Koala: Improved Scheduler Running on Eucalyptus

(Names withheld for anonymous submission)

Abstract

As cloud computing continues to gain momentum, the infrastructure supporting clouds continues to grow. Many modern solutions make heavy use of virtualization, not only to provide isolation between tasks, but also for dividing hardware resources, and balancing machine utilization. Eucalyptus is an infrastructure that has contributed a great deal of work to this area. It is one of the most common open source cloud computing systems, but presently only has rudimentary scheduling policies that are only used when an instance is started. We present *Koala*, an enhanced Eucalyptus system which uses live migration and a dynamic learning scheduler to greatly improve resource utilization within a cluster. Utilizing the benefits provided from live migration, Koala's dynamic scheduler can make effective and reactive scheduling decisions that were previously not possible. Additionally, Koala's use of learning algorithms allow it to get the most out of the scheduler and make it robust to changing needs and demands on the cluster.

1 Related Work

The foundation of our work is the set of schedulers contained in Eucalyptus [11]. Eucalyptus is a cloud computing infrastructure, which uses virtualization technologies such as Xen [4], VMware [13] and KVM/QEMU [6], and provides an interface to EC2/S3 [1]. Eucalyptus ships with two main scheduling algorithms: a greedy algorithm and a round-robin algorithm. The greedy algorithm will use the first available node that can run a given job, and round-robin will find the *next* suitable node. Eucalyptus also implements a power-saving scheduling algorithm based on the greedy scheduler that attempts to shut down idle nodes in order to conserve power. Note that these schedulers are only used to determine where to put a start a job, because Euclayptus does not support live migration. Koala extends Eucalyptus to support live migration, enabling much more aggressive and reactive scheduling algorithms. Additionally Koala uses dynamic information about the system as well as a significantly more sophisticated parameterized scheduling algorithm resulting in a more desirable use of resources.

Researchers from Microsoft built Quincy [8], a cloud scheduling framework for Dryad and MapReduce-like systems. They reduced the scheduling to a min-cost problem in order to expose a variety of adjustable options. They found these options were effective in tuning the scheduler to perform various workloads. In our project, we will build a similarly parameterized scheduler, but will target Eucalyptus, which has fundamentally different scheduling requirements. Additionally our work focuses on building a tool that determines the optimal parameters, not just the scheduling framework itself.

Zaharia et al. developed their LATE scheduler for MapReduce in order to improve task speculation on heterogenous systems [14]. This contrasts our work because we target not just the service level (MapReduce) but the infrastructure itself (at the VM level), where speculation is not traditionally used. Similarly Mantri et al. [3] focus on improving performance by better handling

of stragglers in a MapReduce system. Mantri et al. focus on exploring the cause of an outlier through monitoring and other methods to take appropriate action and free those resources. Early detection of outliers is key to their methodology and is effective for MapReduce workloads.

Load-balancing is a critical issue related to the optimized scheduling. Zhao, et. al [16] developed an adaptive load-balancing system algorithm for Eucalyptus called "COMPARE_AND_BALANCE". Their system focuses on load-balancing as the primary goal, while Koala provides a more general scheduler to fit the needs and success metrics unique to each cluster. Additionally they focused on a distributed algorithm forbidding the use of a centralized scheduling node, which is not a limitation we impose in our system.

Begnum et al. [5] have experimented with moving virtual machine policy inside the virtual machine itself. The benefits of this include improved portability and aptitude for live-migration. Additionally, this increases the ease with which virtual machines can move between non-homogeneous systems in a cloud. They worked with MLN (Manage Large Networks), an open source tool providing atomic operations to enable the management of large quantities of a variety of virtual machines [4, 2, 13]. It provides an interface in the form of a configuration language.

A number of researchers have applied genetic algorithms (GA) to the task of dynamically scheduling on the cloud [10, 17, 15]. In [10] they improved upon Eucalyptus's scheduling algorithms with a focus on topology sensitivity using GA. They claim in their future work they would be interested in machine learning based systems, similar to our system. Additionally they focused on the data-intensive workload produced from MapReduce tasks. Zhong, et al. [17] built their Improved Genetic Algorithm (IGA) to improve the scheduling algorithms commonly used in open-source clouds.

Some scheduling algorithms focus on particular workloads. While some researchers only consider MapReduce-like loads, others focus on scientific workloads [9, 12, 7]. By gathering basic characteristics of the workload, Simmhan et al. [12] build a *blackbox* system to help schedule the workflow. This is similar to our work, although we hope to determine the characteristics of the workflow experimentally, and not require them as inputs from the user, and instead only have the user indicate their desired success metrics.

