limited. The network can be idle when the running tasks are mainly map tasks, and become a bottleneck when most are reduce tasks. Advancing some network-intensive jobs to get network fully utilized not only improves system efficiency, but also decreases JCTs. However, how to accelerate jobs without affecting completions of other jobs further complicates the task of cluster scheduling.

1.3 Problem Statement and Objectives

In this dissertation, we strive to design, implement and evaluate an automatic management framework in virtual and physical clusters respectively. The framework should be able to improve resource utilization, satisfy the SLO of each hosted application in the presence of workload and cloud dynamics, and optimize both system efficiency and JCTs while guaranteeing that each job completes faster, or at least no worse than under DRF. The desired framework needs to be self-adaptive, involving as few manual effort as possible, and the management operations should be efficient in order to be applied in online production systems, and transparent to end users with lowest overhead. Specifically, the framework consists of three parts, with two corresponding to virtual clusters and one to physical clusters.

For a single resource management in virtual clusters, the framework should be able to adapt to workload dynamics and cloud dynamics. It should be able to handle nonlinear relationship between resource allocation and response time with no knowledge of accurate performance model. High scalability is an important requirement for dynamic resource provisioning in large scale data centers.

For multi-resource management in virtual clusters, the framework should be able to get a balanced configuration of multiple resources for each hosted application. Besides satisfying SLOs in terms of response time in the presence of workload dynamics and cloud dynamics, resource management methods need to deal with the interaction between virtual machines on which multi-tier applications span. It requires coordinated management strategy to automatically tune all tiers of one application for satisfying SLOs. During resource contention, it should optimize performance and satisfy SLOs for as many applications as possible.

For MapReduce in physical clusters, the framework should be able to optimize both system efficiency and JCTs while guaranteeing that each job completes faster, or at least no later than under DRF. The scheduling needs to be highly adaptive to workload dynamics. It is able to achieve highly effective resource utilization to optimize system efficiency and improve network utilization to optimize system performance and accelerate network-intensive jobs without decelerating other jobs.

1.4 Our Contributions

In this dissertation work, we designed and implemented an automatic resource management system to improve resource utilization and optimize performance for hosted applications and jobs in clusters. This system is able to address the challenges in both virtual clusters and physical clusters. We summarize the primary contributions of our works as follows:

A single resource configuration

1. We develop adaptive fuzzy control to satisfy response time targets for cloud applications by adjusting vCPU cap of virtual machines despite workload dynamics. It is model-free and self-adaptive to applications.

2. We designed and implemented a self-tuning fuzzy controller to assure QoS dynamically. It is a generic tool and easy to be deployed large scale data centers. Experimental results with TPC-W benchmark applications in Xen Virtualized environment demonstrated that performance of Adaptive Fuzzy control has the best stability with 14% more stable than the second best controller, the shortest settling time with 4 intervals less and minimum overshoot. This work was published in [75] [72].

Multi-resource configuration

1. We develop an integrated MPC and adaptive PI approach (IMAP) for balanced or near balanced provisioning of multiple resources. MPC coordinates with adaptive PI to balance resources. While MPC optimizes resource efficiency and utilization, adaptive PI ensures stable and responsive QoS control.

2. For the performance prediction in the IMAP, both black-box and gray-box models are developed for comparison. A black-box model is simpler to build and more general for applications than the gray-box, whereas the gray-box model is more accurate in performance prediction.

3. We employ a hierarchical way to resolve resource contention. Instead of being allocated individually, the constrained resources are distributed as a whole for an application, preventing the error with an imbalanced configuration from spreading to the whole virtual cluster.

4. We design and implement BConf prototype in a Xen-based cloud testbed. Compared with a representative resource partition approach, BConf reduces resource usages by up to 50%, improves stability by more than 35.6%, and has a much shorter settling time. BConf effectively coordinates resources during resource contention. The saved resources can be used either to pack more workloads, or to promote performance levels. This work was published in [88]

Performance optimization in multi-resource clusters

1. We propose a theorem-proof policy, called credit sharing (CS) policy, that guarantees FJCTs. It recursively accounts for the credit that long jobs contribute to the short jobs, and short jobs pay down or off their borrowed credit later in the process of execution or upon completions.

2. We present a method called Dynamic-Level for calculating the number of head jobs dynamically that optimizes both the effective resource utilization and JCTs according to CS policy and the size of total fragment. The total fragment is estimated as the size of all the normal and non-local fragments resulting from a packing, since a non-local task introduces a “new" non-local fragment by wasting extra resources.

3. We monitor network and schedule additional tasks for some network-intensive jobs to overcome the stage-barrier by borrowing credit from long jobs based on CS policy. The packing scores of the additional tasks are dynamically adjusted and scheduled together uniformly with all other tasks in Dynamic-Level.

1.5 Dissertation Organization

The rest of this dissertation is organized as follows.

Chapter 2 presents an overview on existing approaches on automatic resource management for cloud applications in virtual clusters, performance optimization in physical clusters.

In Chapter 3, we propose a novel adaptive fuzzy control approach to tune resource allocation for applications in cloud. We have conducted extensive experiments to evaluate the performance, and compared it against three other methods. Our experiments indicate that Adaptive fuzzy control has the best stability, shortest settling time and minimum overshoot.

In Chapter 4, we propose that IaaS should dynamically balance multiple virtualized resources across all tiers based on application SLOs. The SLOs are defined in terms of response time. We present BConf, a QoS-aware framework for balanced configuration of multi-resources that integrates model predictive control (MPC) and adaptive proportional integral control (adaptive PI). BConf takes advantage of MPC to actively balance resources for co-hosted applications based on a novel resource metric. Both black-box and gray-box models are developed to capture performance to resources relationship for comparison. Within the framework, constrained resources are arbitrated in a coordinated way to effectively restrict propagation of configuration errors.

In Chapter 5, we propose credit sharing (CS) policy that guarantees FJCTs. In addition to accelerating short jobs, the cluster scheduler is enabled to pack tasks of short jobs, getting system efficiency nearly optimized since task demands of short jobs are in the same magnitude as those of long jobs. Experiments in both simulation and implementation show that it is able to optimize system efficiency and accelerate JCTs while guaranteeing that each job completes faster than, or at least no later than DRF.

Chapter 6 concludes this dissertation with summaries of our approaches and points out future directions.

CHAPTER 2 RELATED WORK

2.1 Automatic Resource Management

With the proliferation of virtualization technologies, traditional resource management has evolved into virtualized resource configuration. Note that current commercial IaaS providers, such as EC2, only sells fix-sized VMs without fine-grained VM management. Many methods are applied to virtualized resource management. Among them, control theory has widespread application.

2.1.1 Control Objective and Theory

Control objective can be reference input and optimization objective. They corresponds to regulatory control and optimal control respectively. Regulatory control problem can transform to optimal control problem, reverse is not true. In resource management, the control objective is dynamically partitioning available virtualized resource across multiple application workloads such that each workload meets its QoS objectives at minimal cost [44]. The objective includes four aspects:

QoS guarantee: Given enough resource, performance targets should be achieved. Otherwise, performance differentiation should be allowed.

High utilization: Minimal cost implies high utilization.

Stability: Performance should be stable without large variations.

Short settling time: When transient workload appears, performance targets should be achieved in a few control intervals.

Control means selecting right input to achieve desired behavior or reference value. Figure 2.1 illustrates a standard feedback control loop. The control objective is the reference input. We refer to the system being controlled as the target system, which has a set of metrics of interest, referred to as measured output and a set of control knobs, referred to as control input. The effectiveness of the control system is largely determined by having a predictable relationship between control inputs and measured outputs. This relationship

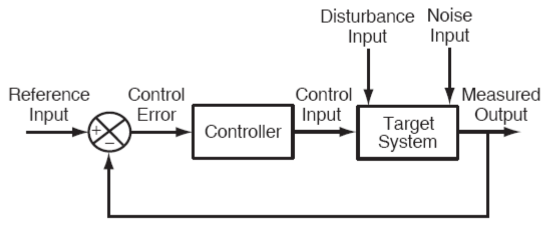


Figure 2.1: Standard feedback control loop

can be impacted by disturbance and noise, something out of control. Controller periodically adjusts the value of control input such that the measured output can match the reference input. That is, it aims to maintain the difference between the two, referred to as control error, at zero [44] [100]. The period is referred to as control interval.

In resource management, the reference input is 90th percentile response time. The target system is the resource pool in the form of VMs in data center. The measured output and control input are measured performance in terms of throughput or response time and VM cap values respectively. The disturbance is the workload and interference of workloads. Noise is distorted measurement.

2.2 Configuration of A Single Virtualized Resource

Most of recent autonomic virtual machine management have been designed to scale a single type of resource, mainly CPU.

2.2.1 SISO Control-based Approaches

Early work [67] [53] centered on the tuning of CPU utilizations to maintain high utilizations. Padala et al. [67] designed an adaptive PI controller to separately regulate CPU utilization for each tier of an application. The work [94] partitioned CPU resources across all tiers of an application based on queuing theory to satisfy the response time target. It is used for applications with all tiers resident in the same machine. Our approach is different from theirs because our work is not constrained from placement.

These methods use single-input and single-output (SISO) techniques. While SISO can guarantee the SLO, it is limited to the regulation of single resource, and hardly applicable to the configuration of multiple resources.

2.2.2 Heuristic-based Approaches

Kalyvianaki et al. [53] applied kalman filter theory to track CPU utilizations across different tiers. The authors in [38] shared the same resource minimization objective as ours. A combined signal processing and statistical learning algorithms [38] was used to predict the requirements of a single resource type. The work [77] extended this method by translating the resource savings to energy reduction. The translation technique is complementary to ours because balancing of multi-resources aims to derive efficient solutions.

2.2.3 RL-based Approaches

The authors [74] configured CPU and memory resources separately by Reinforcement Learning (RL) approaches. One advantage of this method is that it is application-agnostic because it does not need to know the internals of the system. However, despite of no need to setting parameters, it cannot assure SLO stably due to the limitation of RL that cannot guarantee the convergence to performance target.

2.3 Configuration of Multiple Virtualized Resources

Compared with tuning of a single resource, multi-resource configuration provides more space for optimizing resource combinations.

2.3.1 MIMO Control-based Approaches

There are multi-input and multi-output (MIMO) techniques, such as linear quadratic regulation (LQR) [30] in regulatory control. However, LQR involves a lot of manual work, and is not adaptive to cloud dynamics.

The work in [66] is one of the first to introduce MPC for the partitioning of multiple resources. It applied a black-box ARMA model with success to automatically manipulate CPU share and disk I/O bandwidth for multiple applications across multiple machines. Their work is suitable to assure QoS with respect to throughput, but not applicable to response time guarantee. Moreover, though it is able to respond to bottleneck shift, it cannot actively maintain balanced configuration. In contrast, our work is different from their work because we assure stable and responsive control of response time by integrating MPC with adaptive PI. The control techniques used in our BConf framework specifically address the issue of balancing of multi-resources for cloud environments.

The work [62] designed a MIMO controller based on a black-box model that characterizes the interference effect. Efficiency control is invoked to promote QoS if more CPU resource is available. Their work is complementary to ours because the resource savings can also be used to promote QoS. MPC was also widely employed for the purpose of power control [84] [37].

2.3.2 RL-based Approaches

The authors [73] [95] applied reinforcement learning (RL) for multiple resources configuration by Though considering throughput and response time as a whole, it may not guarantee performance target during trial-and-error process due to no theoretical guarantee of stability. Moreover, the coarse-grained discrete configuration by RL may limit the optimization extent of resource usage since response time is sensitive to fine-grained tuning. In comparison, the control techniques have theoretical guarantee of performance target.

2.3.3 VM Migration

The work in [91] is closely related to our work in that it took black-box and gray-box strategies to configure multiple resources. However, it is different from ours because it used VM migration mechanisms.

Previous studies on virtual resource management did not consider the balancing between resources and tiers of multi-tier applications, neither is response time target guaranteed. To our best knowledge, BConf framework represents the first for balanced configuration of multi-resources, guaranteeing response time in virtual clusters.

2.4 Fairness and Performance Optimization in Physical Clusters

The computing system has evolved from a single resource time sharing processor [34], a slot-based sharing cluster [2] [47] to a multi-resource sharing cluster [46] [82] consists of many commodity servers. Clusters are different from time sharing processors in light that nodes are shared in both the temporal and spatial dimensions. The optimization of system efficiency is not trivial in shared clusters because data is scattered over nodes. Multi-resource sharing is different from a single resource sharing because of heterogeneous task demands. It introduces one more dimension to achieve the conflicting objectives of fairness, minimal completion times and system efficiency.

2.4.1 Fairness and Minimal Completion Time

Our work is closely related to Fair Sojourn Protocol (FS) [34], which addresses the conflict between fairness and minimal completion times in a single resource time sharing system. It guarantees that each job completes within FJCT, the one resulting from fair processor sharing policy, such as round-robin. It sorts jobs in the increasing order of FJCTs, and schedules the job with the shortest FJCT to minimize JCTs. We adopts the same philosophy in guaranteeing FJCTS. However, our work is different from FS because we address the challenges to optimize system efficiency that multi-resource sharing has introduced to clusters.

2.4.2 Fairness and Locality in Slot-based Clusters

To resolve the conflict between fairness and data locality in shared clusters, widely deployed techniques [49] [97] employ relaxed max-min fairness based on waiting and time-out.

While Quincy [49] and delay scheduling [97] achieve nearly optimal data locality, they are specially designed for slot-based systems, with the heterogeneity of task demands ignored. As a result, they may cause resource contention by assigning a task a slot with available resources less than the task demand, or end up with low resource utilization if the aggregate demand of tasks is lower than the total capacity of the allocated slots in some resource dimension(s). In addition, it is inefficient and/or ineffective to extend them to multi-resource clusters. Since Quincy [49] necessitates a global solver to get the optimal schedule, generalizing it to multi-resource environment is computationally prohibitive. Delay scheduling [97] was designed for head-of-queue job wait for a few seconds when it cannot launch a local task. However, in a fine-grain multi-resource cluster, the head-of-queue job may end up with launching a non-local task instead if during the waiting period, its resource demands is too large to fit into any of its preferred nodes.

2.4.3 Performance Optimization in Multi-Resource Clusters

To satisfy heterogeneous task demands, modern clusters [46] [82] provides multi-resource sharing by encapsulating task demands to containers. To extend max-min fairness to multi-resource environment, DRF [35] was proposed to maximize the minimal dominant share. DRF degrades performance because it cannot balance the resource utilization in each dimension. Tetris was designed [39] to make tradeoff between the three objectives. Though it enhances system efficiency and JCTs, its percentile knob limits the optimization of JCTs and cannot guarantee FJCTs. By contrast, our theorem-proof credit sharing policy guarantees FJCTs. The scheduler based on CS policy optimizes both system efficiency and minimal JCTs.

CHAPTER 3 ADAPTIVE CONFIGURATION OF A SINGLE RESOURCE

3.1 Introduction

The performance isolation feature and pay-as-you-go price model allows multiple applications to consolidate together in shared virtualized environment to increase utilization, thus reducing infrastructure and operating cost. Moreover, elastic resource management is provided to cope with unexpected resource demand due to transient workload.

Cloud provider may provide multiple classes of service, each with its own characteristics and requirements. SLAs specify both performance targets, known as service level objectives (SLOs), and financial consequences for meeting or failing to meet those targets. It becomes a trend that cloud users are charged by performance due to the interference effect of co-hosted applications in shared infrastructure [62]. Over-provisioning increases resource cost, hence resource waste and underutilization, or under-provisioning leads to SLA violations.

As a result, data center administrators are faced with increasing challenges to meet SLOs in the presence of workload dynamics and cloud dynamics of unpredictable interactions across many applications. These challenges are:

Workload dynamics: The intensity and the mix of typical application workloads vary over the lifetime of an application. As a result, the demands for individual resources also change over time. This implies that static resource allocation can't efficiently work.

Nonlinearity between SLOs and resource allocation: Nonlinearity makes it nontrivial to convert SLOs to corresponding resource share in the shared virtualized platform.

Unpredictable interference across co-hosted applications. Due to lack of perfect performance isolation, applications have unexpected impact on each other while contending for shared resource. Interference itself doesn't pose big challenge, instead the uncertainty of interference causes big trouble. For example, given steady workloads for all applications, the resource distribution satisfying SLOs currently may not meet them next time due to interference from contention. It feels like the impact of uncertain network delay on performance of distributed network system.

This paper proposes a novel adaptive fuzzy control approach to tune resource allocation for applications in cloud. We have conducted extensive experiments to evaluate the performance, and compared it against other three control methods in terms of stability, overshoot, settling time. Our experiments indicate that Adaptive fuzzy control has the best stability, shortest settling time and minimum overshoot.

The structure of rest paper is organized as follows. Section II presents control objectives and theory. Section III discusses the four control methods. Section IV introduces experimental methodology. Section V evaluates and compares the performance in terms of stability and responsiveness. Section VI concludes the paper.

3.2 Adaptive Fuzzy Control

Due to workload and cloud dynamics, relationship can often be linearized at small operating points. It is well known that the linear approximation of a nonlinear system is accurate only within the neighborhood of the operating point. Abrupt changes in workload traffics and the nondeterminism in VM capacity can possibly make the simple linearizion inappropriate. Instead of modeling the system in mathematical equations, fuzzy control employs the control rules of conditional linguistic statements on the relationship of VM capacity and the high-level objectives [52]. The fuzzy control rules are able to describe human expert's experiences and the rule base is easily updated by adding new knowledge. There are works that applied fuzzy control to QoS guarantees in web server [87] and computer networks [23] with success.

Figure 3.1 illustrates the structure of the adaptive Fuzzy Controller. It consists of two layers, namely the fuzzy logic controller, and the output adapter. The capacity allocated in control interval $k + 1$, denoted by $u(k + 1)$, is adjusted according to its error $e(k)$ (i.e., the normalized difference between the reference value and the achieved one) and change of error $\Delta e(k)$ in previous control interval k using a set of control rules embedded in the fuzzy logic controller. $e(k)$ and $\Delta e(k)$ are calculated using the reference value r and the observed value $y(k)$. For the stability of the control system, we define the normalized error $e(k)$ in a range of $[-1, 1]$:

$$e(k) = \begin{cases} \dfrac{r - y(k)}{r} & 0 \le y(k) \le 2r; \\ -1 & y(k) > 2r. \end{cases} \tag{3.1}$$

Based on these, the controller calculates capacity adjustment $\Delta u(k)$ for next control interval. The calculated resource adjustment is then fed into the next layer gain scheduler.

The fuzzy logic controller contains four building blocks. The actual fuzzy logic is implemented as a set of *If-Then* rules about quantified control knowledge about how to adjust the capacity according to $e(k)$ and $\Delta e(k)$. The fuzzification interface converts controller inputs into certainties in numeric values of the input membership functions. The inference mechanism activates the rule-base and applies fuzzy rules according to the fuzzified inputs and generates the fuzzy conclusions for the defuzzification interface. The defuzzification interface converts fuzzy conclusions into the change of capacity in numeric value.

The STFC is built on the static fuzzy logic controller by adding output adaptor. Thus, the capacity allocated to the VM during management interval $k + 1$ is

$$u(k+1) = u(k) + |K_{\Delta u}|\Delta u(k) = \int K_{\Delta u} \Delta u(k) dk. \tag{3.2}$$

3.2.1 Design of the rule base

The design objective is to translate human expert's knowledge into a set of control rules to control the VM capacity without a model of the dynamic cloud environment. In the fuzzy logic controller, the control rules are defined using linguistic variables. For brevity,

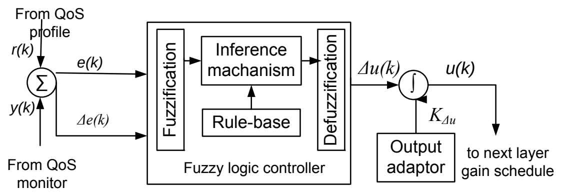
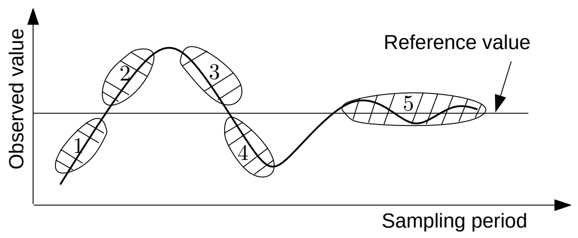
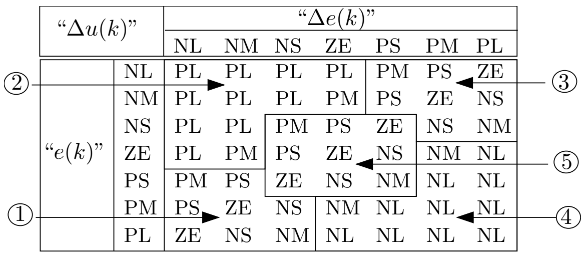


Figure 3.1: The structure of the STFC.



(a) The control effect



"$\Delta u(k)$"		"$\Delta e(k)$"						
		NL	NM	NS	ZE	PS	PM	PL
"$e(k)$"	NL	PL	PL	PL	PL	PM	PS	ZE
	NM	PL	PL	PL	PM	PS	ZE	NS
	NS	PL	PL	PM	PS	ZE	NS	NM
	ZE	PL	PM	PS	ZE	NS	NM	NL
	PS	PM	PS	ZE	NS	NM	NL	NL
	PM	PS	ZE	NS	NM	NL	NL	NL
	PL	ZE	NS	NM	NL	NL	NL	NL

(b) The rule table

Figure 3.2: Design of the fuzzy control rules.

linguistic variables "$e(k)$", "$\Delta e(k)$", and "$\Delta u(k)$" are used to describe $e(k)$, $\Delta e(k)$, and $\Delta u(k)$, respectively. The linguistic variables assume linguistic values NL, NM, NS, ZE, PS, PM, and PL. Their meanings are shown in Table 3.1. They indicate the sign and the size in relation to the other linguistic values.

