# Using Strategy Trees in Change Management in Clouds

Hiroshi Otsuka
Research Center for Cloud Computing
Fujitsu Laboratories Ltd
Kawasaki, Japan
hotsuka@labs.fujitsu.com

Hanan Lutfiyya
Department of Computer Science
The University of Western Ontario
London, Canada
hanan@csd.uwo.ca

*Abstract*— **Change management in a cloud environment is often complicated by the different needs of the cloud clients. Changes are not applied all at once. For example, a client may require that a change to the Platform-as-Service (PaaS) instance assigned to it must only be done on the weekend while another client allows for the change to be done at any time. The time periods at which changes can be applied may be specified in SLAs. A change deployment schedule for making changes to PaaS instances often depends on the cloud provider policies and on the SLAs between the clients and the cloud provider. Different sets of cloud provider policies may result in different deployment schedules. Changes are not always successful. This may result in a change being unsuccessful and a return to a previous state in order to re-start the change. Neither is desirable since it may be impact SLA guarantees such as service availability or service time that could result in the cloud provider paying out penalties. Since changes are not all applied at once it may be desirable to modify the change deployment schedule. For example, if an operator is not highly skilled or if the change's complexity is higher than expected then it may be preferable to apply the change during a time period when there are relatively few customers in order to minimize SLA violations. This paper shows how strategy trees can be incorporated into an autonomic change management system that could result in a switch of cloud provider policy sets to determine a new deployment schedule on the fly. Our experiments show that this approach can save time while minimizing SLA violations.**

***Keywords-change management, policies, cloud***

## I. INTRODUCTION

Cloud computing is "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources"[25]. Cloud resources can be shared by multiple applications from different organizations. We will refer to the organizations as clients of the cloud and the software that manages the resources shared by the clients is referred to as the Cloud OS. There are three service types of cloud offerings [25]: Infrastructure as a Service (IaaS), Software as a Service (SaaS) and Platform as a Service (PaaS). IaaS offers virtualized resources such as virtual machines (VM) and storage space offered as different types of VM per unit time and storage space per dollar. Examples that represent this type of offering include Amazon's Elastic Compute Cloud (EC2) [6] and Elastic Block Storage (EBS) [7]. SaaS is where a provider offers a service and the infrastructure for that service to many clients. An example of a SaaS offering is Salesforce.com [20]. PaaS is where various applications are deployed within a VM (i.e. HTTP, application, and database servers). Typically these applications represent a solution stack. Google App Engine [19] is an example of a PaaS offering. An instance of a PaaS refers to the VM where the applications are installed and the physical server that the VM is placed on. The Cloud OS assigns a PaaS instance to clients in response to their requests. Thus a cloud environment may have multiple PaaS instances.

The configuration of a PaaS environment may need to change. Examples of changes include a software patch for a security problem, installing a new version of software and modifying a configuration setting. These changes are often non-trivial. *Change management* is a disciplined process for introducing required changes into the IT environment [9]. A change is initiated with a formal proposal, which is referred to as a *request for change (RFC)*. The information in an RFC includes the objective of the change, its impact, severity, priority (which is the result of considering the impact and severity), deployment steps, verification steps, etc; The Change Advisory Board determines if the change should proceed. If the change specified in the RFC is approved a change plan is generated. The change plan identifies a sequence of activities, the resources required to implement the change, rollback plans and budget [8]. The change plan also specifies the *deployment schedule*, which specifies deadlines for when the change should be completed.

Deployment schedules are influenced by cloud provider *policies* [10]. Often a set of policies is designed to achieve a specific business objective. One example business objective is that for an RFC the number of service level agreement (SLA) violations should be less than $m$ and the period of time that the change is applied should be less than $t$ time units. Other constraints on deployment schedules may come from SLA. As part of an SLA there may be a specification of when changes may occur e.g., servers may be taken down at most on two consecutive weekends. Another SLA may specify that security patches can be applied at any time. Based on the policies of the cloud provider and the SLAs a global optimization model can be constructed and solved to

determine the change schedule. The result is a deployment schedule where changes to different PaaS instances are done over a period of time. Different policies may result in different deployment schedules. We can think of a set of policies as a strategy.

A change does not always succeed. One possible cause is that an inexperienced engineer may have underestimated the time it takes to complete the activities related to changing a PaaS instance in the specified period of time. A change may be rolled back but rolling back is often expensive. If there are too many changes that cannot be completed or too many rollbacks then it may be feasible to modify some aspect of the change plan. For example, the deployment schedule could be modified to allow more time to be allocated to a change or for a different time period to apply the change e.g., the change should be applied when there are fewer customers, which could be the weekend.

We note that in this work when we refer to a change that may be rolled back, we are referring to an action that places the system in a different state in response to a problem with a change. This may be a previous state but we also allow for other states. One type of rollback may occur when the incorrect configuration of a software stack on a VM could result in the VM being thrown away and replaced with a VM with the original stack. Another type of rollback occurs if a change resulted in the migration of a VM and that migration impacted other VMs on that physical machine then that VM could either be migrated back or terminated. These types of rollback actions are often found in cloud environments.

