A State-Space Approach to SLA based Management

Vibhore Kumar, Karsten Schwan, Subu Iyer†,Yuan Chen†, Akhil Sahai†
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
Hewlett Packard Labs†
1501 Page Mill Road
Palo Alto, CA 94304

Abstract

Large complex systems (such as Enterprise systems) are often composed of several interacting, independent components. In many such systems, although the behavior of the constituent components is well characterized, the behavior that results from interaction between such components is more or less intractable; making it hard for the administrators to efficiently manage the system in conformance with the service level agreements or the SLAs. This paper presents an approach for deriving component-level objectives from system-level objectives or agreements, which if conformed to, imply conformance to the higher-level SLA. Our approach partitions the system's state-space into homogeneous sub-spaces, creates micro-models for such sub-spaces, and then uses such micro-models to translate the higher-level objectives to component-level objectives. We have implemented a system, termed Pranaali*, for evaluating our approach in realistic settings.*

1. Introduction

The need for increased automation, better integration with internal processes and flexibility in interacting with external partners is creating increasingly complex IT systems and applications in today's large enterprises. Examples of such systems include those supporting enterprise websites, inventory or revenue management subsystems [1], and distributed information systems supporting a company's daily operations [8]. Typically, such systems are constructed as collections of components and/or independent software artifacts like web-servers, database systems, and local or remote application services, which interact with each other in many unpredictable patterns when providing the functionality required by the enterprise end users. Further issues include dynamic changes in resource usage and availability, caused by natural system behaviors and failures. Other factors such as resource migration and consolidation also contribute to such behaviors in today's virtualized enterprise. As a result, even when the behavior of constituent components can be well characterized and controlled, it is typically intractable to precisely characterize or limit the dynamic behaviors of the composed enterprise systems and applications. This intractability makes it difficult, if not impossible, for system administrators to efficiently achieve conformance to Service Level Agreements(SLAs) with internal or external enterprise partners. In many cases, systems are over-provisioned to meet SLAs. One such example is the extensive use of over-capacity to guarantee time limits on search requests for the flight search services offered by one of our industry partners, Worldspan [14].

This paper addresses the complexities arising from dynamic component interactions in large-scale enterprise applications by deriving component-level objectives from high level business goals such as SLAs. The solution is based on the assumption that a system's constituent components or subsystems can be individually monitored and managed to the extent needed to attain desired runtime component behaviors. Our approach, as shown in Figure 1 starts by constructing meaningful state-spaces for enterprise system, based on runtime monitored variables. Scalability and manageability are achieved by dynamically partitioning the enormous state-space generated from typical application runs into smaller homogeneous sub-spaces. These homogeneous sub-spaces, which are representative of typical application behavior, are then modeled using probabilistic modeling techniques to create micro-models. For example there may be a micro-model capturing the steady state-behavior of the system, while another micro-model may correspond to the system state in which the back end is being updated by a new batch of updates. These micro-models are then used to derive component-level objectives that characterize a component's contribution to the high level goal. In this way, we dynamically determine the possible set of component level objectives for constituent components, which if conformed to, imply conformance to the higher-level goal. An example is the dynamic determination of CPU shares for a database server in a three tier application for meeting an end-to-end SLA in a given system state sub-space. Determination of the component-level objectives can then be used for designing the overall system or for proactively monitor-