Figure 3.2(a) gives an simple illustration of typical control effect. In this figure, we identify five zones with different characteristics. Zone 1 and 3 are characterized with opposite signs of $e(k)$ and $\Delta e(k)$. That is, in Zone 1, $e(k)$ is positive and $\Delta e(k)$ is negative; in Zone

Table 3.1: The description of linguistic values.

Linguistic value	Description
NL	negative large
NM	negative medium
NS	negative small
ZE	zero
PS	positive small
PM	positive medium
PL	positive large

3, $e(k)$ is negative and $\Delta e(k)$ is positive. In these two zones, it can be observed that the error is self-correcting and the achieved value is moving toward the reference value. Thus, $\Delta u(k)$ needs to be set either to speed up or to slow down current trend.

Zone 2 and 4 are characterized with the same signs of $e(k)$ and $\Delta e(k)$. That is, in Zone 2, $e(k)$ is negative and $\Delta e(k)$ is negative; in Zone 4, $e(k)$ is positive and $\Delta e(k)$ is positive. Different from Zone 1 and Zone 3, in these two zones, the error is not self-correcting and the achieved value is moving away from the reference value. Therefore, $\Delta u(k)$ should be set to reverse current trend.

Zone 5 is characterized with rather small magnitudes of $e(k)$ and $\Delta e(k)$. Therefore, the system is at a steady state and $\Delta u(k)$ should be set to maintain current state and correct small deviations from the reference value.The resulted control rules are summarized in Figure 3.2(b). For example, when "$e(k)$" and "$\Delta e(k)$" are NL and PS, "$\Delta u(k)$" is set to PM.

3.2.2 Fuzzification, inference and defuzzification

At the heart of a fuzzy controller are the membership functions that quantify the *certainty* (between 0 and 1) that an input fall in the corresponding ranges. We select the "triangle" membership function, which is the most widely used in practice. We set the width and height of the "triangle" membership function to be 2/3 and 1 respectively. See our previous work [87] for design details of the membership function. The fuzzification component translates the inputs into corresponding certainty in numeric values of the membership functions. Let $\mu_m(e(k))$ denote the certainty of $e(k)$ of the mth membership function, and $\mu_n(\Delta e(k))$ the certainty of $\Delta e(k)$ of the nth membership function.

The inference mechanism is to determine which rules should be activated and what are conclusions. Let $\mu(m,n)$ denote the certainty of $rule(m,n)$. The *and* operation in the premise is calculated via *minimum*:

$$\mu(m,n) = \min\{\mu_m(e(k)), \mu_n(\Delta e(k))\}. \tag{3.3}$$

Based on the outputs of the inference mechanism, the defuzzification component calculates the fuzzy controller output, which is a combination of multiple control rules, using "center average" method. Let $b(m,n)$ denote the center of membership function of the consequent of $rule(m,n)$. In this case, it is where the membership function reaches its peak. The fuzzy control output is

$$\Delta u(k) = \frac{\sum_{m,n} b(m,n) \cdot \mu(m,n)}{\sum_{m,n} \mu(m,n)}. \tag{3.4}$$

3.2.3 Design of the adapter

The fuzzy logic controller only defines the basic control rules according to the inputs of $e(k)$ and $\Delta e(k)$. It outputs the sign and magnitude of the capacity adjustment $\Delta u(k)$. With cloud dynamics, there could be a lot of fluctuations in the control effect. To achieve accurate, responsive and stable control, the following practical issues should be addressed.

When there are fluctuations in control effect due to abrupt workload or capacity changes, the control should be responsive enough to correct the capacity discrepancy within a small number of steps.

To address the above issues, we design the adaptive controller to have adaptive output magnitude. The adaptive features are realized by dynamically changing output adapter. The adapter implements heuristic control knowledge as follows:

$$K_{\Delta u(k)} = |\frac{c}{\lambda} \cdot e(k)|, \tag{3.5}$$

where c is the capacity allocation for a specific resource. For example, c can be the cap value of the CPU allocation in a Xen platform. The adaptor follows a heuristic rule that the maximum capacity adjustment should not exceed half of current capacity for stability and should be proportional to the control error for adaptability. Note that the direction of the capacity adjustment is still determined by the fuzzy logic.

3.3 Testbed and Experimentation

In this section, we present the testbed and experimental configuration. We compared fuzzy control with adaptive PI, ARMA and kalman-filter from the perspective of stability and responsiveness of control performance.

We selected TPC-W [4] as the host application. TPC-W is an E-Commerce benchmark that models after an online bookstore. It consists of three tiers, Apache web server (version 1.3.11), Tomcat (version 5.5.20) [81] application and MySQL(version 5.0.45) [3] database servers. Apache and Tomcat server are encapsulated into one VM forming a unified front-end, MySQL server into one DB VM. TPC-W defines three different traffic mixes: shopping, browsing and ordering mix, which require different amount of resource. It is empirically determined that the DB tier is the bottleneck under the browsing workload. The CPU resource is changed by CAP value.

Our testbed consists of two virtualized servers, client and NFS servers. The physical machines for virtual hosting were two DELL servers each with two Intel Xeon X5650 CPUs and 32 GB memory. Each CPU has 6 cores with hyper-threading enabled resulting a total capacity of 24 logical CPUs. The front-end VM and back-end DB VM were hosted on separate machines. We configured the front-end VM with 8 core and 4 GB memory. The DB VM, with 8 core and 2 GB memory, resided on the other machine. We used a number of client machines each with 8 cores and 8 GB memory to generate workload for the TPC-W. The NFS server used a RAID5 partition to serve the VM disk images. We used Xen version

4.0 as our virtualization environment. `dom0` and guest VMs were running Linux kernel 2.6.32 and 2.6.18, respectively. All the severs were connected by Gigabit Ethernet network.

We evaluated these methods on the DB tier since it tends to be a bottleneck of this application. We used browsing mix as the workload, since it is more CPU intensive than the other two mixes. The CPU resource allocators were implemented as a user-level daemon in the virtual host (i.e. `dom0` in a Xen environment). It takes the measured application-level performance and the performance objective as input and calculates the capacity adjustment to Xen's management interface. The control interval is set to 30 seconds for all the experiments.

We conducted two experiments on the platform. All these experiments are specifically designed to test the abilities of these controllers.

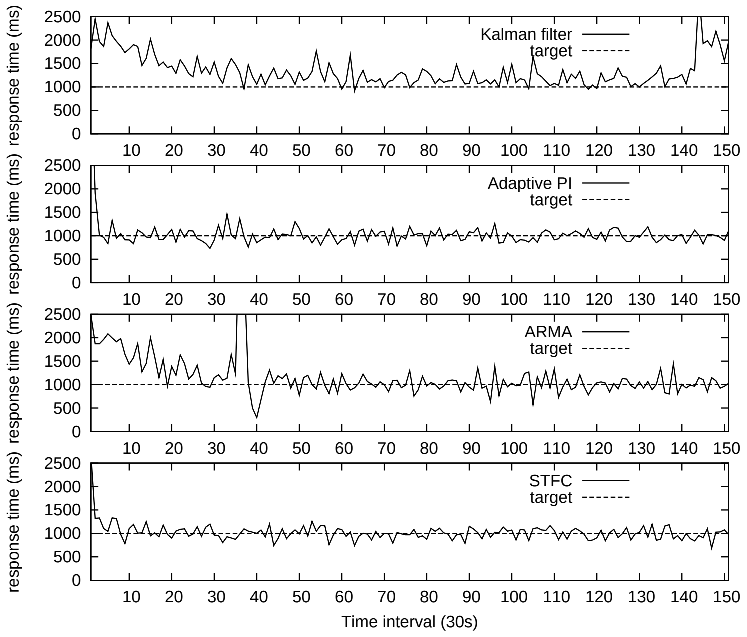
1, we investigated the accuracy, overshoot and stability of these methods while enforcing performance targets in terms of average response time.

2, we evaluated responsiveness of these controllers while adapting resource allocations to time-varying workloads.

3.4 Evaluation Results

Response times behave nonlinearly with respect to resource allocations especially when the system is in a heavy workload state. We selected the target of all the controllers to be 1 second except that we followed the controller in [53] and set the Kalman filter's target c to be 96% CPU utilization, which translates to approximately the 1-second response time. The moving window of the ARMA controller was set to 20 samples.

Experiments are designed to study the efficacy of these methods in the determination of proper CPU capacity settings under both static and dynamic workloads.



(a) Static workload

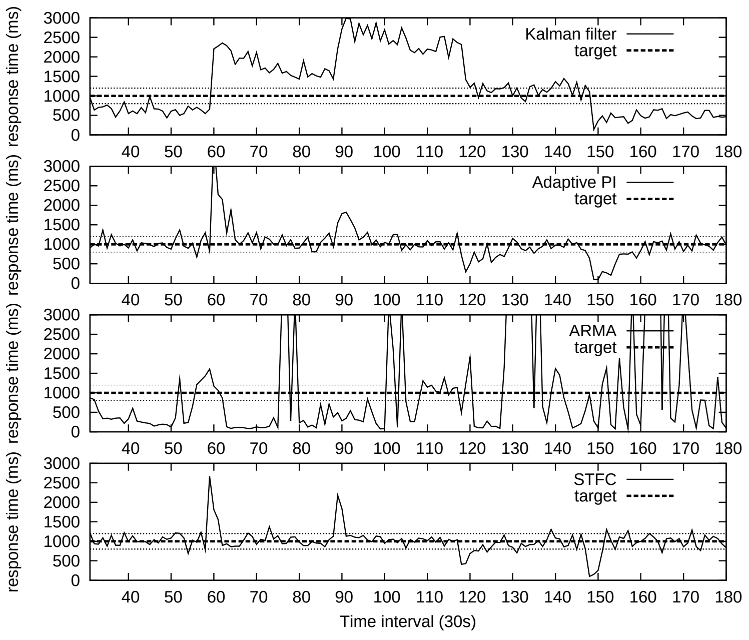
Figure 3.3: Performance comparison of STFC, Kalman filter, adaptive-PI and ARMA in static workload.

3.4.1 ARMA Prediction Accuracy

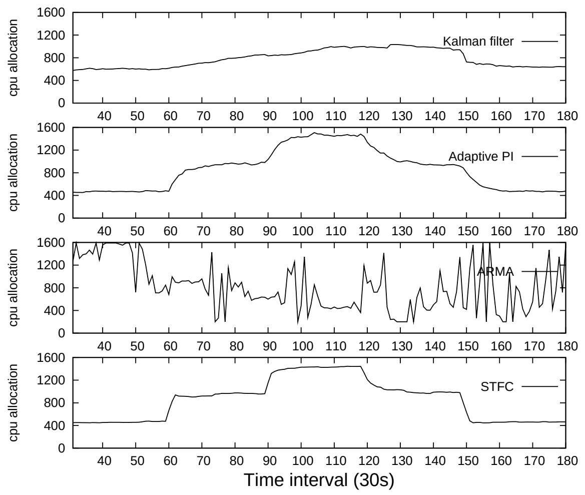
The online performance predictor can adapt the ARMA model parameters to the change of the system states to some extent. We should first study the accuracy of the predictor before applying this model.

We conducted 4 experiments to show the adaptivity of the predictor by running a static workload with client number as 200. In all the experiments, we allocated the VCPU cap to the VM from 500 to 1600, the difference of these experiments lies in the step length. The step length in the first experiment is a fixed value 32(2% of the total cap), second is 64(4% of the total cap), third and fourth are random variable range from $[-160, 160]$(10% amplitude), $[-220, 420]$(20% amplitude) respectively, conforming to uniform distribution.

To assess overall prediction accuracy of the model, we used mean relative error (MRA), defined as $\frac{1}{K}\sum_{k=1}^{K}\frac{|y(k)-\tilde{y}(k)|}{\tilde{y}(k)}$, where K is the total number of samples, $y(k)$ and $\tilde{y}(k)$ denote predicted and measured value at k^{th} interval, respectively. Figure 3.5(b) shows



(a) Dynamic workload



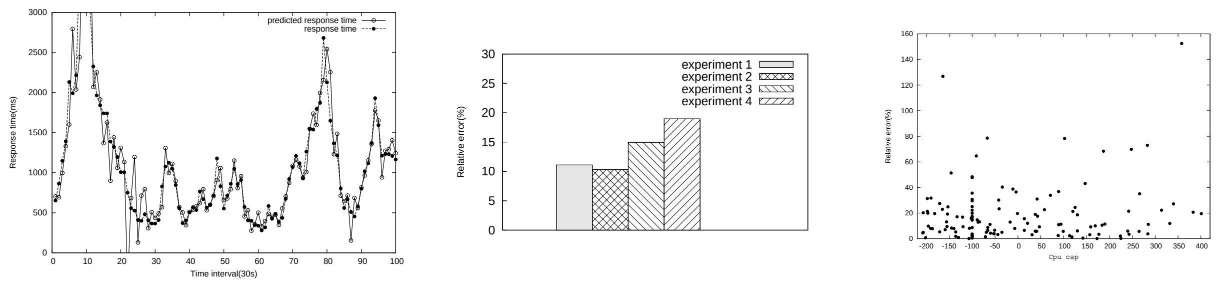
(b) Resource allocation

Figure 3.4: Performance comparison of STFC, Kalman filter, adaptive-PI and ARMA in dynamic workload.

the values of the measure for all the 4 experiments. As an example, we also show in figure 3.5(a) the measured and model-predicted response time for experiment 4. From both figures, we can see that, the model can't always predict the response time accurately, with MRA above 10% and increasing with change of CPU cap to as high as 19%. This is because response times behave nonlinearly with the resource allocation. We are also interested to know how the resource affects the performance. As an example, figure 3.5(c) shows how response time is influenced by the change of resource in experiment 4. Whatever the range of resource change is, they embodies the nonlinear property. Generally speaking, when resource varies largely, relative error will also increase drastically.

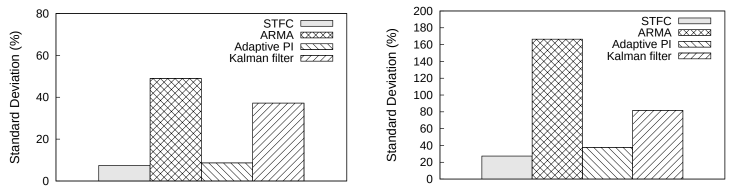
3.4.2 Stability

The workload was set to 200 browsing clients, each with a mean think time of 1 second. The DB VM has 4 VCPUs and its initial capacity was set to 6 cores (a cap of 600). Figure 3.3(a) plots the response time of different control methods with static TPC-W



(a) Predicted and Measured response time (b) Accuracy measure of ARMA model (c) Deviation with change of CPU cap

Figure 3.5: Internal workings of ARMA predictor



(a) Static workload (b) Dynamic workload

Figure 3.6: Relative deviation under static and dynamic workload.

workload. We can see that all the control methods can bring the response time close to the 1 second target but with different variety.

To quantify stability of the performance metric in terms of response time, we defined standard deviation from a reference as the metric.

$$R(e) = \frac{\sqrt{\sum_{k=1}^{n} e(k)^2/n}}{r}, \tag{3.6}$$

where r is the control objective and $e(k)$ is the error. The smaller the $R(e)$, the more the achieved response time concentrates near the target value and better the controller's performance.

Figure 3.6(a) draws the corresponding standard deviations of the controllers. STFC performed best with the standard deviation as 7.4%, Arma's is 48%, Kalman and Adaptive PI are 8.63%, 37.23%respectively. Due to the prediction inaccuracy of ARMA model, it

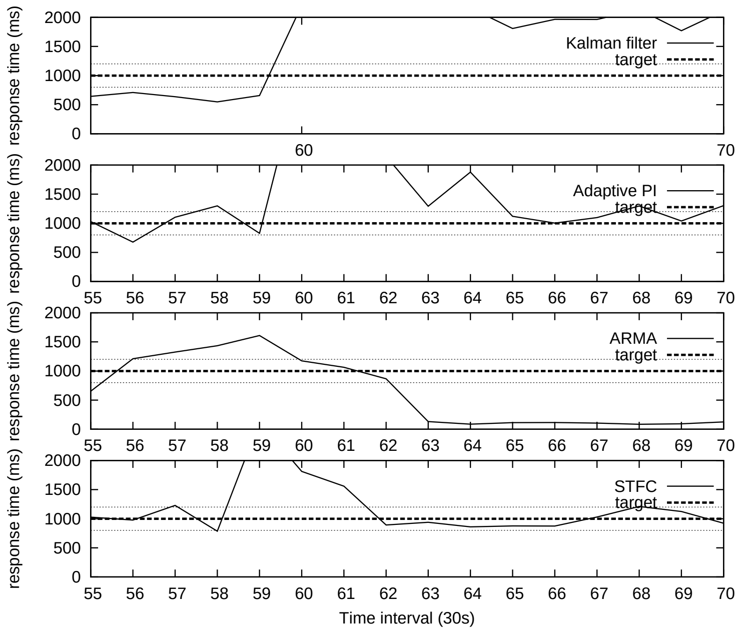


Figure 3.7: Response time around 60^{th} control intervals.

oscillates around the target largely. Kalman filter is also not stable due to the utilization tracking method and coarse estimation of variance of noise. Adaptive PI is somewhat worse than fuzzy control because it doesn't take the trend of error into consideration. The STFC is actually a PI-like controller with nonlinear operating functions while the adaptive-PI only tracks the control error.

3.4.3 Settling time and overshoot

Settling time indicates how quickly the system converges to its steady state value. Short settling times are particularly important for disturbance rejection in the presence of time-varying workloads so that convergence is obtained [43]. We instrumented the workload generators of TPC-W to change client traffic levels at run-time. Figure 3.4(a) plots the response times in a 90-minute period in which the number of clients was changed every 30 intervals. We started with 100 clients and set the client numbers at the $60th$, $90th$, $120th$ and $150th$ interval to be 200, 300, 200 and 100, respectively.

Obviously, result of ARMA and Kalman filter didn't converge to target, as indicated by the heavy dashed line. Result of ARMA always oscillates around the target, because it may take up to M intervals (data collected in past M intervals would be used for regression) for the model to adapt to a new situation. In addition, the maximum overshoot is the largest among all of the methods. Kalman filter didn't converge either. Output deviates largely during large work load change, resulting rise of variance of process noise Q. However, kalman gain didn't increase correspondingly, which approximates to $1/c$ due to ignorable small value of variance of measurement noise R. Therefore, the responsiveness is restricted due to small kalman gain.

By contrast, adaptive PI and fuzzy control can converge to target but with different settling time, even though dynamics is nonlinear and time-varying. The settling time of the system is the time from the change in input to when the measured response time is sufficiently close to its steady-state value(as indicated by the light dashed lines). Figure 3.7 shows the settling time when workload changes at 60^{th} intervals. Around 60^{th}, adaptive PI was disturbed from 58 to 65 experiencing 7 intervals to stabilize. By contrast, STFC only spent 4 intervals. Situation is similar in other workload change period. On average, it takes about 8 intervals for Adaptive PI to adapt to the change; on the contrary, it takes only 4 intervals for our STFC to cope with workload dynamics. The overshoot in fuzzy control is smaller than adaptive PI. The advantages of STFC lies in the rule table, which explored the nonlinearity of performance to resource allocation and considered the trend of error to predict the right resource allocation.

The settling time can be reflected and accounted in the resource allocation. As shown in figure 3.4(b), we can see that at change points of workload level, i.e. 60, 90, 120,150, these methods allocated or reclaimed CPU resource to respond to such change. At the start of each period of new workload, STFC can calculate appropriate resource required by the new workload level after 1 to 3 intervals to converge the response time to desired value. On the contrary, the resource was change by Adaptive PI slower than STFC, about after 6 to 10 intervals, resulting in slow response to workload dynamics.

Figure 3.6(b) draws the corresponding standard deviations of the controllers, which can embody the settling time and overshoot to some extent. STFC performed best with standard deviation of 27%, while ARMA still behaved worst with standard deviation as high as 166%.

In conclusion, STFC outperforms other methods, which has the best stability, shortest settling time, and minimum overshoot.

3.5 Summary

We have conducted extensive experiments to compare the performance of Adaptive Fuzzy control against other three in terms of accuracy, stability and settling time. Our experiments indicate that Adaptive fuzzy control has the best stability, shortest settling time and minimum overshoot. The power is that not only does it explore the nonlinearity of resource allocation to performance, but also it takes into consideration the trend of error to estimate right resource allocation.

CHAPTER 4 DYNAMIC BALANCED CONFIGURATION OF MULTI-RESOURCES

4.1 Introduction

One of the greatest advantages of IaaS is dynamic resource scaling. This feature increases resource utilization by consolidation, while eliminates customers' up-front venture capital due to pay-as-you-go pricing model. Dynamic resource configuration is required to deal with workload dynamics produced by transient workload change, such as flash crowd effects, and cloud dynamics because of the presence of interference among resident applications. From an IaaS provider's point of view, the profit comes from two factors: service level objectives (SLOs) it has committed in service level agreements (SLAs) and resource expenditures it pays for [62]. In order to maximize the profit, the service provider must minimize resource usages without compromising SLOs.

Resource optimization lies in whether it is able to find out a balanced configuration that meets application SLOs without over-supply or under-supply of resources. For a physical machine to work optimally, the capacities of all major components must be in balance by manual configuration [22] [55] [27]. A balanced machine has the same computing time as the I/O time [55]. The ideal state of balance is no less desirable for a virtual machine (VM) that runs an application or an application tier than for a physical machine. A VM is *balanced* when the incremental performance increase due to an incremental increase in the capacity of each resource allocation is equal. A balanced VM will experience a bottleneck in the resource that is reduced and bottlenecks across all other resources when one resource is increased. For example, suppose a VM needs 100 VCPU cap, 1G memory, and 5M disk I/O to be balanced. If memory is decreased to 0.5G, then it becomes the bottleneck. If VCPU cap is increased to 200, then memory and disk I/O may become the bottlenecks. The reason is that increasing VCPU cap for a VM may cause augment of the working set, and vice versa. If the resulting additional requirements for other types, such as memory or disk I/O bandwidth, are not met, then QoS may be not improved particularly when they become bottlenecks. When a VM experiences a bottleneck, it becomes imbalanced. In an imbalanced VM, the bottlenecked resource limits application's performance, while other resources are over-provisioned thus wasted. A balanced configuration avoids bottlenecks that limit performance and makes resource allocations well utilized. Therefore, a balanced configuration maximizes resource utilization and efficiency.