This paper describe a novel change management architecture that allows for strategies to be changed on the fly based on feedback on observed system behavior after a change has been deployed. We assume that a strategy can be represented by a set of policies, which we will refer to as a *policy-set*.

The paper is organized as follows. Section II presents an example showing how SLAs and policy-sets could influence the deployment scheduled. Section III presents the concept of strategy-tree and shows how it can be applied to the change management problem to switch strategies. Section IV describes a change management architecture that uses strategy-trees. Section V shows our change management system implementation, and Section VI describes evaluation results using a simple scenario. Section VII describes related works. Discussion and conclusions are presented in Section VIII and Section IX respectively.

## II. MOTIVATING EXAMPLE

In this section we use a simple example to illustrate how a deployment schedule can be influenced [10]. Assume two clients. The first client is an owner of a web 2.0, beta service with no guarantees. The second client uses the cloud provider for enterprise use.

A *change window* is a time period in which a change may be applied. Change windows may be influenced by SLAs and may be influenced by the priority of the change. For the first client the SLA between the cloud provider and the client may specify that the cloud provider is allowed to update its PaaS instance at anytime in exchange for a low hourly rate. For the second client only changes of high priority, e.g. a critical security patch can be applied at any time. For the first client there may be many change windows while for the second client there are fewer possible change windows.

Another factor that may influence change windows are the cloud provider policies. For example, the cloud provider may have a policy that states that if a change has priority one, which is assumed to be a change of the highest priority, then the change can be applied at any time. Another possible policy is that if a change is of priority one or priority two then the change may be applied at any time.

One problem is that the sequence of activities needed to implement a change may not always succeed. This could result in the unavailability of the PaaS instance, which may impact promises specified in the SLA with respect to availability and service time requirements. There are various reasons for this including engineers that do not have the sufficient skill set or experience, the full impact of the changes was not fully anticipated etc; A possible consequence is a rollback. This could be costly and thus it is desirable that rollbacks are either avoided or minimized.

This work aims to use feedback, through system behavior, about the deployment of the change to determine if a new deployment schedule is needed. This may require a different set of policies to be used to determine a new deployment schedule. For example, the policy set used by the cloud provider to make decisions about the deployment schedule could be the following:

**if** (changePriority == 1 or changePriority == 2)
**then** apply change at any time

**if** (changePriority > 2) **then** apply change only on the weekend

In the rest of this paper we will refer to this as policy-set A. Another policy-set that the cloud provider could use is the following:

**if** (changePriority == 1)
**then** apply change at any time

**if** (changePriority > 1) **then** apply change only on the weekend

In the rest of this paper this will be referred to as policy-set B. Policy-set A results in a deployment schedule that is more aggressive, e.g. the changes are applied faster, than policy-set B. If the changes do not result in loss of availability or increased service time then the aggressive approach is better. However, if the SLA violations exceed some threshold value then it may be better to use policy-set B.

Different policy sets, which come from the cloud provider, often imply different deployment schedules. In this paper we

show how feedback can be incorporated into a change management system that allows for a change in the policy-sets and hence allows for an on the fly change in deployment schedules.

## III. STRATEGY-TREES

In Section II, two policy-sets were introduced. Each policy-set is an expression of a strategy to be implemented. Each strategy is designed for achieving a *directive* where a directive is a set of objectives to be satisfied at run-time. An objective represents a constraint on a metric that represents some aspect of system behavior. A metric may be as simple, low-level, technical metric (i.e., throughput, availability) or as potentially complex as a high-level business metric (i.e., profit, satisfaction).

The strategy-tree is introduced as an abstraction that encapsulates and relates multiple strategies for achieving a directive [15]. Strategy trees were introduced in [11] to evaluate the efficacy of the policy-set in use. This evaluation is based on expectations (defined by an administrator) at various temporal granularities. Strategies can be changed at runt time to better achieve the directive.

This section presents the strategy-tree concept and shows how it can be used to generate new change plans in response to feedback about system behavior after previous changes were applied.

### *A. Defining a Strategy Tree*

A strategy-tree [11] is a directed acyclic graph ST = (V, E) where $V = A \cup O \cup D$. The set of AND type nodes is represented by A, O represents the set of OR type nodes, D represents a set of directive nodes and $E \subset V \times V$. Each type of node is associated with a directive to be achieved over some time interval. However, both AND and OR type nodes also hold additional properties with regards to the structural relationships among nodes in the tree. Specifically, an AND type node provides a conjunctive property in which all child nodes of an AND type node are included in any strategy that includes that particular AND type node while OR type nodes provide a disjunctive property in which each child of the OR type node is a member node of an alternative strategy. We refer to a node as a Directive node if it is also not an OR node or an AND node. The root node and all leaf nodes must be of Directive type node. As long as the root and leaf nodes are not AND or OR type nodes any internal combination of nodes is acceptable. Each leaf node of a strategy tree is bound to a policy set. A strategy, (V',E'), is defined such that the following properties hold:

1. A strategy consists of the nodes $V' = (A',O',D')$ such that $A' \subseteq A$, $O' \subseteq O$ and $D' \subseteq D$.
2. Every child of an AND type node $n \in A$ is included in a strategy.
3. One child of an OR type node $n \in O$ is included in a strategy.
4. Leaf nodes of the strategy tree are bound to policy sets.

