Model Building for Dynamic Multi-tenant Provider Environments

Jayanta Basak
NetApp India Private Ltd.
Advanced Technology Group
Bangalore, India
basak@netapp.com

Kushal Wadhwani
NetApp India Private Ltd.
Advanced Technology Group
Bangalore, India
kushalw@netapp.com

Kaladhar Voruganti
NetApp Inc.
Advanced Technology Group
Sunnyvale, USA
kaladhar@netapp.com

Srinivasan Narayanamurthy
NetApp India Private Ltd.
Advanced Technology Group
Bangalore, India
naras@netapp.com

Vipul Mathur
NetApp India Private Ltd.
Advanced Technology Group
Bangalore, India
vipulm@netapp.com

Siddhartha Nandi
NetApp India Private Ltd.
Advanced Technology Group
Bangalore, India
nandi@netapp.com

ABSTRACT

Increasingly, storage vendors are finding it difficult to leverage existing white-box and black-box modeling techniques to build robust system models that can predict system behavior in the emerging dynamic and multi-tenant data centers. White-box models are becoming brittle because the model builders are not able to keep up with the innovations in the storage system stack, and black-box models are becoming brittle because it is increasingly difficult to a priori train the model for the dynamic and multi-tenant data center environment. Thus, there is a need for innovation in system model building area.

In this paper we present a machine learning based black-box modeling algorithm called M-LISP that can predict system behavior in untrained region for these emerging multi-tenant and dynamic data center environments. We have implemented and analyzed M-LISP in real environments and the initial results look very promising. We also provide a survey of some common machine learning algorithms and how they fare with respect to satisfying the modeling needs of the new data center environments.

Keywords

Resource Modeling, Machine Learning, Storage Management, Black-Box

1. INTRODUCTION

Currently, there is immense pressure on storage providers to increase the utilization of their resources. For example, if a storage provider can run their storage systems at 70 percent utilization (with respect to storage capacity and performance) rather than at 30 percent utilization, then this directly improves their bottom line. In order for storage providers to be able to aggressively increase resource utilization while not violating any performance, protection and availability service level objectives (SLOs), they need to be able to proactively answer the following two questions: 1) whether the SLO requirements of the application being provisioned can be satisfied by the underlying storage system and 2) whether deploying the new workload will negatively impact the SLOs of the already deployed applications. The goal of the system model being discussed in this paper is to be able to pro-actively answer the above two questions.

Historically, people have developed either white-box or black-box based system models to solve the above problem. System designers are increasingly finding that developing and maintaining white-box models are very complex and time consuming tasks, and thus, system models built using white-box techniques are trailing storage controller innovations by a couple of product release cycles. Similarly, models built using the existing black-box modeling techniques are increasingly proving to be inadequate because of the following reasons:

- **Hybrid Storage Systems:** In the past, workload to underlying storage system mapping was more static in nature. For example, people used SSDs or fiber channel disks for satisfying the needs of OLTP workloads, and SATA disks to satisfy the needs of archival workloads. However, going forward, in hybrid storage systems, the same underlying combination of SSDs and SATA disks could be used to satisfy multiple types of workloads. This, in turn, makes it difficult for model builders to a priori train their black-box models for all the different workload and system configuration combinations.
- **Dynamic Data Center Environments:** Service providers start out providing support for a fixed set of storage services. However, eventually customers deploy applications whose needs are not quite satisfied by the service provider given storage service classes, and thus, it is necessary to be able to dynamically create black-box system models on the fly.

Thus, in order for black-box modeling techniques to be relevant in *today's* dynamic (where new workload are dynamically added and removed) and multi-tenant (where the different workload types share a common storage infra-structure) cloud provider environments, they need to have the following properties:

- **Execute in Live Mode:** Storage system environments are increasingly becoming dynamic in nature. This, in turn, usually means that new models need to be dynamically and quickly (in the order of minutes and not hours) created as the workload and system configuration change.
- **Execute in Untrained Region:** Increasingly, black-box model builders are finding that it is impossible to a priori train a black-box model for all possible system and workload combinations. Thus, there is a need for a model to be able to predict in untrained region.
- **Predict Non-Linear Behavior:** As system utilization increases, and as the combination of varying workload types that are co-located increases, the relationship between IOPS and latency will be non-linear in nature. Thus, it is essential for a black-box model to be able to predict when the system behavior is non-linear.
- **Provide Prediction Error:** Black-box models must also provide information about prediction error so that higher level provisioning and migration management tools can choose to ignore the guidance provided by the model.

