MONITORMINING: Creating Domain Knowledge for System Automation using a Gray-box Approach

Sandeep Uttamchandani
IBM Almaden Research Center
San Jose, CA, USA
sandeepu@us.ibm.com

Xiaoxin Yin
University of Illinois at Urbana Champaign
Urbana, IL, USA
xyin1@uiuc.edu

John Palmer
IBM Almaden Research Center
San Jose, CA, USA
jdp@us.ibm.com

Gul Agha
University of Illinois at Urbana Champaign
Urbana, IL, USA
agha@uiuc.edu

Abstract

The effectiveness of automated system management is dependent on the domain-specific information that is encoded within the management framework. Existing approaches for defining the domain knowledge are categorized into white-box and black-box approaches, each of which has limitations. White-box approaches define detailed formulas for system behavior, and are limited by excessive complexity and brittleness of the information. On the other hand, black-box techniques gather domain knowledge by monitoring the system; they are error-prone and require an infeasible number of iterations to converge in real-world systems.

MONITORMINING is a gray-box approach for creating domain knowledge in automated system management; it combines simple designer-defined specifications with the information gathered using machine learning. The designer specifications enumerate input parameters for the system behavior functions, while regression techniques (such as *Neural Networks, Support Vector Machines*) are used to derive the mathematical function that relates these parameters. These functions are constantly refined at run-time, by periodically invoking regression on the newly monitored data. MONITORMINING has the advantage of reduced complexity of the designer specifications, better accuracy of regression functions due to a reduced parameter set, and self-evolving with the changes in the system. Our initial experimental results of applying MONITORMINING are quite promising.

Keywords

Automated system management, Domain knowledge, Gray-box techniques, Autonomic Computing, Models

1. Introduction

System management today is driven by human administrators that continuously monitor the system, analyze its behavior, and take corrective actions to ensure that it converges towards desired threshold goals for performance, availability, security. With the cost of sys-

tem management becoming a significant percentage of the *Total Cost of Ownership* [13], self-management has become a necessity [7]. The idea of self-management is not a new one – Expert Systems [3] have been used to automate various human-intensive processes such as disease diagnosis [4], fault analysis [16], and so on. An important lesson learnt by deploying Expert Systems is summarized by the Knowledge Principle [9]: "*The power of AI programs (i.e. expert systems) to perform at high levels of competence is primarily a function of the program's knowledge of its task domain, and not of the program's reasoning processes.*" In simple words, the effectiveness of an automated system is dependent on the *richness* of domain-specific knowledge encoded within the management framework.

The focus of this paper is an approach for creating the domain knowledge required for automated system management. Existing techniques for encoding domain knowledge fall into two extremities:

- *White-box* approaches where the system-designer defines detailed formulas [10, 12] or rules [8, 19, 26] to describe the characteristics of the system. These techniques are limited by excessive complexity, and brittleness of the domain knowledge to ongoing changes in the system.
- *Black-box* approaches where the system acquires domain-specific knowledge by monitoring the system behavior and using machine learning techniques [18, 27]. This approach is error-prone, and requires an infeasible number of iterations for converging in real-world multi-parameter systems.

MONITORMINING is a *gray box approach* for building domain knowledge; it uses a combination of simple system-designer specifications with the information gathered using machine learning. The domain knowledge consists of mathematical functions (referred to as *models*). For each of these models, the designer specifications list the domain-specific input parameters, while regression techniques such as Neural Networks [21], Support Vector Machines [5] are used to deduce the exact mathematical function that correlates these parameters. These functions are continuously refined at run-time by periodically applying regression to the newly monitored data. The advantages of MONITORMINING are simplistic designer-defined specifications, non-brittleness, and faster convergence of the deduced functions by limiting the number of parameters considered for regression.

This paper addresses the representation, creation, and evolution of domain knowledge for automated system management. To make the discussion concrete, we describe the details in the context of automated storage management. The key contributions of this paper are:

- A model-based represention of the domain knowledge for automated storage management.
- A methodology to create and evolve the domain knowledge using a gray-box approach.