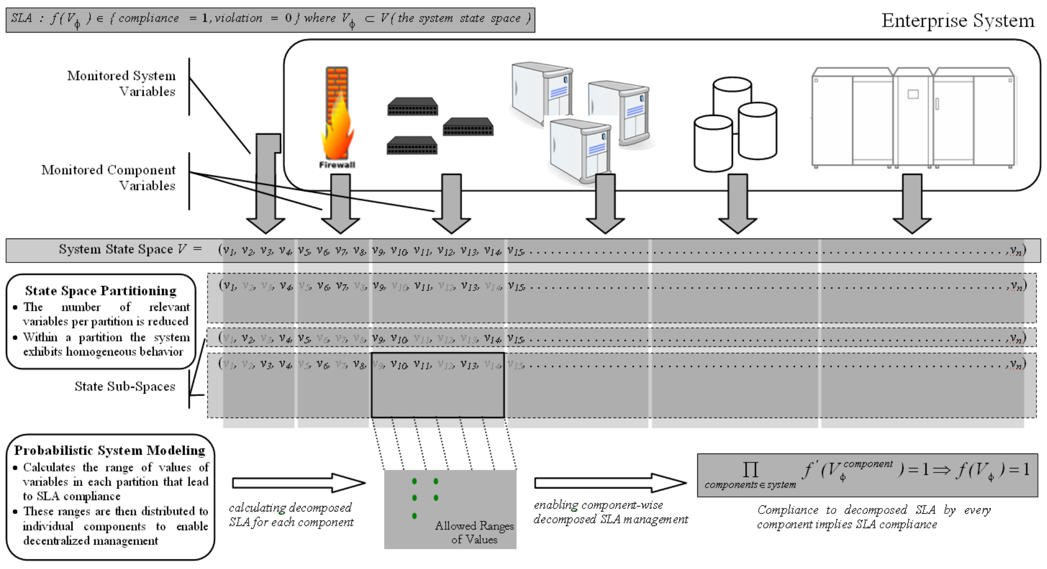

SLA : f(V_ϕ) ∈ {compliance = 1, violation = 0} where V_ϕ ⊂ V (the system state space)
Enterprise System
Monitored System Variables
Monitored Component Variables
Firewall
System State Space V = (v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, ..., vn)
State Space Partitioning
• The number of relevant variables per partition is reduced
• Within a partition the system exhibits homogeneous behavior
State Sub-Spaces
Probabilistic System Modeling
• Calculates the range of values of variables in each partition that lead to SLA compliance
• These ranges are then distributed to individual components to enable decentralized management
calculating decomposed SLA for each component
Allowed Ranges of Values
enabling component-wise decomposed SLA management
∏ components ∈ system f′(V_ϕ^component) = 1 ⇒ f(V_ϕ) = 1
Compliance to decomposed SLA by every component implies SLA compliance

Figure 1. Determining Component-Level Objectives from Service-Level Agreements

ing the system to ensure compliance to a given SLA.

We make the following contributions in this paper. First, we have built an innovative state space partitioning solution that can model the behavior of complex enterprise applications under varying conditions. Second, we use the state spaces to derive component level objectives matching high level goals. Third, we use the information gathered regarding component level objectives in different state spaces to control the application behavior to meet the high level goals.

The following sub-section describes a real-world scenario, provided to us by one industry collaborators, that highlights the need for SLA-decomposition.

1.1 Motivating Example

This work is largely motivated by the needs of one of our industry collaborators, Worldspan [19], a leading provider of information services to the travel industry. The average number of passenger name records in Worldspan's system is around 41.5 million. In the month of March, 2006 alone, their system processed around 9.2 billion messages. To add to the complexity of their enterprise is the 1400 node server farm which searches a frequently updated massive data blob (4GB for domestic and 13GB for international flights) to provide ticket availability and ticket pricing information to their customers, which includes many leading travel portals. One of the critical service level objectives for Worldspan is the `responseTime` of their system. In order to attain this, they typically over-provision their farm to deal with varying workloads. However, given the rate at which the airline industry is expanding, they are predicting that very soon they will need some alternative approaches to ensure compliance to the SLAs. In our collaboration with them, we are trying to develop techniques that automatically, based on previous observations and current operating conditions (like `timeOfDay`, `updateSize`), determine the relationships between controllable component level variables like `cacheRefreshTime`, `allocatedServers`, `searchDepth` and the system-level objectives of `responseTime` and `accuracy`. The idea is to determine ranges for more controllable component-level variables, which if conformed to will ensure compliance to the system level SLA.

2. Solution Overview

In the following sub-sections we formally describe the state-space model that is used by our approach, and provide an outline of the solution.

2.1 System Model & SLA Representation

The following convention is used to describe the SLA and the system state-space model. We use boldface capital letters such as, $\mathbf{V}, \mathbf{S}$ to denote sets, and assignment of values to variables in these sets is denoted by regular capital letters such as $\mathtt{V}_1, \mathtt{S}_1$. Similarly, we use boldface lower case letters such as, $\mathbf{v}_i, \mathbf{o}_i$ to represent variables that occur in the sets, and regular lower case letters such as, $\mathtt{v}_1, \mathtt{o}_1$ denote specific values taken by those variables.

We consider a system whose state can be represented by a set $\mathbf{V}$ of n variables $\{\mathbf{v}_1, ..., \mathbf{v}_n\}$, which are not necessarily independent. Out of these n variables the system's compliance or non-compliance to a SLA can be determined by using only a subset $\mathbf{V}_\phi$ (an example variable in such a subset would be the delay experienced by the users of an enterprise's website) of the state variables in $\mathbf{V}$. Therefore,

$\mathbf{V}_\phi$ is the set of variables of interest as far as the system's operational status is concerned.

A SLA consists of one or more Service Level Objectives (SLOs). We use a tuple $(\mathbf{o}_i, \mathbf{v}^\phi_{\lambda(i)})$ to represent a SLO where $\mathbf{o}_i$ represents the objective specification (say an acceptable operational range), $\lambda(i)$ represents the mapping between the objective $\mathbf{o}_i$ and the variables in $\mathbf{V}_\phi$ and consequently $\mathbf{v}^\phi_{\lambda(i)}$ is the variable over which the objective is defined. The SLA can then be represented as a set $\mathbf{S}_{\mathbf{V}_\phi}$ of m SLOs $\{(\mathbf{o}_1, \mathbf{v}^\phi_{\lambda(1)}), ..., (\mathbf{o}_m, \mathbf{v}^\phi_{\lambda(m)})\}$. We also define a function $\gamma(\mathbf{o}_i, \mathbf{v}^j_\phi)$ that returns `true` if the value represented by $\mathbf{v}^\phi_{\lambda(i)}$ is in conformance with the objective specified as $\mathbf{o}_i$, and is `false` otherwise. The SLA compliance or non-compliance can then be represented as

$$\Gamma(\mathbf{S}_{\mathbf{V}_\phi}) = \prod_{i=1}^{m} \gamma(\mathbf{o}_i, \mathbf{v}^\phi_{\lambda(i)}) \qquad (1)$$

