A Classification-Based Approach to Policy Refinement

Yathiraj B. Udupi
Department of Computer Science
North Carolina State University
Raleigh, NC, 27695-8206, USA
ybudupi@ncsu.edu

Akhil Sahai, Sharad Singhal
UIM, ESSL
Hewlett Packard Laboratories
Palo Alto, CA, 94304, USA
{akhil.sahai, sharad.singhal}@hp.com

***Abstract*— Systems are typically designed based on certain high level goals, such as performance and availability. On the other hand, during operation, usually only low level metrics (e.g., CPU utilization) are measured. The system administrators use their expertise to implicitly map bounds on these metrics such that the high level goals are met. The objective of this research is to create an automated and domain independent approach to derive policy bounds on the low level metrics such that the high level goals are met. These policies may be also be used for monitoring the system for goal assessment purposes. The refinement is carried out using a combination of data classification and test-and-development approaches. An ad hoc system is deployed and a dataset containing values of selected metrics is collected by placing appropriate workloads on the system. The policy bounds are derived by applying classification techniques on this dataset. The classification rules are further refined using statistical distributions to arrive at certain low level rules that are useful for system monitoring and to check the system health when it is deployed and running. We show the validity of our approach for an e-commerce auctioning system (RubiS).**

Keywords- policy refinement, system design, monitoring, SLA, classification.

I. INTRODUCTION

Policies can be specified for system design and to control the system behavior. Currently system administrators are provided with certain high level goals or policies while designing systems. They normally apply their domain knowledge and best-practice rules to design the system. During system operation, however normally the low level metrics are monitored and maintained within bounds, which are also specified as policies. These policies can be very complicated and designers depend on their past experience to arrive at good policies.

In Quartermaster, Singhal et al., visualize policy as the entire set of strict (enforced) constraints and desirable directives that control the behavior of a target entity towards achieving a goal [2]. By formulating policies as constraints on system behavior (as opposed to conditions that arise as a result of system operation), Quartermaster applies the concept of policy to system management in a different manner. It considers the managed system as a set of related entities. Each entity is characterized by a set of attributes and values taken by those attributes, along with actions that were available to the management system to change the attribute values. Policies are defined as constraints that limit the values that the attributes may take in order for the entity (and the system composed from those entities) to behave within an acceptable range.

Quartermaster applies this approach for design and configuration policies [4], uses policies for automatic selection of resources, and for generation of a design specification [1, 5], that could then be deployed. However, it is also important to monitor the system in order to ensure that the system behaves as per the design goals. The monitoring system can help assess compliance with the overall design goals and also to perform a proactive goal assessment.

In this paper, we address the question of generating low level policies that can be used for monitoring the application as part of the design process. The end-user specifies high level goals and during the test-and-development phase, the management system generates the low level policies that are relevant and that may be used to proactively assess the system behavior, as well as used for subsequent design.

A typical SLA may specify various service requirements that include metrics such as availability, response time, throughput, security, and so on. A SLA is composed of a set of Service Level Objectives (SLOs), which specify what constitutes an acceptable service and each SLO is a combination of one or more constraints on component measurements. An example high level SLO defining the performance aspects of an SLA can be stated as - *"service response time < 85 ms between 8 AM and 5PM M-F"*

Any system design involves many components, each potentially affecting the overall performance of the system. Hence any high level constraint potentially relates to certain low level components. By defining policies for the "healthy" ranges on the different low level metrics during system design, it becomes easier for the system operator to proactively avoid potentially costly SLA violations without the need to guess proper ranges for the low level metrics.

The proposed approach provides an automated and domain independent approach to derive these low level policies given the high level SLA goals. The key contribution here is to identify the relevant low level attributes and create policies that define bounds on them based on high level policies. The policy refinement is carried out using a combination of data classification and test-and-

To appear in the Proceedings of Tenth IFIP-IEEE International Symposium on Integrated Network Management (IM 07)

development approach. A system is deployed and data is collected on both the low level metrics and the high level SLOs. Policies are then derived by applying classification techniques on this dataset. The classification rules are further refined using statistical distributions to arrive at certain low level rules that are useful both for checking system health when it is deployed and running, as well as for creating subsequent configurations. The following section describes the policy refinement process in detail.