We also describe an off-the-shelf technique to cater incomplete designer specifications. Finally, we describe the initial experimental results of using MONITORMINING for creating the domain knowledge for a real-world storage system setup.

The outline of the paper is as follows: Section 2 gives the big-picture of automated management in storage systems. Section 3 describes the representation of the domain knowledge. Section 4 describes the gray-box approach for creating the domain knowledge, alongwith details for evolution of the models and handling incomplete designer

specifications. Section 5 describes the initial experimental results. Section 6 covers the related work, followed by conclusion and future work.

2. Background: Automated Management in Storage Systems

Table 1 defines the management terminology used in the rest of the paper.

Term	Description
Service Level Objectives (SLO)	Defines the desired threshold values for the system's performance, reliability, security, availability. The current iteration of MONITORMINING supports performance SLOs only. A performance SLO is of the form: throughput-threshold@latency-threshold i.e. a request-rate below the throughput-threshold should have the average response-time below the latency-threshold.
Workload	There are multiple applications (such as web-server, e-mail) running on the system; the I/O requests generated by each application are referred to as workload. *Workload characteristics* refers to I/O access characteristics namely request rate, average request size, read/write ratio, sequential/random access pattern. The data accessed by the workload is referred to as the *data-set*
Corrective Actions	Changes the behavior of the system so that it converges towards administrator-defined goals. Actions are categorized into: *Short-term actions* that tune the system without physical movement of data, and can take into effect immediately e.g. data-prefetching, throttling. *Long-term actions* generally involve physical movement of data, and have a non-negligible transient cost e.g. data-migration, replication.
Invocation path	The series of components in the system that are used for servicing the workload requests.

Table 1 System Management Terminology

Figure 1 shows a production storage system with multiple applications (such as e-mail, database, web-server) using the storage resources. Each application can have different access characteristics, priorities, and SLOs. The task of a storage virtualization engine (such as SAN.FS [20], SAN Volume Controller [11]) is to map the application-data to the available storage resources. A one-time mapping of data to resources is not optimal and not feasible in most scenarios because of: Incomplete initial information of the access characteristics, component failures and load surges that occur at run-time. Thus there is a need for automated system management to continuously observe, analyze, and act by invoking corrective actions such as throttling, pre-fetching, data replication, etc.

A management framework invokes corrective actions to minimize the effect of system events such as workload variations, component failures, and load surges, on the SLOs of workloads running in the system. Building the action selection function is non-trivial as it

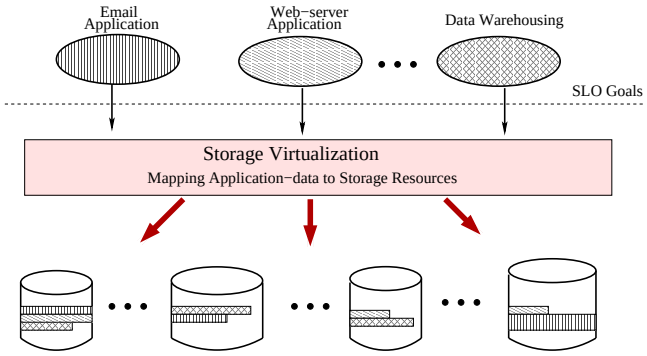


Figure 1: Mapping the data-sets of the workloads to the available resources

needs to take into account: 1) The cost-benefit of actions that is dependent on the system state and the parameters values used for action invocation; 2) The workload trends and load pattern on the system that might make a few actions infeasible in a given state; thus there is no universal "rule-of-thumb" for invoking actions. 3) There are a large number of possible system states (it is not possible to write policy rules for selecting actions in every possible system state), and the need to adapt to changes in the system such as addition of new components, new application workloads.

A model-based approach for automated system management makes decisions using prediction functions for the behavior of the system for given load characteristics and configuration parameters [22, 25]. The key challenges with this approach are the representation of domain-specific details as prediction functions or models, creation of these models, and using the models at run-time to decide the corrective actions. MONITORMINING is a framework for the representation and creation of self-evolving models.