Our objective is to balance multiple resources for VMs under the constraint of SLOs. However, workload dynamics poses great challenges to balancing resources. First, it is not easy to identify the over-provisioned or constrained resource type due to fluctuating workload. Second, a balanced configuration may be imbalanced later because time-varying workload results in time-varying resource requirements. The bottleneck may shift from one resource type to another within a VM, or from one VM to another one in the extension of multi-tier applications.

Moreover, capacity uncertainty caused by VMs interferences in the open shared environment [62] [65] [42] poses additional challenges to virtualized resources balancing. Even if given the same workload intensity and mix over a period, a balanced configuration also may be imbalanced at some point later within the period. The reason is that the capacity of one resource changes (increases or decreases), or the capacities of several resources change but with different extents. Even if with the same extent and configuration is still balanced, performance level is not the same since capacities vary.

Performance in SLO is measured normally in two main metrics: response time and throughput. In many interactive benchmarks, such as TPC-W, dynamic arrival rate is often assumed to follow a Poisson distribution. Accordingly, throughput per time unit is similar with same resource allocations. Compared with throughput, users are more concerned with response time because it reflects the performance perceived by individual clients. Unlike throughput, however, response time behaves nonlinearly with respect to resource allocations.

The nonlinear relationship between resources and performance in terms of response time adds one more dimension to the task of balanced VM configuration [75].

Most of current research on autonomic VM management devoted to configure a single type of resource [67] [53] [38] [77] [94] [75], which may be not optimal considering that other resources remain fixed or uncontrolled so that they could be over-provisioned or under-provisioned. There are a few works able to resize multiple resources [73] [66]. However, their approaches mainly focused on partitioning the total shared resources among co-hosted applications to satisfy SLOs, whether resources are imbalanced or not. Moreover, throughput was considered as the main performance metric. They are hardly applicable to assure stability and responsiveness of response time since it can be significantly different even if throughput is comparable between two different operating points [68].

Multi-tier applications further complicate the problem due to chain reaction during resource contention. An imbalanced configuration for one application may lead to an imbalanced configuration for one tier of another application co-hosted on the same physical server by over-provisioning one or several resources which are needed for the tier of that application to achieve balanced configuration. This may lead to an imbalanced configuration of the other tier(s) of that application residing in a different physical machine by over-provisioning resources to achieve its QoS target. As a result, the error of an imbalanced configuration on one physical machine spreads to another physical machine, and even to the whole virtual cluster. However, no existing work coordinates tiers within an application to restrict error propagation. This motivated us to consider an application as a configuration unit to coordinate configuration among different resources and tiers during contention.

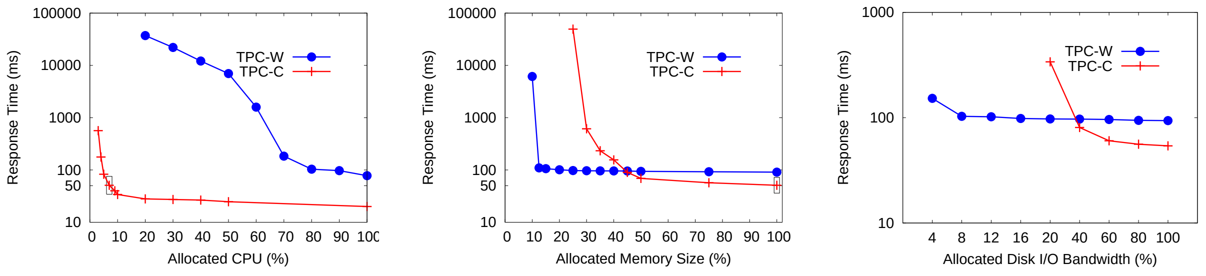
In this work, we propose that IaaS should dynamically balance multiple virtualized resources across all tiers based on application SLOs. The SLOs are defined in terms of response time. We present BConf, a QoS-aware framework for balanced configuration of multi-resources that integrates model predictive control (MPC) and adaptive proportional integral control (adaptive PI). BConf takes advantage of MPC to actively balance resources for co-hosted applications based on a novel resource metric. Both black-box and gray-box models are developed to capture performance to resources relationship for comparison. The gray-box model is built on generic OS-level metrics and hardware events in addition to resource actuators and performance. The resource penalty is introduced to measure the imbalanced degree of a configuration based on the model. It effectively punishes decisions using an imbalanced configuration. Using the model and the metric, BConf dynamically balances resources by minimizing the resource penalty for each resident application under the constraint of SLOs. To deal with the overshoot problem with MPC and improve stability of QoS, adaptive PI is employed to coordinate with MPC by guiding the optimization space to a potential region that shows promise in QoS satisfaction and resource efficiency. Within the framework, constrained resources are arbitrated in a coordinated way to effectively restrict propagation of configuration errors.

We summarize the main contributions of this paper as follows:

1. We propose an integrated MPC and adaptive PI approach (IMAP) for balanced or near balanced provisioning of multiple resources. MPC coordinates with adaptive PI to balance resources. While MPC optimizes resource efficiency and utilization, adaptive PI ensures stable and responsive QoS control.

2. For the performance prediction in the IMAP, both black-box and gray-box models are developed for comparison. A black-box model is simpler to build and more general for applications than the gray-box, whereas the gray-box model is more accurate in performance prediction.

3. We employ a hierarchical way to resolve resource contention. Instead of being allocated individually, the constrained resources are distributed as a whole for an application, preventing the error with an imbalanced configuration from spreading to the whole virtual cluster.



(a) CPU variation (b) Memory size variation (c) Disk I/O bandwidth variation

Figure 4.1: The impacts of resource allocations on response time.

4. We design and implement BConf prototype in a Xen-based cloud testbed. Compared with a representative resource partition approach, BConf reduces resource usages by up to 50%, improves stability by more than 35.6%, and has a much shorter settling time. BConf effectively coordinates resources during resource contention. The saved resources can be used either to pack more workloads, or to promote performance levels.

The rest of the paper is organized as follows. Section II presents the motivation. Section III overviews BConf framework. Section IV introduces the IMAP controller. Section V designs resource arbitrator. Section VI describes the experimentation methodology. Section VII evaluates the performance in terms of resource efficiency, stability, settling time, and overall performance during resource contention. Related work is summarized in Section VIII. We conclude this paper in Section IX.

4.2 Motivation

To illustrate how each resource influences performance, we conducted a set of experiments that used two benchmarks, TPC-W and TPC-C. Details of the testbed and benchmark descriptions can be found in Section VI. The response time reference values or targets were set to 100ms and 50ms for TPC-W and TPC-C respectively. Unless otherwise mentioned, TPC-W was assigned 80% CPU, 25% memory, and 16% disk I/O bandwidth, and TPC-C 6% CPU, 50% memory and 60% bandwidth. Then we changed each of the resource allocations, one at a time, to study the impact of the resource on each application's response time.

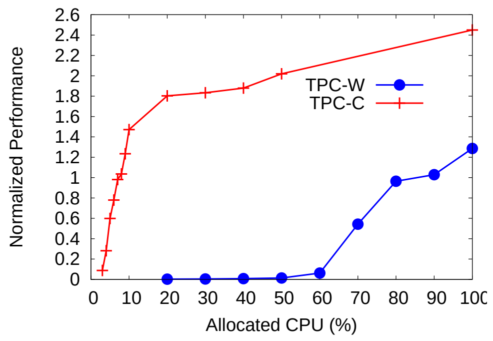


Figure 4.2: Normalized performance vs. CPU.

Figures 4.1 plot the measured response time (MRT) against the amount of resources (CPU, memory, disk I/O).

Several observations are obtained from these figures. First, the impact of resources on the applications' MRT varies from resource to resource and from application to application. Second, it is possible to satisfy the same target for an application by different configurations. For example, as shown in the rectangles, TPC-C is able to achieve 50ms response time by using either 7% CPU and 50% memory size in Figure 4.1(a), or 6% CPU and 100% memory size in Figure 4.1(b). However, the first configuration is more balanced than the second because the bottleneck lies in the CPU.

As said before, response time does not behave linearly with resource allocations. However, there is a reciprocal relationship between the response time and resources [86]. To illustrate the reciprocal relationship, Figure 4.2 plots the normalized performance. It is defined as the reference value by the response time. The larger the normalized performance, the shorter the response time. Though the normalized performance is not linear in the whole operating space, it is nearly piecewise linear in the under-load and overload conditions respectively, such as the linear performance in TPC-W within [0, 10%] and [10%, 100%] CPU share ranges. The piecewise linearity also applies to memory and I/O bandwidth.

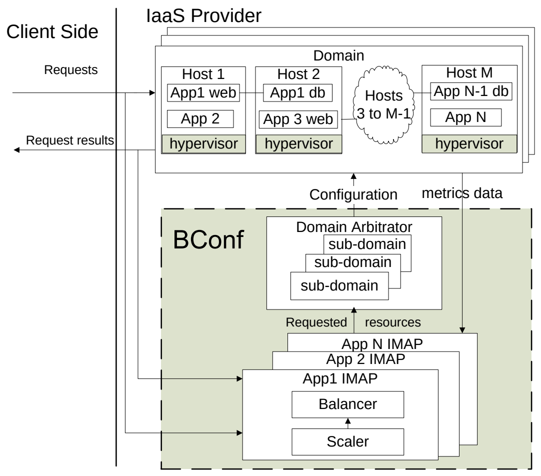


Figure 4.3: System architecture of BConf.

4.3 The BConf Framework

BConf can balance any resource to satisfy SLOs with respect to average response time. After applications are encapsulated to VMs and placed onto the virtualized hosts initially, BConf balances resources dynamically among co-hosted applications to satisfy SLOs. The placement may place a multi-tier application to multiple physical hosts with each tier residing in a different host. The placement is a graph, in which each host is a node and each multi-tier application spanning multiple hosts is an edge. We call the maximal connected subgraph a domain. BConf configures resources within a domain. It assumes that workload migration rests on a separate replacement or migration system when resources capacity can not meet SLOs.

In order to address the challenges in section I, BConf framework has been designed as a two-layer architecture, as shown in Figure 4.3. The lower layer calculates the requested resources to satisfy SLOs. It includes a set of application IMAP controllers (AppIMAPs). The upper layer is a domain arbitrator that determines the final configuration during resource contention. At first, the AppIMAP sets SLO and configuration parameters for the corresponding application. During each control interval, the real-time performance of each application is collected to the corresponding AppIMAP. The AppIMAP consists of a balancer and a scaler. The balancer tunes resource configuration in a balanced way by MPC based on a performance model established in Section IV. The resource penalty is defined to measure the imbalanced degree of a configuration. The scaler narrows resource variation to the promising region by adaptive PI control. During resource contention, constrained resources are distributed as a whole to each application by the domain arbitrator. The final configurations are conveyed to the corresponding hosts to regulate resources. In Sections IV and V, we will describe the IMAP controller and the domain arbitrator respectively.

4.4 The IMAP Controller

For easy reference, we use following mathematical symbols for key parameters and variables in the paper. R is a set of all resource types configured, e.g., $R = \{$*cpu, memory, disk I/O*$\}$; T_a is a set of all tiers of application a, e.g. T_a= *{web, db}*; $y_a(k)$ is the response time of application a measured in interval k; $\hat{y}_a$ is response time target for application a; $\bar{y}_a(k)$ is the normalized performance for application a in interval k, and $\bar{y}_a(k) = \hat{y}_a/y_a(k)$; $w_{a,r,t}(k)$ is requested allocation of resource type r to tier t of application a in interval k that is normalized to the total shared capacity of resource r, $0 \leq w_{a,r,t}(k) \leq 1$; $\boldsymbol{w}_a(k)$ is a column vector of all the requested resource allocations, and all vectors are in bold.

There are three challenges to design an effective IMAP controller. First, the balancer configures resources based on a performance model. Due to the nonlinearity with response time to resource relationship, it is challenging to predict response time accurately. Nevertheless, we take advantage of OS and hardware metrics to establish a gray-box model, which predicts it more accurately than a black-box model. Second, different resources are not directly comparable to each other. Only when they are comparable can they be balanced. Thus, we need a way to formulate different resources in the same unit for encoding them into the resource metric. Third, there is overshooting problem with MPC. It is challenging to guarantee the response time stably and responsively.

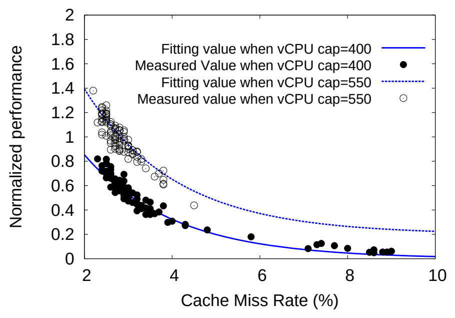


Figure 4.4: TPC-W performance with the change of last-level cache miss rate.

In order to compare different resources, one way is to normalize each resource to its corresponding part in the balanced configuration. For example, assume a balanced configuration is (100 VCPU cap, 1G memory, and 5M I/O), then the configuration (200 VCPU cap, 1.5G memory, 6M I/O) is normalized to (2, 1.5, 1.2). However, we do not know the balanced configuration and the balanced configuration changes with time. As a result, we normalize the resource allocation to the larger one between the total shared capacity of resource r and the same multiple of its average entitlement. We use the symbol $\bar{w}_{a,r,t}(k)$, which is the normalized requested allocation of resource type r to tier t of application a in the interval k, and $0 \leq \bar{w}_{a,r,t}$.

4.4.1 Performance Model

It is very important to select a "less wrong" model for MPC [43]. Nevertheless, more features do not necessarily render more accurate model. Feature selection is a key.

4.4.1.1 Gray-box Model

A system provides a rich set of OS-level metrics and hardware performance counters that represent the application's performance at running time. Performance varies greatly with resource load conditions, so utilizations are measured. Moreover, data locality has a huge impact on the response time of applications, so cache related performance counters are used. Finally, one more counter that seems to be related to response time are also checked. To establish the performance model, we use following metrics:

- Resource utilization $u_{a,r,t}$: It is utilization of resource type r in tier t of application a, $0 \leq u_{a,r,t} \leq 1$. They are used to estimate and distinguish resource load conditions.
- Last-level Cache Misses ($miss_a$): This metric measures the accesses of memory, as well as disk, when LLC cannot hold the whole working set for application a.
- Last-level Cache References (ref_a): It measures the numbers of data accesses for application a.
- CPU I/O wait ($iow_{a,t}$): When CPU stalls due to an outstanding of disk I/O requests in the VM where tier t of application a resides, response time will increase.

We first study the relationship of normalized performance with Last-level Cache Miss Rate (LCMR), i.e. $\dfrac{miss_a}{ref_a}$. Figure 4.4 depicts $\bar{y}_a$ against LCMR in DB tier of TPC-W with 500 clients of browsing mix, which ran exclusively with same fixed memory and disk bandwidth, and two different VCPU caps. As shown in Figure 4.4, $\bar{y}_a$ decreased exponentially with the LCMR, and increased almost linearly with resource allocations. This relationship is attributed to the memory hierarchy. So far, cache is at least one order of magnitude faster than memory, which in turn is at least five orders of magnitude faster than disk. If the application exhibits good locality, i.e., low miss rate, CPU fetches the data directly from cache most of the times. Otherwise, it needs memory access and potential disk access, which dramatically increases response time. We then add $iow_{a,t}$ to the performance relationship, but the coefficient for $iow_{a,t}$ is small, so we remove it.

Based on the exponential relationship with LCMR and linear relationship with resources observed in Figure 4.4, we predict $\bar{y}_a$ as follows.

$$\bar{y}_a(k) = \sum_{i=1}^{n} \alpha_i exp(\lambda(c_0 - c(k-i))) + \sum_{j=0}^{m-1} \mathbf{d}_j^T \bar{\mathbf{w}}_a(k-j). \tag{4.1}$$

where the parameters α_i and $\boldsymbol{d}_j^T$, the orders n and m characterize the dynamic behavior of the system; $\bar{\boldsymbol{w}}_a(k)$ is a column vector of $\bar{w}_{a,r,t}(k)$. $c(k)$ is the LCMR in the interval k. Because of the "lack of memory" property with exponential function, performance relative to the target depends on the difference of the corresponding cache miss rates. Therefore, c_0 is set as the cache miss rate that corresponds to the performance target. λ is determined by Gauss-Newton method, which solves this non-linear least squares problem in a few seconds in a testbed server (see Section IV for configuration) . In the above model, α_i and $\boldsymbol{d}_j$ are updated by constrained least squares, which ensures nonnegative coefficients considering that $\bar{y}_a$ generally increases with resource entitlements. Due to incremental change of the database, λ is updated online per 10 hours to capture the new relationship. For convenience, we refer to such a model as "LCMR-Gray".

For each application a, $2^{|R||T_a|}$ LCMR-Gray models are learned so as to characterize the piecewise linearity as observed in Section II.

4.4.1.2 Black-box Model

For comparison, we develop a black-box model. It only considers resources (inputs) and performance (output) without knowing internals of the application or relying on OS-level support inside the VM. ARMA is a typical black-box model, which is defined as following.

$$\bar{y}_a(k) = \sum_{i=1}^{n} \alpha_i \bar{y}_a(k-i) + \sum_{j=0}^{m-1} \mathbf{d}_j^T \mathbf{w}_a(k-j). \tag{4.2}$$

A black-box model is more general since it is OS and application-agnostic. However, response time may be not observed timely especially because of the presence of long dead-time of requests in a website [43].

4.4.2 The Balancer

The balancer forecasts the future performance by choosing one of the above models. The data samples collected in past S intervals, similar to the sliding window size, are used to obtain the model parameters. However, when black-box model is used in our framework, $w_{a,r,t}(k)$ must be replaced with $\bar{w}_{a,r,t}(k)$ so as to work with it.

It applies MPC [21] to balance resource allocations for each application. We achieve this goal by finding the configuration $\bar{\mathbf{w}}_a$ that minimizes the following penalty function:

$$P_a(k) = \Gamma_a(k) + q\theta_a(k), \tag{4.3}$$

$$\Gamma_a(k) = \|\bar{\mathbf{w}}_a(k)\|^2, \tag{4.4}$$

$$\theta_a(k) = \|\bar{\mathbf{w}}_a(k) - \bar{\mathbf{v}}_a(k-1)\|^2, \tag{4.5}$$

where q is stability factor; $\bar{\boldsymbol{v}}_a(k)$ is normalized actual allocations to application a in interval k by the same value as the $\bar{w}_a(k)$. $\bar{\boldsymbol{v}}_a(k)$ is a column vector of $\bar{v}_{a,r,t}(k)$.

Minimize P_a subject to

$$\bar{y}_a(k) = 1, \tag{4.6}$$

$$lb_{a,r,t}(k) \leq \bar{w}_{a,r,t}(k) \leq ub_{a,r,t}(k), \forall r \in R, \forall t \in T_a, \tag{4.7}$$

where $ub_{a,r,t}(k)$ is the upper bound on $\bar{w}_{a,r,t}(k)$; $lb_{a,r,t}(k)$ is the lower bound. The two bounds are determined by the scaler to directing the search space to a small promising region, realizing stable and responsive control.

We define the total utility function $P_a(k)$ as a linear combination of resource penalty $\Gamma_a(k)$ and control penalty $\theta_a(k)$. The parameter q controls balancing pace. The requirement for a balanced configuration is reflected in $\Gamma_a(k)$. Intuitively, the resource type with higher correlation to application performance is allocated more than other type with lower correlation because the allocation of the higher correlated resource gives better performance improvement. This implies that BConf prefers to increase fewer resource that is more correlated and reduce less correlated resource to achieve the same performance until the configuration is balanced. Once the balance is destroyed due to bottleneck shifts, the new bottlenecked resource will have higher correlation and BConf will repeat the same process to restore balance.

Minimizing $\Gamma_a(k)$ is to balance resources, which can easily be proved by using Cauchy-Schwartz inequality. This utility function can achieve the minimal $\Gamma_a(k)$, so the previous balanced configuration is still balanced. When this minimum value cannot be achieved, the optimization makes trade-off between the two penalties. If q is small, resources are balanced aggressively, resulting in unstable QoS. However, a large value of q slowdowns and even prevents balancing.

Constraint(4.6) enforces the QoS target. The solution is to minimize the total cost while meeting the SLO under the two bounds. Generally, the SLO can be met in the searching space except when workload changes largely. Extremely, SLO will be satisfied in a few intervals by taking the margin values.

The utility function (4.3) is represented ultimately by a quadratic function $Q(\bar{\mathbf{w}}_a(k))$ by substituting (4.4) and (4.5). The objective function is quadratic and convex, so we use a quadratic programming solver to calculate the solution.