A strategy-tree must contain at least one OR type node and at least three directive type nodes. This is necessary because one of the purposes of a strategy-tree is to encapsulate multiple strategies and in order to have different strategies an OR type node is needed to provide alternatives. An example strategy tree (which will be described in more detail) is seen in Figure 1. The strategy tree has two alternative strategies.

Periodically an evaluation should take place to determine if the directive is being met. The frequency of evaluation is represented by an attribute, which is found in each node. This attribute is referred to as the *quantum attribute value*. The Management Time Unit (MTU) represents the value of the quantum attribute at the leaf nodes. The quantum attribute value of a parent node should always be greater than or equal to the associated quantum attribute value of any of its direct child nodes. In practice the quantum attribute value of the parent node modulo the quantum attribute value of the child node should be zero. The value of the quantum attribute value attribute in a node is an integer coefficient of MTU values. We use the term *epoch* to refer to the number of increments of the MTU equivalent to a given node's quantum attribute value. This attribute, allows for the nodes to be ordered temporally from root (maximum value) to leaf (minimum value) within a strategy-tree.

To understand the relationship between the directive at the root node and the directives in nodes at lower levels consider the following analogy [12]. The objective of a student taking a course might be to get an A+ in the course. The marking scheme may allocate 15% for quizzes, 35% for the midterm exam and 50% for the final exam. The weekly quizzes and midterm are points of feedback to the student. This feedback can be used to assess how well the student is learning the course material and whether a change in strategy is necessary i.e., attend more lectures, study more, sleep more. The quizzes give snapshots of weekly performance, a series of four of these weekly snapshots provides a more comprehensive view while the midterm being much more involved and covering more information attempts to provide a deeper and more complete assessment of a student's understanding up to that point.

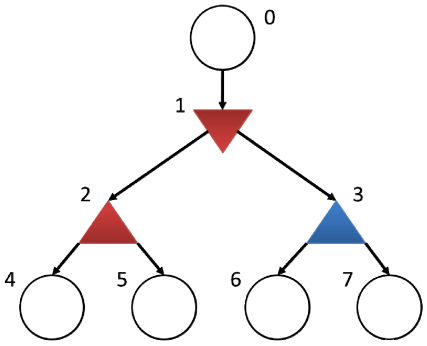
To support evaluation of directives, the cloud provider defines a SAT-element (i.e., satisfaction element) that encapsulates the evaluation related to the node's directive. This evaluation occurs at the frequency specified by the quantum attribute value. A second type of element, a DEC-element (i.e., decision making element), is defined for OR-type nodes in the tree. Like SAT-elements, DEC-elements are evaluated according to the node's quantum attribute value. The decision making defined within these DEC-elements analyzes the effectiveness of the current strategy and makes a choice about whether to persist in using the current strategy or to switch to an alternative strategy. The strategy-tree, through its hierarchical relationships among nodes, allows for the monitoring and evaluation of the active strategy's performance at (potentially) several granularities of time. Furthermore, a decision-making capability is available at the OR-nodes.

These nodes act as choice points which allow for alternative strategies to be employed on the fly.

### B. *Example Strategy Tree for Change Management*

This section describes how strategy trees can be used for Change management. **Example 1** presents an example of a strategy-tree.

**Example 1:** An example strategy-tree is presented Figure 1. Let $S_i$ denote the i-th strategy in the strategy-tree. This strategy-tree has two alternative strategies that we will denote as $S_1$ and $S_2$. As in **[12]**, a shorthand notation used to represent a strategy is to list the participant node indices for a particular strategy enclosed in round braces. For example, $S_1$=(0,1,2,4,5) and $S_2$=(0,1,3,6,7). Node 1 is an OR-node. Each outgoing edge represents a different strategy. The policy set associated with $S_1$ is policy-set A and the policy set associated with $S_2$ is policy-set B. The leaf node 4 and leaf node 5 are bound to policy-set A while the leaf node 6 and leaf node 7 are bound to policy-set B.




| Index | Quantum Attribute Value |
|---|---|
| 0 | 168 |
| 1 | 168 |
| 2 | 168 |
| 3 | 168 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |
| 7 | 1 |

**Figure 1: The Strategy Tree Used for Evaluation**
Node Numbers are shown at each node. Directive type node is shown as circle. AND type node is as triangle, OR type node as is upside down node, relatively.

The strategy-tree, through its hierarchical relationships among nodes, allows for the monitoring and evaluation of the active strategy's performance at (potentially) several granularities of time. Furthermore, a decision-making capability is available at the OR-nodes. These nodes act as choice points which allow for alternative strategies to be employed on the fly. Example 2 describes the quantum attribute values and MTUS for each of the nodes for the strategy tree presented in Figure 1.

**Example 2:** The SAT-element of nodes 4,5,6,7 of Figure 1 evaluate the directives associates with those nodes once every MTU while the SAT element associated with node one evaluates its directive every 168 MTUs. An MTU is assumed to be one hour. The DEC-element associated with node 1 makes a decision, based on current and past feedbacks from the Cloud OS, to determine whether for the next epoch strategy $S_1$ or strategy $S_2$ should be used.