3. Representation of the Domain Knowledge

The domain knowledge consists of mathematical functions (i.e. models) that capture the system details required for deciding corrective actions at run-time. In the case of storage systems, the domain knowledge consists of models for: 1) The response time of the component as a function of incoming load at the component *(component model)*; 2) The load on the individual components in the workload's invocation path *(workload model)*; 3) The cost and benefit of action invocation *(action model)*. This section covers the details of each of these models

3.1 Component Model

A component model predicts the response time of the component as a function of the incoming load at the component. The component's response time is dependent on the service-time and wait-time incurred by the workload stream. The service time is a function of the workload characteristics, and is of the form:
$Stime_{Wi} = c(req\_size, req\_rate, rw\_ratio, random/sequential, cache\_hit\_rate...)$

The wait time represents the time spent in the queue due to interleaving with other workload streams arriving at the component. MONITORMINING approximates this non-trivial computation by estimating the wait time for each individual stream as per a multi-class queueing model [17]. The resultant response time is approximated as follows. The *utilization U* of the component is:

$$Utilization\,(U) \;=\; \sum_{i=1}^{n} \lambda_{Wi} Stime_{Wi}$$

where λ_{Wi} is the arrival rate and $Stime_{Wi}$ is the service-time for the workload stream Wi. The resultant response time $Rtime$ of the component for the workload stream Wi is represented as:

$$Rtime_{Wi} \;=\; \frac{Stime_{Wi}}{1-U}$$

3.2 Workload models

Representation and creation of workload models has been an active area of research [6]. In MONITORMINING, workload models predict the load on each component as a function of the request rate that each workload injects into the system. For example, to predict the rate of requests at component i originated by workload j:
Component_load$_{i,j}$ = $w_{i,j}(workload\_request\_rate_j)$
In real scenarios, function $w_{i,j}$ changes continuously as workload j changes or other workloads change their access patterns (e.g., a workload with good temporal locality will push other workloads off the cache). To account for these effects, MONITORMINING represents function $w_{i,j}$ as a *moving average* [24] that gets recomputed by regression every n sampling periods.

3.3 Action Model

An action model captures the transient costs, and expected benefit of invoking the action; these effects are a function of the current system state and the values of the invocation parameters. The effect of invoking the action is represented as a change in one of the following:

1) *Component models* e.g., data prefetching improves the response-time of the component for sequential workloads, and represented as a change in the component model.

2) *Workload models* e.g., migration of data reduces the workload's dependency on the current component as data is moved to the new component; this is represented as a change in the workload model.

3) *Workload access characteristics* e.g., the throttling action is represented as a change in the workload request rate.

In the examples described above, throttling and data prefetching have a negligible transient cost. But actions like migration incur the transient cost of reading data from the source and writing it to the target. Both the transient cost as well as the permanent benefit function are represented in terms of a workload model; the transient cost is formalized as an additional workload stream on the source and target components.

4. Creation of the MONITORMINING models

The functions for the component, workload, and action models can potentially consist of a large number of parameters. For example, in the case of migration action, the monitoring infrastructure will collect detailed state information (order of hundreds of parameters) from indiviual components in the invocation path. A pure black-box approach will try to find a function that relates all of them and will be quite inaccurate; on the other hand, the white-box approach would define the exact function between the relevant subset of parameters, but would be complex to define and brittle to the system changes.

MONITORMINING uses a hybrid approach where the designer defines a list of correlated parameters along with a hint of the nature of relationship (as shown in figure 2), while data regression techniques are used to deduce the function. The intuition of MONITORMINING is that the list of correlated parameters is dependent on the actual implementation and is non-brittle w.r.t to the underlying physical infrastructure, while the co-efficents of the parameter functions are brittle and need to be evolved at run-time.