In this paper we propose a new black-box modeling algorithm, M-LISP (A Machine Learning based Incremental Storage Provisioning Framework), that uses a combination of the classification and regression tree (CART) [3] and Kriging [10] machine learning techniques to satisfy the above mentioned black-box modeling requirements. M-LISP observes the behavior of the storage system live and builds a black-box model by observing various storage system counters (e.g. read latency, IOPS, amount of data transferred, etc). M-LISP first uses CART algorithm to prune the system counters which affect latency and IOPS of the system. CART allows M-LISP to build models in the order of minutes and not hours, and it also allows M-LISP to track and predict non-linear system behavior. M-LISP then uses modified Kriging [10], another machine learning technique that we borrowed and enhanced from Geostatistics domain, at the relevant leaf nodes of the CART to extrapolate the behavior (predict latency versus IOPS curve in untrained region). Kriging allows M-LISP to provide confidence bands for its predictions. The combination of CART and modified Kriging is a new contribution also from a machine learning standpoint.

M-LISP is able to extrapolate the storage system behavior for incremental provisioning. M-LISP constructs a workload signature (a set of storage counters), which determine the latency and throughput of the system. We find out the workload signature using the CART model [3]. We build CART model in such a way that in each leaf node there is a certain amount of variance in latencies of observed samples (we perform early stopping of the growth of the tree). For extrapolating the behavior, we iteratively generate synthetic samples. A synthetic sample is a sample in the extrapolated region which the system has not observed yet. For example, if the system observes IOPS up to 1000, we generate a synthetic sample which contains the IOPS, workload signature, and latency at 1050 IOPS. We then proceed to 1100 IOPS and so on. At each step we take samples from the relevant leaf node of the CART and perform unconstrained Kriging for the purpose of extrapolation. In Kriging model, we take into account synthetic samples that have been generated previously. Along with the generation of synthetic samples, we associate a confidence band with the prediction of each latency value.

2. BACKGROUND AND RELATED WORK

People have proposed numerous white-box and black-box modeling techniques in the past. White-box models have been proposed to model disks and simple storage arrays [6, 9, 11, 14]. It is difficult to build white-box models for complex storage systems because it is hard to understand the intricate interplays between various hardware and software system components. Thus, the availability of these models typically lags the release of corresponding features in the storage system.

Table 1, describes how some popular existing black-box techniques fare with respect to the features that are required in predicting system behavior in dynamic and multi-tenant cloud provider environments. Due to space constraints we are not describing the details of the various black-box modeling techniques in this paper.

BASIL [4], Pesto [5], Relative Fitness [7] and CMUCART [16, 17] are three recent black-box techniques that have been proposed for modeling storage systems. BASIL provides predictions in the interpolation (trained) region. Furthermore, BASIL does not provide confidence level with its predictions. Pesto can extrapolate in the unseen region under the assumption that latency has a linear relationship with the outstanding IO. However, Pesto is not able to provide any confidence level about the prediction in the extrapolated region. Both in relative fitness [7] and CMUCART [17], the performance of the storage system is predicted based on observed samples from the past. These approaches are good for prediction in the interpolation region and they leverage CART (classification and regression tree) based machine learning. In these approaches certain counters are considered, and a CART model is built to model the latency and throughput. Each leaf node of the CART consists of past samples which have similar latency values. For a new sample, it is assigned to a leaf node of the existing CART, and the corresponding latency in that leaf node is predicted. In other words, if the new sample is similar to the past observed samples only then the prediction becomes accurate.

In machine learning literature [15, 2], function approximation refers to predicting the functional value for an unobserved sample. In support vector regression (SVR) [15, 12] with RBF kernels, the prediction is usually good for the interpolation region. Support vector regression with polynomial kernels can be used for extrapolation, however, it is not possible to associate confidence level with the prediction using support vector regression. From the storage provisioning perspective, it is essential to associate a confidence level with the prediction to suggest to the user if the prediction is reliable or not. Kriging [18, 10] is able to associate a confidence level with each prediction. With a modification in

Table 1: Classification of Black Box techniques for Storage Modeling

	Polynomial Regression	SVR [12]	CART [16, 17, 3]	Kriging or GP[10, 18]	Relative Fitness [7]	BASIL [4]	Pesto [5]	M-LISP
Live Mode	N	N	Y	N	Y	Y	Y	Y
Extrapolate	N	N	N	N	N	N	Y	Y
Non-Linear	Y	Y	Y	Y	Y	N	N	Y
Confidence Interval	N	N	N	Y	N	N	N	Y

