Model-driven Configuration of Cloud Computing Auto-scaling Infrastructure

Brian Dougherty[1] and Jules White[2] and Douglas C. Schmidt[1]

[1] Vanderbilt University briand,schmidt@dre.vanderbilt.edu
[2] Virginia Tech julesw@vt.edu

Abstract. Cloud computing uses virtualized computational resources to allow an application's computational resources to be provisioned on-demand. Auto-scaling is an important cloud computing technique that dynamically allocates computational resources to applications to precisely match their current loads. This paper presents a model-driven engineering approach to optimizing the configuration and cost of cloud auto-scaling infrastructure. The paper provides the following contributions to the study of model-driven configuration of cloud auto-scaling infrastructure: (1) it shows how virtual machine configurations can be captured in feature models, (2) it describes how these models can be transformed into constraint satisfaction problems (CSPs) for configuration and cost optimization, (3) it shows how optimal auto-scaling configurations can be derived from these CSPs with a constraint solver, and (4) it presents a case-study showing the cost reduction produced by this model-driven approach.

1 Introduction

Current trends and challenges. Cloud computing is a computing paradigm that uses virtualized server infrastructure to dynamically provision virtual OS instances [8]. By allocating virtual machines to applications on demand, cloud infrastructure users can pay for servers incrementally rather than investing the large up-front costs to purchase new servers. Moreover, rather than over-provisioning an application's infrastructure to meet peak load demands, an application can *auto-scale* by dynamically acquiring and releasing virtual machine instances as load fluctuates.

A key concern when auto-scaling an application is ensuring that virtual machines can be provisioned and booted quickly enough to meet response time requirements as the load changes. If auto-scaling is too slow to keep up with load fluctuations, applications may experience a period of poor response time while waiting for additional computational resources to come online. One way to mitigate this risk is to maintain an auto-scaling queue containing prebooted and preconfigured virtual machine instances that can be allocated rapidly, as shown in Figure 1.

When a cloud application requests a new virtual machine configuration from the auto-scaling infrastructure, the auto-scaling infrastructure first attempts to fulfill the request with a prebooted virtual machine in the queue. For example, if a virtual machine with Fedora Core 6, JBoss, and MySQL is requested, the auto-scaling infrastructure will attempt to find a matching virtual machine in the queue. If no match is found, a new virtual machine must be booted and configured to match the allocation request.

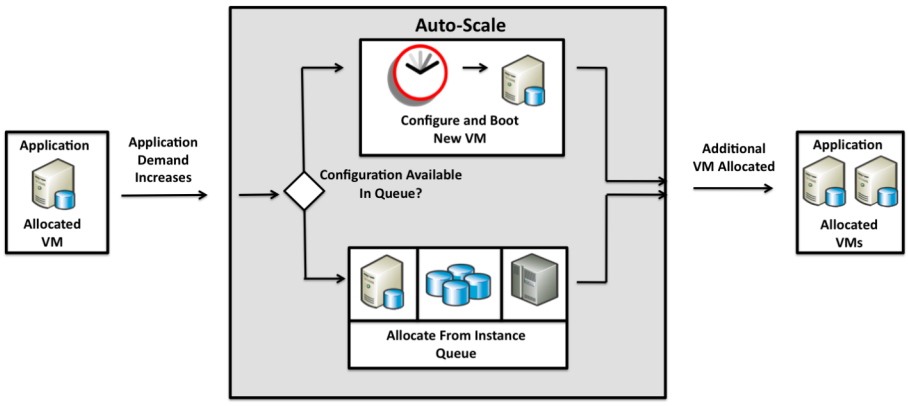

Fig. 1: Auto-scaling in a Cloud Infrastructure

Open problems. A key challenge for developers is determining the size and properties of an auto-scaling queue shared by multiple applications that each may have different virtual machine configurations [3]. For example, a web application may request virtual machine instances configured as database, middle-tier Enterprise Java Beans (EJB), or front-end web servers. Determining how to capture and reason about the configurations that comprise the auto-scaling queue is hard due to the large number of configuration options (such as MySQL and SQL Server databases, Ubuntu Linux and Windows operating systems, and Apache HTTP and IIS/Asp.Net web hosts) offered by cloud infrastructure providers.

It is even harder to determine the optimal queue size and types of virtual machine configurations that will ensure that virtual machine allocation requests can be serviced quickly enough to meet a required auto-scaling response time limit. A variety of strategies can be employed, such as filling the queue with virtual machine configurations that are most common across the applications or that take the most time to provision/boot. Cost optimization is challenging because each configuration placed into the queue can have varying costs based on the hardware resources and software licenses it uses.