4.1 Designer-defined Specifications

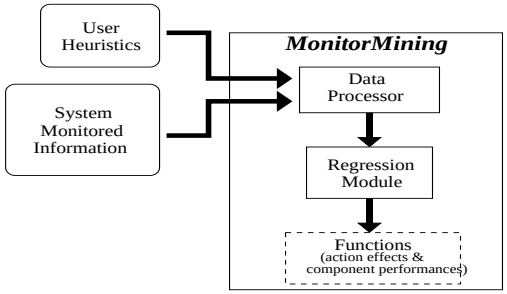


Figure 2: The overall procedure of deriving action and component functions.

The designer-specifications enumerates a list of related input-output parameters for the action, component, and workload models e.g. *Parameter X is related to the target Parameter Y*. Additionally, the specifications can have an optional hint for the type of relationship e.g. *There is a quadratic relationship between Parameter X and Parameter Y.* Figure 3 gives example specifications for the migration action.

4.2 Extracting functions using Regression

Using the designer specifications, MONITORMINING analyzes the performance log to derive the models. The schema for the performance logs is as shown Figure 4.

The parameters short-listed by the designer-specifications are extracted from the performance log and fed to the regression algorithms. MONITORMINING implements two approaches for regression, – Support Vector Regression (SVR) [5] that is relatively easy to implement, and the traditional Neural Network [21] with back-propagation.

- The key idea of SVR is to find the balance point between the training error and the complexity of the function; in other words, it avoids finding complex functions with

```
<Action: Migration>
<Transient-behavior>
 <output_parameter ="request-rate" @ source>
    <input_parameters>
      <parameter name="migration_speed" func="linear" />
      <parameter name="data_size" />
   </input_parameters>
 <output_parameter ="request-size" @ source>
    <input_parameters>
      <parameter name="disks_per_lun">
      < parameter name = "stripe_size" />
    </input_parameters>
 <output_parameter ="read/write_ratio" @ source>
   <input_parameters>
      <parameter name="workload_characteristics" />
    </input_parameters>
 <output_parameter ="random/sequential_ratio" @ source>
   <input_parameters>
      <parameter name="workload_characteristics" />
    </input_parameters>
<\Transient-behavior>

<Permanent-behavior>
 <output_parameter ="request-rate","request-size", "read/write_ratio",
"random/sequential_ratio" @ source>
    <input_parameters>
      <CONSTANT />
    </input_parameters>
<\Permanent-behavior>
<\Action>
```

Figure 3: Specifications for the migration action(for simplicity we only enumerate the source component parameters).

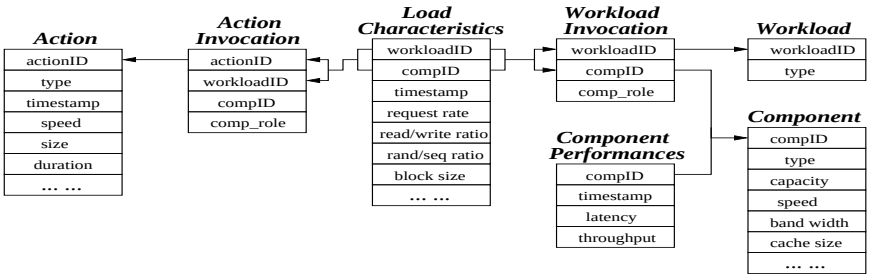


Figure 4: The schema of the database of monitored information.

low error only on training data but high error on real world data. SVR is able to identify linear functions, polynomial functions, and functions of arbitrary shapes as directed by user. It is usually inefficient for large datasets.

- Neural networks can find functions of arbitrary shapes by adapting its network structure with the data. It is efficient and can perform reinforcement learning to adapt to changing environments.The structure of a neural network is shown in Figure 5. A neural network contains an input layer, one or more hidden layers, and an output layer.