2 Background

Koala runs on Eucalyptus, a system which provides an Infrastructure as a Service (IaaS) cloud interface. This exposes virtualized hardware resources to the user, allowing them to build and manage the system as they see fit. Eucalyptus is made of a hierarchy of controllers, including the Node Controllers (NCs), Cluster Controllers (CCs), Cloud Level Controller (CLC), Storage Controllers (SCs), and the Walrus Storage Controller. They communicate through web front ends, generated by Web Services Description Language (WSDL). An overview of these components and how they interacted is shown in Figure 1.

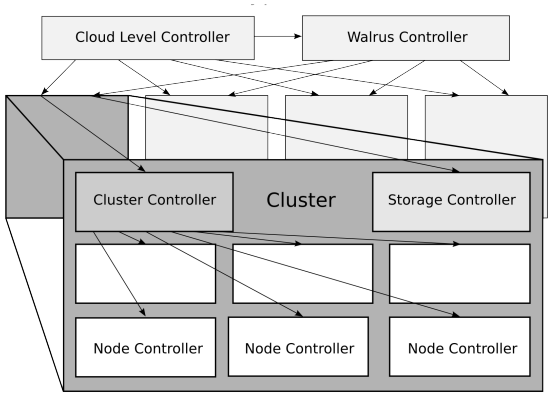

Figure 1: Eucalyptus System Overview

2.1 Node Controller

Node Controllers are at the bottom of the Eucalyptus hierarchy. They are responsible for communicating with the hypervisor running on the node through the libvirt virtualization library. Prior to our modifications and ignoring inter-VM communication, the Node Controllers communicate exclusively with their respective Cluster Controller. They maintain the state of the VM and the host system's resources and communicate them back to the Cluster Controller.

2.2 Cluster Controller

Cluster Controllers are responsible for maintaining the state of a number of nodes. The send heartbeats and query information about the instances and resources on each node and passes some of this information up to the CLC. The CC is responsible for scheduling instances to the nodes it maintains. It caches information on the nodes and instances in order to not have to poll all the nodes as it scales to larger numbers.

2.3 Cloud Level Controller

The Cloud Level Controller is responsible for managing the various controllers in the Cloud and providing a web front-end to administer the Cloud. The CLC talks with the CCs, SCs, and Walrus Controller and communicates and interfaces between them. It also manages network resources and is responsible for many of the security functionality of Eucalyptus.

2.4 Storage Controller

A Storage Controller would normally be group with each cluster in a cloud. The SC exposes a block level storage, which the VMs can mount and store persistent data. The SC is designed to implement the Amazon Elastic Block Storage (EBS) specification.

2.5 Walrus Controller

Walrus is a Eucalyptus's put-get bucket storage system. It is designed to be compatible with Amazon's S3 storage system. Eucalyptus uses Walrus for storage of user data and access control for VMs.

3 Design

Koala builds upon Eucalyptus in three main ways. First, Koala extends Eucalyptus by adding live migration, which serves as a fundamental building block for the other components. Second, Koala adds a parameterizable scheduler to the cluster controller that leverages live migration to move tasks around to achieve a more desirable configuration. Third, Koala monitors the performance of the system and uses learning algorithms to tune the parameters to the scheduler, using this feedback to improve Koala's scheduling decisions.

Figure 2 shows these components and where they fit into the overall system architecture, and we describe each of them below.

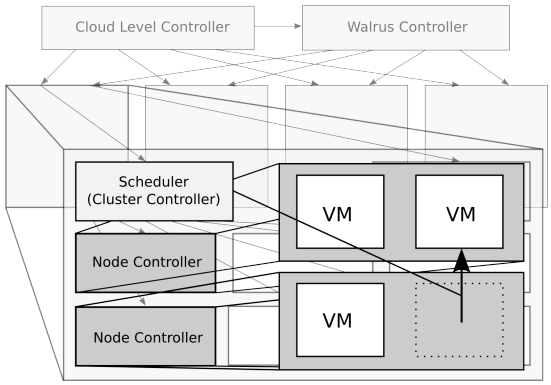

Figure 2: Koala System Overview

3.1 Live Migration

Live migration is the process of transparently moving a running VM between two machines.

In Koala, in order to efficiently manage the resources of a cloud, we need to be able to move VMs around to put them in a desired configuration. This must be done without disrupting the internal state of the VM, any active network connections, and that while moving VMs we have minimal downtime. Koala uses live migration to achieve this. This live migration primitive is a key part of the Koala architecture, enabling the other components to dynamically manage the resources of the cluster.