Solution approach → Auto-scaling queue configuration derivation based on feature models. This paper presents a Model-Driven Engineering (MDE) approach called the *Smart Cloud Optimization for Resource Configuration Handling* (SCORCH) that captures virtual machine configuration options for a set of cloud applications and derives an optimal set of virtual machine configurations for an auto-scaling queue. SCORCH provides three contributions to the study of MDE cost optimization of cloud auto-scaling. First, we describe an MDE technique for transforming feature model representations of cloud virtual machine configuration options into constraint satisfaction problems (CSPs) [6, 5]. Second, we describe another MDE technique for analyzing application configuration requirements and virtual machine operating costs to determine what virtual machine instance configurations to include in an auto-scaling queue in order to meet an auto-scaling response time guarantee. Third, we present empirical re-

sults from a case study using Amazon's EC2 cloud computing infrastructure that shows our MDE techniques minimize operational cost while ensuring that an auto-scaling response time requirement is met.

Paper organization. The remainder of the paper is organized as follows: Section 2 describes the challenges of modeling, analyzing, and deriving an optimal auto-scaling queue configuration; Section 3 presents SCORCH's MDE approach to derive optimal auto-scaling queue configurations; Section 4 presents empirical results from applying SCORCH to derive an auto-scaling queue for ecommerce applications that show the cost effectiveness of the technique; Section 5 compares SCORCH with related work; and Section 6 presents concluding remarks.

2 Challenges of Configuring Virtual Machines in Cloud Environments

This section describes three key challenges of capturing virtual machine configuration options and using this configuration information to optimize the setup of an auto-scaling queue. Section 3 then presents SCORCH's MDE approach to resolving these challenges.

2.1 Challenge 1: Capturing Virtual Machine Configuration Options and Constraints

A cloud application can request virtual machines with a wide range of configuration options, such as type of processor, amount of memory, OS, and installed middleware. For example, the Amazon EC2 cloud infrastructure (`aws.amazon.com/ec2`) supports 5 different types of processors, 6 different memory configuration options, and over 9 different OS types, as well as multiple versions of each OS type [4]. These EC2 configuration options cannot be selected arbitrarily and must adhere to a multitude of configuration rules. For example, a virtual machine running on Fedora Core 6 OS cannot run MS SQL Server. Tracking these numerous configuration options and constraints is hard. Sections 3.1&3.2 describe how SCORCH uses feature models to alleviate the complexity of capturing and reasoning about configuration rules for virtual machine instances.

2.2 Challenge 2: Selecting Virtual Machine Configurations to Guarantee Auto-scaling Speed Requirements

A key determinant of auto-scaling performance is the types of virtual machine configurations that are kept ready to run. If an application requests a virtual machine configuration and an exact match is available in the auto-scaling queue, the request can be fulfilled nearly instantaneously. If the queue does not have an exact match, it may have a running virtual machine configuration that can be modified to meet the requested configuration faster than provisioning and booting a virtual machine from scratch. For example, a configuration may reside in the queue that has the correct OS but needs to unzip a custom software package, such as a pre-configured Java Tomcat Web Application Server, from a shared filesystem onto the virtual machine. Auto-scaling requests

can thus be fulfilled with both exact configuration matches and subset configurations that can be modified faster than provisioning a virtual machine from scratch.

Determining what types of configurations to keep in the auto-scaling queue to ensure that virtual machine allocation requests are serviced fast enough to meet a hard allocation time constraint is hard. For one set of applications, the best strategy may be to fill the queue with a common generic configuration that can be adapted quickly to satisfy requests from each application. For another set of applications, it may be faster to fill the queue with the virtual machine configurations that take the longest to provision from scratch. Numerous strategies and combinations of strategies are possible, making it hard to select configurations to fill the queue that will meet auto-scaling response time requirements. Section 3.3 show how SCORCH captures cloud configuration options and requirements as cloud configuration feature models, transforms these models into a CSP, and creates constraints to ensure that a maximum response time limit on auto-scaling is met.

2.3 Challenge 3: Optimizing Queue Size and Configurations to Minimize Cost

A further challenge for developers is determining how to configure the auto-scaling queue to minimize the costs required to maintain it. The larger the queue, the more it costs. Moreover, each individual configuration within the queue can vary in cost. For example, a “small” Amazon EC2 virtual machine instance running a Linux-based OS costs $0.085 per hour while a "Quadruple Extra Large" virtual machine instance with Windows costs $2.88 per hour.

It is hard for developers to manually navigate the tradeoffs between costs and auto-scaling response time of different queue sizes and sets of virtual machine configurations. Moreover, there are an exponential number of possible queue sizes and configuration options that complicates deriving the minimal cost queue configuration that will meet auto-scaling speed requirements. Section 3.3 describes how SCORCH uses CSP objective functions and constraints to derive a minimum cost queue configuration that meets virtual machine configuration constraints generated by transforming feature model representations of application configuration rules and options into a CSP.