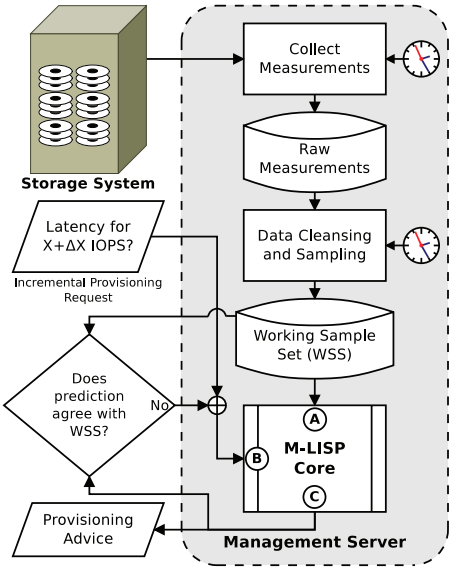

Figure 1: Architecture diagram showing the interaction of M-LISP with the storage system being monitored.

computation of model variogram from the experimental variogram, Gaussian Process (GP) model has been developed [8] which also associates confidence level with prediction, although GP is suitable for interpolation only.

3. DESCRIPTION OF M-LISP

The goal of the M-LISP framework is to learn from data collected in the past and predict the possible future behavior of the storage system. The behavior is characterized by throughput and latency. The system should ideally prescribe how the latency will change if more workload is given to the system. We model the latency behavior against the IOPS of the system with an assumption that an increase in the workload will increase the IOPS. In real-life, the workload may refer to number of clients, number of email accounts or other such variables. Mapping of these real-life factors on to IOPS is not within the purview of this paper.

3.1 Architecture of M-LISP Framework

Here, we introduce the role of various components of the M-LISP framework. Figure 1 depicts a high level view, while details of the core M-LISP algorithm are discussed in subsequent subsections. Broadly, our proposed solution can be understood in three parts: measurement collection, pre-processing and the core M-LISP algorithm. These are depicted as rectangles in the figure. The workflow starts by monitoring various metrics as internal counters of the *storage system*. These counters include both system-specific metrics (such as CPU utilization, and other software/ hardware state measurements) and workload-specific metrics (such as volume throughput and latency). The workflow proceeds as follows.

Periodically (say every 30 seconds), the full set of measurement data is collected by a *management server*, and stored as raw measurements. This raw data is pre-processed and sampled at regular intervals (e.g. every hour) by a *data cleansing and sampling* operation (details below). This gives the reduced *working sample set (WSS)* (a subset of relevant system counters and their corresponding values) of measurements, which is used by the core M-LISP module as an input (point **A** in Fig. 1) to learn a model of the behavior of the storage system in the recent past.

The data cleansing operation, checks for counters whose values are missing, constant, or inactive. The counters which are not active, or are constant, are eliminated. If the number of instances of missing values of a counter is larger than a certain threshold, that particular counter is also eliminated. If a counter has very few missing values, the corresponding measurements are eliminated. Once the counter data is ready, M-LISP samples the data and constructs a *working sample set* (WSS). If the size of the working set is large then accuracy of prediction becomes better at the cost of large time for prediction [1]. As a trade-off we observed that a working sample set size of 500-1000 samples works with reasonable accuracy and time. At present we sample uniformly, however, accuracy can possibly be improved by having more number of samples from the recent past.

The process of learning in the M-LISP core based on WSS can be triggered (point **B** in the figure) in two cases: (a) when an incremental provisioning request is received, or (b) when previous predictions are found to be inaccurate based on new data received in a recent WSS. The latter feedback loop enables the live-learning feature of our methodology where if the error in the predict crosses above a threshold (specified as policy input) M-LISP re-starts a new training run to update the model.

Given its black-box nature, the core M-LISP algorithm is system and workload agnostic – it can potentially model any metric at system/ workload/ volume/ LUN level. Depending on what kind of prediction is required, the level and metric of interest needs to be specified as part of the *incremental provisioning request*. With this input, M-LISP internally builds and persists the model (details later). Once the internal model is ready, predictions can be quickly obtained as described in section 3.2. In this paper we use latency as the performance metric to be predicted, when throughput (expressed as IOPS) increases by a small amount (see Figure 1). The entire pre-processing, learning and prediction process is described in detail in the subsequent subsections, along with the core M-LISP algorithm.

[1]For example, 10% accuracy with 20 hour of computational time vs. 15% accuracy with one hour computational time.

M-LISP can be run either on the storage server itself or in a separate management server. Furthermore, if the system and workload configurations are a priori well known then M-LISP can be used in an off-line manner to create the appropriate model and ship it along with the system. The model can subsequently be fine-tuned, if necessary, in a live manner.