Table 4.2: Scaling Setup

	Under-load		Saturation		Overload
$e(k-1)$	< -0.1	> 0.5	< 0	> 0	
$lb(k)$	$-s$	0	$-0.5s$	0	$0.5s$
$ub(k)$	$2e * s$	s	0	s	s

4.4.3 The Scaler

$$s_{a,r,t}(k) = g_r(\lfloor |10e_a(k-1)| \rfloor + 1)\bar{v}_{a,r,t}(k-1)/\bar{y}_{a,ref} \tag{4.8}$$

$$lb_{a,r,t}(k) = max\{m_r, \bar{v}_{a,r,t}(k-1) - s_{a,r,t}(k)\}, \tag{4.9}$$

$$ub_{a,r,t}(k) = \bar{v}_{a,r,t}(k-1) + s_{a,r,t}(k). \tag{4.10}$$

The scaler narrows optimization to a promising region by setting the two bounds. The two bounds are controlled by adaptive PI [86], depending on step. Step $s_{a,r,t}(k)$ is the range that $\bar{w}_{a,r,t}(k)$ is allowed to variate. It is determined by the error $e_a(k)$, $\bar{y}_{a,ref}$ and $\bar{v}_{a,r,t}(k-1)$; $e_a(k-1) = \bar{y}_{a,ref} - \bar{y}_a(k-1)$; $\bar{y}_{a,ref}$ is the reference value of $\bar{y}_a$, and $\bar{y}_{a,ref} = 1$, corresponding to $y_a(k) = \hat{y}_a$; $\bar{v}_{a,r,t}(k)$ is as defined in (4.5). Even if $e_a(k-1)$ is zero, current configuration is still allowed to variate so as to balance resources. g_r is a margin of resource type r, which are set as 0.05 and 0.1 for CPU and disk respectively in our experiments, since the effect of CPU on performance is more quickly observed than disk. m_r is minimal allocation of resource type r, which ensures the allocation will not cause starvation.

$lb_{a,r,t}(k)$ and $ub_{a,r,t}(k)$ are by default set as (4.9) and (4.10) respectively. However, under different load situations, they change to the setup in the Table 4.2 on the basis of $\bar{v}_{a,r,t}(k-1)$.

For any resource type r in tier t of application a, if r is overloaded then $\bar{w}_{a,r,t}(k)$ is increased by at least half of the step; if it is underloaded and with the performance better than target, then resource should be decreased by at least in proportion to the error, but if

SLO is violated, ie. $e_a(k-1) > 0.5$, r is at least not reduced; if it is saturated, the setting is the same as in under-load, except for that the step is reduced to half. This heuristics guides resource allocation toward high-utilization while satisfying SLOs, thus optimizing resource usage.

4.5 The Domain Arbitrator

For any resource type in any host within an domain D, the total requirement should satisfy the following capacity constraint.

$$\sum_{(a,t):host_{a,t}=h} w_{a,r,t} \leq 1, \forall h \in D, \forall r \in R, \tag{4.11}$$

where $w_{a,r,t}$ is transformed from $\bar{w}_{a,r,t}$; $host_{a,t}$ is the hosting node for tier t of application a. If the constraint is violated, we say h is a contended host for resource r. Otherwise, h is contention free so that all requirements are granted. However, surplus resources are not allocated because their usage would increase other costs, such as power and cooling cost. A contention free host is removed from the domain graph until it is cut into isolated sub-domains D_i, with each sub-domain containing just contended hosts.

The constrained resources are distributed within each sub-domain D_i. Because performance is limited by the bottlenecked resource, the constrained resource in any tier of an application should receive the same share p_a of its requirement so that the each one has the same extent of bottleneck. The objective is to minimize the performance penalties Ψ_{D_i} across all applications hosted in D_i in a fair way:

$$\Psi_{D_i} = \sum_{a \in A_{D_i}} ((1-p_a) \sum_{(r,t) \in RT_a} (\frac{\partial \bar{y}_a}{\partial w_{a,r,t}} w_{a,r,t}))^2. \tag{4.12}$$

Minimize Ψ_{D_i} subject to

$$\sum_{(a,t):host_{a,t}=h} p_a w_{a,r,t} \leq 1, \forall h \in D_i, \forall r \in C_h, \tag{4.13}$$

$$0 < p_a \leq 1. \tag{4.14}$$

where A_{D_i} is a set of all resident applications in the domain D_i; C_h is a set of all constrained resource types in host h; RT_a is a relation set between the set of resource types and the set of tiers of application a, $RT_a = \{(r,t) : r \in C_{host_{a,t}}\}$. The excess of any constrained resource r in any host with $\sum_{(a,t):host_{a,t}=h} p_a w_{a,r,t} < 1$ is distributed in proportional to $\frac{\partial \bar{y}_a}{\partial w_{a,r,t}}$.

4.6 Experimentation Methodology

4.6.1 Cloud applications

We selected two representative server workloads as the hosted applications. TPC-W is an E-Commerce benchmark that models after an online bookstore. It consists of two tiers, Tomcat application and MySQL database servers, which are encapsulated into two VMs. We use browsing mix as the workload. TPC-C [5] is another online transaction processing benchmark that contains lightweight disk reads and sporadic heavy writes. Its performance is sensitive to memory size and disk bandwidth.

To create dynamic variations in resource demands, we instrumented the workload generators of TPC-W to change client traffic level at run-time.

4.6.2 Testbed configurations

One cluster (CIC100) was used for the experiments. It consists of 16 DELL servers, each configured with two-socket Intel quad-core Xeon X5450 CPUs and 8 GB memory, connected by a 1Gb Ethernet.

We used Xen version 4.0 [92] [17] [93] as our virtualization environment. Dom0 and guest VMs were all running Linux kernel 2.6.32. Dom0 was configured with 8 VCPU and

4 GB memory, so it would not become the bottleneck for guest VMs. Perfctr-Xen was installed on both Dom0 and DomUs to enable the monitoring of LCMR on guest VMs, which each ran a daemon to compute LCMR during each control interval. To enable on-the-fly reconfiguration of CPU and disk bandwidth, all the VMs were para-virtualized. The VM disk images were stored locally on a second hard drive on each host. We created the dm-ioband device mapper [80] on the partition containing the images to control disk I/O bandwidth. We used Xen's xm command to collect CPU utilization and disk usage statistics.

4.6.3 Experimental methodology

To evaluate the efficacy of BConf, we attempt to answer the following questions:(1) How accurate is the performance model in predicting performance? (2) How well does BConf perform under enough supply of resources? (3)When there is resource contention, can BConf properly arbitrate the constrained resources and maximize the overall system performance? We compared BConf with a representative partition approach named AutoControl [66]. We implemented the original controller and improved it by incorporating the reciprocal performance to resources relationship. In all experiments, all applications warmed up before the start of online model training to build up necessary working set, about for 5 minutes. For fair comparison, both BConf and AutoControl used the same databases and initial resource configurations for TPC-W and TPC-C respectively.

The control interval was set to 30 seconds for all experiments. The performance targets for both TPC-W and TPC-C were set to 100 and 50 mili-seconds respectively. The stability factor for AutoControl was set to 2, which makes the best trade-off between settling time and stability according to [66]. For BConf, q was set to 4.

Table 4.3: MAPE and R^2 values(%) of ARMA and LCMR-Gray models for TPC-W.

	ARMA(1,1)	ARMA(2,2)	LCMR(1,1)	LCMR(2,2)
MAPE	23.98	29.93	19.26	25.16
R^2	66.60	63.56	69.58	65.28

Table 4.4: MAPE and R^2 values(%) for TPC-C.

	ARMA(1,1)	ARMA(2,2)	LCMR(1,1)	LCMR(2,2)
MAPE	32.23	45.04	29.95	36.73
R^2	36.65	15.82	51.76	23.64

4.7 Evaluation Results

4.7.1 Model accuracy

We performed two series of system identification experiments for "LCMR-Gray" or "ARMA" model under various orders for TPC-W and TPC-C, with both CPU and disk entitlements varied randomly.

To assess overall prediction accuracy, we computed two measures, the coefficient of determination (R^2) and the mean absolute percentage error (MAPE), for each application. These two measures of both ARMA and LCMR-Gray models with different orders are shown in Table 4.3 for TPC-W, and Table 4.4 for TPC-C. From Table 4.3, To compare the accuracy of different models, we take LCMR-Gray(1,1) as a baseline and define accuracy difference between LCMR-Gray and ARMA models as $\dfrac{MAPE_{ARMA} - MAPE_{LCMR-Gray(1,1)}}{MAPE_{LCMR-Gray(1,1)}}$. For TPC-W, LCMR-Gray(1,1) is 55.4%, 24.5% more accurate than ARMA(2,2), ARMA(1,1) respectively. For TPC-C, LCMR-Gray(1,1) is 50.4% and 7.5% more accurate respectively. Since LCMR-Gray(1,1) performs best, we use LCMR-GRAY(1,1) in our BConf framework.

4.7.2 Performance under enough resources

We have done two experiments under workloads with static and dynamic number of clients respectively using the small setup shown in Figure 4.5(a). This setup uses two physical machines to host one TPC-W, and two TPC-C benchmarks.

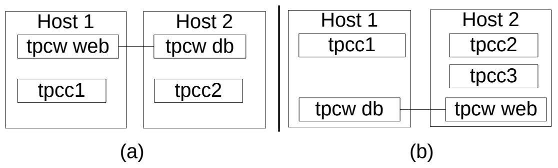
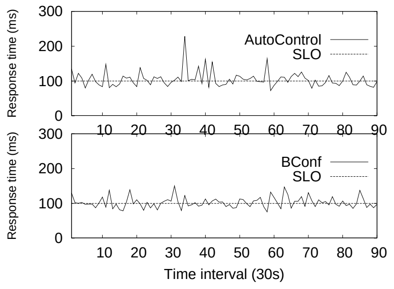
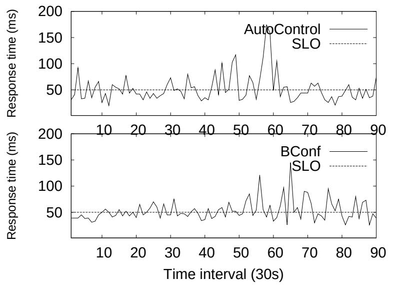


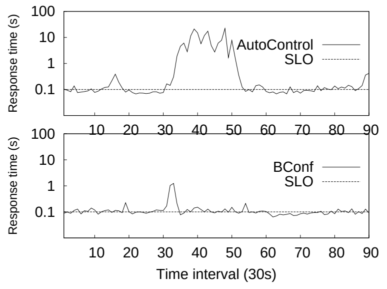
Figure 4.5: Experimental setup.



(a) tpcw under static workload



(b) tpcc1

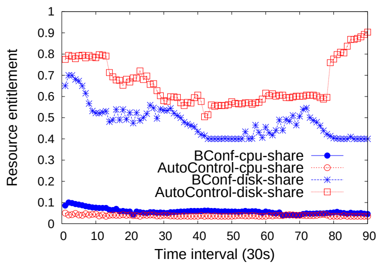


(c) tpcw under dynamic workload

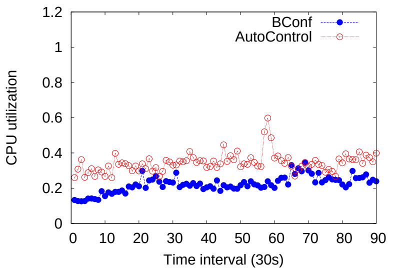
Figure 4.6: Performance comparisons of BConf and AutoControl for tpcc1, tpcw under static and dynamic workload.

Figure 4.6(a) and 4.6(b) show the response times of the two methods with static workloads. The SLO is the dotted straight line. The static workloads were set to 500 browsing clients for tpcw, 100 clients and 30 warehouses for both tpcc1 and tpcc2. Figure 4.6(c) plots the response times with the number of tpcw clients changed every 30 intervals. We started with 300 clients and set the client numbers at the 30th, 60th intervals to be 500, 300 respectively. From Figure 4.6(a) and 4.6(b), we observe that both can bring the response times close to the targets, but with different deviations. Figure 4.6(c) shows that BConf converges the performance to the target more quickly.

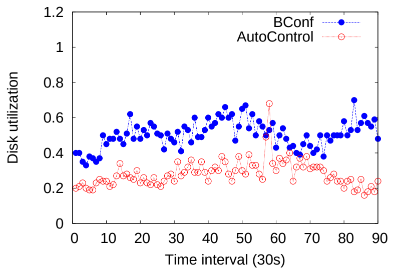
For tpcc1, Figure 4.7 plots allocations and utilizations. As shown in Figure 4.7(a), BConf allocated 2% more CPU than AutoControl, but consumed 30% less disk bandwidth. The configuration by BConf was more balanced and efficient than AutoControl, since CPU contributes more to the performance than disk allocations. The CPU and disk utilizations are shown in 4.7(b) and 4.7(c) respectively. The scaler tries to increase the resource utilizations thus reducing resources but avoids resources to run in overloaded condition, In BConf, CPU utilizations was lower than by AutoControl, but with much higher disk utilization.



(a) resource allocation



(b) CPU utilization



(c) disk utilization

Figure 4.7: tpcc1 resource allocations and utilizations comparisons of BConf and AutoControl.

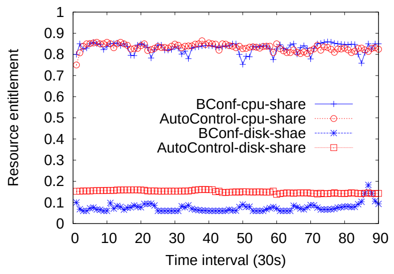


Figure 4.8: Resource share for tpcw-db under static workload.

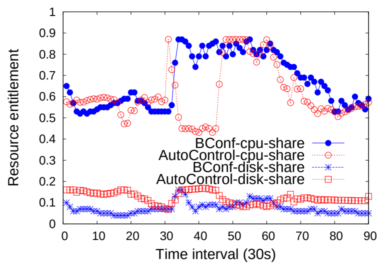


Figure 4.9: Resource share for tpcw-db under dynamic workload.

The performance and configuration of tpcc2 is similar to tpcc1. For tpcw with the static workload, the comparison of the actual allocations between BConf and AutoControl is shown in Figure 4.8. The resource allocations for the web tier were insignificant compared with the db tier, but BConf consumed less overall resources than AutoControl.

For the dynamic workload, Figure 4.9 plots allocations for TPC-W. As shown in Figure 4.9, BConf guaranteed QoS in the balancing process by increasing fewer VPU cap and reducing more disk I/O in the first 30 intervals while AutoControl violated QoS due to CPU reductions in intervals 16-18. When the balanced state was disturbed after client number mounted from 300 to 500 in the 30th interval, BConf quickly supplemented CPU and disk bandwidth resources substantially in 4 intervals to meet the SLO. At the same time, AutoControl first decreased VCPU cap and increased disk I/O due to the wrong model learning that response time correlated with itself and disk I/O more than CPU allocation,

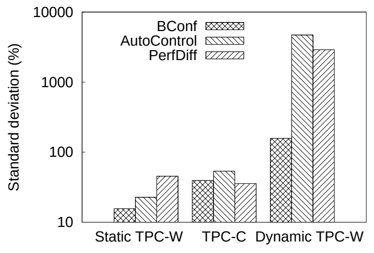


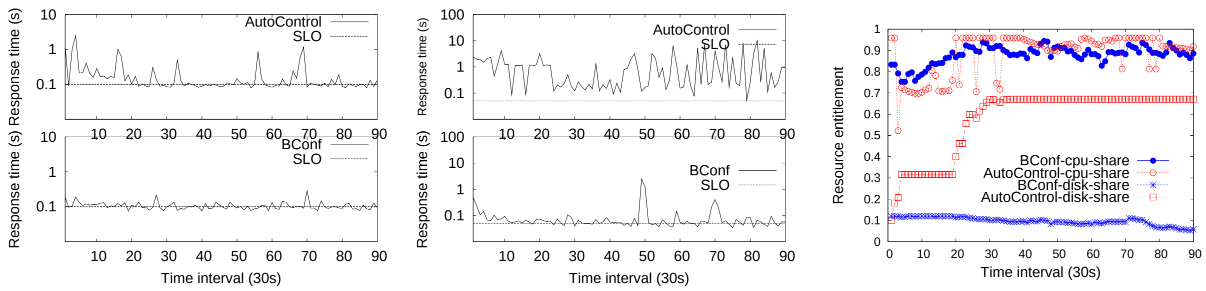
Figure 4.10: Standard deviations.

incurring long dead time of requests. Only after a long time, did AutoControl recover from severe SLO violation in the 54th interval. When client number declined to 300 in 60th interval, AutoControl got rid of surplus CPU quickly at the cost of disk I/O, but BConf diminished both CPU and disk I/O to balance the configuration. In the long run, BConf and AutoControl had almost the same CPU allocations, but BConf consumed half disk I/O as much as AutoControl. The performance for TPC-C consolidated with TPC-W under dynamic workload was almost the same as consolidated with TPC-W using static workload, so we do not show the results.

From above analysis, we see that BConf outperformed AutoControl by consuming half as much I/O bandwidth as AutoControl, about 50% less I/O than AutoControl, thus doubling disk utilization for tpcw under both static and dynamic workloads. For tpcc1, BConf consumed 20% less I/O than AutoControl, with the disk utilization increased by 22%.

It can be concluded that the performance, resource balancing and allocation efficiency consideration in BConf justifies that the resource minimization is constrained by and consistent with SLOs satisfaction for both static and dynamic workloads. By contrast, AutoControl maintains relative steady resource allocations, which can not deal with workload dynamics but only increases the risk of violating SLOs and wastes unnecessary resources.

Stability is important because it reflects how stable the performance is that the cloud users experience. Instability may discourage customers's use of the cloud. We define standard



(a) tpcw performance (b) tpcc1 performance (c) Comparisons for tpcw-db

Figure 4.11: Results under scenario(b).

deviation from a reference as the metric.

$$R(e) = \frac{\sqrt{\sum_{k=1}^{n} e(k)^2/n}}{r}, \tag{4.15}$$

where r is the control objective and $e(k)$ is the error. The smaller the $R(e)$, the better the stability. From Figure 4.10, we can see that BConf performed 45.4%, 35.6%, and 29 times better than AutoControl for static tpcw, tpcc1, dynamic tpcw respectively, due to the reason that BConf scales resourcs towards performance target and use a more accuate model.

Settling time indicates how quickly the system converges to its steady state value [43]. It measures the SLO violation time.

From Figure 4.6(c), we can observe that it took BConf 4 intervals to get the target when client number climbed from 300 to 500, but about 23 intervals for AutoControl. Therefore, BConf has shorter settling time than AutoControl.

4.7.3 Results under resource contention

We have done the experiment of scenario(b) in the setup of Figure 4.5(b). Two physical machines are used to host a TPC-W benchmark and three TPC-Cs. The workload for tpcw is increased to 550 browsing clients, and the workload for each tpcc is the same as in scenario(a). This setup may slightly constrain the CPU resource in Host1 and the disk I/O resource in Host2, so tpcw may contend resources in both tiers.

Figures 4.11(a) and 4.11(b) compare the response times of the two methods. From Figure 4.11(a), we observe that both approaches can satisfy SLOs for tpcw, but still BConf has better stability. For tpcc1, however, AutoControl can not meet the target as shown in Figure 4.11(b).

To shed light on the reason, the configurations are depicted in Figure 4.11(c). The two approaches treated Host1 very differently. In AutoControl, tpcc1 at first exhibited a stronger relationship with CPU than with disk I/O. It competed for CPU resource with tpcw in Host1, and got the CPU resource because tpcw correlated more with disk I/O bandwidth. But soon tpcw learned that its performance degraded a lot, so tpcw updated the relationship with CPU and got CPU resource from tpcc1. Besides, it also had strong relationship with disk I/O, and was granted with the disk resource to improve its performance. By contrast, tpcc1 did not get a chance to update the model because the requirements were not granted. Therefore, it was more and more unlikely to obtain the resource. The relative fixed relationship misled the resource arbitrator to make a bad choice.

BConf first did not obtain the right relationship either, but even though it had given tpcw a bad choice, it later invoked the more appropriate model when CPU was overloaded. It is the model shifting and the intelligent scaling direction provided by the scaler that guides the arbitrator to make a good choice. Though a wrong model sometimes punishes itself, it later will learn the right relationship to get the reasonable resource requirements granted. Another advantage of BConf is that it just needs required resources that satisfies its target, and releases surplus resources as regulated by the scaler. So tpcw did not compete for disk I/O with tpcc1. Besides, BConf considers the resource allocations in both of web and db tiers as a whole, so the tiny requirements in web tier of tpcw was not throttled by the tpcc2 and tpcc3. Even if they were not granted at one time, tpcw could learn the right requirements later and both tpcc2 and tpcc3 would hand over the excess resources. The tpcc2 and tpcc3 by BConf was slightly better than by AutoControl due to the relieved contention in Host2.

As a result, BConf does not need to migrate applications from Host1 to other machines. It is able to consolidate more applications. To compare the resource savings, we adopt a cost model based on that of Amazon's EC2 to our system. We set the resource price for the customer at 1.00 dollar per hour and the resource cost is 0.5 dollar per hour. Then Host 1 can earn 0.5 more dollar per hour using BConf strategy.

4.8 Summary

In this paper, we present an integrated MPC and adaptive PI control approach for balanced allocations of virtualized resources to guarantee response time target. We evaluated BConf in a cloud testbed using two online transaction processing benchmarks and compared it with a representative partition controller for performance. Experimental results show the advantages of BConf in resource efficiency and coordination. BConf also demonstrates its stability and responsiveness over a variety of workloads in contrast to the controller.

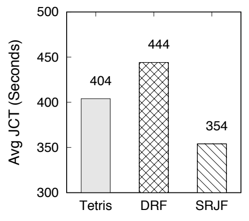
Though the gray-box model is response time oriented, the resource balancing and coordination policy can be applied to satisfy the QoS in terms of other metrics, such as throughput, by using the black-box model. In the future, we have following directions to work on: (1) extend our gray-box model to data-intensive applications, such as MapReduce Applications; (2) configure resources by considering the energy and performance as a whole.

CHAPTER 5 CANAL: CREDIT SHARING-ORIENTED NETWORK AND LOCALITY AWARE SCHEDULING IN MULTI-RESOURCE CLUSTERS

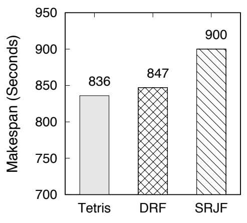
5.1 Introduction

MapReduce [28] remains as a dominant parallel and distributed programming paradigm for applications to process big data in clusters [64] [58] [85] [10] [54] [98] [83] [8] [31] [56] [57]. It is one of the core technologies powering IT companies like Facebook. MapReduce jobs are transformed into a set of tasks in today's cluster systems, such as Mesos [46] and Yarn [82]. The cluster scheduler uses a two-level architecture that consists of job scheduling and task scheduling, where tasks are basic objects of resource allocation. Moreover, a fine-grained multi-resource allocation model is provided to satisfy heterogeneous demands of various tasks. It unlocks space for resource combination and optimization between tasks and servers.