3 The Structure and Functionality of SCORCH

This section describes the MDE techniques that SCORCH uses to address the challenges of optimizing an auto-scaling queue described in Section 2. SCORCH resolves these challenges by using models to capture virtual machine configuration options explicitly, model transformations to convert these models into constraint satisfaction problems (CSPs), and constraint solvers to derive the optimal queue size and contained virtual machine configuration options to minimize cost while meeting auto-scaling response time requirements.

The SCORCH MDE process is shown in Figure 2 and described below:

1. Developers use a SCORCH *cloud configuration model* to construct a catalog of configuration options that are available to virtual machine instances.

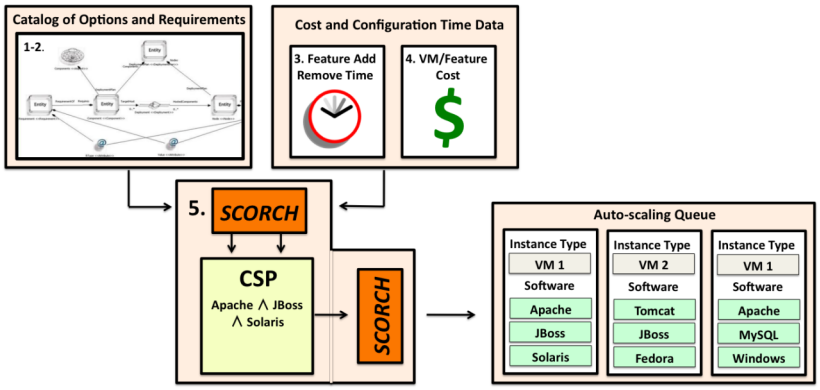

Fig. 2: SCORCH Model-Driven Process

2. Each application considered in the auto-scaling queue configuration optimization provides a *configuration demand model* that specifies the configuration for each type of virtual machine instance the application will request during its execution lifecycle.
3. Developers provide a *configuration adaptation time model* that specifies the time required to add/remove a feature from a configuration.
4. Developers provide a *cost model* that specifies the cost to run a virtual machine configuration with each feature present in the SCORCH cloud configuration model.
5. The cloud configuration model, configuration demand models, and load estimation model are transformed into a CSP and a constraint solver is used to derive the optimal auto-scaling queue setup.

The remainder of this section describes the structure and functionality of each of the models defined and used by SCORCH.

3.1 SCORCH Cloud Configuration Models

A key consideration in SCORCH is modeling the catalog of virtual machine configuration options. Amazon EC2 offers many different options, such as Linux vs. Windows operating systems, SQL Server vs. MySQL databases, and Apache HTTP vs. IIS/Asp.Net webhosts. This model provides developers with a blueprint for constructing a request for a virtual machine instance configuration and checking its correctness. The queue configuration optimization process also uses this model to ensure that valid configurations are chosen to fill the queue.

To manage the complexity of representing virtual machine instance configuration options, SCORCH uses a modeling paradigm from software product-lines called *feature models* [5]. Feature models describe commonality and variability in a configurable software platform. For example, a feature model can capture the different OS types and versions available for a virtual machine instance, as shown in Figure 3.

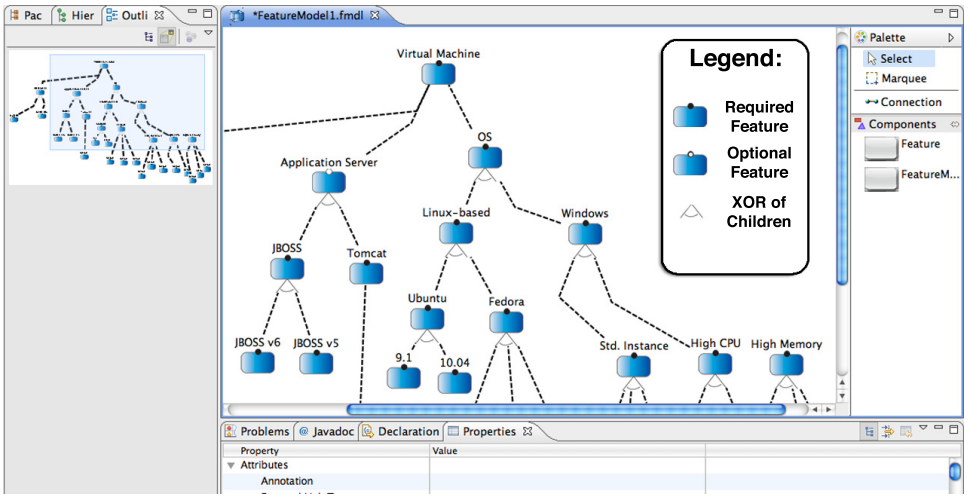

Fig. 3: Example SCORCH Cloud Configuration Feature Model

Feature models capture the points of commonality and variability using an abstract called a *feature*. Each feature in the feature model represents an increment of functionality or a point of variability. Features can describe both high-level functional variations in the software, *e.g.*, whether or not the underlying software can load balance HTTP requests. A feature can also represent implementation-specific details, *e.g.*, whether or not Ubuntu 9.10 or Fedora is used.