However, in large enterprise systems it is often not possible to deterministically steer the variables in $\mathbf{V}_\phi$ to ensure SLA compliance at all the times. A commonly used approach to facilitate the management of such systems is to simplify and express the SLOs contained in SLA $\mathbf{S}_{\mathbf{V}_\phi}$ in terms of component-specific system variables $\mathbf{V}_\tau \subset \mathbf{V}$ that are more easily controllable (an example of such a variable would be the response time for a well managed database backend, or even the CPU allocation to the middle-tier). We call these variables the *controllable variables* and this simplification results in each SLO $(\mathbf{o}_i, \mathbf{v}^\phi_{\lambda(i)}) \in \mathbf{S}_{\mathbf{V}_\phi}$ being expressed as a set of q distinct SLOs $\{(\mathbf{o}^i_1, \mathbf{v}^\tau_{\lambda'(1)}), ..., (\mathbf{o}^i_q, \mathbf{v}^\tau_{\lambda'(q)})\} \subset \mathbf{S}_{\mathbf{V}_\tau}$, where $\lambda^{'}(i)$ represents the mapping between the new objectives $\mathbf{o}^i_j$ and the variables in $\mathbf{V}_\tau$. For each new simplified SLO, the following equation should hold true

$$\gamma(\mathbf{o}_i, \mathbf{v}^\phi_{\lambda(i)}) = \prod_{j=1}^{q} \gamma(\mathbf{o}^i_j, \mathbf{v}^\tau_{\lambda'(j)}) \qquad (2)$$

Now, if the new simplified SLOs are grouped together by the component to which the variable $\mathbf{v}^\tau_{\lambda'(j)}$ belongs, the resulting groups of SLOs are the objectives for the corresponding components. The component-level objectives are useful for simplifying and decentralizing the task of SLA management.

To put the above discussion in context, such a system model can be readily applied to the example described in Section 1.1. The set of variables monitored by the enterprise constitute the set $\mathbf{V}$, and the SLA is described over the two monitored variables $\{\texttt{responseTime}, \texttt{accuracy}\}$, which constitute the set $\mathbf{V}_\phi$. Since both the members of the set $\mathbf{V}\phi$ cannot be easily controlled, we resort to finding the relation between them and the more easily controllable variables in the set $\mathbf{V}_\tau$ such as $\{\texttt{cacheRefreshTime}, \texttt{searchDepth}, \texttt{allocatedServers}\}$. The component-level objectives essentially determine the allowed ranges of values for the variables in $\mathbf{V}_\tau$ given the SLA $\mathbf{S}_{\mathbf{V}_\phi}$ and the current operational conditions as represented by $\mathbf{V}$.

2.2 Outline of the Solution

Our solution is based on the state-space model described above. The solution requires us to identify the overall system variables $\mathbf{V}$, variables $\mathbf{V}_\phi$ over which the SLA is defined and identify the variables $\mathbf{V}_\tau$ that are more easily controllable. Once such variables are identified and the underlying enterprise system is provisioned to monitor the system variables, our approach for determining sub-SLAs can be put into use for the underlying enterprise system.

Our approach consists of two phases. In the first phase, we monitor the system for a sufficiently large amount of time, encompassing a variety of operational conditions and collect monitoring data. In the second phase, we analyze the data. The resulting data is a collection of several instances $\texttt{I} = \{\texttt{V}_1, ..., \texttt{V}_n\}$ of the state-space set $\mathbf{V}$ (usually $|\texttt{I}| \sim 10^3$). Now, in order to simplify and express the SLA $\mathbf{S}_{\mathbf{V}_\phi}$ in terms of $\mathbf{V}_\tau$ one must use the set $\texttt{I}$ to build the translation function. However, given the scale of the enterprise systems and the fact that such a translation function is intuitively dependent on the prevailing operational conditions, determining the function is not straight-forward. To address this problem, our solution makes use of a novel state-space partitioning algorithm [11] that partitions the state-space into several smaller 'homogeneous' regions that have a reduced number of controllable variables. As a result, we are able to limit both the number of observations and the number of variables from the set $\mathbf{V}_\tau$ (the newly created partitions have several state-space variables that do not vary within the partition) that need to be considered for determining the translation function, contributing to scalability and dynamism. Our solution then makes use of tree augmented naive Bayesian networks or TANs to build the per-partition system models, termed micro-models, which act as the functions that translate the SLOs. The TAN models return the sub-SLA $\mathbf{S}_{\mathbf{V}_\tau}$, along with a probability p that represents the confidence of our TAN model in the returned sub-SLA in achieving the SLA $\mathbf{S}_{\mathbf{V}_\phi}$. The probability p can be compared against a threshold to control the admittance of sub-SLAs. Finally, by building state-space partitions that have lesser cardinality of relevant variables we are also able to limit the overheads imposed by our approach.

3. Algorithms

In this section we describe in detail the various algorithms used by our approach. We start with the system state-space partitioning algorithm and thereafter we describe our algorithm for constructing micro-models and determining component-level objectives.

3.1 System State-Space Partitioning

The system state-space partitioning algorithm aims to achieve two goals

- *Better System Models* - It is often too hard to build a single monolithic model for the entire state space because their behavior is dependent on the prevailing conditions.
- *Limiting the number of Controllable variables* - Creating partitions with limited number of controllable variables that can be modified makes the problem of finding sub SLAs more tractable.