Multi-resource scheduling has been extensively studied [6] [26] [24] [39] [41] [40] [51] [79]. In a shared cluster, scheduling has three different objectives: system efficiency [39], fairness [35] [90] [19] [18] [24] [20], and minimal job completion times (JCT) [40]. System efficiency is measured by throughput or makespan, the time to finish a set of jobs. However, each objective may be in conflict with the other two in existing scheduling policies. While packing a set of tasks of complementary resource demands optimizes system efficiency by maximizing resource utilization of any resource type, fairness may not be preserved because these tasks are not from the job with minimal dominant share according to the dominant resource fairness (DRF) policy, and JCT may be not minimal because they are not from the shortest job according to shortest remaining job first (SRJF). In addition, the objectives of fairness and minimal JCT may not be achieved together because the scheduling order under DRF is different from the one under SRJF. Particularly, SRJF may cause starvation or extreme unfairness for long jobs if short jobs keep coming.



(a) Average JCT



(b) Makespan

Figure 5.1: Average JCT and makespan under different schedulers.

The multi-resource version of max-min fairness, i.e., DRF, is based on instantaneous share at any time point. On one hand, it causes resource fragments, which are resources in a node that cannot be utilized by any task, thus degrading system performance. On the other hand, the envy-free property of fairness, i.e., no user should prefer the resource allocation of another user, may increase JCT. The reason is that JCT is improved by trading resource allocation of long jobs with that of short jobs. Since users cannot experience the instantaneous share, they care more about whether a job or its last task finishes within its *fair sharing job completion time* (FJCT), the one resulting from DRF. Our observation is that by keeping in mind the FJCT of each job, the constraints placed by DRF on the optimization of both cluster efficiency and JCTs can be removed. Specifically by dynamically sorting jobs in the increasing order of their FJCTs (instead of instantaneous share), packing tasks of relatively short jobs by taking credit from long jobs not only accelerates short jobs, but also guarantees that long jobs will not decelerate. Moreover, packing optimizes system performance by maximizing resource utilization in each dimension.

To make tradeoff between the three conflicting objectives, a scheduler called Tetris [39] was designed to employ dot product to pack tasks of jobs at a fixed-value (75th) percentile of dominant share. It remains the best practice as shown in Figure 5.1. Whenever resource is available, it computes the packing score and job score (related to job length) for each task, and selects the one with the largest weighted score. While Tetris improves system efficiency and JCTs, it has two major disadvantages. First, the percentile knob limits improvement of JCT. When the dominant shares of short jobs go beyond the fixed-value percentile, they are not allowed to run further even though they will not decelerate long jobs. Therefore it misses the opportunity to accelerate short jobs. Second, the knob cannot guarantee the FJCTs. Packing heuristics favor certain tasks. For example, dot product favors large resource demand. As a result, it may cause long and indefinite wait time for the jobs that have extremely small task demands as shown in Section 2.

A recent proposed work, Altruistic Scheduling (AS) [40], allows long jobs to yield resource allocation to short jobs. While it improves the average JCT, the improvement is achieved at the cost of long jobs, because it provides guarantee of FJCTs only in *offline scheduling*, rather than in online scheduling. The AS results also show that a significant percentage (16%) of jobs is slowed down by up to 1.61 times. Moreover, AS serves as an outside layer of an existing scheduler, such as DRF, focusing on job acceleration with no consideration of cluster efficiency. Since AS is contingent upon other scheduler, it is *complementary* to our independent scheduling work. Our objective is to optimize both system efficiency and JCTs, and theoretically guarantee FJCTs in *online scheduling*.

In a multi-resource cluster, system efficiency depends on effective utilization of any resource type. A MapReduce job is divided into Map and Reduce tasks. A Map task can run on the same node as its input data (node locality), on the same rack (rack locality) or on a different rack (off-rack). Compared with a node-local task, a non-local one needs additional network resources and extra time for data transfer. Tradeoff should be made when the resource demand of a non-local task is larger than a local task. There were studies on achieving good data locality without impairing fairness in a slot-based cluster [49] [97]. A slot is a fixed amount of a single resource type, such as memory in Hadoop. In multi-resource clusters, the heterogeneity of multi-resource requirements by tasks may render them ineffective. To choose between a local and a non-local task, current work [39] imposed a fixed-value penalty to discount the packing score of a non-local task. Though the fixed-value improves the average performance, its fixity may cause sub-optimal performance due to lack of adaptability to workload dynamics. Therefore, data locality poses great challenges to cluster scheduling.

Network utilization is crucial to performance in large-scale data-intensive clusters because network bisection bandwidth tends to become a bottleneck [29] [71] [78]. In MapReduce jobs, network is mainly used by reduce tasks for intermediate data transfer. There is stage barrier to network utilization because reduce tasks cannot start if map tasks have not finished. The network can be idle when the running tasks are mainly map tasks, and become a bottleneck when most are reduce tasks. Therefore, scheduling tasks of the same stage in most or all jobs may cause imbalanced network utilization and elongation of network-intensive jobs. Existing schedulers, including Tetris and AS, are oblivious of this barrier. Though Tetris scheduled the last few tasks when the map stage is about to completion, the effect is limited. As to AS, it even will degrade system performance when only long jobs are network-intensive, because accelerating short jobs causes network to be more under-utilized. Advancing some network-intensive jobs to get network fully utilized not only improves system efficiency, but also decreases JCTs. However, how to accelerate jobs without affecting completions of other jobs further complicates the task of cluster scheduling.

Our main contribution in this paper is to exploit above key observation and propose a simple theorem-proof policy, called credit sharing (CS) policy, that guarantees fair sharing JCTs. It recursively accounts for the cumulative credit that long jobs contribute to the short jobs, and short jobs pay down or off their borrowed credit later in the process of accelerated execution or upon completions. The credit is measured by how much resource shares have been borrowed and for how long. In addition to accelerating short jobs, the cluster scheduler is enabled to pack tasks of relatively short jobs, getting system efficiency nearly optimized since task demands of short jobs are in the same magnitude as those of long jobs. Because a non-local task introduces a “new" non-local fragment by wasting extra resources, we estimate the size of all the normal and non-local fragments resulting from a packing. According to CS policy and the size of total fragment, we present a method called

Dynamic-Level for calculating the number of head jobs dynamically that optimizes both the effective resource utilization and JCTs. To overcome the stage-barrier, we monitor network and schedule additional tasks for some network-intensive jobs by borrowing credit from long jobs based on CS policy. The packing scores of the additional tasks are dynamically adjusted and scheduled together uniformly with all other tasks in Dynamic-Level.

5.2 Motivation and Challenges

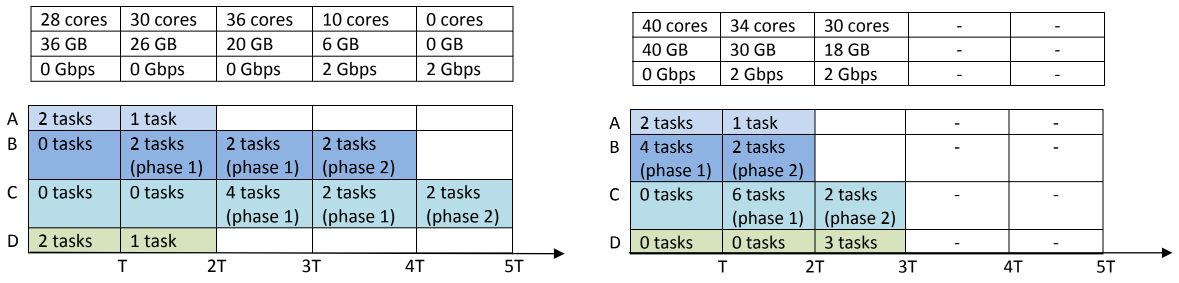
5.2.1 Limitations of Current Cluster Schedulers

As mentioned in the introduction, DRF impedes performance of shared clusters. While Tetris [39] alleviates the impediments by setting a percentile-knob, it may elongate some jobs by preventing them from further execution, or cause indefinite wait. We use an example to illustrate this.

Consider a two-node cluster with 2Gbps network. Each node has 20 Cores and 20 GB RAM. There are four jobs. Job A has 3 tasks that require [4 Cores, 12 GB] each; job B has 4 map tasks that require [8 Cores, 4 GB] each, and 2 reduce tasks; job C has 6 map tasks that require [5 Cores, 3 GB] each, and 2 reduce tasks; job D has 3 tasks that require [10 Cores, 6 GB] each. All reduce tasks need 1Gbps network and extremely little CPU or memory. Assume that the duration of each task is T. Tetris will consider the first 3 of the 4 jobs (75% percentile) sorted by their dominant shares, and select the task with the highest packing score each time. Tetris will schedule 2 tasks for job A and D, then 1 task for each of them. Figure 5.2(a) shows its task schedule.

Consider a FJCT-aware scheduler that dynamically schedules tasks from the head jobs. It will first schedule 2 tasks for job A, and all the 4 tasks for job B to use up all the CPU and MEM. Then the reduce tasks of job B become runnable to use up the network resource. Such a schedule is shown in Figure 5.2(b). The jobs finish at $2T$, $2T$, $3T$, $3T$.

Under the new schedule, average JCT drops by 23% over Tetris ($13T \rightarrow 10T$). The makespan reduces by 40% over Tetris ($5T \rightarrow 3T$).



(a) Tetris Schedule (b) FJCT-aware Schedule

Figure 5.2: Comparison of schedules under Tetris and an alternative.

Table 5.5: Comparison of JCTs

JCT	Tetris	CANAL	DRF
A	2T	2T	3T
B	4T	2T	5T
C	5T	3T	4T
D	2T	3T	3T

Table 5.5 compares their JCTs with DRF. Under Tetris, job C completes later than FJCT under DRF. Suppose jobs A and D keep coming, Tetris will delay job C indefinitely due to its small task demand. By contrast, a FJCT-aware scheduler will not.

5.2.2 Data Locality and Effective Resource Utilization

For any job, non-local Map tasks require additional disk and network resources than data-local ones. We call the resource demand of local tasks the essential resources, the minimum required by any task executing the same application on data of the same size. The utilization of essential resource is effective whereas non-local data access is not effective. We illustrate this problem by running 20 jobs of 4 data-intensive benchmark applications on Tetris: RandomForest, TeraSort, WordCount and GrepSearch, with 5 jobs for each application over input data range from 1GB to 20GB. Details of the testbed and benchmark descriptions can be found in Section 5 and Table 5.7. Figure 5.3 shows effective disk and network utilizations respectively, as well as non-local input data access for map tasks. Rack-local and off-rack data access accounts for up to 20% of disk bandwidth, and off-rack data

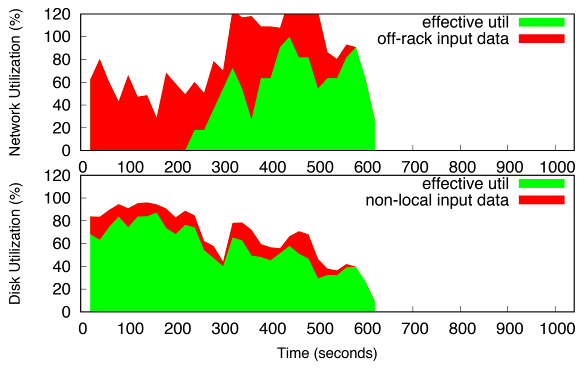


Figure 5.3: Non-local data access for map tasks and effective resource utilization.

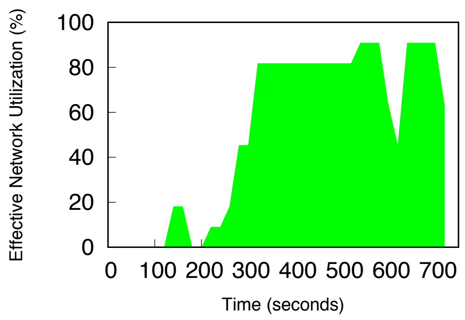


Figure 5.4: Effective network utilization by Tetris.

for up to 80% of network bandwidth of the cluster switch. The non-local data access greatly degraded the performance by decreasing effective disk and network utilizations.

5.2.3 Stage Barrier to Network Utilization

Reduce tasks may have large network requirement due to a great amount of intermediate data transfer across the cluster switch. Network is bottlenecked resource for network-intensive jobs. However, there is stage barrier to fully utilize network because only after all Map tasks finish can reduce tasks start to execute. Network may be idle if all jobs are still in Map stage or running Map tasks. To demonstrate this problem, we ran 8 jobs of 2 applications: TeraSort and WordCount on Tetris, with 4 jobs for each application over input data range from 6GB to 20GB. Figure 5.4 shows effective network utilizations across their lifetime. The cluster switch is nearly idle in the first 260 seconds, accounting for 35% of total time.

5.3 The Credit Sharing Policy

A job can have DAG tasks [59] [61] [25] [14] [36]. As demonstrated in Dryad [47] [48], the DAG of any job is mapped to a topology consisting of multiple sequential stages; each stage has many concurrent tasks. Particularly, MapReduce jobs have two stages: map and reduce. We consider the general scenario that tasks have *different* demand vectors in different stages. Task demands and durations can be leveraged based on prior runs of recurring jobs [9] [32] [39]. In addition, tasks in the same stage have the *same* demand vector and duration [13] [60] [35], Our study is reasonable and practical because existing SampleSort [33] algorithm can assure evenly partitioning of intermediate data. In the estimation of FJCTs, we first consider the case with MapReduce jobs, then generalize it to multi-stage scenario.

Suppose a multi-resource cluster and N jobs, from j_1 to j_N, with the resource set as $R = \{cpu, mem, disk, network\}$. Job j_i has M_i stages, and n_{is} tasks in stage s; each task in stage s has essential resource demand vector $\boldsymbol{D_{is}} = [d_{is}^{cpu}, d_{is}^{mem}, d_{is}^{disk}, d_{is}^{network}]$ and duration l_{is}. The demand vector is normalized to the corresponding capacity of the entire cluster. Note that above vector is a row vector. We represent all vectors in boldface. The effective utilization is denoted as $\boldsymbol{U} = [u^{cpu}, u^{mem}, u^{disk}, u^{network}]$. For convenience, we call the product of the demand vector of a task of job j_i in stage s and its duration the product vector, which is $l_{is}\boldsymbol{D_{is}}$. The product vector of job j_i, denoted as $\boldsymbol{P_i}$, is the sum of product vectors across all tasks, $\boldsymbol{P_i} = \sum_{s=1}^{M_i} n_{is} l_{is} \boldsymbol{D_{is}}$. The largest component of the product vector of a job is called its dominant demand product P, abbreviated as demand product.

5.3.1 Bounded-space Dynamic MVBP Model

Task scheduling is a variation of classic Multi-dimensional Vector Bin Packing (MVBP) problem called bounded-space dynamic MVBP, because the number of servers is bounded; jobs and tasks dynamically arrive and depart.

One goal of task scheduling is to maximize throughput or minimize makespan. The classic MVBP problem is NP-complete and APX-hard [89]. Packing heuristics or approxima-

Table 5.6: Resource allocation and cumulative credit of job B

Time Window	DRF	CS	share credit	product credit	cumulative credit
[0, T]	0.2	0.8	0.6	0.6T	0.6T
[T, 2T]	0.2	1	0.8	0.8T	1.4T
[2T, 3T]	0.2	-	-0.2	-0.2T	1.2T
[3T, 4T]	0.2	-	-0.2	-0.2T	1T
[4T, 5T]	1	-	-1	-T	0T

tion algorithms, such as dot product, are near optimal because the results on average are only a few percent worse than the minimum number of servers [76]. Meanwhile, they maximize server utilization in each resource dimension at each scheduling step, and accumulatively step by step, the utilizations over the lifetime of the server.

Since the total product vector across all jobs is fixed, the makespan (MS) depends on the effective utilizations of all dimensions. Therefore, we have:

$$MS = max\{\frac{\sum_{i=1}^{N}\sum_{s=1}^{M_i} n_{is} l_{is} d_{is}^r}{u^r}, \forall r \in R\}. \tag{5.1}$$

Maximizing effective utilization in each resource dimension amounts to minimizing makespan.

5.3.2 Credit Sharing Theorem

Under the new CS policy, suppose N jobs are sorted as job j_1, j_2, ..., j_k, ..., j_N in the increasing order of FJCTs. Note that above subscripts are ranks of the order. FJCTs are dynamically calculated in online scheduling according to DRF (subsection 5.3.3). If short jobs borrow some resource shares from long jobs, then not only short jobs run faster, but also long jobs will not be delayed under certain constraint. The borrowed share is called share credit, or allocation credit. The product of the allocation credit and its duration is called product credit, abbreviated as credit. The credit accumulated over the time is called cumulative credit. Table 5.6 shows resource allocation of job B in Section 5.2.1 under

DRF and CS, as well its share credit and cumulative credit. After job B finishes at $2T$, its cumulative credit starts to decrease because it pays off its credit.

The “if" condition in the hypothesis is called prerequisite. The *constraint* of borrowing is that after the advancement, if not completion, of short jobs, long jobs later have enough resource demand to utilize the credit lent to short jobs. The product of dominant share and its duration is called dominant allocation product, abbreviated as allocation product or dominant product. Note its difference from dominant demand product. The dominant product accumulated over the time is called cumulative dominant product. Formally, the cumulative credit of a job over time is the difference between its cumulative dominant product under the new policy and the corresponding one under DRF over the same time. The *constraint* will be discussed in detail in subsection 5.3.4. In the following, the prerequisite is formulated mathematically, and the hypothesis is proved based on mathematical induction.

Theorem Assume the above constraint is satisfied by long jobs. The cumulative credit of the first i ($\forall i, 1 \leq i \leq N$) shortest jobs over any time t, denoted as $C_i(t)$, is larger than or equal to 0, i.e., the first i jobs can borrow credit, is a *sufficient* prerequisite for all jobs to run faster than or at least the same as they did under DRF.

For convenience, the new cumulative dominant product of the first i jobs over time t is denoted as $F_i'(t)$, $F_i'(t) = \sum_{k=1}^{i} \int_0^t f_k'(\tau)d\tau$; the corresponding one under DRF is denoted as $F_i(t)$, $F_i(t) = \sum_{k=1}^{i} \int_0^t f_k(\tau)d\tau = i \cdot \int_0^t f(\tau)d\tau$, where $f_k'(\tau)$ is the new dominant share allocated to job k, $f(\tau)$ is the dominant share allocated to each job by DRF. Note that each notation under CS is primed compared with the corresponding one under DRF. Then, we have $C_i(t) = F_i'(t) - F_i(t)$. Since $C_i(t) \geq 0$, the prerequisite is simplified as:

$$F_i'(t) \geq F_i(t). \tag{5.2}$$

Proof We start with a fixed-set of N jobs, then consider dynamic arrival of new jobs in online scheduling. Suppose the first job j_1 finished at time t_1 under DRF as shown in

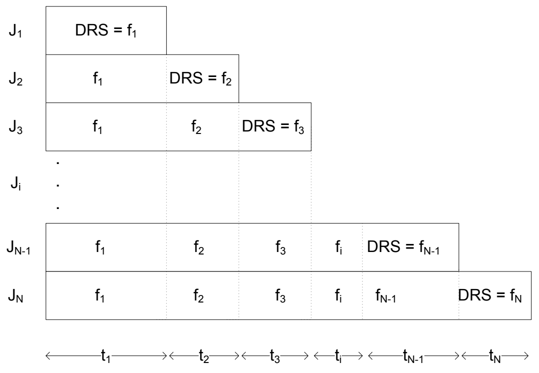


Figure 5.5: Job dominant resource shares and completion times under DRF.

Figure 5.5, and job $j_i, 2 \le i \le N$ finished at time $\sum_{k=1}^{i} t_k$, or t_i later than its immediate predecessor job j_{i-1}. The dominant share each job received was f_1 before the first job finished, and was f_i during the subsequent time $t_i, 2 \le i \le N$, after job j_{i-1} finished.

The index i is assumed to be $1 \le i \le N$ without particular specification. The FJCT of job j_i is denoted as $T_i = \sum_{k=1}^{i} t_k$. A job finishes when its cumulative dominant product is equal to its demand product. Therefore, we have:

$$F_i(T_i) = \sum_{k=1}^{i} P_k. \tag{5.3}$$

Suppose the new completion time of job j_i is T_i' *under CS*. Given (5.2), we want to prove $T_i' \le T_i$. Note that P_i is a constant. The new product that the first job accumulates over time $t_1 = T_1$ is $F_1'(T_1)$. According to (5.2) and (5.3), $F_1'(t_1) \ge f_1 t_1 = P_1$. Therefore, the first job must finish within t_1 under CS. We have:

$$T_1' \le T_1. \tag{5.4}$$

Assume $T_1' \le T_1$, ..., $T_{i-1}' \le T_{i-1}$, $T_i' \le T_i$, that is, the first i jobs complete before their FJCTs. In the CS policy, the first $i+1$ jobs since the start time 0, accumulate total dominant product $F_{i+1}'(T_{i+1}) \ge F_{i+1}(T_{i+1}) = \sum_{k=1}^{i+1} P_k$ over time T_{i+1} based on (5.2) and (5.3). Since the first i jobs already finish within $T_i \leq T_{i+1}$, job j_{i+1} must finish within T_{i+1}, otherwise above formula does not hold. We have:

$$T'_1 \leq T_1, ..., T'_i \leq T_i \Longrightarrow T'_{i+1} \leq T_{i+1}. \tag{5.5}$$

From (5.4) and (5.5), $T'_i \leq T_i$, $1 \leq i \leq N$ is concluded according to mathematical induction.