As shown in Figure 3, feature models use a tree-like structure to define the relationships between the various features and encode configuration rules into the model, *e.g.*, a virtual machine configuration can include only a single operating system, such as Ubuntu 9.10 or Fedora. Some features may require other features to be present in order to function, *e.g.*, the `JBOSS v6` feature cannot be chosen without also selecting the `JBOSS` feature.

A configuration of the software platform is defined by a selection of features from the feature model. The most basic rule of configuration correctness is that every selected feature must also have its parent feature selected. This rule also implies that every correct feature selection must include the root feature. Moreover, the feature selection must adhere to the constraints on the parent-child relationships encoded into the feature model. A listing of the basic feature selection rules is shown in Figure 3.

Developers use the SCORCH cloud configuration model to express the available configuration options for virtual machine instances as a feature model. SCORCH is designed for environments, such as private clouds, where it is feasible for applications to enumerate their features and to use this model for queue configuration optimization. SCORCH is also applicable to clouds where a subset of customers pay a premium to have their virtual machine instance configurations considered in the queue optimization process.

The configuration adaption time model's information is captured as attributes of the features in the SCORCH cloud configuration model. Although attributes are not technically a model, we describe them as such to simplify the CSP discussions. Each feature can be annotated with an integer attribute that specifies the time in milliseconds to add/remove the given feature from a configuration. Features that are not realistically feasible to add/remove from a booted VM configuration are annotated with configuration adaptation times of T=∞. For example, installing a Red Hat Package Manager archive on a virtual machine to add a feature might take a few seconds, whereas changing the OS may not be as feasible due to the overhead incurred.

The cost model is also captured using attributes in the SCORCH cloud configuration model. Again, we describe it as a model to simplify the CSP descriptions. Each feature that impacts the cost of a configuration is annotated with a cost attribute that specifies the cost per hour to have a booted virtual machine configuration in the queue with that feature. For example, these attributes can be used to model the cost of the "Small" vs. "Quadruple Extra Large" computing node size features of an Amazon EC2 virtual machine configuration.

3.2 SCORCH Configuration Demand Models

Applications are auto-scaled at runtime by dynamically requesting and releasing virtual machine instances. When a new virtual machine instance is requested, the desired configuration for the instance is provided. SCORCH requires each application to provide a model of the virtual machine instance configurations that it will request over its lifetime.

Developers construct SCORCH configuration demand models to dictate what virtual machine configurations an application will request. The configuration demand models use a textual domain-specific language to describe each configuration that will be requested as a selection of features from the SCORCH cloud configuration model. For example, a valid configuration demand model that includes one virtual machine configuration for a large JBOSS application server and one configuration for a Jetty server is shown in Figure 4.

```
LargeJBOSSServer {
    Ubuntu 9.0:selected;
    JBOSS v6:selected;
    Processor:selected;
    Large:selected;
}
JettyServer {
    Ubuntu 9.0:selected;
    Jetty v6.1:selected;
}
```

Fig. 4: SCORCH Textual Configuration Demand Model

The basic structure of the model is one or more identifiers for configurations that will be requested. Each identifier is followed by a set of braces that enclose the names of the features selected/deselected in the configuration. Features either may be specifically selected for a configuration by marking them as `selected` or excluded by marking them as `not selected`. If a configuration does not explicitly mark a feature as `selected` or `not selected`, the SCORCH optimization process will determine whether or not to include it. For example, the `JettyServer` does not specify a value for the processor size and will accept any available processor.

3.3 Runtime Model Transformation to CSP and Optimization

A key value of using feature models for capturing virtual machine configuration options is that selecting a group of features to optimize an objective function can be phrased as a CSP. In the context of SCORCH, the cloud configuration model and configuration demand models are converted into a CSP where a solution is a valid set of configurations for the virtual machine instances in the auto-scaling queue. The objective function of the CSP is designed to attempt to derive a mix of configurations that minimizes the cost of maintaining the queue while ensuring that any hard constraints on the time to fulfill auto-scaling requests are met.