A strategy-tree allows a run-time evaluation of the active strategy may be made at multiple temporal granularities. Further, due to its temporal partitioning it allows for different directives to be evaluated on various time scales to better achieve the root directive.

### C. *Using a Strategy Tree in Change Management*

In order to use strategy-trees it must be possible to monitor and evaluate system behavior. A metric represents an attribute that characterizes an aspect of system behavior. This work defines the *change quality ratio,* $R_q$, as the ratio of the number of deployments finished successfully ($N_{suc}$) to the total number of deployments ($N_{all}$) i.e. $R_q = N_{suc} / N_{all}$. Figure 2 represents a state transition diagram associated with a change deployment. $N_{all}$ is the number of change deployments that terminated either in the state of VERIFIED, ROLLEDBACK, or ROLLBACKFAILED. The value of $N_{suc}$ represents the number change deployments that terminated in VERIFIED.

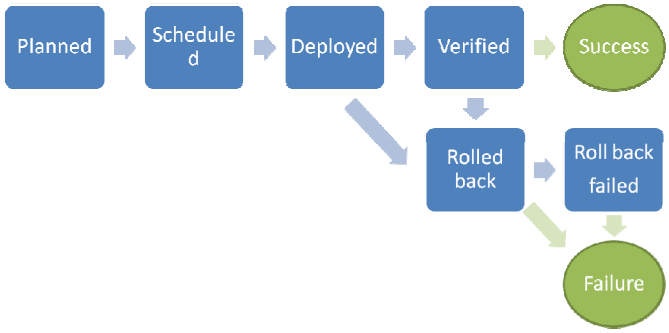



**Figure 2: Change State Transition Diagram**

The number of change deployments rolled back successfully is regarded as a failure. Many rollback occurrences are often the result of a bad design of the deployment tasks. Thus $N_{suc}$ does not include the number of deployments that ended in the ROLLEDBACK states. Future work could have different types of rolled back states where each state represents a range of steps e.g., 1 to 5, 6 to 10. States with few rollback steps would be considered successful.

We used the change quality ratio to define the following directive for node 0 of the strategy tree presented in Figure 1.

*Obtain a change window that is less than t time units while the SLA is being met*

We assume that the *t* time units are of sufficient duration. The OR-node in node 1 uses results of evaluations of lower-level nodes to evaluation satisfaction towards the above directive. This evaluation occurs every 168 MTU periods and is used to determine if a new strategy should be used.

For the purposes of evaluation the administrator defines a SAT-element (i.e., satisfaction element) that encapsulates an evaluation that is useful in determining if a directive is being achieved. The evaluation encapsulated by SAT-elements is performed once per epoch (based on the quantum attribute

value of the node). **Example 3** describes SAT-elements used in the strategy-tree presented in Figure 1.

**Example 3:** The SAT-elements associated with node 4 and node 6 are concerned with detecting SLA violations with respect to availability. These SAT-elements compute the availability rate of the PaaS platform in the past hour (which is the MTU value). If the availability rate is greater than the availability threshold (which is defined in the SLA), success is signaled. Otherwise, a failure is signaled. The SAT-elements associated with node 5 and node 7 are concerned with evaluating the latest change quality ratio. The SAT-element associated with node 5 computes the latest quality ratio, and compares it to a swiftness threshold (which is defined in the SLA). If the ratio is greater than the swiftness threshold, then success is signaled. Otherwise, failure is signaled. The SAT-element associated with node 7 computes the latest quality ratio, and compares it to an availability threshold. If the ratio is greater than the threshold value, success is signaled. Otherwise, a failure is signaled. The quantum attribute value of nodes 2 and 3 are 1 MTU and take AND of results from its two child nodes. Node 2 and 3 signals the result.

A SAT-element maintains a list where each element of the list represents the result of an evaluation. This list is provided to the parent node. The size of this list is the quantum attribute of the parent nodes. Thus nodes 2 and 3 in Figure 1 maintain a list with size 168 since the quantum attribute value of node 1 is 168.

A DEC-element is also provided by the cloud provider and is associated OR type nodes, for evaluating whether to maintain or switch the current strategy, and if, a change is needed, to which strategy this switch should be made.

**Example 4:** The only DEC-element associated with node 1 (which has quantum attribute value 168) is concerned with determining which strategy to employ each week while attempting to converge on achieving the directive of node 0. This DEC-element uses a context of (i) current active strategy and (ii) results lists from SAT-elements. If the current sub-strategy is $S_1$ and results list has greater than or equal to 50% success entries then the current strategy is maintained. If results list from node 2 has less than 50% of its entries indicating success then a switch to $S_1$ is made. If the current sub-strategy is $S_2$ and results list has less than 50% of its entries indicating success then the current strategy is maintained. If results list has greater than or equal to 50% of its entries indicating success then a switch to $S_1$ is made.

The value of 168 MTUs in the DEC element allows changes scheduled but not yet done to be postponed no more than a week to a time period that is less busy e.g., weekend. The amount of time to postpone depends on the type of change. This has an impact on the MTU value and quantum attribute values. Strategy trees are flexible in the values used.