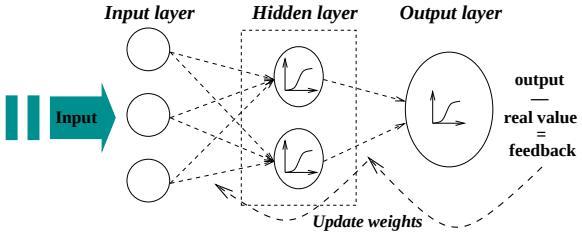


Figure 5: Adaptive learning of neural networks.

MONITORMINING uses a brute force approach to determine the function (in case the designer specifications do not specify them). It applies different function forms to the data and chooses one with the "best-fit." The list of candidate functions used are: (1) linear (x), (2) quadratic ($x^2 + ax$), (3) power (x^a), (4) reciprocal ($\frac{1}{x}$), (5) logarithm ($\ln(x)$), (6) exponential (a^x), and (7) simple combinations of two of them, such as reciprocal linear ($\frac{1}{x+a}$).

In summary, neural networks and support vector machines can both identify functions of arbitrary shapes. But they usually have better performances when the data can be well modelled by some simple models. The time complexity for Neural networks should be linear to the data size (but usually it will iterate many rounds for optimization). The time complexity for support vector machines is quadratic w.r.t. number of data points.

4.3 Bootstrapping and Evolution of models

The initial baseline values for the action, workload, and component models are generated as follows:

- Component models: The initial values are generated either from the component's performance specifications provided by the vendor, or by running calibration tests and measuring the component's behavior for different permutations of workload characteristics. The calibration tests generate I/O requests with different permutation of <request_size, read_write_size, random_ssequential_ratio, num_threads>. For each of the IO permutations, the iops, wait-time, and service-time counters are collected from the component.
- Action models: The effect of action is mainly dependent on the implementation details of the actions rather than the deployment specific details. As such, the baseline values for the action models can be pre-packaged by running in-house experiments to invoke the action for different workload characteristics and invocation parameter values.

- Workload models: The initial values of the workload models is based on libraries of workload characteristics for different applications such as e-mail, web-server, online-transactions, etc.

The models are continuously updated using the newly monitored; this improves the accuracy of the regression functions (increasing the number of data-points that have been seen in the past), and also accounts for changes in the system (especially the workload models). Evolving models using neural networks is based on the difference between the predicted value and the actual monitored value; this difference is used for *back propagation* i.e. change the link weights between units of different layers. MONITORMINING uses two approaches to evolve the models: 1) A computationally efficient approach is to invoke regression after every m additional data-points are collected from the system; this approach is used for the component and action models as they are relatively static compared to the workload models 2) Another approach is to update the model after every prediction; in this the difference between the predicted value and the actual value is used as an error-feedback to adjust the coefficient values in the model using re-enforcement based neural networks. The experimental section compares results of both these approaches.

4.4 Handling incomplete specifications

The system designer may not provide a complete set of relevant parameters. Missing parameters lead to inaccuracy of the models and reflect as larger differences between the predicted value and the actual value. A data mining approach called *Iceberg Cubing* [2] is used for this purpose. The approach can be formally stated as: *Given a set of records with K parameters $x_1, \ldots, x_K$ and a target value y, find out all groups of at least m records that have identical or similar values on at least $K - \delta$ parameters ($\delta = 1$ or 2). We say two values v_1, v_2 of parameter x_k are similar to each other if $v_1 - v_2 \leq \epsilon \cdot range(x_k)$. ($m$ is set to 5 in* MONITORMINING*).*

To illustrate this, consider the designer-specifications are shown in figure 9. In these specifications, num_threads is not specified as a relevant parameter. MONITORMINING uses Bottom-up computation (BUC) as an Iceberg Cubing algorithm, and its internal working is described as follows.

```
<component name="disk">
  <output_parameter ="IOPS">
    <input_parameters>
      <parameter name="RW_ratio" />
      <parameter name="SR_ratio" />
      <parameter name="block_size" func="linear" />
    </input_parameters>
  </output_parameter>
 </component>
```

Figure 6: Incomplete component specifications.

100 records are randomly selected and plotted in Figure 7. It is hard to judge whether

num_thread and IOPS (output parameter) are related, when the effects of three other parameters are present.