The conversion of feature selection problems into CSPs has been described extensively in prior work [1, 12]. A CSP is a set of variables and a set of constraints governing the allowed values of the variables. For example, $X + Y < 10$ is a CSP with two variables. A valid labeling of values for X and Y must adhere to the constraint that their sum is less than 10. Feature configuration problems are converted into CSPs where the selection state of each feature is represented as a variable with domain {0,1}. The constraints are designed so that a valid labeling of these variables yields a valid feature selection from the feature model.

A CSP for a feature selection problem can be described as a 3-tuple:

$$P = < F, C, \gamma >$$

where:

- F is a set of variables describing the selection state of each feature. For each feature, $f_i \in F$, if the feature is selected in the derived configuration, then $f_i = 1$. If the i^{th} feature is not selected, then $f_i = 0$.
- C captures the rules from the feature model as constraints on the variables in F. For example, if the i^{th} feature requires the j^{th} feature, C would include a constraint: $(f_i = 1) \Rightarrow (f_j = 1)$.
- γ is an optional objective function that should be maximized or minimized by the derived configuration.

Building a CSP to derive a set of configurations for an auto-scaling queue uses a similar methodology. Rather than deriving a single valid configuration, however, SCORCH attempts to simultaneously derive both the size of the auto-scaling queue and a configuration for each position in the auto-scaling queue. If SCORCH derives a

size for the queue of K, therefore, K different feature configurations will be derived for the K virtual machine instances that need to fill the queue.

Since SCORCH attempts to minimize cost while ensuring that auto-scaling time constraints are not exceeded, additional information must be added to the CSP to represent the expected types of configurations that will be requested from the queue. The configuration demand models are encoded into the CSP to express this information. The configuration demand models are encoded into the CSP as sets of labeled variables representing the feature configurations of virtual machines that each application may request from the queue.

The CSP for a SCORCH queue configuration optimization process can be described formally as the 8-tuple

$$P =< S, Q, C, D, L, T, M, \gamma >$$

, where:

- S is the size for the auto-scaling queue, which represents the number of virtual machine instances that will be prebooted and available in the queue. This variable is derived automatically as part of the SCORCH optimization process.
- Q is a set of sets that describes the selection state of each virtual machine instance configuration in the queue. If there are Z distinct types of configurations specified in the configuration demand models, then the size of Q is Z. Each set of variables, $Q_i \in Q$, describes the selection state of the features for one of the virtual machine instances in the queue. For each variable, $q_{ij} \in Q_i$, if $q_{ij} = 1$ in a derived configuration, it indicates that the j^{th} feature is selected by the i^{th} virtual machine instance configuration.
- C captures the rules from the feature model as constraints on the variables in all sets $Q_i \in Q$. For example, if the kth feature requires the j^{th} feature, C would include a constraint: $\forall Q_i \in Q,\ (q_{ik} = 1) \Rightarrow (q_{ij} = 1)$.
- D contains the set of configuration demand models contributed by the applications. Each demand model $D_i \in D$ represents a complete set of selection states for the features in the feature model. If the j^{th} feature is requested by the i^{th} demand model, then $d_i j \in D_i, d_i j = 1$. The demand models can be augmented with expected load per configuration, which is a focus of our future work.
- L is the cost model that specifies the cost to include the feature in a running virtual machine instance configuration in the auto-scaling queue. For each configuration $D_i \in D$ a variable $L_i \in L$ specifies the cost of that feature. These values are derived from the annotations in the SCORCH cloud configuration model.
- T is the configuration time model that defines how much time is required to add/-remove a feature from a configuration. The configuration time model is expressed as a set of positive decimal coefficients, where $t_i \in T$ is the time required to add/remove the i^{th} feature from a configuration. These values are derived from the annotations in the SCORCH cloud configuration model.
- γ is the cost minimization objective function that is described in terms of the variables in D, Q, and L.
- M is the maximum allowable response time to fulfill a request to allocate a virtual machine with any requested configuration from the demand models to an application.

3.4 Response Time Constraints and CSP Objective Function

Given a CSP to derive configurations to fill the auto-scaling queue, an objective function can be defined to attempt to minimize the cost of maintaining the auto-scaling queue. Moreover, we can define constraints to ensure that a maximum response time bound is adhered to by the chosen virtual machine queue configuration mix and queue size that is derived.

We can describe the expected response time, Rt_x, to fulfill a request D_x from the configuration demand model as:

$$Rt_x = \min(CT_0 \ldots CT_n,\ boot(D_x)) \tag{1}$$

$$CT_i = \begin{cases} \forall q_{ij} \in Q_i,\ q_{ij} = d_{xj} & 0\ (a), \\ \exists q_{ij} \in Q_i,\ q_{ij} != d_{xj} & \sum t_j(|q_{ij} - d_{xj}|)\ (b) \end{cases} \tag{2}$$

where:

- Rt_x is the expected response time to fulfill the request.
- n is the total number of features in the SCORCH cloud configuration model
- CT_i is the expected time to fulfill the request if the i^{th} virtual machine configuration in the queue was used to fulfill it.
- $boot(D_x)$ is the time to boot a new virtual machine instance to satisfy D_x and not use the queue to fulfill it.