3.2 Parameterized Scheduler

Koala has a parameterized scheduler. The primary goal of the scheduler is to manage the assignment of VMs to nodes in an attempt to achieve a more desirable system. The scheduler has two components, a static scheduler and the dynamic scheduler. The static scheduler answers the question "on which node should this instance start?", while the dynamic scheduler answers the more complicated question "Is there some migration that would result in a more optimal configuration?". The dynamic scheduler needs to take into account many things, such as the cost of migration and what it means to be an optimal configuration. This is a challenge because distributed systems are hard to predict, especially when building a general scheduler. Koala handles this in its scheduler by exposing the various weights and configuration knobs into scheduling parameters that can be changed at runtime to make decisions. In exposing these parameters, we attempt to abstract as many assumptions as possible into configurable values that enable the scheduler to be as general it can be.

3.3 Dynamic Learning

Given our parameterized scheduler, we have the task of setting the configuration values to something reasonable for a particular system and set of workloads, all of which might change dynamically. We believe asking the user to set these values imposes an unreasonable burden, due to both the complexity of such a task and the ever-changing needs of an Eucalyptus system. Koala simplifies this task by leveraging learning algorithms. It uses monitoring information to attempt to find ideal values for the configuration parameters, with respect to some optimality metric. The problem of how to determine the optimality metric is in itself, a complicated issue. This is due to the fact that there are many different definitions of optimal (attempts to reduce power consumption might be at odds with attempts to increase system performance), and so we generalize these to a smaller set of knobs which are exposed to the end-user. This greatly simplifies the burden on the user, and because this too is dynamically configurable, can allow for greater control of the system as requirements and priorities change.

3.4 Summary

Koala has three components, all of which build upon each other. The addition of live migration allows dynamic resource management, but is far too complicated for Eucalyptus's primitive scheduler to reason about. The parameterized scheduler attempts to uses live migration to achieve a more desirable system configuration, but exposes a new challenge in the form of configurable parameters. Learning algorithms, when provided sufficient data and optimality metrics, should provide near optimal scheduling parameters. We believe this is a powerful combination that not only greatly simplifies and automates the resource management of the cluster, but also enables significantly improved resource utilization than would otherwise be possible.

4 Implementation

4.1 Live Migration

In the implementation of our project, we hit an unexpected roadblock. Eucalyptus did not support live migration, a method for moving run-

ning VMs between nodes. Unfortunately we had intended to utilize live migration heavily within our scheduler, so we set forth to implement it. We modified Eucalyptus to support live migration. This involved leveraging libvirt, the virtualization toolkit used by Eucalyptus, in order to perform the migration. Additionally, extensive modification of Eucalyptus was required to communicate with libvirt and to manage its state during migration.

Libvirt, the Eucalyptus node controller, and the Eucalyptus cluster controllers all have very different views of the state of VMs. Libvirt supports specific states of the VMs in respect to their status on an individual machine. This means that migration is transparent, and during migration various states (running, paused, etc) are visible on both machines taking part in migration. The default node controller and cluster controller are (by design) unaware of migrating VMs, and quickly report failures when VMs are migrated unexpectedly (this can be done easily by issuing libvirt commands outside of eucalyptus) as their state becomes confused. The node controllers had to be modified to account for the respective states (both in respect to libvirt and migration), and to be robust to a wide variety of cases such as internal shutdown or VM/node failure. The cluster controller has to manage its respective state for both nodes in addition to transmitting the relevant Node Controller state for the VMs and the controller itself. Additionally the cluster controller has to reason about this migration state when reporting to the cloud the status of a particular instance.

The very nature of migration involves slightly breaking the Eucalyptus model of communication between controllers. Initially, nodes only communicated with their respective cluster controller and (ignoring what occurs in the instances themselves) are completely oblivious to each others existence. The process of migration fundamentally involves sending the entirety of the state of a VM (in the form of a disk image, kernel, ramdisk, and memory pages) to another node. Short of redirecting this communication through the cluster, which would generate significant bandwidth, or using special routing hardware to mask the transmission, migration would not be possible without exposing the existence and location of other nodes.

Both nodes taking part in migration need to have access to the VM image. We set up a network file-system over NFS, where we store the VM images. Accessing a VM image of the network is a very bandwidth expensive operation, so we utilized the Eucalyptus image caching functionality, in order to keep bandwidth costs of both operation and migration to a minimum.