3.1.1 The State-Space Partitioning Algorithm

A system state $\mathtt{V}_i$ can be defined as the binding of appropriate values to the variables contained in the set $\mathbf{V}$. Let, $\mathtt{I} = \{\mathtt{V}_1, ..., \mathtt{V}_n\}$ be the set of many such observed system states contained in the unpartitioned system state-space. The partitioning algorithm aims to partition many such observed system states into smaller sets to achieve the objectives mentioned in the previous section. A partition inherits the sets $\mathbf{V}$ and $\mathbf{V}_\phi$ from the unpartitioned system state-space but the set of variables in $\mathbf{V}_\tau$ can vary between the partitions. We define the range ρ for any discrete or continuous state variable $\mathbf{v} \in \mathbf{V}$ as follows -

$$\rho(\mathbf{v}, \mathtt{I}) = \begin{cases} max(\mathbf{v}, \mathtt{I}) - min(\mathbf{v}, \mathtt{I}) & \text{continuous} \\ unique(\mathbf{v}, \mathtt{I}) & \text{discrete} \end{cases}$$

where $unique(\mathbf{v}, \mathtt{I})$ implies the number of discrete unique values the variable $\mathbf{v}$ takes in the set $\mathtt{I}$. The normalized distance ϕ between any two instances $\mathtt{v}_1, \mathtt{v}_2$ of the state variable $\mathbf{v}$ is defined as follows, given $\rho(\mathbf{v}, \mathtt{I}) > 0$.

$$\phi(\mathtt{v}_1, \mathtt{v}_2) = \begin{cases} \frac{(\mathtt{v}_1 - \mathtt{v}_2)}{\rho(\mathbf{v}, \mathtt{I})} & \text{continuous} \\ 0 & \text{if } \mathtt{v}_1 = \mathtt{v}_2 \text{ discrete} \\ \frac{1}{\rho(\mathbf{v}, \mathtt{I})} & \text{if } \mathtt{v}_1 \neq \mathtt{v}_2 \text{ discrete} \end{cases}$$

We use the operators defined above to define an operator $\Phi_\mathbf{R}$ that calculates the normalized distance between any two instances $\mathtt{s}_1, \mathtt{s}_2$ of the set $\mathbf{V}$ along the dimensions $\mathbf{R}$, where $\mathbf{R} \subseteq \mathbf{V}$. Finally, the partitioning distance υ between any two system states is defined as follows.

$$\upsilon(\mathtt{s}_1, \mathtt{s}_2) = \eta \times \Phi_\mathbf{V}(\mathtt{s}_1, \mathtt{s}_2) + \mu \times \Phi_{\mathbf{V}_\tau}(\mathtt{s}_1, \mathtt{s}_2) \qquad (3)$$

where, η and μ can take values from the range [0,1] and these are used to configure υ for the two objectives mentioned in the previous section. To evaluate if we need to partition a given system state-space $\mathtt{I}$, we try find a subset $\mathbf{V}'_\tau$ of $\mathbf{V}_\tau$ such that

$$\sum_{\forall \mathtt{s}_i, \mathtt{s}_j \in \mathtt{I}} \delta_{\mathbf{V}_\tau - \mathbf{V}'_\tau}(\mathtt{s}_i, \mathtt{s}_j) \leq \Delta_{max} \qquad (4)$$

$$|\mathbf{V}'_\tau| \leq \varphi \qquad (5)$$

where Δ_{max} is a user defined parameter that represents the maximum allowed representation error for the controllable variables and φ represents the maximum number of allowed controllable variables per partition. We employ a greedy approach for finding $\mathbf{V}'_\tau$, i.e. we add the member of $\mathbf{V}_\tau$ to $\mathbf{V}'_\tau$ which causes the greatest reduction in the L.H.S. of the equation 4. We repeat the above process until the L.H.S. becomes lesser than Δ_{max}, at this point we look at the cardinality of the set $\mathbf{V}'_\tau$ - if the cardinality is less than φ we do not partition the system state-space, otherwise we proceed to partition the system state-space. The $\mathbf{V}'_\tau$ so determined becomes the $\mathbf{V}_\tau$ for the partition. We start by finding a pair of states $\mathtt{s}_1$ and $\mathtt{s}_2$ from the set of all such pairs contained in the set $\mathtt{I}$ such that $\upsilon(\mathtt{s}_1, \mathtt{s}_2)$ is maximized. The pair $\mathtt{s}_1$ and $\mathtt{s}_2$ acts as the seed for the two new system state sub-spaces $\mathtt{I}_1$ and $\mathtt{I}_2$ that will be created. We then iterate through the remaining operational states in the set $\mathtt{I}$, adding the operational state $\mathtt{s}_i$ to $\mathtt{I}_1$ if $\Phi_\mathbf{V}(\mathtt{s}_i, \mathtt{s}_1) \leq \delta_\mathbf{V}(\mathtt{s}_i, \mathtt{s}_2)$, otherwise $\mathtt{s}_i$ is added to the partition $\mathtt{I}$. One can alternatively use the centroid of existing operational states in the evolving partitions to determine the membership. Once the two new partitions $\mathtt{I}_1$ and $\mathtt{I}_2$ have been created, we find the set $\mathbf{V}_\tau$ for them using the greedy approach described above. If the criteria defined by Δ_{max} and φ is not met by any partition then we repeat the above scheme for that partition.