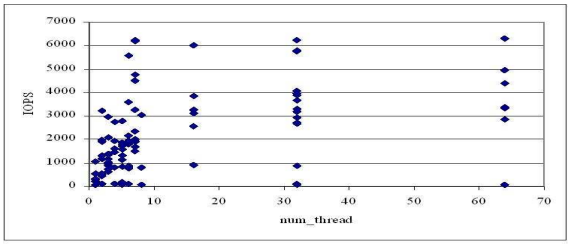
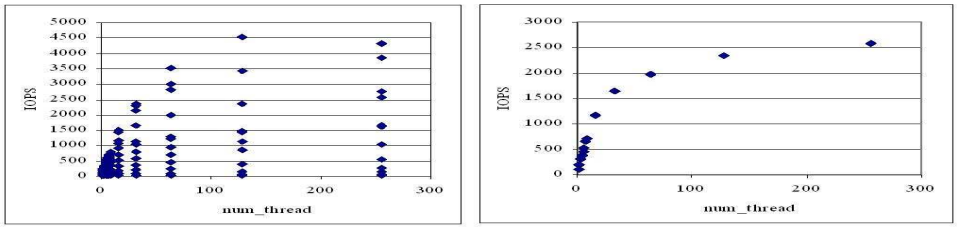


Figure 7: Plot of IOPS vs. num_thread.

As such, in order to identify the relationship between num_thread and IOPS, BUC finds all the records with a certain RW_ratio and SR_ratio (but different block_size), and plot them in Figure 8 (a). From this plot it is clear that num_thread and IOPS are related, but it is still hard to find how they are related. In Figure 8 (b) BUC plots records with identical values on all parameters except num_thread, and it becomes obvious that IOPS is a sub-linear function of num_thread; regression techniques can be used to the exact function.



a) with different block sizes b) with a certain block size

Figure 8: Plot of IOPS vs. num_thread by fixing the values of other parameters such as RW_ratio, SR_ratio.

5. Experimental Evaluation

The current set of experiments serve as a partial proof-of-concept for MONITORMINING. In these experiments, MONITORMINING was used to create the component model for a 30-drive RAID 0 Logical Volume running on an IBM FAStT 900 storage controller. The performance logs consisted of 3168 data-points, each of which has four parameters (number of threads, read/write ratio, sequential/random ratio, and block size) and two target values (IOPS and latency). The regression calculations were performed on a P4 2.8 GHz workstation with 512MB main memory, running Windows XP Professional. The

regression algorithms used in MONITORMINING were *SVM-light* * for support vector regression, and a version of Neural Networks implemented by CMU. In each of the experiments, the data-points are divided into five parts; four parts are used for training the regression algorithms and one part for testing the accuracy of the functions.

5.1 Creation of models and identifying function forms

In this experiment, MONITORMINING is given the designer specifications are shown in 9. Using the monitored data-points, MONITORMINING identifies the relationship functions between the individual parameters, and the composite function that relates the target value with all the input parameters. The results are summarized in Table 2.

```
<component name="disk">
  <output_parameter ="Response-time">
    <input_parameters>
      <parameter name="num_thread" />
      <parameter name="RW_ratio" />
      <parameter name="SR_ratio" />
      <parameter name="block_size" func="linear" />
    </input_parameters>
  </output_parameter>
 </component>
```

Figure 9: Component specifications where all the relevant parameters are specified.

	SVR	*Neural Networks*
Average error	0.393	0.159
Median error	0.352	0.121
Runtime (sec)	360	1.80

Table 2 Predicting component models for complete designer-specifications.