## IV. ARCHITECTURE

This section describes the architecture for Change Management Environment that uses strategy-trees. The Monitoring Client examines availability of PaaS instances that are allocated to clients. The Change Request Client and the Change Manager provide basic Change Management functionality. In addition, the Policy Manager adds an automated management policy switching function. The rest of this section briefly describes the components of the automatic change management environment that is depicted in Figure 3.

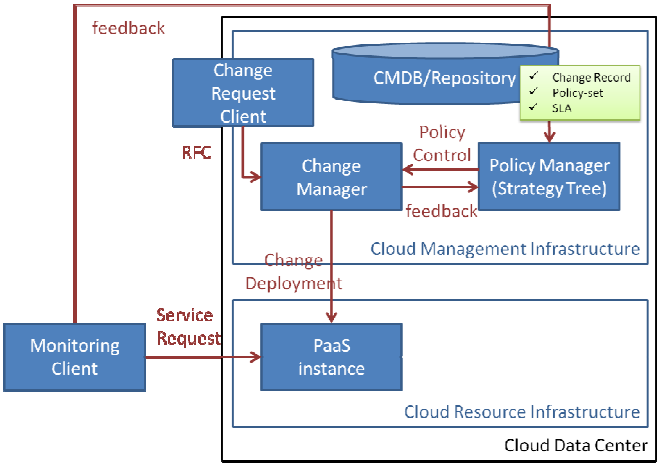



**Figure 3: Autonomic Change Management Environment**

**Configuration Management Database (CMDB)**: We assume that the Cloud OS has its own management database/repository as prescribed by ITIL. The CMDB is implemented as a set of multiple databases, such as a configuration database, performance database, and so on. All feedbacks are retrieved from CMDB. Change records are also available in the CMDB.

**Change Manager**: The Change Manager provides a change processing engine as well as an automated change deployment mechanism for this work. The Change Manager carries out these tasks: (i) The Change Manager manages all change records using CMDB as database; (ii) The Change Manager schedules deployment of each change according to the current policy-set; (iii) The Change Manager triggers the change deployment process at the scheduled time.

**Policy Manager**: The Policy Manager implements strategy-trees. The Policy Manager gathers the feedbacks it needs, and performs a directive evaluation every epoch for each node of the current strategy-tree. The Policy Manager chooses a strategy/policy-set for the next epoch. The Policy Manager notifies a Change Manager that a new strategy/policy-set was chosen.

**Monitoring Client**: The Monitoring Client monitors PaaS instances to evaluate whether SLAs are met or not. This client tests availability of instances and records the results to CMDB.

**Change Request Client**: This client usually manipulated by the engineer who is involved in the system, requests configuration changes from the Change Manager. An Incident Management System could issue changes as a consequence of actions taken for incidents.

## V. IMPLEMENTATION

Prototypes of all components were implemented. All code was written in Ruby. The strategy-tree (Policy Manager) is implemented based on the code found in [16].

Ruby on Rails [21] was used to implement Change Manager and CMDB/Repository interface. SQLite [22] was used to implement the database. Parts of tasks related to deployment, verification, and rollback were written as shell scripts**.** The Monitoring Client, Change Manager stored data into SQLite via CMDB interface.

VMware Player [23] is used to simulate all of components. PaaS instances provided HTTP service on Apache HTTP Server [24].

Changes for the PaaS instances were randomly generated. Changes from clients requested modification of the HTTP server configuration file (by modifying a parameter value) and restarting the HTTP server. Each change was received and stored into the CMDB. Each change was scheduled to be deployed under the current policy-set. When the time came, changes were deployed, verified, and rolled back if necessary. Availability of PaaS instance was monitored and stored into CMDB to evaluate satisfaction of SLA.

## VI. EVALUATION

This section describes evaluation of our proposed approach by comparing the results of using only policy set A, only using policy set B and using a strategy tree that alternates between policy set A and policy set B as needed.

### *A. Scenario*

Suppose there were two clients. As aforementioned, the first client provided service to end users with no guarantee. This client required swift changes. The second client is related to enterprise customer, and emphasized availability.

One PaaS instance was allocated for the clients. These instances were to be used for about a month. One month is the typical rental period unit. A cycle of 28 days, which is nearly equal to one month, was employed instead.

For the experiments both clients used the same SLA, which was defined under this assumption as follows:

*Cloud Provider guarantees 95% of availability over all instances for the client during a period of 28 days. If this agreement is not met, penalty is paid for the client. The hourly rate of violation fee is one monetary unit (which is the same amount of allocation fee). Penalty would be calculated as the product of the hourly rate and the service unavailable period.*

Under the contract between the cloud provider and clients, the cloud provider may have a regular weekly maintenance of the infrastructure during weekend (announced in advance). Under this contract, availability was calculated without monitoring results during weekend.

### *B. Experiment*

Ten trials were performed under the test scenario described in the previous subsection. Each trial was run three times for policy-set A, policy-set B and the strategy-tree.

The Change Request Client generated four changes per day following a Poisson distribution. Because of the absence of a cloud change model, we chose to generate changes following the typical random distribution.

Changes were planned to modify an HTTP service parameter. The priority of changes was two. There is another parameter to simulate deployment failure described below.

For the strategy-tree the initial set was policy-set A. Each change deployment involved a parameter change and a reboot of HTTP server for the instance was required.