Once the system state-space has been partitioned we build a system *micro-model* corresponding to each partitioned sub-space. A system model in our framework consists of several micro-models each one of which models a sub-space of possible system states. The micro-model to be applied is determined based on the current system state. Since, we attempt to model only a small partition of the entire system state-space at a time we are able to build models even for systems with a very high number of variables. This makes our approach highly scalable. A similar approach was presented in [21], which made use of an ensemble of probabilistic models to detect SLO violations, and was shown to perform significantly better than the approach which used a single monolithic model. The approach works by adding new models when the existing models do not accurately capture the current system behavior.

3.2 Constructing Micro-Models

We want to create *micro-models* such that they can predict the range of acceptable values for the variables in $\mathbf{V}_\tau$ given the values for the variables in $\mathbf{V} - \mathbf{V}_\tau$. To find such ranges we resort to making use of probabilistic modeling techniques. We use a variant of the Bayesian network [9] called the Tree Augmented Naive Bayes [6] or TANs to probabilistically model the system state-space. A Bayesian network is represented as an acyclic graph whose vertices encode random variables and the edges represent statistical dependence relations among the variables and local probability distributions for each variable given values of its parents. The main advantage of using a Bayesian network (or one of its variants) is that their representation provides and easy way to inspect the relationships between the involved variables. This allows an expert to embed her knowledge or the common wisdom into the self-management framework by proposing an initial model, which can be further refined using learning techniques. Furthermore, by simple inspection an expert can single out any faults in the learnt system model. Our choice for making use of TANs was driven by the fact that unrestricted forms of Bayesian network are computationally very costly to build as they need to evaluate all the dependencies amongst the set of random variables. A TAN, on the other hand allows only a tree structured dependence amongst the set of random variables (other than the class variable) and is therefore cheaper to build and has been shown to perform almost as well as the unrestricted version. A TAN model when used as a classifier is able to determine the following probability

$$p = Pr(\mathbf{c}|\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n) \tag{6}$$

for the set $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n, \mathbf{c}\}$, from a given training set. The variable $\mathbf{c}$ assumes a special status in this equation and is called the class variable and the other variables are called the attributes.

To create the micro-model for our partitions, we designate the output α from the system status function $\Gamma(\mathbf{S}_{\mathbf{V}_\phi})$ (refer equation 1) as the class variable and the variables in the set $\mathbf{V}$ are designated as the attributes. The resulting micro-model is able to determine the following probability

$$p = Pr(\alpha|\mathbf{V}) \tag{7}$$

the above equation determines the probability of SLA compliance or violation given the system state. To determine the suitable range of values that the variables in $\mathbf{V}_\tau$ can take while ensuring SLA compliance, we make use of the following procedure. If a SLA violation is detected or if the current system state, say $\mathtt{V}_{now}$ requires a change of the micro-model to be used, we recalculate our sub-SLAs. We retrieve the set $\mathbf{V}_\tau$ for the micro-model under consideration and generate an exhaustive enumeration of the possible values $\{\mathtt{V}_1^\tau, ..., \mathtt{V}_n^\tau\}$ that the controllable variables can take. We then generate a set of possible system states $\{\mathtt{V}_1, ..., \mathtt{V}_n\}$ by substituting into $\mathtt{V}_{now}$ the values for the controllable variables from the set constructed earlier. We set the value of α to SLA-compliance and evaluate the probability p for each possible system state. The ones with probability p greater than κ (which is a user defined confidence-threshold) are recorded in set $\mathbf{N}$ for determining the sub-SLAs.

3.3 Component Level Objectives

The problem of finding healthy ranges for sub-components requires us to segregate the controllable variables according to sub-components and find range of values for each controllable variable such that they are independent of the value taken by other controllable variables. The per-component controllable variables along with the respective ranges constitute the sub-SLA for the component. However, finding the allowed independent range of values from the set $\mathbf{N}$ is not straight-forward. For $|\mathbf{N}| = 1$, this problem is trivial and each controllable variable is assigned the values that appears in the solution $\mathtt{V}_1$. For larger values of $|\mathbf{N}|$, the solution to finding appropriate ranges is based on finding a clique [7]. All distinct values taken by the controllable variables in the set $\mathbf{N}$ are denoted as vertices of a graph, all such vertices which belong to the same variable are connected so as to form a clique between them. We also form cliques between the set of controllable variable values corresponding to each $\mathtt{V}_i \in \mathbf{N}$. In the resulting graph, we find all the possible cliques and choose the clique which maximizes the product $\prod_{j=1}^{|\mathbf{V}_\tau|} n_j$, where n_j is the number of values for the $\mathbf{v}_j^\tau$ that appear in the clique. The set of values that appear corresponding to a variable in the chosen clique constitute the acceptable range of values for the component variable in question.

4. Implementation: *Pranaali*

We have implemented our approach in a system termed *Pranaali*[1]. Pranaali is implemented in C++ and it relies on jBNC [10] (a Java based open-source implementation of Bayesian Network) for constructing the TANs. The system during the training phase takes as input a set of data points which contains monitoring information observed from the system under consideration, service level objectives, and additional metadata including the type and name of the monitored variables and details regarding the controllable variables. Every state in the input data set is augmented with SLA conformance/violation information based on the suppplied SLOs. The user also needs to provide values for the partitioning parameters η, μ, Δ_{max} and φ as defined in Section 3.1.1. The module partitions the training data set and after discretization and conversion to C4.5 format submits it to the jBNC Classifier for generating TANs. Each TAN is then associated with a centroid from the data partition that was used for its construction. This marks the end of the training phase. The real-time component of the module provides regular updates about the monitored system regarding its state. If a SLA violation is detected the module recalculates the value ranges for various controllable variables and passes them on to respective components.