II. Policy Refinement: A Classification-Based Approach

Policy refinement involves breaking the high level SLA goals into smaller low level policies that feature several system attributes and metrics. A policy specifying a constraint at a high level can be affected by several low level metrics and the challenge is to optimally constrain the low level metrics just enough to satisfy the high level metrics. For example, a high level policy rule can specify that "*service response time < 85 ms"*. Policies can be specified to constrain each of the low level metrics that affect the high level metric. But the actual thresholds for the various low level metrics can vary in different scenarios and for different possible workloads specified. The policy refinement mechanism should identify all the important low level attributes and their policies, and refine them to specify the most optimal constraint thresholds for the relevant metrics. We explain the detailed mechanisms of policy refinement applied in this work below in the following subsections.

A. Test-and-Development Phase: Data Collection and Preprocessing

An ad hoc system configuration is first created on the basis of the given high level SLA goals. We record values for the low level system attributes by placing workloads on the system that are spread around the ranges of the target workload. Next, we preprocess the data for classification. A few metrics can be eliminated if they do not significantly vary for different workloads and can be performed by looking at the distribution statistics of the variables (explained with example results in Section III).

B. Classification Phase

This phase involves running a classification algorithm on the dataset collected above, and deriving the policies that are useful for our purposes. In this dataset, we include a Boolean target variable that is either TRUE or FALSE and is computed by determining if the high level SLA metrics are satisfied or not. As an example of a classification algorithm, decision tree classification algorithm is applicable. Here, the TRUE paths, i.e., all the paths leading to TRUE leaves provide us the required policies on low level metrics.

C. Policy Derivation and Refinement Phase

This phase includes the processes of deriving policies from the output of the classification phase. In the case of a decision tree approach for classification, we derive policies from all the TRUE paths. These TRUE policies are a conjunction of inequalities on various attributes that are picked by the classification phase. A refinement strategy that is applied at this level uses the distribution statistics of the attributes on these TRUE paths. For all attributes in the TRUE tuples (tuples in the dataset that correspond to a TRUE target value), the distributions statistics such *minimum* and *maximum* values are computed. The inequalities of attributes that appear on the TRUE policies are further refined by appending the minimum and maximum values giving definite bounds for those attributes, resulting in the required *refined* TRUE policies. These policies can be used for monitoring system health as well as for designing subsequent configurations. We can provide certain further refined categories of policies by aggregating the different policies generated to arrive at rules on individual attributes. We arrive at two kinds of aggregate ranges for the attributes namely – *Allowable ranges* (performing a *union* operation on their individual ranges picked from each of the TRUE policies), and *Restricted ranges* (performing an *intersection* operation instead).

III. Experimentation and Results

We performed a workload analysis using the ***RUBIS*** [3] testbed – an auction platform implemented as a multi-tier application. For the current purposes, we considered a two-tier application having an *apache* web server tier and a *mysql* database tier. The RUBIS framework offers two kinds of workload mixes – a browsing mix (read-only interactions), and a bidding mix (includes 15% read-write). The workload can be varied by changing the number of clients or by using different mixes. We considered three different high level SLA goal parameters in our experiments namely: No. of Clients, Response Time, and Throughput. For the purposes of the experiments illustrated in the example below, the following SLA goals were considered: *No. of Clients* < = 400, *Response Time* < = 24 ms, and *Throughput* >= 17 req/sec.

We ran experiments with workloads around the above specified high level SLA goal, varying the number of clients from 100 to 900, for four different mixes. The example results shown in the following sections are for an experiment with a bidding mix. We selectively collected the low level system metrics such as *CPU utilization*, *Net_Stats* (packets info), *IO_Stats* (bytes read and written) (a total of 18 metrics) and the *mysqladmin* extended-status metrics (a total of 15 metrics). We can potentially collect many other metrics including some metrics from the apache server and other system metrics.

A. An Example Policy Refinement Scenario

We illustrate the policy derivation and refinement mechanism using an example scenario, where we consider a dataset containing the tuples and the corresponding high level target SLA Boolean values for the first 18 metrics. We ran two different decision tree classification algorithms on this dataset namely, the Random Tree classification (RT), and the J48 algorithm. For example, applying the J48 algorithm resulted in a decision tree as shown in Figure 1.

To appear in the Proceedings of Tenth IFIP-IEEE International Symposium on Integrated Network Management (IM 07)

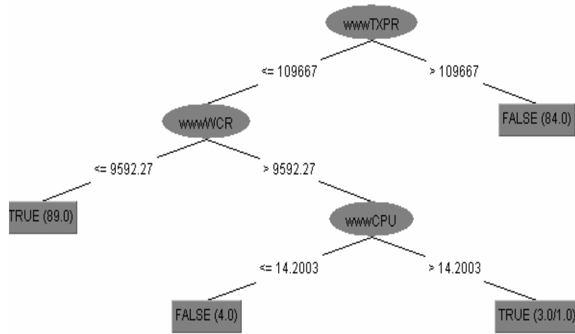

Figure 1 A decision tree generated using J48 algorithm

Applying the RT algorithm results in a tree (not shown in the paper for the limitation of space) with 16 leaves out of which 8 are TRUE leaves and the rest are FALSE leaves. We are interested in the TRUE paths that lead to eight TRUE policies that are conjunctions of inequalities consisting of certain relevant low level system attributes. We compute the distribution statistics such as the *minimum* and the *maximum* values of the attributes that appear in the TRUE policies, and append them in the TRUE policies to get the REFINED TRUE policies.