The expected response time, Rt_x is equal to the fastest time available to fulfill the request, which will either be the time to use a virtual machine instance in the queue CT_i or to boot a completely new virtual machine instance to fulfill the request $boot(D_x)$. If a configuration exists in the queue that exactly matches the request (a), the the time to fulfill the request is zero (or some known constant time). If a given virtual machine configuration is not an exact match (b), then the time to fulfill the request with that configuration is equal to the time required to modify the configuration to match the requested configuration D_x. For each feature q_{ij} in the configuration that does not match what is requested in the configuration, t_j is the time incurred to add/remove the feature.

Across the Z distinct types of configuration requests specified in the configuration demand models we can therefore limit the maximum allowable response time with the constraint:

$$\forall D_x \in D,\ M \geq Rt_x \tag{3}$$

With the maximum response time constraint in place, the SCORCH model-to-CSP transformation process then defines the objective function to minimize. For each virtual machine instance configuration, Q_i, in the queue, we define its cost as:

$$Cost(Q_i) = \sum_{j=0}^{n} q_{ij} L_j$$

. The overall cost minimization objective function, γ, is defined as the minimization of the variable $Cost$, where:

$$\gamma = Cost = Cost(Q_0) + Cost(Q_1) + \cdots + Cost(Q_k)$$

.

The final piece of the CSP is defining the constraints attached to the queue size variable S. We define S as the number of virtual machine instance configurations that have at least one feature selected:

$$S_i = \begin{cases} \forall q_{ij} \in Q_i,\ q_{ij} = 0 & 0, \\ \exists q_{ij} \in Q_i,\ q_{ij} = 1 & 1 \end{cases} \tag{4}$$

$$S = \sum_{i=0}^{Z} S_i$$

Once the CSP is constructed, a standard constraint solver, such as the Java Choco constraint solver (`choco.sourceforge.net`), can be used to derive a solution. Section 4 presents empirical results from applying SCORCH with Java Choco to a case study of an ecommerce application running on Amazon's EC2 cloud computing infrastructure.

4 Results

This section presents a comparison of SCORCH with two other approaches for provisioning virtual machines to ensure that load fluctuations can be met without degradation of quality of service. We compare the cost effectiveness of each of the approaches for provisioning an infrastructure for supporting a set of ecommerce applications. We selected ecommerce applications due to the high fluctuations in workload that occur due to the varying seasonal shopping habits of users. To compare the cost effectiveness of these approaches, we chose the pricing model and available virtual machine instance types associated with Amazon EC2.

We investigated three-tiered ecommerce applications consisting of web front end, middleware, and database layers. The applications required 10 different distinct virtual machine configurations. For example, one virtual machine required JBOSS, MySql, and IIS/Asp.Net while another required Tomcat, HSQL, and Apache HTTP. These applications also utilize a variety of computing instance types from EC2, such as high-memory, high-CPU, and standard instances.

To model the traffic fluctuations of ecommerce sites accurately we extracted traffic information from Alexa (`www.alexa.com`) for newegg.com (`newegg.com`), which is an extremely popular online retailer. Traffic data for this retailer showed a spike of three times the normal traffic during the November-December holiday season. During this period of high load, the site required 54 virtual machine instances. Using the pricing model provided by Amazon EC2, each server costs $1.44 an hour to support the heightened demand.

4.1 Experiment: Virtual Machine Provisioning Techniques

Static provisioning. The first approach consists of provisioning a computing infrastructure equipped to handle worst case demand at all times. In our scenario, this technique would require that all 54 servers were run continuously to ensure that response time is

maintained. This technique is similar to computing environments that do not permit any type of auto-scaling. Since the infrastructure can always support the worst-case load, we refer to this technique as *static provisioning*.

Non-optimized auto-scaling queue. Another approach is to augment the auto-scaling capabilities of a cloud computing environment with an auto-scaling queue. In this approach, auto-scaling is used to adapt the number of resources to meet the current load that the application is experiencing. Since additional resources can be allocated as demand increases, we need not boot all 54 servers continuously. Instead, an auto-scaling queue with a virtual machine instance for each of the ten different configurations required by the application must be available to be allocated on demand. We refer to this technique as *non-optimized auto-scaling queue* since the auto-scaling queue is not optimized.