5. Experimental Evaluation

Our goal was to study the suitability of our approach in determining more tractable component-level objectives for large enterprise scale systems. In this section, we present our findings based on the experiments conducted using

[1]Pranaali is a Sanskrit word meaning *Mechanism*

the well-known RUBiS [15] application running within the Xen [2] virtual machine environment. Our approach, for instance, was able to recognize an overload at the backend database server and as a result were able recalculate a new set of per component thresholds to maintain conformance to the overall SLA. We start with a description of our RUBiS/Xen testbed, which is followed by a brief description of the workload. We present our experimental results starting from Section 5.3.

5.1 Experimental Setup

The experimental setup consisted of 5 Emulab [5] nodes, each with a 2800MHz Pentium-4 processor, 512MB RAM and running the 2.6.18-4-xen-686 Linux kernel. The virtual machine was started with the Xen SEDF scheduler running in non work-conserving mode. The RUBiS instance consisted of an Apache server, two load-balanced Tomcat servers and an instance of the MySQL server; each hosted on a different machine. The RUBiS client along with the monitoring program (IFLOW [12]) was configured to run on the one remaining node. The nodes were connected by Gigabit Ethernet links.

The 4 nodes running the RUBiS components were instrumented to monitor the `vmstat` records and the VM statistics; the Apache server status, and the Tomcat and the load balancer status were monitored and reported using the appropriate plugins (`mod_status`, `mod_jk`), `mysqladmin` was used to track the status of the MySQL Server. The response-time and throughput metrics were collected at the client node, which also hosted the IFLOW agent for collecting the monitored data. There were 137 monitored variables, collected every 5 seconds, which included quantities like CPU and memory allocation to virtual machines, load-balancing factor, bytes transferred, requests processed, etc.

5.2 Workload

The training data sets were generated using a synthetic workload applied to the RUBiS instance, and during the duration of the experiment an automated script was responsible for modifying the environment parameters like allocated CPU, allocated Memory, request-rate and external load on the Middle Tier and the Database Tier. We collected 5 such training data sets, each for a duration of approximately 1 hour. We also collected 4 more data sets, each for a duration of 10 minutes under variety of different perturbations, which were to serve as test data sets.

We used the EPA-HTTP web traffic trace from the LBL Repository [13] when determining component-level objectives under traffic spikes, varying transaction-mix and varying external load at the database tier. The EPA-HTTP web trace contains traffic for an entire day. However, for the purpose of experimentation we scaled down the trace to run in 1 hour while preserving the shape of the workload. We called the trace EPA-HTTP-ONE, shown in Figure 3.

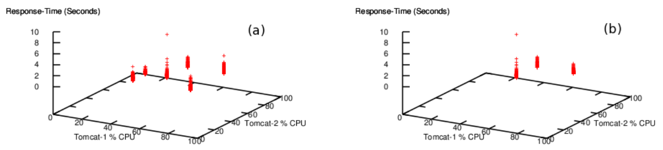


Figure 2. Response-time variation with change in CPU allocation in unpartitioned (a) and the partitioned (b) data-set. Observe the more intuitive variation of response-time in (b) as compared to (a). Corroborates our claim of sub-space homogeneity.

5.3 Results

In the following section we report the microbenchmark results using the Pranaali system, followed by experiments that evaluate the suitability of our approach in deriving the component-level objectives.

5.3.1 Microbenchmarks

The first experiment was focused on evaluating the usefulness of our partitioning scheme in clustering together a set of homogeneous states. We used TANs generated from partitioned and unpartitioned training data sets for the purpose of classifying the states of the test data set as the ones causing SLA violation or conformance. In the results reported in Table 1, we compare the accuracy of classification. Clearly, the TANs generated from the partitioned data set are significantly more accurate at classifying the states. This can be attributed to the partitioning scheme which aims to cluster together a set of homogeneous states that can be modeled more easily as compared to the entire training data set. The experiments were performed using the following parameters $\eta = 1.0, \mu = 0.2, \varphi = 5$. In Figure 2, we show the actual plot of data along 3 dimensions, comparing the entire training data set to the partitioned data set.

In the second experiment we analyzed the effect of setting up a threshold for the classification probability. A classification was termed successful only if the TAN model returned that classification with a probability higher than the threshold. As shown in Figure 4, with increasing value of threshold probability the accuracy of classification increased. This observation is useful for fine-tuning the correctness of the component-level objectives; from stringent (a very high value for parameter κ) to relaxed. As a result of setting up thresholds for classification probability a significant number of test data states were left unclassified, and such numbers increased with an increase in the threshold. As many as 35% of the states remained unclassified for a threshold value of 0.95. We believe that the number of these unclassified states can be significantly reduced by

Table 1. Effect of partitioning on classification accuracy

	Original	Partition		
Δ_{max}	-	0.4	0.3	0.2
Accuracy %	72.0	77.8	80.1	81.3
Partitions	-	3	4	6