For example, consider a TRUE path resulting in the following UNREFINED TRUE policy:

"(*dbCPU* >= 7.03) AND (*wwwCPU* < 27.2) AND (*wwwRXPR* >= 8769.72) AND (*dbTXPR* < 9666.55) AND *dbRBR* >= 10477.8) AND (*dbTXPR* >= 3790.93) AND (*wwwWCR* >= 11424.7)"

In the next iteration, we combine attributes that appear more than once in the path and also append their distribution statistics, to get the refined TRUE policy:

"(7.03 <= *dbCPU* <= 12.434) && (3.61269 <= *wwwCPU* < 27.2) && (8769.72 <= *wwwRXPR* <= 183002.0) && (3790.93 <= *dbTXPR* < 9666.55) && (10477.8 <= *dbRBR* <= 154804.0) && (11424.7 <= *wwwWCR* <= 12352.3)"

Allowable and *Restricted* ranges can be arrived for each of the low level metric that appears in the TRUE paths by aggregating each of them. For example, for the metric *dbCPU* (mysql CPU utilization) *allowable* range is [1.08, 12.434], for *wwwCPU* it is [3.61269, 27.2), and so on.

B. Classification Algorithms

We tried three decision tree classification algorithms namely J48, REP, and RT; resulting in different sets of policies. They were evaluated based on the number of TRUE policies selected, the number of attributes appearing in these TRUE policies, the standard classification algorithm evaluation techniques such as number of incorrectly classified instances, error rate and so on. We observed that the RT algorithm provides a larger number of TRUE policies and attributes picked over the J48 and the REP algorithms. While at first glance, this may suggest that the Random Tree algorithm is more suitable, this is not necessarily true without taking into account the error characteristics of the algorithms. For this analysis, we segmented the data into a training set (to generate the decision trees) and a test set (for testing performance of the classifiers). We used a 75-25 split and a 80-20 split of the data into the training and test sets for this analysis. Table 1 shows the results of the analysis.

Random Tree Classifier		
	75%-25% data split	**80%-20% data split**
Number of instances	45	36
Correct classifications	41 (91%)	32 (89%)
Incorrect classifications	4 (9%)	4 (11%)
Kappa statistics	0.82	0.78
J48 Classifier		
	75%-25% data split	**80%-20% data split**
Number of instances	45	36
Correct classifications	44 (98%)	35 (97%)
Incorrect classifications	1 (2%)	1 (3%)
Kappa statistics	0.96	0.94

Table 1 Classifier performance observed for the different classification algorithms

We find that the random tree classifier has a much higher error rate as compared to J48 decision tree classifier. Since J48 classifier provides better classification and lower error rates we find it to be a better classifier for policy refinement because even though it generates lesser number of policies, they will be more reliable.

The effects and relevance of these policies can be further evaluated by actually configuring a system based on the policies generated and testing for their consistency, and will be considered as future work.

C. Distribution Correlation Study

We observe the distribution characteristics of some of the variables by varying the number of clients. These distributions help us in two ways. First we can eliminate some of the attributes that do not vary with workload, and hence are not important for the purposes of the required policies. After eliminating the non-varying attributes we arrived at a list of 33 metrics. The second use of studying the distributions helps us see the patterns of attributes value variation. We observed that attributes that do not correlate with one another also do not appear together in the generated policies. Hence we infer that the refined policies obtained using our approach are better indicators of SLA performance than policies that create thresholds on the distribution

To appear in the Proceedings of Tenth IFIP-IEEE International Symposium on Integrated Network Management (IM 07)

statistics of the low level attributes independently of such correlations.

This correlation study can also be applied to perform a "bottleneck analysis" by running a bigger system for a wider range of workload and will be considered as future work.

IV. Related work

The area of policy refinement has become very important recently with a growing interest in policy-based approaches to systems management. Darimont and Lamsweerde studied formal refinement patterns for a goal-driven requirements elaboration using the KAOS language [6]. Several formal patterns were identified for refining goals into subgoals.