The policies used in this paper are based on the observation that organizations assign priorities to types of changes. Change scheduling is based on priority. For a cloud environment, organizations (cloud customers) can use SLAs to specify priorities. This leads to this paper's policies. If the use of strategy trees were not effective for these policies, it most likely would not be effective for more complex policies.

The first five trials assumed that changes are not well designed. Change deployment failed 50% of the time and the rollback task required two hours to complete.

The last five trials assumed that changes are well designed by a skilled engineer. Change deployment failed 10% of time and the rollback task required two hours to complete.

The quality of the changes is reflective of the fact that there is a mix of good and bad changes. Changes with long rollback time allow us to evaluate the impact of the rollback process.

If the strategy-tree switches policy-sets, change deployment can be scheduled differently. Thus, the use of the strategy-tree results in the dynamic adjustment of the change window. Each run using the strategy-tree required 672 periodic reviews (i.e., 4 weeks) by node 1 (see Figure 1). The DEC-element evaluated its directive 168 times in a week. The policy-set was switched at most three times during one trial. Change deployment was postponed by at most one week.

Figure 4 shows the results. In this scatter diagram, the vertical axis shows availability rate during 28 days as a percentage and the horizontal axis shows the mean duration of change in the number of days. Duration of a change is a period of time from the change's request to completion.

One run plots one dot in this diagram. One trial plots three dots in this diagram. If a dot is plotted at the point below 95%, SLA is not met at that 28 days period. If not, SLA is met. If a dot is plotted at the point of 1.0 in horizontal axis, changes took one day on average.

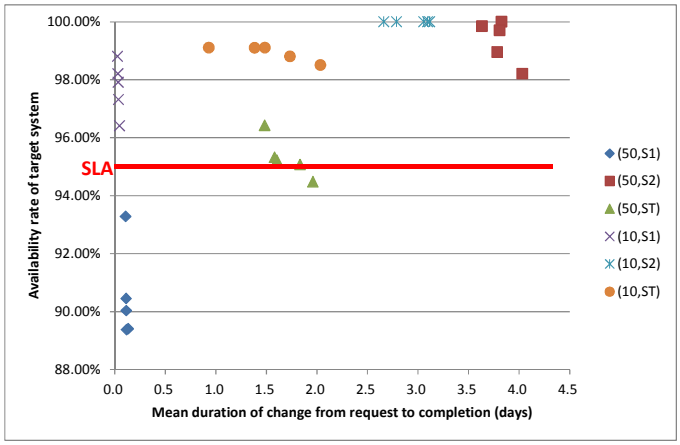



**Figure 4: Results of simulation**

Results with Strategy Tree shows that SLA was met and its deployment finished in lesser duration than results of policy-set B (Availability-first policy-set) except for one trial

There are six groups of dots in the diagram in relation to parameters. Each group is described by a pair of parameter using change failure rate of as *r* and strategy as *s*; (r,s) = (10,$S_1$), (10,$S_2$), (10,ST), (50,$S_1$), (50,$S_2$), (50,ST). ST means the strategy-tree was used. $S_1$ means that only policy-set A was used and $S_2$ means that only policy-set B was used.

For the first client scenario, results with four parameter pairs (10,$S_1$), (10,ST), (50,$S_1$), (50,ST) are compared. This client prefers changes to be applied as quickly as possible. Thus policy-set A was used. $S_1$ was the default strategy when the strategy-tree was used. In the result of (10,$S_1$) and (50,$S_1$), each mean of change windows was within one day on average. However, the availability of (50, $S_1$) was below 95%. Using the strategy-tree, availability rate was kept above 95% as (50, ST). If $S_2$ was adopted, the mean of change duration was larger than with the use of the strategy-tree.

For the second client scenario, results with four parameter pairs (10, $S_2$), (10, ST), (50, $S_2$), (50, ST) are compared.

The second client prefers availability of service and thus policy-set B was used. $S_2$ was the default strategy when strategy-tree was used. In the result of (10, $S_2$) and (50, $S_2$), the availability rate was on average above 95% for each run, which was the value was specified in the SLA. However, changes took more than 3.5 days to be finished on average in each run. When the strategy-tree was used, availability rate was kept above 95%, and changes took less than 3.5 days to be finished.

## VII. Related Work

There is considerable research related to different aspects of IT change management. These are briefly described in this section.

Sauve et al 2006 [8] discusses how business considerations can be taken account in the scheduling of changes. More specifically this work mathematically relates low-level IT metrics such as availability to business objectives e.g., business loss. Our work is based on a similar concept. The directives at the high-level of the strategy tree are intended to represent business objectives. Although not explicitly quantified in the paper, a low change quality ratio does have an impact on results of change management process**.** It is discussed in this study.

Cordeiro et al 2008 [1] proposes a change template as a mechanism for formally specify and reuse experience related to changes. This can be used to aid in the development of future change plans. The effectiveness of using change plans is demonstrated in Cordeiro et al 2009 [4]. Our work benefits in that experience often allows for that development of change plans that reflect specific circumstances. Different strategies may correspond with different circumstances suggesting the need for different change plans.