Now consider ***dynamic job arrival***. Suppose a job j arrives at time t_0, and the FJCTs of all jobs are updated, with j longer than l jobs. Therefore, j becomes j_{l+1}, and all the jobs longer than j shift right by 1, becoming j_{l+2}, ..., j_N, j_{N+1}. The FJCTs are sorted as T_1, ..., T_l, T_{l+1}, ..., T_{N+1}. Note that T_i of each job j_i that has not finished before t_0 changes with the arrival of job j. We want to prove that $T'_i \leq T_i$ still holds. Applying induction to the first l jobs as to the previous N jobs, $T'_i \leq T_i$ holds when $i \leq l$. The first $l+1$ jobs accumulate total dominant product $F'_{l+1}(T_{l+1}) \geq F_{l+1}(T_{l+1}) = \sum_{k=1}^{l+1} P_k$ over time T_{l+1} based on (5.2) and (5.3). Since the first l jobs already finish within $T_l \leq T_{l+1}$, job j_{l+1} must finish within T_{l+1}. Therefore, $T'_i \leq T_i$ holds when $i = l+1$. Similarly, $T'_i \leq T_i$ holds for the remaining jobs from j_{l+2} to j_{N+1} by applying induction.

By applying divide and conquer again, as well as induction in each branch, $T'_i \leq T_i$ holds for all jobs after arrival of each new job. Therefore, the prerequisite guarantees FJCTs in online scheduling.

End of Proof

Interpretation The theorem recursively checks the cumulative credit that long jobs contribute to the top i jobs. Upon accelerated completions, short jobs pay off the credit drawn from long jobs.

Suppose c_i is the cumulative credit of the single job j_i. Note that $c_i(t)$ is different from $C_i(t)$. When $i = 1$, $c_1(t) = C_1(t) \geq 0$. When $i \geq 2$,

$$c_i(t) = C_i(t) - C_{i-1}(t). \tag{5.6}$$

In other words, from the second job, $c_i(t)$ can either be 0, positive or negative, interpreted as no credit, borrowing and lending credit respectively. Hence, at any time point t, there are 3^{N-1} combinations about credit polarity. At some time later, when $c_i(t)$ decreases from positive to 0 or from 0 to negative, job j_i pays off the credit or lends credit respectively; when $c_i(t)$ increases from negative to 0 or from 0 to positive, job j_i collects the credit or draws credit respectively. Unlike DRF that always schedules one particular job, CS allows scheduling any job that meets the criteria.

Because CS unleashes the power of packing, the resource utilization $\boldsymbol{U_{cs}}$ in each dimension is larger than $\boldsymbol{U_{drf}}$. Since the scheduler does not know or need to know $\boldsymbol{U_{drf}}$, we use the maximum utilization over all resource types in $\boldsymbol{U_{cs}}$, the utilization of bottleneck resource, as the upper bound of utilization of each resource for calculating FJCTs, denoted as $T_i^{cs-util}$. Moreover, we have:

$$T_i' \leq T_i^{cs-util} \leq T_i. \tag{5.7}$$

5.3.3 Calculation of Fair Sharing Completion Times

Each FJCT $T_i^{cs-util}$ is computed according to the divisibility version of DRF [70]. DRF equalizes the dominant share f_i across all jobs as shown in Figure 5.5. Moreover, if a job cannot use up the allocation due to lack of task demands, the surplus is redistributed to other jobs similarly [35]. For multi-stage jobs, the dominant resource type of each job is determined by task demand in current stage. In addition, each stage has fair sharing completion time (FCT). The FCT of current stage is estimated from both the remaining dominant demand product and dominant share in current stage. The dominant share, as well as the remaining demand product, is recalculated whenever a job “shifts" from one stage to another stage or “departs” the system according to DRF, or arrives to the system. The three types of events constitute job dynamics. The FJCT is determined by calculating FCT for each stage and adding them up. If current stage is the final stage, for example reduce stage of map-reduce jobs, then FJCT is estimated directly. If it is not, the FCT for each remaining stage will be estimated.

Suppose the current time is t, and dominant share is f. We take map-reduce jobs as example for estimating FCTs of further stages. We simulate DRF to update dominant share and remaining demand products of jobs with the time advancement ever since from t. Since job inter-arrival times is roughly exponential, the number and task demands of jobs at any time are almost identical. Therefore, we do not need to predict job arrivals, and do simulation at two types of job events: stage shifting and departure. The stage shifting event of a job will affect dominant shares of all jobs that are running network-intensive (NI) reduce tasks at the event, because it starts to utilize network bandwidth. The job departure event will also affect the shares of NI jobs since the network utilization will change. The dominant share of other jobs that are running map tasks or non-network-intensive (NNI) tasks are not affected by both events given the same job and task demand distribution of other resources except for network. Therefore, we can use the same dominant share f in the time advancement for NI jobs at map-stage and NNI jobs at any stage. Their map-FCTs keep the same due to the same share f, and reduce-FCTs are computed accordingly by using f.

In order to estimate the reduce-FCTs for NI jobs, we advance time and update parameters as follows in the increasing order of occurring times of the above two types of job events. First, update the remaining demand product. Second, derive the network shares of NNI jobs at reduce stage from the fixed network to dominant share ratios of their reduce tasks since dominant share is known as f, and deducted from the normalized network capacity of entire cluster, which is 1. Third, divide the remaining network share evenly among NI jobs at reduce stages according to DRF due to the same dominant resource type − network, with the share of other resource types not exceeding their corresponding shares at time t. Repeat the same process until the reduce-FCTs of every NI job at t is predicted. If the network is fully utilized by some NI jobs during some time piece in the prediction, then long NI jobs can lend credit to short ones. Otherwise they cannot because they would take risk of decelerating completions and network would be more underutilized due to fewer ready NI reduce tasks. Suppose the set of NI jobs are j_{i_1}, ..., j_{i_k}, ..., j_{i_n}, the dominant share is f' during time piece t' and $nf' = 100\%$. Then job j_{i_k} can wait the first $k-1$ jobs to use up the $(k-1)f'$ share, the credit it can lend is $(k-1)f't'$. Accumulating all the credits during all time pieces with full network utilization, we get the total credit for each job to lend, called the credit threshold Γ_{i_k}. Moreover,

$$-c_{i_k} \leq \Gamma_{i_k}. \tag{5.8}$$

For multi-stage jobs, the further estimation can be done similarly by recognizing resources that are only used by some particular stages.

In above further estimation with time advancement, we assume the same dominant share for NNI jobs. Actually, the dominant share may change with job dynamics in future even during a single stage of a job as indicated in Figure 5.5. Our argument is that job order is the same as long as the new changed dominant share is the same across jobs and can be used up by jobs. The CS theorem depends on the order of jobs, instead of FJCTs.

For jobs that cannot use up its dominant share, their FJCTs tend to be the constant during job dynamics and the constraint will be assured by our CNANL scheduler.

5.3.4 Multi-Objective Optimization in Online Scheduling

In the online scheduling, jobs are submitted to the cluster dynamically and ordered according to their FJCTs.

For each node with available resource, we prefer to pack tasks of short jobs to minimize JCTs, and also maximize effective resource utilization on the node. However, the two objectives may not be achieved together. To resolve the conflict, we present a method, called Dynamic-Level, that examines the candidates of the first L short jobs, $1 \leq L \leq N$. L is doubled in each iteration so as to differentiate completion times between iterations, also make a fast convergence. As L increases, system performance improves, but JCTs inflate. To make a tradeoff, we use two metrics – remaining work and ineffective resource utilization using a weighted sum. The remaining work w of a job is the total work across all unscheduled tasks; the work of a task is the sum of components of its demand vector normalized to the corresponding capacity of the node. We use ineffective utilization instead of effective one. Since there are already some tasks scheduled and running on the node, the decision based on latter would be affected by previous ones and insensitive to current choice. The ineffective utilization is v on the node, which will be explained in next section. We choose the best combination with the lowest value of $(v + \lambda \cdot w)$. To ensure that the weighted sum is not dominated by either one, we use $\lambda = \dfrac{\bar{v}}{\bar{w}}$, where $\bar{v}$ and $\bar{w}$ is the average value.

Since the task demands are at the same magnitude, the shorter job has higher probability to be scheduled, and gets more share. Therefore,

$$f'_1(t) \geq f'_2(t) ... \geq f'_i(t) ... \geq f'_N(t). \tag{5.9}$$

Because $\sum_{k=1}^{N} f'_k(t) = \sum_{k=1}^{N} f_k(t) = N \cdot f(t)$, we have:

$$\sum_{k=1}^{i} f'_k(t) \geq \sum_{k=1}^{i} f_k(t) = i \cdot f(t) \ . \tag{5.10}$$

The prerequisite (5.2) is deduced from (5.10) by integrating both sides over time.

In practice, it is not necessary to guarantee (5.9). First, doing so at any time may degrade performance. The nodes are considered one by one in cluster scheduling. The first node may be more suitable to a long job to fully utilize its resource, while the immediate second one to a short job. Exchanging the order deteriorates performance. Second, the probability that (5.9) is valid increases greatly with both the job rank and time as shown.

To simplify the analysis without loss of generality, we only consider whether or not $f_i'(t) \geq f_{i+1}'(t)$, instead of the concrete value. Let X_i be a random variable of job j_i on the sample space $\Omega = \{0, 1\}$. $X_i = 1$ means $f_i'(t) \geq f_{i+1}'(t)$; $X_i = 0$ means $f_i'(t) < f_{i+1}'(t)$. The base probability of $X_i = 1$ is p and p is much larger than $\frac{1}{2}$. Let $X = \sum_{k=1}^{i} X_k$. The first i jobs satisfy (5.10) when at least half of their random variables equates 1, i.e., $X \geq \lceil \frac{i}{2} \rceil$. $Pr(X \geq \lceil \frac{i}{2} \rceil) = \sum_{k=\lceil \frac{i}{2} \rceil}^{i} \binom{i}{k} p^k (1-p)^{i-k}$.

If the rank i is large enough ($i > 5$), the binomial distribution of X approximates normal distribution with the mean and variance as ip and $ip(1-p)$ respectively [1]. Moreover, the coefficient of variation $\sqrt{\frac{1-p}{ip}}$ decreases, and X centers around the mean $ip > \frac{i}{2}$. $Pr(X \geq \lceil \frac{i}{2} \rceil)$ approximates 1. Suppose $p = 0.8$, when $i = 10$, $Pr(X \geq 5) > 96.7\%$. Suppose $p = 0.9$, when $i = 10$, $Pr(X \geq 5) > 99.8\%$.

In online scheduling, we check the satisfaction of the prerequisite (5.2) after each epoch. The epoch is configured as the average task length of all jobs, denoted as e. It is neither long because we cannot do so until tasks finish, nor short because we can observe most jobs satisfy (5.2) after one task length. At the end of each epoch, we check the prerequisite recursively. Suppose $C_{i-1} \geq 0$, if $c_i \geq 0$, then $C_i \geq 0$ based on (5.6). Therefore, we only need to deal with $c_i < 0$. If $C_i < 0$, we compensate the prerequisite gap $|C_i|$. When $c_i < 0$, i.e., job j_i lends credit. Trace analysis of MapReduce workload at Google and Facebook [7] shows that task number is positively correlated to job length, with the correlation coefficient as highly as 0.86 and 0.75 respectively (to do). Therefore, most of long jobs have enough tasks to use up the credit. Nevertheless, we guarantee the constraint in case some may not.

Constraint Satisfaction We first distribute the credit evenly across the fair sharing remaining time and get the credited share for each stage. If one stage does not have enough tasks to use up allocated credit, the unconsumed credit is redistributed to other stages. The same process is repeated until all stages are examined. If there is still unconsumed credit, then the job must be compensated the product discrepancy or gap between the credited product, the product of credited share and the length of one epoch, and actual product during the next epoch.

Moreover, if job j_i is NI job, we also need to check c_i against Γ_i based on (5.8). Among the three gaps, including prerequisite gap if there is, we compensate the largest one, denoted as G_i, by scheduling corresponding additional tasks during next epoch (subsection 5.4.3). There is a delay of one epoch on average, and two epochs at worst or most, for the few jobs to satisfy the constraint and prerequisite.

In this way, L is dynamically determined to optimize both effective utilization and JCTs without missing FJCTs.

5.4 Locality-aware and Network-aware Packing

Since various packing heuristics, such as dot product and cosine, have similar approximation ratios, we use dot product in our packing.

5.4.1 Locality-aware Packing

Given the L head jobs considered in job scheduling, the objective of tasks scheduling is to pack tasks from these L jobs to minimize ineffective utilization by taking data locality into consideration. Suppose the resource vector of a fragment resulting from a packing is $\boldsymbol{V} = [v^{cpu}, v^{mem}, v^{disk}, v^{network}]$. At node level, $v^{network}$ is close to 1 most of the time because network utilization is low due to high oversubscription ratio. It cannot be used to distinguish packing efficiency so that we exclude it. The size of a fragment is the average of its components except for network dimension, $v = \sum\limits_{r \in \hat{R}} v^r / |\hat{R}|$, where $\hat{R} = R - \{network\}$.

Scheduling of a non-local task introduces a new type of "fragment", called a non-local fragment. Each non-local task of job j_i in stage s requires additional network resource $d_{is}^{network}$ for fetching and transferring data and extra time Δl_{is} beyond the essential demand vector

$[d_{is}^{cpu}, d_{is}^{mem}, d_{is}^{disk}]$ and duration l_{is} if it ran locally. The corresponding resource overhead $o_{is} = l_{is} d_{is}^{network} + \Delta l_{is} \sum_{r \in \hat{R}} d_{is}^{r} / |\hat{R}|$.

The size of the non-local fragment is estimated as the ratio between its overhead and total work.

$$v_{is} = o_{is} / (o_{is} + l_{is} \sum_{r \in \hat{R}} d_{is}^{r} / |\hat{R}|). \tag{5.11}$$

Based on this metric, we can opportunistically pack tasks. At the very start, we find the minimal ineffective utilization, v^{local}, by packing only local tasks from all jobs. A non-local task has the same alignment score as the local one due to same essential demand. The smaller v_{is} is than v^{local}, the larger chance it has. The probability is

$$p_{is} = 1 - v_{is} / v^{local} \ . \tag{5.12}$$

In practice, we generate a random number g between [1, 10]. If $g \leq 10 p_{is}$, then it is packed, otherwise it is discarded. In this way, we find the best packing of tasks, whether local or not, that results in minimal ineffective utilization for each L.

5.4.2 Network-aware Scheduling

To overcome the stage barrier to network utilization, we monitor the utilization of the cluster switch. Whenever network is not fully utilized with u at the end of each epoch less than a threshold, say 90%, we accelerate NI jobs. For convenience, let reduce-NI and map-NI denote NI jobs at reduce and map stages respectively. Moreover, the largest share across all resources except for network is called three-dimensional dominant share or node share because the capacity of network bandwidth is determined by cluster switch while all other resources are determined by nodes; the corresponding product is called node product. The node product of each job j_i during n-th epoch is $\Delta \hat{F}_i'(n) = \hat{F}_i'(n) - \hat{F}_i'(n-1)$. Note that three-dimensional notation is hatted in comparison to the corresponding four-dimensional one if it has. Moreover, Δ is prefixed to the notation over n-th epoch compared with those over n epochs. Suppose it is at the end of n-th epoch, and the available network product during next epoch is $P^{network}$.

Acceleration of reduce-NI jobs. Above all, we accelerate reduce-NI jobs from top to down by supplementing their resource allocations over next epoch. Since we are not sure of the fair sharing dominant share each job will receive during (n+1)-th epoch, we conservatively use the lower bound, the node share $\frac{1}{N}$, so that no job will lend more than its dominant share over next epoch. According to the prerequisite of CS policy, the total estimated dominant product for job j_i to j_N to use is at most $\frac{N-i+1}{N}e$ over next epoch. Since dominant product is estimated as node product, each NI job j_i can use all the estimated dominant product credit provided by j_i to j_N as node product while using available network bandwidth as much as possible. Therefore, the upper bound or quota on the node product that job j_i is eligible to be supplemented during $(n+1)$-th epoch, denoted as $Q_i(n+1)$, is obtained by deducting from the total estimated dominant product the product compensations $\sum_{k=i}^{N} G_k$ and its own node product:

$$Q_i(n+1) = \frac{N-i+1}{N}e - \Delta\hat{F}_i'(n) - \sum_{k=i}^{N} G_k \tag{5.13}$$

Suppose the reduce-NI jobs are j_{i_1}, ..., j_{i_k}, ..., j_{i_x}. We wish to compute the product supplement for each job j_{i_k} over $(n+1)$-th epoch, denoted as $S_{i_k}(n+1)$. The eligible node product for each NI job j_{i_k} to supplement, denoted as $E_{i_k}(n{+}1)$, depends on its node product quota and how much node product credit that its predecessor jobs have borrowed from jobs j_{i_k} to j_N over $(n+1)$-th epoch, denoted as $\Delta\hat{C}_{i_k-1}(n+1)$. Since the first NI job j_{i_1} does not have any predecessor, $\Delta\hat{C}_{i_1-1}(n+1) = 0$. We have,

$$E_{i_k}(n+1) = Q_{i_k}(n+1) - \Delta\hat{C}_{i_k-1}(n+1). \tag{5.14}$$

Besides, the corresponding network product of $S_{i_k}(n+1)$ is subject to network constraint,

$$h_{i_k} \cdot S_{i_k}(n+1) \leq P^{network} - \sum_{q=1}^{k-1} h_{i_q} \cdot S_{i_q}(n+1), \tag{5.15}$$

where h_{i_k} is the ratio between the network to node share of its reduce task. $\Delta\hat{C}_{i_{k+1}-1}(n+1)$ is updated as:

$$\begin{aligned}\Delta\hat{C}_{i_{k+1}-1}(n+1) &= \Delta\hat{C}_{i_k-1}(n+1) + S_{i_k}(n+1) \\ &\quad -(Q_{i_k}(n+1) - Q_{i_{k+1}}(n+1)).\end{aligned} \tag{5.16}$$

The iteration stops when either $E_{i_k}(n+1)$ in (5.14) or the corresponding remaining network product on the right-side of (5.15) is not above 0.

Acceleration of map-NI jobs. If there is available network over the next epoch after accelerating reduce-NI jobs or no NI job is at reduce stage, accelerating map-NI jobs will further improve network utilization. To this end, we need to decide which map-NI jobs are accelerated and how much product to supplement for each job. On the one side, accelerating the map-NI job that is closest to the completion of its map stage will make use of idle network in the short term; on the other side, accelerating the job that has the least remaining map work and the most network demand not only optimizes network utilization in the long term, but also affects the optimization of other JCTs the least.

Therefore, we design two phases for accepting and selecting jobs respectively. In the first phase, only the jobs whose acceleration will open a time window for network utilization improvement are accepted to the second phase. For each map-NI job j_g, the completion time of map stage under CS policy t'_g is predicted based on its moving average finishing speed over epochs; the corresponding FCT of map stage is t_g. Moreover, we also calculate its accelerated time of map stage after supplementing its eligible product t''_g. According to

CS policy, some jobs execute faster than they do under DRF while others may run slower. Therefore, if $t'_g < t_g$, then the time window is $t_g - t''_g$, otherwise is $t'_g - t''_g$. If the time window is positive, then j_g is accepted. The available network j_g is shown on the Figure (to do)

In the second phase, the job that has the highest ratio between the total network demand and remaining map work across all accepted jobs. Suppose the selected job j_g is the k-th job in the updated accelerated jobs j_{i_1}, ..., j_{i_k}, ..., j_{i_m}. Its eligible node product is obtained by deducting from its node product quota all the product supplements of below jobs, in addition to the credit borrowed by above jobs:

$$\begin{aligned} E_{i_k}(n+1) = Q_{i_k}(n+1) - \Delta\hat{C}_{i_k-1}(n+1) \\ - \sum_{q=k+1}^{m} E_{i_q}(n+1). \end{aligned} \tag{5.17}$$

Because the calculation of supplemented product is based on CS policy, other jobs are not decelerated.

5.4.3 Allocation of Product Compensation and Supplement

For allocating the compensated and supplemented product, we have two options—reservation and adjustment of packing scores. Reserving resources may be inefficient because some resources are wasted during the reservation period. The problem is even worse if task demand is large. Moreover, the resulting resource utilization may be not high due to no packing. By contrast, the alternative does not have such issues.

5.5 Evaluation

The CANAL scheduler was implemented on Hadoop Yarn 2.6.0. We use both simulations of large-scale cluster settings and our testbed of a 16-node private cluster running real MapReduce applications to show CANAL exhibits significant advantage over competitive schedulers, such as Tetris.

This work does not raise any ethical issues.

Table 5.7: MapReduce benchmark summary.

Name	Type	Description
TeraSort	I/O	Sort the input data into a total order
TeraGen	I/O	Generate and write data into system
Grep	I/O	Extract matching regular expression
RandomWrite	I/O	Random write words into file
WordCount	I/O	Count words in the input file
PiEst	CPU	Estimate Pi using Monte Carlo method
Bayes	CPU	Contruct Bayes Classifier on input data
Kmean	CPU	Cluster analysis using K-mean method
RandomForest	CPU	Classification using Random Forest
Matrix	CPU	Matrix add and multiplication

5.5.1 Methodology

Alternate Schedulers We compared CANAL with Tetris, SRJF, Delay scheduling based on DRF (Delay/DRF), locality-aware only scheduling (LAO) and network-aware only scheduling (NAO). Each job has the same 4-dimensional task demands across all schedulers.

Workload We evaluated CANAL scheduler using 10 MapReduce applications, most of which were widely used in evaluations of MapReduce framework by previous works [69] [50] [12] [16] [45] [9 Table 5.7 shows their main characteristics. The number of map and reduce tasks were configured in a fixed ratio in each application.

Testbed In the private cluster, each node has 12 CPU cores, 32 GB memory and three 1 TB 7200 RPM disk drives with a 110MB/s peak bandwidth for each one. Since hyperthreading is enabled, each node has 24 logical CPUs, and runs Linux 3.16. The cluster spans 2 racks, each with 8 nodes. Each rack has a rack switch with a 1Gbps links. Racks in turn are connected with a 1Gbps Ethernet. The HDFS block size was set to 256 MB. For the benchmark, the resource requests and durations of both local and non-local tasks are monitored by running each with sufficient resource allocation.