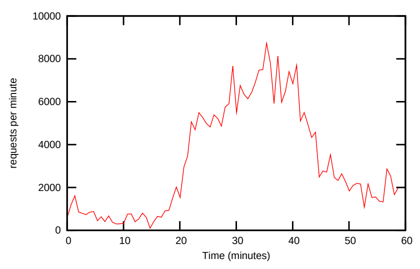


Figure 3. EPA-HTTP-ONE workload: Requests per minute vs time

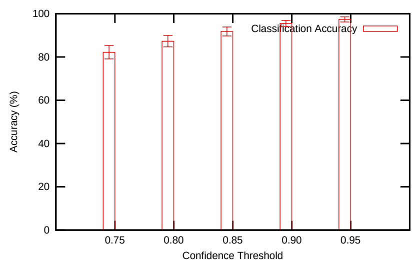


Figure 4. Variation of classification accuracy with increasing κ

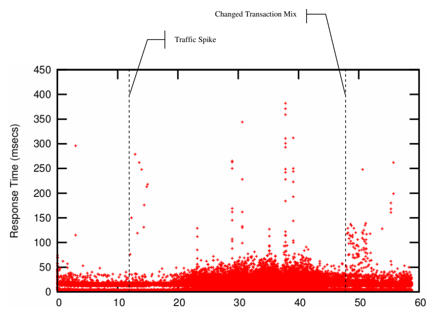


Figure 5. Response-time for workload variations without Pranaali

providing a more comprehensive training data set. However, we leave this analysis as part of our future work.

The remaining experiments use the training data set from Section 5.2 to construct the system models for deriving component-level objectives. The high-level SLA for these experiments was to maintain a response-time of less than 75 milliseconds. The set $\mathbf{V}_\tau$ for these experiments consisted of 8 variables which included the CPU and Memory allocated to the 4 VMs.

5.3.2 Workload Variations

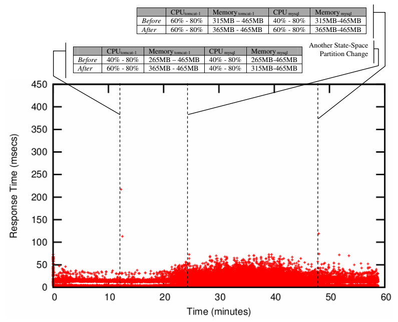
We wanted to observe and evaluate the response of our approach to variations in the workload characteristics. Specifically, we observed the response of our system to sudden spike in traffic and its response to change in workload transaction mix. To conduct the experiment, we modified the EPA-HTTP-ONE trace to include a synthetic traffic spike at the 12th minute, which lasted for 3 minutes. Furthermore, we modified the RUBiS client to increase the ratio of request for database intensive pages for 4 minutes, starting at the 48th minute. We ran the experiment twice - without and with the Pranaali system in place.The Pranaali system was able to determine at runtime the CPU and memory allocation ranges for the VMs that were hosting the RUBiS components. All the VMs at the start of both the experiments were configured to use 50% CPU and 365MB out of the total 465MB of the available memory. Figure 5 shows the variation in response time without the Pranaali system. The Pranaali system was able detect the SLA violations and the migration of the RUBiS system to new state-space partitions (like the one characterized by high traffic) and was therefore able to suggest new component-level objectives and avoid SLA violations. The results and the new component-level objectives for relevant variables are shown in Figure 6.

5.3.3 Variation in External Load

In this experiment we used the Pranaali system to automatically detect and provision the resources to counter the delay introduced by application of external load to the database (like database updates or analytic queries against production database). Our modeling techniques were able to detect the migration of system into a different partition and the component-level objectives, so determined, were able to achieve SLA conformance. The results are shown in Figure 7 and 8, the external database load (a series of complex DB Queries) was applied at the 52nd minute and lasted for 2 minutes. Clearly, with the Pranaali system in place, we were able to avoid SLA violation that occurred in the system without Pranaali. The Pranaali system in this case had automatically increased the CPU and Memory allocated to the VM hosting the database, the new component-level ranges are also shown in the figure.

6. Related Work

Automated diagnosis of performance problems and its application to self-healing systems is a topic of considerable research interest. A number of approaches have been proposed in this domain including use of analytical models, machine learning techniques and feed-back control systems. Proactive management of Service Level Agreements is also a topic of current research. Notable efforts in applying analytical models include the work on using performance models to guide resource provisioning and capacity planning [18, 20, 4]. However, these efforts, mainly focused on multi-tier web applications, rely on making use of execution models for the underlying components to arrive at per-tier allocation decisions. Reliance on such models, typically attained with component profiling methods, makes it difficult to extend these approaches to other enterprise systems that can benefit from decomposition. Further, the performance models being used are typically based on the steady-state behavior of constituent components and systems, which makes it impossible to use them to characterize interesting or important conditions caused by system dynamics. Y. Udupi et. al [17] propose a classification based approach to policy refinement. To the best of our knowledge, our work is the first that can predict application behavior under normal conditions as well as under stress. Further more, most of the existing approaches only deal with a small subset of system and application level metrics. On



	$\text{CPU}_{\text{tomcat-1}}$	$\text{Memory}_{\text{tomcat-1}}$	$\text{CPU}_{\text{mysql}}$	$\text{Memory}_{\text{mysql}}$
Before	60% - 80%	315MB – 465MB	40% - 80%	315MB-465MB
After	60% - 80%	365MB - 465MB	60% - 80%	365MB-465MB