The mapping of change plans to workflows is described in [2]. Other work related to automation includes [3][5].

The execution of a change plan may fail for a number of reasons. Machado et al 2008 [7] addresses this issue with showing how rollback mechanisms can be incorporated into IT change management systems.

Lunardi et al (2010) [6] identifies the need to develop multiple strategies for automated planning of assignment of human resources to IT activities. The importance is based on the inability to find a "one-size-fits-all" strategy.

As far as we can tell none of the existing work discusses being able to switch strategies in change deployment schedules on the fly.

## VIII. Discussion

The development of our prototype is ongoing. Based on our observations and experiences so far with the prototype we have drawn insights from our current prototype into the applicability of the approach and possible extensions to the prototype.

A cloud provider needs low cost management methods, and autonomic change management is able to cut the labor cost by operating tasks without engineer. One of the problems is to parametrize problems solved manually many times on the cloud e.g. design of change scheduling policy. We designed our example focusing on this problem.

For practical use, several problems should be considered. These are discussed in this section.

**Scalability for the Cloud System:** A strategy-tree uses feedback about system behavior to evaluate satisfaction of directives and changes policy-sets (and hence strategies) based on that evaluation. The architecture presented Figure 3 is scalable. The data needed for evaluation is stored in the CMDB. The CMDB could consist of multiple repositories stored on multiple machines. SAT-elements of child nodes of an AND-node can be evaluated in parallel and on multiple machines.

**Choosing threshold values:** This study assumed some threshold values are given. However, it is difficult to determine appropriate values sometimes. It is also needed to

change the values in accordance with requirement changes on business.

**Suitability of our Approach:** Strategy-trees appear to be well-suited for change management in cloud environments, which are large and potentially require many changes that take place over time. This suggests that feedbacks can be plentiful in a short period of time and thus there will be a good deal of information to evaluate satisfaction of directives. Furthermore, there are various assumptions (e.g., experience level of operators). Different assumptions result in different change management strategies. Often it is not until run-time that the validity of an assumption can be evaluated. The strategy-tree allows for run-time evaluation and a modification of strategies (through a change of a policy-set) at run-time when incorrect assumptions result in directives not being satisfied.

**Corrupt/defect detection of strategy-tree:** Consider a scenario where none of the policy-sets will satisfy SLAs or the number of RFCs is so high that not all changes that is expected to be applied on a weekend can be done. The strategy-tree is not able to find a policy-set that satisfies the directive. Currently the ability to learn new strategies or even the best switch among existing strategies is not incorporated in strategy-trees or the autonomic change management system presented in this paper. One possible is to take the data in the CMDB from a real-environment to be used in a simulator to evaluate other possible policy-sets. We would like to explore the use of learning techniques, which would be used offline, to determine the best switch or even possible new strategies. An example of an approach that could be used as a basis is found in [13].

**Measuring Business Impact:** The change quality ratio was the indicator used to determine if a policy set was effective. There are other possible indicators that may be used to represent business (e.g., [8]). Future work could explore the use of different indicators. For example, a rollback was considered an unsuccessful change. We may want to consider the number of steps involved in a rollback to define a metric that considers partial successes.

## IX. CONCLUSIONS

We proposed a change management system that schedules change deployment automatically using the policy-set given by a strategy-tree. Simulation results show that changes can be processed such that SLAs are satisfied and changes can be deployed in a shorter change window using our autonomic change environment system. Although the evaluation was limited to specific scenario the results suggest that the use of strategy tree in an autonomic change environment is worth pursuing. We do expect to evaluate the system for various types of scenarios. Future work includes the following.

**Reutilization considering Policy-Set Assumptions:** The strategy-tree designed in this work makes assumptions that the sequence of activities needed to implement a change is not changed. For example, if there are engineers performing the changes that are highly skilled then they may be able modify the activities required for the change before making further deployments of the change. Currently this is not considered in this study and is a topic for future work.

**Improvement of Change Models:** The change in the experiments presented in this paper is relatively simple with a change of only one parameter value i.e., changing the maximum number of connections to an HTTP server. Changes are often more complex e.g., entire software upgrade. These kinds of changes require a set of simpler changes that must either all be done together or not at all. It is possible that the changes within this set must be done in a specific order. The work described in [14] formalizes the change scheduling problem as an optimization model that minimizes business impact, where business impact is defined using some metric. The optimization model considers more complex changes described above. One possible use of policies is to specify the number of concurrent changes that can take place, specify the cost if a change is not done within a specific amount of time or specify how human resources are to be assigned [6]. Further work will examine this in more detail in that new optimization models will be generated and solved to determine a new deployment schedule. Approximation algorithms could be explored to find a solution that is close to optimal but produces a new deployment schedule quickly.

Furthermore, determining the quality and distribution of changes would require extensive analysis of actual log files. The experiments in this paper are needed to show that it is possible to develop an effective autonomic change management system before going to this next step.

**Multiple strategy-trees on the Cloud:** Currently this work focuses on change management. The policy-sets described are specifically designed for achieving directive that are relevant for change management. There may be another policy-set focused on resource management. There is the potential for multiple policy-sets that can be in conflict with each other. This requires cooperation between the Policy Manager, resource allocation component of the Cloud OS and the change management environment system. This is a topic for further study.