Simulation To show the effectiveness of CANAL in large-scale clusters with full-bisection bandwidth networks, we developed a simulator that models resource allocations of nodes and networks. The simulator simulates two clusters of two-level tree topology with

Table 5.8: Job type and size distribution in simulation

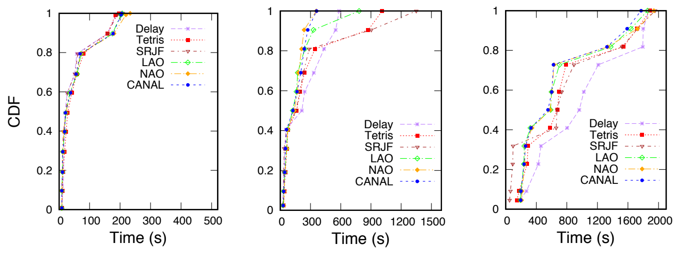
Bin	# Maps	%	#	# of I/O(size)	# of CPU(size)
1	1	39%	76	28 RWrite(1)	20 Matrix(1) 28 RForest(1)
2	2	16%	32	TeraGen(2)	
3	3-20	14%	28		10 Bayes(7) 18 Kmean(15)
4	21-60	9%	18		RForest(23)
5	61-150	6%	12	WCount(80)	
6	151-300	6%	12	6 WCount(240) 6 TeraSort(300)	
7	301-500	4%	8		Pi(500)
8	501-1500	4%	8	WCount(640)	
9	> 1501	3%	6	2 TeSort(1800)	4 RF(12000)

200 and 1200 nodes respectively. The first cluster is used to simulate above workload based on Facebook trace. There are 200 machines at the bottom level, with each modeling the node in the testbed cluster. Every 20 machines form a rack and are linked with a rack switch with 1 Gbps links. Every 10 rack switches are connected to the cluster switch. The default oversubscription of the physical network is 10, i.e., the cluster switch has 2Gbps links.

The second 1200-node cluster spans 30 racks. It is used to replay a randomly selected 24-hours long workload trace from Google clusters, with each node modeling Google machines. Since the task demands and resource utilization are not shown in the trace, all nodes are connected to 1 Gbps ethernet.

5.5.2 Simulation Results

We started by generating a common submission schedule for 200 jobs based on the workload trace from Facebook reported in [97]. The job order was randomly generated to eliminate the skewness of the occurrences that a small job submitted after a large one. The job size, in terms of number of map tasks, and the corresponding distribution are shown in the second and third columns of Table 5.8. We adjusted the mean inter-arrival time between jobs based on the cluster scale and set the mean to 4 seconds. It makes our submission



(a) Bins 1-3 (b) Bins 4-6 (c) Bins 7-9

Figure 5.6: CDFs of JCTs in various bin ranges in simulation.

schedule 810 seconds long. The actual number of running jobs, the number of I/O-bound and CPU-bound jobs are listed in the fourth to sixth columns respectively in Table 5.8.

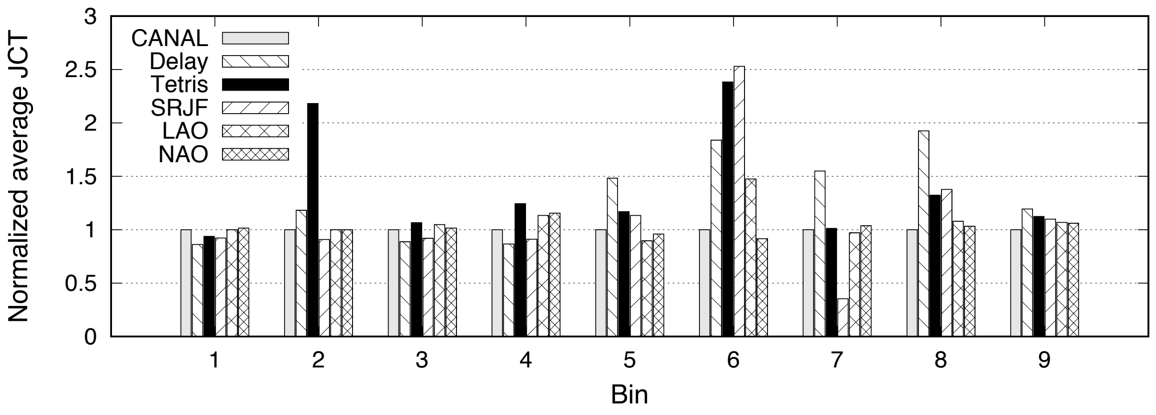


Figure 5.7: Normalized average JCT in each bin under different schedulers in simulation.

JCT Figure 5.6 shows a CDF of JCTs for various ranges of bins. We observe that credit sharing has a negligible effect on small jobs (bin 1-3) as expected. Small jobs perform nearly equally under all schedulers. Under DRF, Tetris and credit-based schedulers, all runnable tasks of small jobs are launched together once they are submitted because they cannot use up their shares. Under SRJF, they are more of the case. However, CANAL significantly speeds up larger jobs over network-oblivious schedulers, including Delay, Tetris, SRJF and LAO. The effect on JCTs is more subtle, and to illustrate it clearly, we have plotted Figure 5.7, which shows the average JCT in each bin under different schedulers. The results are normalized with respect to the performance under CANAL. Compared with Delay and Tetris, CANAL accelerates about 60% of the medium jobs (bin 4-6) by up to 1.84 and 2.38

times respectively partly because it lets shorter jobs run first by borrowing resources from longer jobs, and partly because network-intensive jobs (TeraSort in bin 6) are accelerated to avoid network contention. Tetris is even worse than Delay for network-intensive jobs because the former lets short non-network-intensive jobs run first so that network contention is more severe. For the large jobs (bin 7-9), CANAL improves more than 90% of them by up to 1.92 and 1.32 times respectively in comparison to Delay and Tetris due to the same reasons. Compared with LAO, it improves 32% of medium and 40% of large jobs by up to 1.47 and 1.08 times respectively at a little cost of slowing down 28% of medium jobs by up to 0.9 times because LAO does not avoid network contention though the contention is alleviated by credit sharing policy. The performance of NAO is very close to CANAL. Nevertheless, it leads to improvement of 1.03-1.06 times over NAO. Though SRJF optimizes the JCTs for NNI jobs, the NI jobs are slowed down because it is not aware of network contention. Be contrast, CANAL speeds up NI jobs by up to 2.52 times over SRJF.

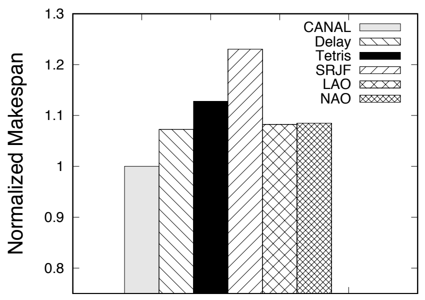


Figure 5.8: Makespan under different schedulers in simulation.

Makespan Figure 5.8 shows the makespan under different schedulers. The results are normalized to the one under CANAL. We can see that CANAL improves makespan by up to 23% over network-oblivious schedulers and 8% over NAO. SRJF significantly degrades system performance because it is unaware of network contention that long jobs experience later and schedules short jobs greedily instead. Tetris is 12.8% worse than CANAL due to the same problem. Delay alleviates this problem since long network-intensive jobs get the same resource shares as short jobs. LAO performs similarly as Delay due to no acceleration of network-intensive jobs either. NAO is a little worse than CANAL, attributing to the fact that it only considers network contention.

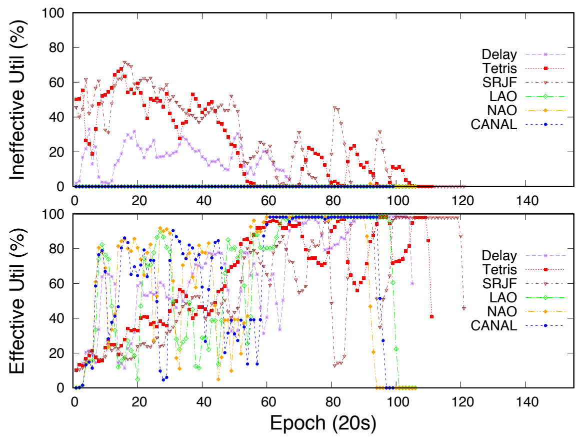


Figure 5.9: Ineffective and effective network utilizations under different schedulers in simulation.

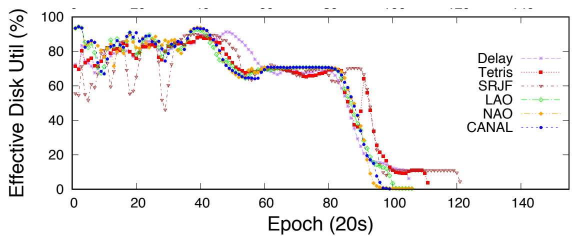


Figure 5.10: Effective disk utilizations under different schedulers in simulation.

To investigate the reasons, we first have plotted both ineffective utilizations due to off-rack input data and effective utilizations of network under all schedulers in Figure 5.9. We can see that credit-sharing schedulers are able to enhance effective network utilization at the first half of cluster scheduling because they schedule relatively short network-intensive jobs. Moreover, they greatly reduce ineffective utilization. On one hand, the locality-aware strategy in both CANAL and LAO avoids scheduling off-rack tasks because off-rack data transfer degrades performance; on the other hand, the network-boosting strategy in CANAL and NAO makes use of cluster switch effectively so that off-rack input data does not have enough bandwidth. By contrast, non-credit-sharing schedulers have low effective utilizations due to no such preference. Moreover, they result in high ineffective utilizations because

SRJF and Tetris cannot avoid off-rack data transfer, and Delay cannot optimize data locality in a multi-resource cluster. At the second half, there is severe network contention due to TeraSort jobs in bin 9 so the network utilization in credit-sharing schedulers is close to 100%. However, non-credit-sharing schedulers cannot reach full network utilization due to off-rack data transfer in Tetris and SRJF, and Delay schedule the job with least dominant share.

In the disk dimension, the effective utilizations are shown in Figure 5.10. Compared with SRJF and Tetris, CANAL dynamically improves effective disk utilizations in the first half of scheduling by up to 20% with no need of parameters. Delay scheduling cannot maintain high effective disk utilizations in multi-resource environment by greedy data locality policy. Two main factors are attributed to its worse performance compared with CANAL. First, it cannot make use of the complementary disk demands of tasks. Second, the head-of-queue job launches more non-local tasks because any of its preferred nodes does not have enough available resources to accommodate any of its tasks. The disk utilizations of LAO is close to CANAL since both use the same strategy. NAO can also have high utilizations because the network-intensive jobs it accelerates are also disk-intensive.

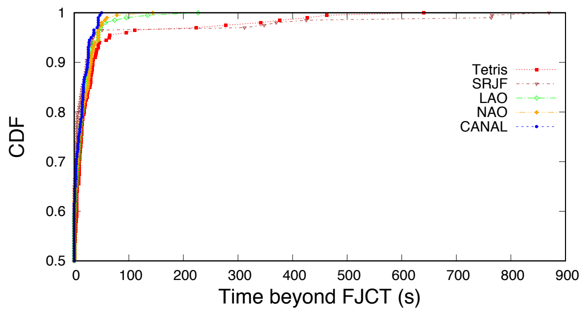
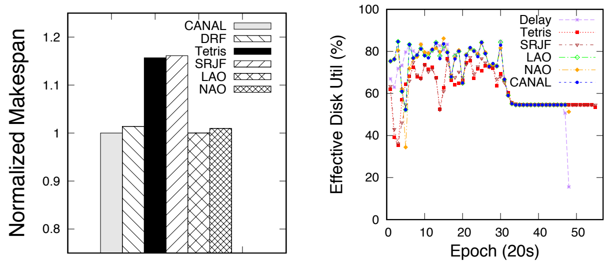


Figure 5.11: CDFs of time beyond FJCT under different schedulers in simulation.

JCT vs. Fairness Figure 5.11 shows the CDFs of the amount of time that goes beyond FJCT under different schedulers. We can see that Tetris and SRTF have 8% (16) and 3.5% (7) jobs that surpasses over FJCT under DRF by more than 40s (twice of the average task length), with the largest amount as 640s and 870s respectively. By contrast, credit-sharing schedulers, including CANAL, LAO and NAO, greatly reduces both the number of jobs



(a) Makespan (b) Disk Util

Figure 5.12: Benefits of locality-aware packing.

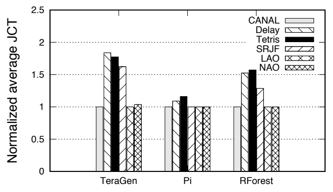
that violate FJCT and the amount of time that exceeds beyond FJCT. Among the three, CANAL has the minimal number of jobs, 2.5% (5) jobs, that exceeds FJCT, with the largest amount as only 50s because it considers both network and locality. Credit-sharing schedulers outperform the other two because they are guided by a theory that guarantees FJCT. In practice, there is still a chance of missing FJCT because there is no node-local nodes or lack of sufficient network bandwidth for transferring remote data for jobs that pass FJCT. Unlike rigid demand for memory, network bandwidth is elastic. Scheduling the few jobs that is about to pass FJCT even if the network is not sufficient, the number of jobs that violate FJCT can be reduced to zero.

5.5.3 Benefits of Locality and Network Aware Packing

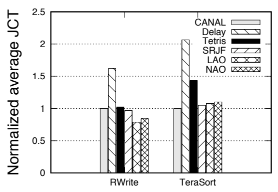
In this section, we demonstrate effectiveness of LAP and NAP in the particularly designed experiments.

Benefit from LAP. LAP is the component in CANAL conducting the locality-aware packing. To isolate the effect from network-aware scheduling, we selected the *TeraGen*, *Pi* and *RForest* applications in the evaluation because none of them are network-intensive. Each *TeraGen* job consisted of 400 map tasks, each *Pi* and *RForest* contained 100 map tasks. We submitted 50 *TeraGen*, 50 *Pi* and 100 *RForest* jobs.

Figure 5.12(a) shows the makespan under different schedulers. The results are normalized to the one under CANAL. Since there is no network effect, CANAL and LAO are



(a) LAP



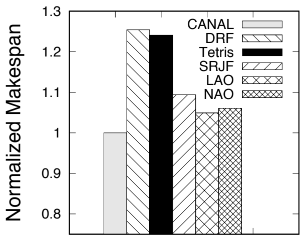
(b) NAP

Figure 5.13: Normalized JCT of LAP and NAP

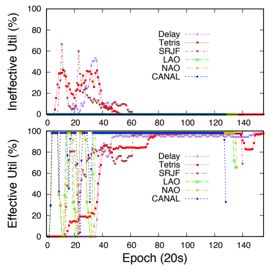
reduced to the same scheduler. We can see that CANAL improves makespan by about 16% over locality-oblivious schedulers except for NAO. Both NAO and Delay are only 1% worse because the former applies the same Dynamic-Level to optimize effective disk utilization as CANAL, the latter optimizes data locality. The performance of different schedulers are reflected in the corresponding disk utilizations as shown in Figure 5.12(b). Figure 5.13(a) shows the normalized JCT of each type of jobs under different schedulers. CANAL speeds up *TeraGen* and *RForest* by 1.28-1.83 times compared with non-credit-sharing schedulers. The speedup for Pi jobs is limited because Pi jobs are short due to small task demands.

Benefit from NAP. NAP module is responsible for network-aware packing. Accelerating NI jobs to have more available reduce tasks not only makes full use of network bandwidth in highly over-subscribed clusters, but also improves network utilization in clusters with full-bisection bandwidth network, such as fat-trees [11]. To demonstrate its effectiveness, we limited the influence from data locality by running 80 *RWrite* and 20 *TeraSort* jobs. *RWrite* does not require input data. Each *RWrite* contained 150 map tasks and each *TeraSort* contained 600 map tasks.

Figure 5.14(a) shows the normalized makespan under different schedulers. We can see that CANAL reduces makespan by 9%-25% over network-oblivious schedulers except for LAO. LAO is only 5% worse because it also accelerates short network-intensive jobs based on credit sharing. NAO is 6% worse because it does not consider data locality. The reason is demonstrated as high effective network utilizations of credit sharing schedulers as shown in Figure 5.14(b). Figure 5.13(b) shows the normalized JCT of each type of jobs under different



(a) Makespan



(b) Network Util

Figure 5.14: Benefits of network-aware packing.

Table 5.9: Job type and size distribution in testbed

JobSize	%	#	# of I/O(size)	# of CPU(size)
1-2	55%	22	11 RForest(1)	11 Matrix(1)
3-20	14%	6		Kmean(15)
21-150	15%	4	TeraSort(120)	
151-300	6%	2		RWrite(250)
301-500	4%	2	WCount(320)	
> 500	8%	4	2 TeraSort(600)	2 TeraGen(600)

schedulers. CANAL speeds up jobs by up to 2.06 times compared with non-credit-sharing schedulers. Compared with CANAL, SRJF improves *RWrite* jobs by 3% and slows down *TeraSort* jobs by 5%.

5.5.4 Testbed Experiment

The job order was generated as before in the simulation. The number of jobs and the mean inter-arrival time between jobs were adjusted to 40 and 14 seconds respectively. It makes our submission schedule 560 seconds long. The number of running jobs, the number of I/O-bound and CPU-bound jobs are listed in the fourth to sixth columns respectively in Table 5.9.

Figure 5.15 shows the average JCT under different schedulers. The results are normalized with respect to the performance under CANAL. Compared with Delay and Tetris, CANAL accelerates jobs by up to 2.66 and 1.25 times respectively because it schedules shorter jobs based on credit-sharing and network-intensive jobs (TeraSort) to avoid network

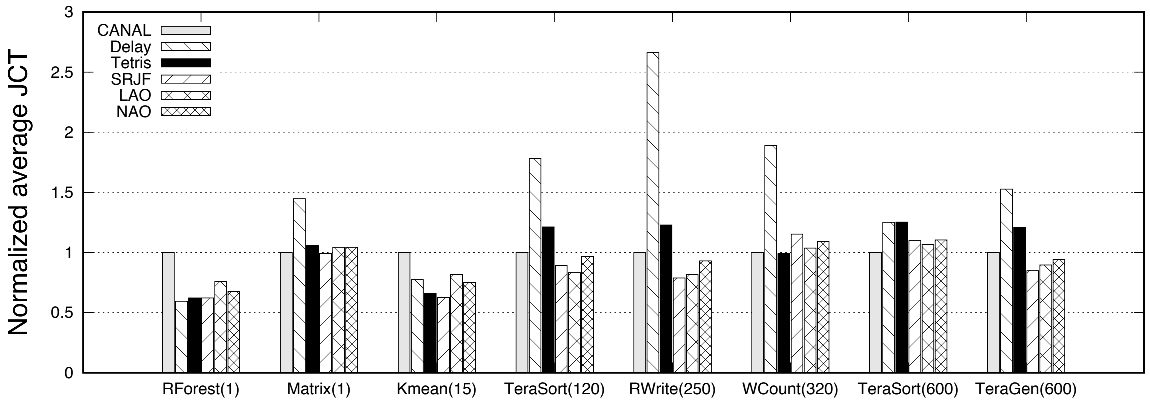


Figure 5.15: Average JCT under different schedulers in Testbed.

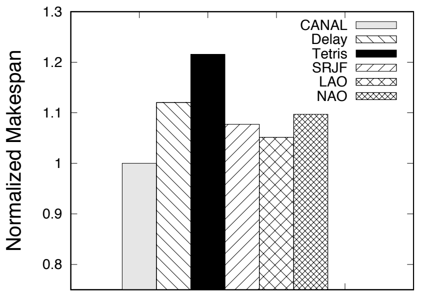


Figure 5.16: Makespan under different schedulers in Testbed.

contention. Compared with LAO, it improves jobs by up to 1.47 and 1.08 times respectively at a little cost of slowing down jobs by up to 0.9 times because LAO does not avoid network contention. The performance of NAO is very close to CANAL. Nevertheless, it leads to improvement of 1.04-1.10 times over NAO. Though SRJF speeds up NNI jobs, the NI jobs are decelerated because it is not aware of network contention. Be contrast, CANAL improves NI jobs by up to 1.09 times over SRJF.

Figure 5.16 shows the makespans by different schedulers. They are normalized to the one under CANAL. We can see that CANAL reduces makespan by 5-21.5% over network-oblivious schedulers and 9.7% over NAO. Tetris significantly degrades system efficiency because it is unaware of network contention that long jobs experience later and schedules short jobs instead. Delay and SRJF are 12% and 7.7% worse than CANAL due to the same problem. LAO performs similarly as Delay due to no acceleration of network-intensive jobs either. NAO is 9% worse than CANAL because it only considers network contention.

5.6 Summary

In this paper, we present credit sharing policy for resolving conflicts among three different objectives: system efficiency, minimal JCTs and fairness. According to this policy, we optimize both effective resource utilizations and JCTs while guaranteeing FJCTs. Network-intensive jobs are further accelerated to improve JCTs and system efficiency in accordance with CS policy. We designed CANAL and evaluated it in both trace-driven large-scal simulation and testbed implementation using representative benchmarks and compared it with several schedulers. Experimental results show the advantages of CANAL in optimizing both system efficiency and JCTs over other schedulers, particularly for network-intensive jobs. CANAL also guarantees FJCTs resulting from DRF while other cannot. In the future, we have following directions to work on: (1) extend CANAL to multi-stage jobs, such as DAG jobs; (2) further improve the performance of DRF while satisfying the fairness requirement.

CHAPTER 6 CONCLUSIONS AND FUTURE WORK

This dissertation aims to build an automatic resource management and performance optimization system in clusters in order to guarantee application performance, improve system efficiency and optimize completion times of individual jobs. In this chapter, we summarize the approaches presented in this dissertation and point out directions for future work.