4.2 Scheduling

Currently, our scheduler is quite elementary and only exposes two thresholds, one to adjust the scheduling frequency, and one in which to optimize between throughput and energy efficiency. Our model of throughput is currently built on the assumption that instances assigned to individual machines will perform better than the same number of instances on fewer machines, and our energy model is based on the assumption that instances assigned to individual machines on consume more energy than the same number of instances on fewer machines. We will implement a much greater number of thresholds to implement. One of these thresholds will adjust the VM densities and the hardware resources allocated to the specific VMs. Another threshold would prioritize data locality over general computational throughput. These knobs will allow us to modify the ways in which VMs are allocated, in terms of both allocations and migrations, in order to better meet the needs of the cluster.

4.3 Data Mining + Learning Algorithms

A lot of work has been done in the areas of data mining and machine learning. We intend to leverage this work by exposing various schedul-

ing and machine events. We will use established machine learning techniques to generate inputs for the thresholds in our scheduler in order to adapt to the properties of our workload and cluster. A more optimized scheduler can better allocate the VMs to the resources of the cluster, and should achieve higher throughput.

4.4 Summary

Although both live migration and more advanced scheduling improve resource consumption on the cluster, these techniques are at a cost. Live migration causes the nodes involved to incur the cost of managing the migration, as well as imposes a burden on the network while transferring across the related VM state. The scheduling itself, particularly with the introduction of learning algorithms adds overhead to the cluster controller. Finally, migration itself might incur some degraded performance in the guest machine.

The actual computation cost of migration is very small in comparison to an instance running for an extended amount of time. Detail of this can be found in our evaluation. In addition to the performance gains, we gain the ability to switch the system to be optimized for energy efficiency on the fly. The larger costs associated with Koala would be the analysis of the event data and the generation of decision trees and other comparable structures. As our implementation permits, we plan to evaluate the trade-off between these computations and the benefits gained from more optimal scheduling. It's important to note that many of these costs (what does it 'cost' to migrate a machine with respect to some optimality metric?) will be handled by Koala's learning dynamic scheduler, which we believe will help mitigate many of these costs.

5 Evaluation

Our current setup for Koala uses three computers. The Eucalyptus Cloud Level Controller, Cluster Controller, Walrus Controller, and Storage Controller all are running on a CentOS 5 machine, with a 3GHz dual core processor and 8GB of ram. We have two laptops running the Node Controllers on a Debian Squeeze machine, one with a 2.4GHz dual core processor and 4GB of ram, and the other with a 1.8GHz dual core processor and 2GB of ram. The node controllers both run instances in QEMU with KVM support.

In addition to changing the scheduling frequency, our current scheduler can be configured to either optimized for throughput or energy efficiency. We conducted the following two experiments by varying the threshold between throughput and energy efficiency and using a scheduling frequency of 30 seconds.

In our first test, we configure our scheduler to maximize throughput. We start three VMs, of which two are placed on one node and one on the other. We then terminate the VM (asynchronously through the web interface) on the node with only one, so one node has two VMs and the other node has none. At the next iteration of or scheduler, one of the two VMs on other node is migrated to the empty node. When configured for throughput, the scheduler attempts to run the fewest VMs per core per machine. The scheduler operates attempting to balance the ratio of cores utilized across all machines, and is robust both machines with varying numbers of cores and VMs configured to use varying numbers of cores.

In our second test, we configure our scheduler to optimize energy consumption. We again start with three VMs, of which two are placed on one node and one on the other. Now, we terminate one of the VMs (again asynchronously through the web interface) on the node with two, so each of the two nodes is left with a single instance. At the next iteration of the scheduler, one of the two nodes migrates its VM to the other node, leaving it without any VMs, and it could then be suspended or powered off. We again expect this to be robust to larger and homogeneous systems where the VMs and nodes both support varying

numbers of cores.

Finally, to test the robustness of the scheduling components and particularly the live migration feature we added to Eucalyptus, we ran an experiment with a special scheduler designed to randomly move VM's around amongst the available resources. We ran this experiment for over an hour and a half, resulting in over 180 migrations. At the end of the experiment, all of the migrations were successful. The guest VM's were still running, and the node controllers and cluster controllers all had clean state, with no failures in any of the migration attempts. There was no failure at an hour and a half, but we believed that long enough to demonstrate the robustness and completeness of our live migration support implementation.

Future evaluation will be performed to verify these beliefs, as we get access to additional machines. As the scheduler and learning algorithms are more fully implemented, we will begin to compare our implementation to a default Eucalyptus system.