SCORCH. In this approach we use SCORCH to minimize the number of virtual machine instances required in the auto-scaling queue while ensuring that response time is met. By optimizing the auto-scaling queue with SCORCH, the size of the queue can be reduced by 80% to two virtual machine instances.

4.2 Cost Comparison of Techniques

Figure 5 shows the average loads of the ecommerce applications per month.

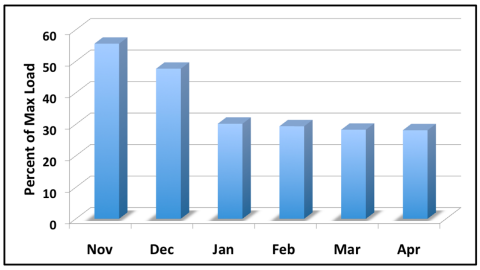

Fig. 5: Average Load Per Month

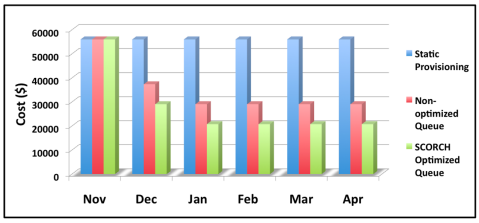

Fig. 6: Monthly Cost Comparison of Virtual Machine Instance Provisioning Techniques

The maximum load for the 6 month period occurred in November and required 54 virtual machine instances to support the increased demand. The monthly operational costs of applying each response time minimization technique can be seen in Figure 6.

Since the maximum demand of the ecommerce applications required 54 virtual machine instances to function, the static provisioning technique was the most expensive, with 54 virtual machine instances prebooted at all times. The non-optimized auto-scaling queue did not require as many virtual machine instances and therefore reduced cost. Applying SCORCH yielded the lowest cost by requiring the fewest number of virtual machine instances to be placed in the auto-scaling queue.

Figure 7 compares the total cost of applying each of the virtual machine provisioning techniques for a six month period. The non-optimized auto-scaling queue and

SCORCH techniques reduced the price of utilizing an auto-scaling queue to maintain response time in comparison to the static provisioning technique. Figure 8 compares the savings of using a non-optimized auto-scaling queue versus an auto-scaling queue generated with SCORCH. While both techniques reduced cost by more than 35%, de-

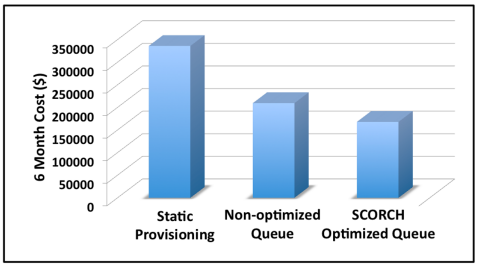

Fig. 7: Total Cost for Provisioning Approaches

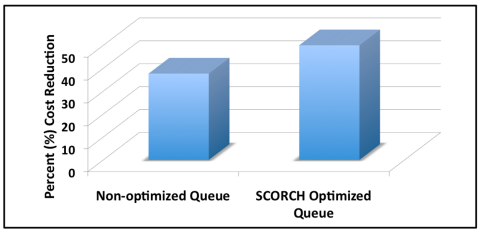

Fig. 8: Total Cost Reduction Comparison

riving an auto-scaling queue configuration with SCORCH yielded a 50% reduction of cost compared to utilizing the static provisioning technique. This result reduced costs by over $165,000 for supporting the ecommerce applications for 6 months.

5 Related Work

This section compares SCORCH to related techniques for handling increased demand in cloud computing environments and automated configuration derivation.

Virtual machine forking handles increased workloads by replicating virtual machine instances onto new hosts in negligible time, while maintaining the configuration options and state of the original virtual machine instance. Cavilla et al [7] describe SnowFlock, which uses virtual machine forking to generate replicas that run on hundreds of other hosts in a less than a second. This replication method maintains both the configuration and state of the cloned machine. Since SnowFlock was designed to instantiate replicas on multiple physical machines, it is ideal for handling increased workload in a cloud computing environment where large amounts of additional hardware is available. SnowFlock is ideal for situations in which a single simple task, such as a page request or query, is computationally expensive, but must exit rapidly.

SnowFlock is effective for cloning virtual machine instances so that the new instances have the same configuration and state of the original instance. As a result, the configuration and boot time of a virtual machine instance replica can be almost entirely bypassed. This technique, however, requires that at least a single virtual machine instance matching the configuration requirements of the requesting application is booted. In contrast, SCORCH uses prebooted virtual machine instances that are likely to match the configuration requirements of arriving applications. Our future work will augment SCORCH with virtual forking techniques, such as SnowFlock, to help reduce/remove

the need to have multiple, pre-booted instances with identical configuration requirements included in the auto-scaling queue.