6.1 Conclusions

Cloud and cluster computing are two important computing paradigms. Although they have gained adequate popularity today, there are still some key obstacles to large scale adoption and deployment. Dynamic resource sharing and allocation is one of the top challenges. The success of cloud and cluster computing severely depends on the effectiveness and efficiency of automatic resource management and cluster scheduling strategies.

Automatic resource configuration is pivotal to the quality and availability of cloud service. To satisfy SLOs for applications in cloud, we propose a novel adaptive fuzzy control approach to allocate adequate CPU resources (no surplus). It is able to adapt resource allocations not only to the workload and cloud dynamics, but also provide response time guarantee for applications. We have conducted extensive experiments to compare the performance of Adaptive Fuzzy control against other three methods. Our results show that Adaptive fuzzy control has the best stability, shortest settling time and minimum overshoot. The power is that not only does it explore the nonlinearity of resource allocation to performance, but also it takes into consideration the trend of error to estimate right resource allocation.

For the multi-resource configuration, we propose that IaaS should dynamically balance multiple virtualized resources across all tiers based on application SLOs. The SLOs are defined in terms of response time. We present BConf, a QoS-aware framework for balanced configuration of multi-resources that integrates model predictive control (MPC) and adaptive proportional integral control (adaptive PI). BConf takes advantage of MPC to actively balance resources for co-hosted applications based on a novel resource metric. Both black-box and gray-box models are developed to capture performance to resources relationship for comparison. The gray-box model is built on generic OS-level metrics and hardware events in addition to resource actuators and performance. The resource penalty is introduced to measure the imbalanced degree of a configuration based on the model. It effectively punishes decisions using an imbalanced configuration. Using the model and the metric, BConf dynamically balances resources by minimizing the resource penalty for each resident application under the constraint of SLOs. Within the framework, constrained resources are arbitrated in a coordinated way to effectively restrict propagation of configuration errors. Compared with a representative resource partition approach, BConf reduces resource usages by up to 50%, improves stability by more than 35.6%, and has a much shorter settling time. BConf effectively coordinates resources during resource contention. The saved resources can be used either to pack more workloads, or to promote performance levels.

Modern cluster systems, such as Mesos and Yarn, provides fine-grain resource allocation mechanism to satisfy heterogenous demands of various tasks. MapReduce remains dominant parallel and distributed programming paradigm for applications to process big data in clusters. It is one of the core technologies powering IT giants like Facebook. In shared clusters, scheduling has three different objectives−system efficiency, fairness, and minimal job completion times (JCT). In each pair of two objectives, one may conflict with the other under state-of-the-art policies or strategies. To resolve the conflicts, we propose a theorem-proof policy called credit sharing (CS) policy, that guarantees JCTs under DRF. It recursively accounts for the credit that long jobs contribute to the short jobs, and short jobs pay down or off their borrowed credit later in the process of execution or upon completions. In addition to accelerating short jobs, the cluster scheduler is enabled to pack tasks of short jobs, getting system efficiency nearly optimized since task demands of short jobs are in the same magnitude as those of long jobs. Since a non-local task introduces a “new" non-local fragment by wasting extra resources, we estimate the size of all the normal and non-local fragments resulting from a packing. According to CS policy and the size of total fragment, we present a method called Dynamic-Level for calculating the number of head jobs dynamically that optimizes both the effective resource utilization and JCTs. To overcome the stage-barrier, we monitor network and schedule additional tasks for some network-intensive jobs by borrowing credit from long jobs based on CS policy. The packing scores of the additional tasks are dynamically adjusted and scheduled together uniformly with all other tasks in Dynamic-Level. We conduct a detailed evaluation in both trace-driven simulation and prototype implementation. Compared with the most competitive scheduler, we find that CANAL significantly speed up jobs by up to 2.36 time, particularly for network-intensive jobs, and reduce makespan by 7%. Moreover, each job under CANAL does not decelerate compared with DRF.

6.2 Future Work

There are several directions and challenges along the line of this dissertation.

For a single-resource configuration, we currently tune CPU resource by adaptive fuzzy control. It should be able to tune any other resource for any application. In the future, we would like to apply it to data-intensive applications, such as MapReduce job.

For the multi-resource configuration, we consider resources as the only cost. Different configurations may cause different energy consumptions. In the future, we would like to extend the cost to the energy so as to save the cost of both resources and energy.

For the cluster scheduling, the credit sharing policy has big potential. It not only unleashes the power of packing, but also guarantees the JCT under DRF. Current multi-resource fairness−DRF, is instantaneous fairness. It not only degrades system efficiency by resulting in large resource fragments, but also increases JCTs. Other policies are interval-based fairness [79], or tradeoff between fairness and efficiency [51]. None of them assures fairness. Based CS policy, we can realize result-oriented fairness that improves JCTs by controlling the completion time of the last task of each job after their advanced execution. The final completion of each job is thus the same as the packing version of DRF. We can expect that the JCTs of network-intensive jobs can largely be reduced compared with DRF while satisfying the essential envy-free feature of fairness.

REFERENCES

[1] https://en.wikipedia.org/wiki/Binomial_distribution.

[2] "Apache hadoop," *http://hadoop.apache.org.*

[3] "Mysql." [Online]. Available: http://www.mysql.com

[4] "The transaction processing council (tpc)." [Online]. Available: http://www.tpc.org/tpcw

[5] "The transaction processing council (tpc)." [Online]. Available: http://www.tpc.org

[6] "Yarn capacity scheduler." [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html

[7] "Traces of google workloads," *http://code.google.com/p/googleclusterdata/*, 2011.

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899

[9] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou, "Re-optimizing data-parallel computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228327

[10] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "Blinkdb: Queries with bounded errors and bounded response times on very large data," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/2465351.2465355

[11] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967

[12] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. G. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: coping with skewed content popularity in mapreduce clusters," in *EuroSys*, 2011, pp. 287–300.

[13] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica, "Pacman: Coordinated memory caching for parallel jobs," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 20–20. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228326

[14] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 265–278. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924962

[15] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *OSDI*, 2010.

[16] S. Babu, "Towards automatic optimization of mapreduce programs," in *SoCC*, 2010, pp. 137–142.

[17] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP*, 2003.

[18] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, "Hierarchical scheduling for diverse datacenter workloads," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 4:1–4:15. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523637

[19] T. Bonald and J. Roberts, "Multi-Resource Fairness: Objectives, Algorithms and Performance," in *ACM Sigmetrics*, ser. ACM Sigmetrics, Portland, United States, 2015. [Online]. Available: https://hal.inria.fr/hal-01243985

[20] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 285–300. [Online]. Available: http://dl.acm.org/citation.cfm?id=2685048.2685071

[21] E. F. Camacho and C. B. Alba, "Model predictive control," in *New York:Springer-Verlag*, 2004.

[22] C. Charles E., "Supercomputing-balancing resources," in *Spectrum*, 1992.

[23] C.-L. Chen, "Ieee 802.11e edca qos provisioning with dynamic fuzzy control and cross-layer interface," in *ICCCN*, 2007.

[24] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "Hug: Multi-resource fairness for correlated and elastic demands," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, ser. NSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 407–424. [Online]. Available: http://dl.acm.org/citation.cfm?id=2930611.2930638

[25] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 393–406. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787480

[26] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 98–109, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2043164.2018448

[27] W. J. Dally, "A universal parallel computer architecture," in *New Generation Computing*, 1993.

[28] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004, pp. 137–150.

[29] ——, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[30] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server," in *In Proceedings of the Network Operations and Management Symposium, Florence, Italy*, 2002.

[31] A. Eldawy and M. F. Mokbel, "Spatialhadoop: A mapreduce framework for spatial data," in *in ICDE*, 2015.

[32] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: Guaranteed job latency in data parallel clusters," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 99–112. [Online]. Available: http://doi.acm.org/10.1145/2168836.2168847

[33] W. D. Frazer and A. C. McKellar, "Samplesort: A sampling approach to minimal storage tree sorting," *J. ACM*, vol. 17, no. 3, pp. 496–507, Jul. 1970. [Online]. Available: http://doi.acm.org/10.1145/321592.321600

[34] E. J. Friedman and S. G. Henderson, "Fairness and efficiency in processor sharing protocols to minimize sojourn times," 2002.

[35] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*,

ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 323–336. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972490

[36] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 365–378. [Online]. Available: http://doi.acm.org/10.1145/2465351.2465387

[37] J. Gong and C.-Z. Xu, "vpnp: Automated coordination of power and performance in virtualized datacenters," in *IWQoS*, 2010.

[38] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *CNSM*, 2010.

[39] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 455–466. [Online]. Available: http://doi.acm.org/10.1145/2619239.2626334

[40] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, "Altruistic scheduling in multi-resource clusters," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 65–80. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026884

[41] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "Graphene: Packing and dependency-aware scheduling for data-parallel clusters," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 81–97. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026885

[42] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Middleware*, 2006.

[43] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, "Feedback control of computing systems," in *IEEE Press/Wiley Interscience*, 2004.

[44] J. L. Hellerstein, S. Singhal, and Q. Wang, "Research challenges in control engineering of computing systems," in *Transactions on Network and Service Management, 6(4)*, 2009.

[45] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics," in *CIDR*, 2011, pp. 261–272.

[46] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[47] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1272998.1273005

[48] ——, "Dryad: Distributed data-parallel programs from sequential building blocks," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1272998.1273005

[49] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 261–276. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629601

[50] H. Jin, X. Yang, X.-H. Sun, and I. Raicu, "Adapt: Availability-aware mapreduce data placement for non-dedicated distributed computing," in *ICDCS*, 2012, pp. 516–525.

[51] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework," *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 1785–1798, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1109/TNET.2012.2233213

[52] C.-H. Jung, C.-S. Ham, and K.-I. Lee, "A real-time self-tuning fuzzy controller through scaling factor adjustment for the steam generator of npp," *Fuzzy Sets Syst.*, vol. 74, 1995.

[53] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *ICAC*, 2009.

[54] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-milne, and M. Yoder, "Impala: A modern, open-source sql engine for hadoop," in *In Proc. CIDRâĂŹ15*, 2015.

[55] H. Kung, "Memory requirements for balanced computer architectures," in *ACM SIGARCH Computer Architecture News*, 1986.

[56] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, March 2018.

[57] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, Jan 2018.

[58] S. Li, S. Hu, S. Wang, L. Su, T. F. Abdelzaher, I. Gupta, and R. Pace, "WOHA: deadline-aware map-reduce workflow scheduling framework over hadoop clusters," in *ICDCS*. IEEE Computer Society, 2014, pp. 93–103.

[59] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," in *In UAI*, 2010.

[60] K. Morton, M. Balazinska, and D. Grossman, “Paratimer: A progress indicator for mapreduce dags,” in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’10. New York, NY, USA: ACM, 2010, pp. 507–518. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807223

[61] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, “Naiad: A timely dataflow system,” in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP ’13. New York, NY, USA: ACM, 2013, pp. 439–455. [Online]. Available: http://doi.acm.org/10.1145/2517349.2522738

[62] R. Nathuji, A. Kansal, and A. Ghaffarkhah, “Q-clouds: Managing performance interference effects for qos-aware clouds,” in *EuroSys*, 2010.

[63] NIST, http://www.nist.gov/itl/iad/cloud-050212.cfm.

[64] S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen, “Cost-effective cloud hpc resource provisioning by building semi-elastic virtual clusters,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’13. New York, NY, USA: ACM, 2013, pp. 56:1–56:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503236

[65] D. Ongaro, A. L. Cox, and S. Rixner, “Scheduling i/o in virtual machine monitors,” in *VEE*, 2008.

[66] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, “Automated control of multiple virtualized resources,” in *EuroSys*, 2009.

[67] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, “Adaptive control of virtualized resources in utility computing environments,” in *EuroSys*, 2007.

[68] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, “Performance evaluation of virtualization technologies for server consolidation,” in *HP Labs, Tech. Rep. HPL-2007-59*, 2007.

[69] B. Palanisamy, A. Singh, L. Liu, and B. Jain, “Purlieus: locality-aware resource allocation for mapreduce in a cloud,” in *SC*, 2011, p. 58.

[70] D. C. Parkes, A. D. Procaccia, and N. Shah, “Beyond dominant resource fairness: Extensions, limitations, and indivisibilities,” *ACM Trans. Econ. Comput.*, vol. 3, no. 1, pp. 3:1–3:22, Mar. 2015. [Online]. Available: http://doi.acm.org/10.1145/2739040

[71] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, “Faircloud: Sharing the network in cloud computing,” in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM ’12. New York, NY, USA: ACM, 2012, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342396

[72] J. Rao, Y. Wei, J. Gong, and C. Xu, “Qos guarantees and service differentiation for dynamic cloud applications,” *IEEE Transactions on Network and Service Management*, vol. 10, no. 1, pp. 43–55, March 2013.

[73] J. Rao, X. Bu, and C.-Z. Xu, “A distributed self-learning approach for elastic provisioning of virtualized cloud resources,” in *MASCOTS*, 2011.

[74] J. Rao, X. Bu, C.-Z. Xu, and L. Wang, “Vconf: A reinforcement learning approach to virtual machines auto-configuration,” in *ICAC*, 2009.

[75] J. Rao, Y. Wei, J. Gong, and C.-Z. Xu, “Dynaqos: Model-free self-tuning fuzzy control of virtualized resources for qos provisioning,” in *IWQoS*, 2011.

[76] L. U. Rina Panigraphy, Kunal Talwar and U. Wieder, “Heuristics for vector bin packing,” in *MSR Technical Report*, 2011.

[77] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, “Cloudscale: Elastic resource scaling for multi-tenant cloud systems,” in *SOCC*, 2011.

[78] I. Stoica, H. Zhang, and T. S. E. Ng, “A hierarchical fair service curve algorithm for link-sharing, real-time and priority services,” in *Proceedings of the ACM SIGCOMM ’97 Conference on Applications, Technologies, Architectures, and Protocols for*

Computer Communication, ser. SIGCOMM '97. New York, NY, USA: ACM, 1997, pp. 249–262. [Online]. Available: http://doi.acm.org/10.1145/263105.263175

[79] J. Tan, L. Zhang, M. Li, and Y. Wang, "Multi-resource fair sharing for multiclass workflows," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 4, pp. 31–37, Jun. 2015. [Online]. Available: http://doi.acm.org/10.1145/2788402.2788408

[80] The dm-ioband bandwidth controller., http://sourceforge.net/apps/trac/ioband/wiki/dm-ioband.

[81] Tomcat, http://tomcat.apache.org/.

[82] 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*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633

[83] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys '15. New York, NY, USA: ACM, 2015, pp. 18:1–18:17. [Online]. Available: http://doi.acm.org/10.1145/2741948.2741964

[84] R. Wang, D. M. Kusic, and N. Kandasamy, "A distributed control framework for performance management of virtualized computing environments," in *ICAC*, 2010.

[85] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 190–203, Feb. 2016. [Online]. Available: http://dx.doi.org/10.1109/TNET.2014.2362745

[86] Z. X. Wang W. and S. S., "Utilization and slo-based control for dynamic sizing of resource partitions," in *Proc. of the 16th IFIP/IEEE Distributed Systems: Operations and Management*, 2005.

[87] J. Wei and C.-Z. Xu, “eqos: Provisioning of client-perceived end-to-end qos guarantees in web servers,” *IEEE Transaction on Computer*, vol. 55, 2006.

[88] Y. Wei and C. Xu, “Dynamic balanced configuration of multi-resources in virtualized clusters,” in *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems(MASCOTS)*, vol. 00, Aug. 2013, pp. 60–69. [Online]. Available: doi.ieeecomputersociety.org/10.1109/MASCOTS.2013.14

[89] G. J. Woeginger, “There is no asymptotic ptas for two-dimensional vector packing,” in *Information Processing Letters*, vol. 64:6, 1997, pp. 293âĂŞ–297.

[90] J. Wolf, Z. Nabi, V. Nagarajan, R. Saccone, R. Wagle, K. Hildrum, E. Pring, and K. Sarpatwar, “The x-flex cross-platform scheduler: Who’s the fairest of them all?” in *Proceedings of the Middleware Industry Track*, ser. Industry papers. New York, NY, USA: ACM, 2014, pp. 1:1–1:7. [Online]. Available: http://doi.acm.org/10.1145/2676727.2676728

[91] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S, “Yousif: Black-box and gray-box strategies for virtual machine migration,” in *NSDI*, 2007.

[92] Xen, http://www.xen.org/.

[93] Xenoprof, http://xenoprof.sourceforge.net.

[94] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu, “Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach,” in *ICDCS*, 2011.

[95] C.-Z. Xu, J. Rao, and X. Bu, “Url: A unified reinforcement learning approach for autonomic cloud management,” in *Journal of Parallel and Distributed Computing (JPDC)*, 2012.

[96] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling,” in *EuroSys*, 2010, pp. 265–278.

[97] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling,” in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys ’10. New York, NY, USA: ACM, 2010, pp. 265–278. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755940

[98] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228301

[99] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, “Improving mapreduce performance in heterogeneous environments,” in *OSDI*, 2008.

[100] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin., “What does control theory bring to systems research,” in *SIGOPS Operating Systems Review*, 2009.

ABSTRACT

AUTOMATIC RESOURCE MANAGEMENT AND PERFORMANCE OPTIMIZATION IN CLUSTERS

by

YUDI WEI

August 2019

Advisor: Dr. Cheng-Zhong Xu

Major: Computer Engineering

Degree: Doctor of Philosophy

Virtual machine is a primary way to increase resource utilizations in data centers by encapsulating multi-resource demands for applications and providing performance isolation. Moreover, the resource configuration can change on the fly to satisfy performance target. Container is another popular way for fine-grained multi-resource allocation. In this dissertation work, we aim to design and implement an automatic resource management system to improve application performance, optimize system efficiency and job completion times in virtual and physical clusters respectively.

For large-scale applications hosted in data center, automatic resource configuration is crucial to service availability and quality. The workload dynamics, cloud dynamics and nonlinear relationship between resource allocation and response time requires an automatic and effective resource allocation strategy. To improve the quality of application services experienced by clients, we propose a novel adaptive fuzzy control approach to tune CPU resource allocation for applications in cloud. It is able to adapt resource allocations not only to the workload variations, but also provide response time guarantee for applications. We have conducted extensive experiments to evaluate the performance, and compared it against other three control methods in terms of stability, overshoot, settling time. Our experiments indicate that Adaptive fuzzy control has the best stability, shortest settling time and minimum overshoot.

Multi-resource configuration provides more space for optimizing resource combinations. The optimization of resource efficiency and utilization has great significance to IaaS providers. To this end, we propose a framework, BConf, for dynamic balanced configuration of multi-resources for the provisioning of response time guarantee in virtualized clusters. BConf employs an integrated MPC (model predictive control) and adaptive PI (proportional integral) control approach (IMAP). MPC is applied to actively balance multiple resources using a novel resource metric. For the performance prediction, a gray-box model is built on generic OS and hardware metrics in addition to resource actuators and performance. We find out that resource penalty is an effective metric to measure the imbalanced degree of a configuration. Using this metric and the model, BConf tunes resources in a balanced way by minimizing the resource penalty while satisfying the response time target. Adaptive PI is used to coordinate with MPC by narrowing the optimization space to a promising region. Within BConf framework, resources are coordinated during contention. Experimental results with mixed TPC-W and TPC-C benchmarks show that BConf reduces resource usages by about 50% and 30% for TPC-W and TPC-C respectively, improves stability by more than 35.6%, and has a much shorter settling time, in comparison with a representative partition approach. The advantages of BConf in resource coordination are also demonstrated.

Modern cluster systems, such as Mesos [46] and Yarn [82], provides fine-grain resource allocation mechanism to satisfy heterogenous demands of various tasks. Moreover, it supports diverse programming models, such as Map-Reduce and MPI. Jobs are transformed into a set of tasks. In shared clusters, scheduling has three different objectives−system efficiency, fairness, and minimal job completion times (JCT). In each pair of two objectives, one may conflict with the other under state-of-the-art policies or strategies. To address the conflicts among the three, we propose a simple theorem-proof policy, called credit sharing (CS) policy, in online scheduling that guarantees the JCTs resulting from DRF. The policy allows short jobs to borrow resource credit from long jobs as long as the long jobs later have enough resource demand to consume the credit that short jobs pay down or off in the process of execution or upon accelerated completions. In addition to accelerating short jobs, the power of packing is unleashed for maximizing system efficiency by packing tasks of short jobs. In multi-resource environment, preservation of data locality has evolved into optimization of effective resource utilizations. Moreover, stage barrier of multi-stage jobs, such MapReduce, may cause imbalanced utilization of network bandwidth in clusters. According to CS policy, we present CANAL (Credit sharing-oriented Network And Locality-aware scheduling) in multi-resource clusters. We conduct a detailed evaluation in both trace-driven simulation and prototype implementation. Compared with the most competitive scheduler, we find that CANAL significantly speed up jobs by up to 2.36 time, particularly for network-intensive jobs, and reduce makespan by 25%. Moreover, each job under CANAL does not decelerate compared with DRF.

AUTOBIOGRAPHICAL STATEMENT

Yudi Wei is a Ph.D. candidate of Department of Electrical and Computer Engineering at Wayne State University. She received the M.E. degree in Computer Science from Beihang University, Beijing, China, in 2008.

She is a member of the Cloud and Internet Computing group. Her research interests include cloud computing, distributed systems and cluster scheduling. She has published several papers in conferences in these areas. She has also served as a peer reviewer for a few conferences and journals. She is also a receiver of Travel Award for Excellence in Graduate Student Research.

Besides the academic research, she served as a lab instructor for the undergraduate courses Introducing Microcomputers and Digital Logic for six consecutive semesters and four semesters respectively. She also worked as a teaching assistant for grading course Computer Networking and the Internet in 2010 and 2012, and tutoring course Scalable Internet Services and Architecture in 2011. She has been receiving excellent evaluations each semester from her students.