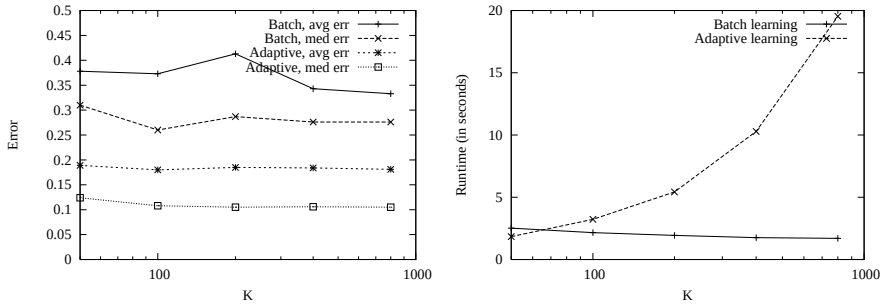
5.2 Evolving the models

For this experiment, we create a data-set in which some aspects of component behavior are made to change over time. We divide our current data-points according to their sequential/random ratios; they are divided into six partitions in this way, each having a certain sequential/random ratio ($0, 0.2, \ldots, 1$). Then we randomly choose a partition, and draw a random number (0 to 400, uniformly distributed) of records from that partition and add to our new dataset. We repeatedly do this until all records are added. If there are not enough records in a partition, just add all remaining records. Then the parameter of sequential/random ratio is removed from the new dataset. In general, this dataset

*http://svmlight.joachims.org

can be considered to contains records of different workloads, each having different sequential/random ratio. A good adaptive learning method should be able to adapt itself according to the changes of the component behavior.

The average error and median error with static learning (i.e. models created in testing phase are not refined) was 0.203 and 0.174 respectively. In the batch mode learning in which the model is re-generated after every K records ($K = 50, 100, 200, 400, 800$). Similarly, in the adaptive learning mode, the neural network continuously refines the weights using back propogation. The accuracy and running time of the two experiments are shown in Figure 10.



a) Relative Error vs. K b) Runtime vs. K

Figure 10: Accuracy and runtime of batch learning and adaptive learning

From the experimental results, adaptive learning achieves highest accuracy (higher than batch learning and even static learning). This is because it keeps adapting the model to new data when the component changes its behavior. It is quite efficient when $K \leq 200$, and its accuracy does not improve for larger values of K.

6. Related Work

Encoding of the domain-specific knowledge has been an active area of research within Expert Systems [3]. In system management, the white-box approach for creating domain knowledge is manifested as Event-Condition-Action (ECA) rules [14] that define the system behavior in different system states (originally proposed by Carl Hewitt as *pattern-based procedure invocation* [15]). These rules serve as "canned recipes" for automated management i.e. at run-time, the management software simply determines the rule that is applicable in the current state, and invokes it. Similarly, the black-box approach is mainly manifested as Case-Based Reasoning [18, 27], where the management software determines the action to be invoked by scanning a history of previous system states that are similar to the current state.

The gray-box approach as proposed in this paper is new to the domain of system management; there are a few manifestations of the gray-box approach in other domains. For example the Snowball project [1] extracts information from the text; it starts off with initial sets of patterns, and recursively refines the patterns based on the input text. The

details of the technique are tied to the domain of text extraction and analysis. Another similar concept is referred as *Lifelong Learning* [23] where the information is continuously evolved using the hypothesis from previous learning tasks.

7. Conclusion and Future Work

Model-based system management is one of the promising approaches to automated system management. In a model-based approach, the management decisions are based on predictions for the behavior of the system, given the load characteristics and configuration parameters. The key requirements for applying the model-based approach in real-world scenarios are: 1) Models need to simple yet semantically rich for making decisions; 2) Models should be easy to maintain, and update for changes in the system properties; 3) Techniques to handle bootstrapping for the models; evolving the models at run-time when additional monitoring information is collected; and ability to discover missing system parameters on which the model is dependent. Existing model-based frameworks have a limited scope and not applied comprehensively to the domain of run-time system management.

The objective of this paper is to address the issues related with representation, creation, and evolution of models for automated system management. We propose MONITORMINING as a gray-box approach for creating models; it combines designer specifications with the information generated using machine learning techniques.