	$\text{CPU}_{\text{tomcat-1}}$	$\text{Memory}_{\text{tomcat-1}}$	$\text{CPU}_{\text{mysql}}$	$\text{Memory}_{\text{mysql}}$
Before	40% - 80%	265MB – 465MB	40% - 80%	265MB-465MB
After	60% - 80%	365MB - 465MB	40% - 80%	315MB-465MB

Figure 6. Response-time for workload variation with Pranaali

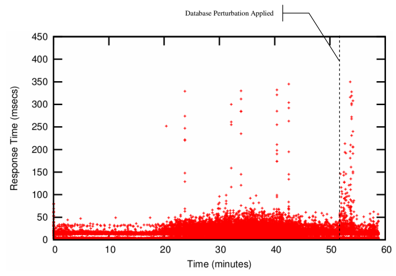
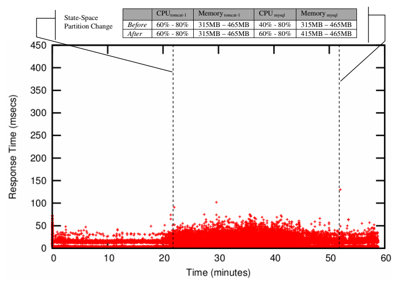


Figure 7. Response-time for external DB load without Pranaali



	$\text{CPU}_{\text{tomcat-1}}$	$\text{Memory}_{\text{tomcat-1}}$	$\text{CPU}_{\text{mysql}}$	$\text{Memory}_{\text{mysql}}$
Before	60% - 80%	315MB – 465MB	40% - 80%	315MB – 465MB
After	60% - 80%	315MB – 465MB	60% - 80%	415MB – 465MB

Figure 8. Response-time for external DB load with Pranaali

the contrary, our approach allows us to model many metrics simultaneously. In the area of statistical and machine learning research, Chen et al. [3] analyzes run-time execution paths of complex distributed applications to automatically detect failures by identifying statistically abnormal paths; faulty paths can then aid a human analyst in diagnosing the underlying cause. Similarly, the SLIC project [16] uses statistical techniques including Bayesian networks to automatically extract signatures for root cause analysis.

7. Conclusions & Future Work

In this paper we described an approach for deriving for component-level objectives from system level objectives or agreements. The approach offers scalability and better manageability by partitioning the system state-space into more homogeneous regions which can be more easily modeled as compared to the entire state-space. We made use of probabilistic modeling techniques to dynamically infer the relationship between the variables of interest and the controllable variables, and used the models, so developed, to derive component-level objectives. As part of the future work we are trying to evaluate the usefulness of micro-models at limiting the monitoring overhead and making use of dynamic Bayesian networks to incorporate time into our models.

References

[1] S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham. E2eprof: Automated end-to-end performance management for enterprise systems. In *DSN*, 2007.

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

[3] M. Chen, E. Kiciman, E. Fratkin, E. Brewer, and A. Fox. Pinpoint: Problem determination in large, dynamic, internet services. In *DSN*, 2002.

[4] Y. Chen et al. SLA decomposition: Translating service level objectives to system level thresholds. In *ICAC*, 2007.

[5] Emulab - network emulation testbed home. `http://www.netlab.cc.gatech.edu/`, as retrieved on 09/15/2007.

[6] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3), 1997.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[8] A. Gavrilovska, K. Schwan, and V. Oleson. A practical approach for zero' downtime in an operational information system. In *ICDCS*, 2002.

[9] D. Heckerman. A tutorial on learning with bayesian networks. Technical report, Microsoft Research, 1995.

[10] jBNC: Bayesian network classifier toolbox. `http://jbnc.sourceforge.net/`, as retrieved on 09/15/2007.

[11] V. Kumar, B. F. Cooper, G. Eisenhauer, and K. Schwan. iManage: Policy-driven self-management for enterprise-scale systems. In *Middleware*, 2007.

[12] V. Kumar et al. Implementing diverse messaging models with self-managing properties using iflow. In *ICAC*, 2006.

[13] EPA-HTTP - a day of HTTP logs from the EPA WWW server. `http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html`, as retrieved on 09/15/2007.

[14] M. Mansour, K. Schwan, and S. A. Aziz. I-queue: Smart queues for service management. In *ICSOC*, 2006.

[15] RUBiS - home page. `http://rubis.objectweb.org/`, as retrieved on 09/15/2007.

[16] SLIC. `http://www.hpl.hp.com/research/slic/`, as retrieved on 09/15/2007.

[17] Y. Udupi, A. Sahai, and S. Singhal. A classification-based. approach to policy refinement. In *IM*, 2007.

[18] B. Urgaonkar et al. Dynamic provisioning of multi-tier internet applications. In *ICAC*, 2005.

[19] Worldspan by Travelport. `http://www.worldpsan.com`, as retrieved on 09/15/2007.

[20] A. Zhang, P. Santos, D. Beyer, and H. Tang. Optimal server resource allocation using an open queueing network model of response time. In *HP Labs Technical Report, HPL-2002-301*, 2002.

[21] S. Zhang, I. Cohen, J. Symons, and A. Fox. Ensembles of models for automated diagnosis of system performance problems. In *DSN*, 2005.