References

[1] Amazon web services (ec2/s3). `http://aws.amazon.com/`.

[2] User-mode linux. `http://user-mode-linux.sourceforge.net/`.

[3] 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*, OSDI'10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[5] K. Begnum, N. A. Lartey, and L. Xing. Cloud-oriented virtual machine management with mln. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 266–277, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] F. Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[7] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the use of cloud computing for scientific workflows. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 640–645, Washington, DC, USA, 2008. IEEE Computer Society.

[8] 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*, SOSP '09, pages 261–276, New York, NY, USA, 2009. ACM.

[9] G. Juve and E. Deelman. Scientific workflows and clouds. *Crossroads*, 16:14–18, March 2010.

[10] G. Lee, N. Tolia, P. Ranganathan, and R. H. Katz. Topology-aware resource allocation for data-intensive workloads. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems*, APSys '10, pages 1–6, New York, NY, USA, 2010. ACM.

[11] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[12] Y. Simmhan and L. Ramakrishnan. Comparison of resource platform selection approaches for scientific workflows. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 445–450, New York, NY, USA, 2010. ACM.

[13] B. Walters. Vmware virtual platform. *Linux J.*, 1999, July 1999.

[14] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

[15] C. Zhao, S. Zhang, Q. Liu, J. Xie, and J. Hu. Independent tasks scheduling based on genetic algorithm in cloud computing. In *Proceedings of the 5th International Conference on Wireless communications, networking and mobile computing*, WiCOM'09, pages 5548–5551, Piscataway, NJ, USA, 2009. IEEE Press.

[16] Y. Zhao and W. Huang. Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud. In *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, NCM '09, pages 170–175, Washington, DC, USA, 2009. IEEE Computer Society.

[17] H. Zhong, K. Tao, and X. Zhang. An approach to optimized resource scheduling algorithm for open-source cloud systems. In *Proceedings of the The Fifth Annual ChinaGrid Conference*, CHINAGRID '10, pages 124–129, Washington, DC, USA, 2010. IEEE Computer Society.

6 Business Plan

Modern computing is moving towards larger and more complex tasks. These tasks require more storage and computational power than ever before. Cloud Computing is a developing field, built around providing solutions to this problem. Cloud Computing is closely tied with distributed and grid computing, and shares technology and infrastructure with them. Many advancements have been made in these areas, and Cloud Computing is currently a very popular venture for computing start-ups.

Eucalyptus is an infrastructure provided by Eucalyptus Systems Inc, which allows for deployment across a large variety of systems and provides a consistent interface for access to the underlying network. The current version of Eucalyptus has a primitive scheduler, only using rudimentary placement algorithms and only invoked to determine which node to start a VM on. In order to fully utilize a large cluster of machines, properly delegating tasks and balancing resource utilization is key.

We propose Koala, an enhanced Eucalyptus system, which uses a learning scheduler to maximize throughput and utilization of the machines in a cluster. Koala's algorithm gathers monitoring information and uses this data to adjust how the scheduler behaves. Our system allows for increased utilization of machines in a cluster, reducing the hardware, energy, and management expenses associated with the operation of a large cluster of machines.

The current version of Eucalyptus is currently only capable of static scheduling. Due to VM churn, these allocations can drift away from optimal configurations. For example, if a 100 instances were running on 100 machines at 25% utilization and energy efficiency was important, the Koala scheduler could migrate the instances to just 25% of the machines, and the remaining 75% of the machines could be powered off in order to conserve energy. In the case where the optimal throughput was desired, it could perform the same operation in reverse. Neither of these operations are possible in the existing Eucalyptus system, and are one of the key benefits of using Koala.

In addition to the automated improvements to efficiency and throughput, the Koala scheduler can be influenced externally to account for additional factors. It can be instructed to minimize network bandwidth (for other tasks) or reduce power consumption to reduce operation costs. For example, if resources are not scarce, or a workload is not time critical, the scheduler can be instructed to prioritize power consumption, and not use a machine if it cannot sufficiently utilize that machine's resources.

However, the needs, priorities, and workloads of a cloud can often change during operation. A company might want to conserve power (and money) by default, but change to maximize performance at some point when a deadline comes up, perhaps. The ability to reconfigure this dynamically allows the cluster to be optimized to the customer's particular needs and use cases.

Companies often invest large amounts of capital into building and maintaining their cloud infrastructure, and Koala allows those companies to significantly increase the efficiency of their cluster. This can result in lots of savings for those users in reduced costs of hardware, energy, and management.