Automated feature derivation. To maintain the service-level agreements (SLAs) provided by cloud computing environments, it is critical that techniques for deriving virtual machine instance configurations are automated since manual techniques do not support the dynamic scalability that makes cloud computing environments attractive. Many techniques exist for the automated derivation of feature sets from feature models such as [2, 10, 9, 11]. These techniques convert feature models to CSPs that can be solved using commercial CSP solvers. By representing the configuration options of virtual machine instances as feature models, these techniques can be applied to yield feature sets that meet the configuration requirements of an application.

Existing techniques, however, focus on meeting configuration requirements of one application at a time. These techniques could therefore be effective for determining an exact configuration match for a single application. SCORCH analyzes CSP representations of feature models to determine feature sets that satisfy some or all of feature requirements of multiple applications. This information is critical to raise the hit rate of the auto-scaling queue while minimizing additional cost by determining virtual machine configurations that match configuration requirements of multiple applications.

6 Concluding Remarks

Auto-scaling cloud computing environments helps minimize response time during periods of increased application demand, while reducing cost during periods of light demand. The time to boot and configure additional virtual machine instances to support applications during periods of high demand, however, can negatively impact response time. This paper describes how the *Smart Cloud Optimization of Resource Configuration Handling* (SCORCH) MDE tool intelligently populates an auto-scaling queue to provide pre-booted virtual machine instances that can be used immediately, allowing applications to bypass boot-time penalties and significantly reducing allocation time.

SCORCH uses feature models to represent the configuration requirements of multiple software applications and operational costs of utilizing different virtual machine configurations, transforms these representations into CSP problems and analyzes them to determine a set of virtual machine instances that maximizes auto-scaling queue hit rate while minimizing additional operating cost. These virtual machine instances are then placed in an auto-scaling queue to expedite auto-scaling in response to increased application demand. SCORCH differs from existing techniques, such as virtual machine forking, by examining and leveraging the commonalities of the configuration requirements of multiple applications to determine virtual machine instances that maximize hit rate to be included in an auto-scaling queue.

The following are lessons learned from using SCORCH to construct auto-scaling queues:

- **Auto-scaling queue optimization effects operating cost.** Using an optimized auto-scaling queue greatly reduces the total operational cost compared to using a full queue or non-optimized auto-scaling queue. SCORCH reduced operating cost by 50% or better.

- **Dynamic pricing options should be investigated.** Cloud infrastructures may change the price of procuring virtual machine instances based on current overall cloud demand at a given moment. Our future work is incorporating a monitoring system to allow SCORCH to take advantage of such price drops when appropriate.
- **Predictive load analysis should be integrated.** The workload of a demand model can drastically effect the resource requirements of an application. In future work, SCORCH will take into account predictive load analysis to auto-scaling queues that cater to the workload characteristics of applications.

SCORCH is part of the ASCENT Design Studio and is available in open-soure format from `code.google.com/p/ascent-design-studio`.

References

1. D. Benavides, P. Trinidad, and A. Ruiz-Cortes. Automated Reasoning on Feature Models. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering*, Porto, Portugal, 2005. ACM/IFIP/USENIX.
2. D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, volume 3520, pages 491–503. Springer, 2005.
3. J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. Obbink, and K. Pohl. Variability issues in software product lines. *Lecture Notes in Computer Science*, pages 13–21, 2002.
4. S. Hazelhurst. Scientific computing using virtual high-performance computing: a case study using the Amazon elastic computing cloud. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, pages 94–103. ACM New York, NY, USA, 2008.
5. K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (FODA) feasibility study, 1990.
6. V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32, 1992.
7. H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. SnowFlock: rapid virtual machine cloning for cloud computing. In *Proceedings of the fourth ACM european conference on Computer systems*, pages 1–12. ACM, 2009.
8. 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-Volume 00*, pages 124–131. IEEE Computer Society, 2009.
9. J. White, D. Benavides, B. Dougherty, and D. Schmidt. Automated Reasoning for Multi-step Configuration Problems. In *Proceedings of the Software Product Lines Conference (SPLC)*, San Francisco, USA, Aug. 2009.
10. J. White, B. Dougherty, and D. Schmidt. Selecting highly optimal architectural feature sets with Filtered Cartesian Flattening. *The Journal of Systems & Software*, 82(8):1268–1284, 2009.
11. J. White, D. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated diagnosis of product-line configuration errors in feature models. In *Proceedings of the Software Product Lines Conference (SPLC)*, pages 225–234. Citeseer, 2008.
12. J. White, D. C. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortez. Automated Diagnosis of Product-line Configuration Errors in Feature Models. In *Proceedings of the Software Product Lines Conference (SPLC)*, Limerick, Ireland, Sept. 2008.