These formal refinement patterns became the basis for many techniques of policy refinement involving temporal logic and event calculus approaches [7, 8]. Several researchers have considered a domain-knowledge based refinement. Bandara and others study refinement on DiffServ QoS management domain [9]. In contrast, the proposed approach presents an automated, generic approach of refinement mainly for performance related goals.

Feather and others integrated the KAOS goal-driven specification methodology and the FLEA runtime event-monitoring system to identify runtime deviations from requirements specifications [10]. Their approach is capable of monitoring the system requirements at runtime to reconcile the requirements and the system's runtime behavior. The proposed approach provides runtime monitoring policies for the system to detect any anomalous behavior and to perform a proactive goal assessment.

Beigi and others study different policy transformation techniques using a combination of clustering and case based reasoning to identify the correct system configurations [11]. The proposed approach is distinct and is to undertake staging around a required SLA and to create workloads to be sent to the system that lead to violation and non-violation of SLAs. All the metrics so collected are then classified using decision tree approaches.

V. Conclusion

We have presented an automated and a non-domain specific approach to derive a set of low level policies from the given high level goals using a combination of a test-and-dev methodology and a classification-based policy derivation and refinement mechanism. It provides a refined set of policies that can be used for generating system design specifications and system monitoring for a proactive goal assessment. We observe that not all low level attributes are selected by the classification algorithm, and hence we conclude that only a few selected attributes correlate with each other and hence form relevant policies for our purposes. This is an improvement over providing just the distribution statistics such *min* and *max* for all the low level attributes. There are certain problems that could be associated with this approach. If the ad hoc system that we begin with is not a big enough system, for certain workload, we can observe that certain low level attributes may become the bottleneck. To avoid such situations we propose to start our test and dev phase with a larger ad hoc system. While using such large systems our approach can be used to detect bottlenecking attributes in addition to deriving the refined TRUE policies. Currently, this approach works best for performance related high level goals; while we believe that this approach is generalizable to other kinds of goals.

As future work, we plan to validate our approach by redesigning the system based on the derived low level policies and testing the results from the new system and if the policies are consistent. Our approach can be potentially beneficial for the root-cause detection and analysis in the case of a failure of the high level goals.

Acknowledgment

We would like to express many thanks to the anonymous reviewers and other colleagues at HP Labs for their valuable comments.

References

[1] Lyle Ramshaw, Akhil Sahai, Jim Saxe, and Sharad Singhal. Cauldron: A Policy-Based Design Tool. In *Proceedings of the 7th IEEE International Workshop on Policies for Distributed Systems and hNetworks (POLICY)*. June 2006. To appear.

[2] S. Singhal et al. Quartermaster: A Resource Utility System. Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management, Nice, France, May, 2005. pp 265—278.

[3] RUBIS Project. URL: http://rubis.objectweb.org/.

[4] Akhil Sahai, Sharad Singhal, Vijay Machiraju, Rajeev Joshi. Automated Generation of Resource Configurations through Policies. In *HP-Labs Report HPL-2004-55.* 2004.

[5] Akhil Sahai, Sharad Singhal, Rajeev Joshi, Vijay Machiraju. Automated Policy-Based Resource Construction in Utility Computing Environments. In *HP-Labs Report HPL-2003-176.* 2003.

[6] Robert Darimont and Axel van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. In *Proceedings of SIGSOFT, CA, USA.* 1996.

[7] Arosha K. Bandara, Emil C. Lupu, Jonathan Moffett and Alessandra Russo. A Goal-based Approach to Policy Refinement. In *Proceedings of POLICY,* 2004.

[8] Javier Rubio-Loyola, Joan Serrat, Marinos Charalambides, Paris Flegkas, George Pavlou, and Alberto L. Lafuente. Using Linear Temporal Model Checking for Goal-oriented Policy Refinement Frameworks. In *Proceedings of POLICY,* 2005.

[9] Arosha Bandara, Emil Lupu, Alessandro Russo, Paris Flegkas, Marinos Charalambides, and George Pavlou. Policy Refinement for DiffServ Quality of Service Management. In *Proceedings of IEEE/IFIP Integrated Management Symposium (IM'2005),* Nice, France, 2005.

[10] M.S. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard. Reconciling System Requirements and Runtime Behavior. In *Proceedings of IWSSD9,* Isobe, Japan, April

[11] M Beigi, S Calo and D verma. "Policy Transformation Techniques in Policy-based Systems Manegement", Policy 2004 workshop, Yorktown, NY, 2004.

To appear in the Proceedings of Tenth IFIP-IEEE International Symposium on Integrated Network Management (IM 07)