## REFERENCES


[1] W. Cordeiro, G. Machado, F. Daitx, et al., "A Template-Based Solution to Support Knowledge Reuse in IT Change Design," Proceedings of the IFIP/IEEE Network Operatons and Management Symposium (NOMS 2008), p. 355-362, 2008.

[2] W. Corderio, G. Machado, F. Andreis, A. Santos, C. Both, L. Gaspary, L. Granville, C. Bartolini and D. Trastour, "A Runtime Constraint-Aware Solution for Automated Refinement of IT Change Plans", Proceedings of the 19th International Workshop on Distributed Systems: Operations and Management (DSOM 2008), pp. 69-82, 2009

[3] W. Cordeiro, G. Machado, F. Andreis et al., "ChangeAdvisor: A Solution to Support Alignment of IT Change Design with Business Objectives/Constraints", Proceedings of the 20th International

Workshop on Distributed Systems: Operations and Management (DSOM 2009), 27-28, October 2009.

[4] W. Cordeiro, G. Machado, F. Andreis, A. dos Santos, B. Coth, L. Gaspary, L. Granvile, C. Bartolini, D. Trastour, "ChangeLedge: Change Design and Planning in Networked Systems Based on Reuse of Knowledge and Automation", Computer Networks 53, pp. 2782-2799, 2009.

[5] A. Keller, J. Hellerstein, J. Wolf et al, "The CHAMPS System: Change Management with Planning and Scheduling", Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), p. 395-408, 2004.

[6] R. Lunardi, F. Andreis, W. Cordeiro, J. Wickboldt, B. Dalmazo, L. Gaspary, L. Granville, "On Strategies for Planning the Assignment of Human Resources to IT Change Activities", Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2010), 8 pages, 2010

[7] G. Machado, F. Daitx, W. Cordeiro, C. Both, L. Gaspary, L. Granville, C. Bartolini, A. Sahai, D. Trastour, K. Saikoski, "Enabling Rollback Support in IT Change Management Systems", Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2008), pp. 347-354, 2008.

[8] J. Sauve, R. Reboucas, A. Moura, C. Bartolini, A. Boulmakoul, and D. Trastour, "Business-Driven Decision Support for Change Management" Planning and Scheduing of Changes", Proceedings of the 17th International Workshop on Distributed Systems: Operations and Management (DSOM 2006), pp. 173-184, 2006.

[9] D. Trastour, M. Rahmouni, C. Bartolini, "Activity-Based Scheduing of IT Changes", AIMS 2007, LNCS 4543, pp. 73-84, 2007.

[10] H. AbdelSalam, K. Maly, R. Mukkamala,M. Zubair, D. Kaminsky, "Scheduling-capable Auton0mic Manager for Policy-Based IT Change Management System", Entrprise Information Systems, 4:4, 423-444.

[11] B. Simmons, H. Lutfiyya, "Strategy-Trees: A Feedback Based Approach to Policy Management", Proceedings of the 3rd IEEE international workshop on Modelling Autonomic Communications Environments (MACE 2008), S. Meer, M. Burgess, and S. Denazis (Eds.). Springer-Verlag, Berlin, Heidelberg, 26-37.

[12] B. Simmons, H. Lutfiyya, "Achieving High-Level Directives Using Strategy-Trees", Proceedings of the 4th IEEE International Workshop on Modelling Autonomic Communications Environments (MACE 2009), J. C. Strassner and Y. M. Ghamri-Doudane (Eds.). Springer-Verlag, Berlin, Heidelberg, 44-57.

[13] M. Rahmouni, and C. Bartolini, "Learning from Past Experiences to Enhance Decision Support in IT Change Management", Proceedings of the IFIP/IEEE Network Operatons and Management Symposium (NOMS 2010), pp. 408-415, 2010.

[14] R. Reboucas, J. Sauve, Antao Moura, C. Bartolini, D. Trastour, "A Decision Support Tool to Optimize Scheduling of IT Changes", IEEE/IFIP International Conference on Integrated Network Management (IM 2007), pp. 343-352, 2007.

[15] B. Simmons. "Strategy-Trees: A Novel Approach to Policy-Based Management". PhD Thesis. The University of Western Ontario. Department of Computer Science. 17 June, 2010. London, Ontario, Canada.

[16] B. Simmons, Private Correspondence, January 2011.

[17] Amazon, "Elastic compute cloud (ec2)," 2010. http://aws.amazon.com/ec2/ [online March 11].

[18] "Amazon elastic block store (ebs). "http://aws.amazon.com/eb2 [online March 2011].

[19] Google, "Google app engine." http://code.google.com/appengine/ [online March 2011]

[20] Salesforce, http://www.saleforce.com/ [online July 2011].

[21] Ruby on Rails, http://www.rubyonrails.org/ [online March 2011].

[22] SQLite, http://www.sqlite.org/ [online March 2011].

[23] VMware, "VMware Player", http://www.vmware.com/products/player/ [online March 2011].

[24] Apache, "Apache HTTP Server", http://httpd.apache.org/ [online March 2011].

[25] NIST, "The NIST Definition of Cloud Computing (Draft)", http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf [online July 2011]