As areas of future work, first, we plan to evaluate the ability of MONITORMINING to discover missing parameters and the computational complexity of the Iceberg approach [2]. Second, we want to generalize the MONITORMINING approach to other domains such as web-server management and grid computing – the primary challenge is representation of the domain-specific details as models (similar to the component, workload, and action models used for storage virtualization).

References

[1] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000.

[2] Kevin Beyer and Raghu Ramakrishnan. Bottom-up computation of sparse and iceberg cube. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 359–370. ACM Press, 1999.

[3] J. S. Brown. The low road the middle road and the high road. In P. H. Winston and K. H. Prendergast, editors, *The AI Business*, pages 81–90. MIT Press, Cambridge, MA, 1984.

[4] B.G. Buchanan and E.H. Shortliffe, editors. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Massachusetts, 1984.

[5] Chris Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[6] Maria Calzarossa and Giuseppe Serazzi. Workload characterization: A survey. *Proc. IEEE*, 81(8):1136–1150, 1993.

[7] International Business Machines Corp. Autonomic Computing:IBM's Perspective on the

State of Information Technology. http://www.research.ibm.com/autonomic/manifesto/, 2001.
[8] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 1995:18–??, 2001.
[9] E. A. Feigenbaum. The art of artificial intelligence: Themes and case studies of knowledge engineering. In *Proc. of the 5th IJCAI*, pages 1014–1029, Cambridge, MA, 1977.
[10] M.R. Genesereth and M.L. Ginsberg. Logic Programming. *Comm. ACM*, 28(9), September 1985.
[11] J.S. Glider, F. Fuente, and W.J. Scales. The software architecture of a san storage control system. *IBM System Journal*, 42(2):232, 2003.
[12] C. Green. Application of theorem proving to problem solving. In B. L. Webber and N. J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 202–222. Kaufmann, Los Altos, CA, 1981.
[13] Gartner Group. Total Cost of Storage Ownership – A User oriented Approach. Research note, Gartner Group, 2000.
[14] F. Hayes-Roth. Rule-based Systems. *Comm. ACM*, 28(9), September 1985.
[15] C. Hewitt. Procedural embedding of knowledge in planner. In *Proc. of the 2nd IJCAI*, pages 167–182, London, UK, 1971.
[16] Martin Hofmann, Glen Collins, Juan Vargas, John Bourne, and A. Brodersen. A paradigm for building diagnostic expert systems by specializing generic device and reasoning models. In *Proc. 1st Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 37–42. ACM Press, 1988.
[17] Raj Jain. *The Art of Computer System Performance Analysis*. Wiley, 1991.
[18] D.B. Leake. *Case-Based Reasoning: Experiences, Lessons and Future Directions*. AAAI Press, 1996.
[19] E. Lupu M. Sloman. Security and management policy specification. *IEEE Network*, March 2002.
[20] D.A. Pease, J.Menon, B. Rees, L.M. Duyanovich, and B.L. Hillsber. Ibm storage tank-a heterogeneous scalable san file system. *IBM Systems Journal*, 42(2):250–267, 2003.
[21] S.J. Russell and P. Norvig. *Artificial Intelligence: a modern approach*. Prentice-Hall, 1995.
[22] David Gerand Sullivan. Using probabilistic reasoning to automate software tuning. September 2003.
[23] S. Thrun. Lifelong learning: A case study, 1995.
[24] Nancy Tran and Daniel A. Reed. ARIMA time series modeling and forecasting for adaptive i/o prefetching. In *Proceedings of the 15th international conference on Supercomputing*, pages 473–485. ACM Press, 2001.
[25] S. Uttamchandani, K. Voruganti, S. Srinivasan, J. Palmer, and D. Pease. Polus: Growing storage QoS management beyond a 4-year old kid. In *Proc. of 3rd File and Storage Technologies (FAST)*, March 2004.
[26] D. Verma. Simplifying network administration using policy based management. (2), March 2002.
[27] D. Verma and S. Calo. Goal Oriented Policy Determination. In *Proc. 1st Workshop on Algorithms and Architectures for Self-Managing Sys.*, pages 1–6. ACM, June 2003.