3.2 Core M-LISP Algorithm

The overall M-LISP method is shown in Figure 2. As the core M-LISP receives the provisioning request, it finds out the performance counter which needs to be modeled and also finds out the extrapolated region for which prediction has to be performed. It feeds the relevant information to the M-LISP algorithm for subsequent computation of the extrapolated region.

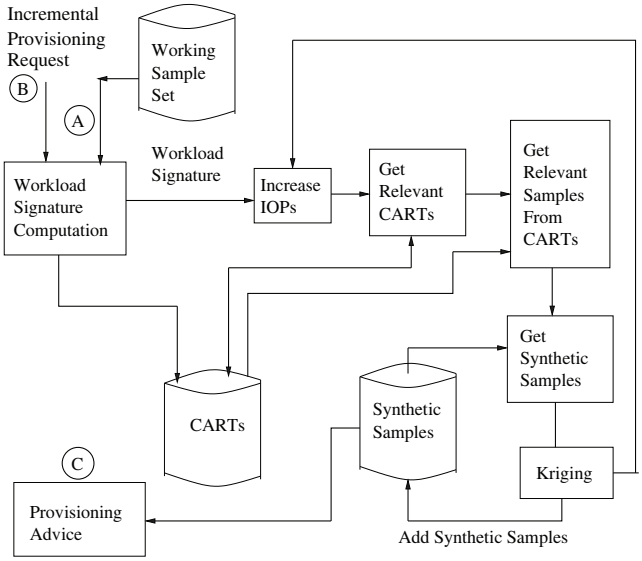

Figure 2: Different blocks of M-LISP core.

Once the working sample set is selected, M-LISP constructs a workload signature. A workload signature is essentially a handful set of counters which determine the latency against IOPS. In the workload signature, each counter is also decided by the other counters including latency and IOPS in the set. We determine workload signature in such a way that no other counter outside the signature has any influence on any counter in the signature. We use classification and regression tree (CART) [3] to determine the workload signature. The CARTs thus generated are stored as the metadata.

Once the workload signature is decided, we increment IOPS by certain fixed step, and for each increased value, we determine the new counter values. These new counter values in turn, decide the latency. We perform this process of synthetic sample generation using CART and Kriging. Kriging has been used in the geostatistics domain, and we use this technique along with CART not only to predict the latency in the extrapolated region, but also to produce a confidence band along with the prediction.

As we increase the IOPS, the relevant samples that can be used for predicting the extrapolated region are extracted from the CARTs stored as metadata. For each counter, we store one CART model. For each increment in IOPS, we also get the previously generated synthetic samples which are also stored separately. The samples obtained from CARTs and the previously generated synthetic samples together are used for training the Kriging model. We used unconstrained Kriging for each counter separately to predict the new value for the increased IOPS. The new counter values, in turn, are used to predict the new latency value for the increased IOPS. Thus the new counter values together with the new latency value along with the increased value of IOPS constitute a new synthetic sample which is stored in the repository of synthetic samples. The synthetic samples are considered together to generate the predicted behavior in the extrapolation region (provisioning advice). We store the confidence band, as produced by Kriging, along with the prediction of the latency.

3.2.1 Workload Signature Generation

In M-LISP, we first find out the workload signature in terms of a set of counters which determine the latency and IOPS of the system. In order to determine the workload signature, we find out a transitive closure of the counters. A transitive closure of the counters is a minimal set of counters together with the IOPS and latency such that every member of the set except the IOPS (which is the independent variable according to our model) is dependent on at least one member within the set and no other counter outside the set. In the process of extrapolation, we first extrapolate each counter in this set except the latency. We then extrapolate latency based on the IOPS and extrapolated counter values. Since the set is minimal there is no redundant counter in this set on which latency and other counters do not depend. For example, in a storage system, let the *read latency* depend on the *IOPS* and the counter *read latency histogram (2-4ms)*. Let the counter *read latency histogram (2-4ms)*, in turn, depend on the *data rate* and *cache hit ratio*. Let both the counters *data rate* and *cache hit ratio* depend on the *IOPS*, where *read write ratio* is constant. In that case the workload signature will be {*IOPS, read latency, read latency histogram (2-4ms), data rate, cache hit ratio*}.

We explain the algorithm for finding out the workload signature below.

1. Initialize the workload signature $S = \{IOPS, Latency\}$.
2. Find out the counters which determine *Latency* along with $IOPS$. Let the set of counters be C. Construct S such that $S = S \bigcup C$.
3. For each counter $c_i \in S$ except $IOPS$ and *Latency*, if c_i has not already been visited, find out the set of counters C_i which determine c_i. Expand $S = S \bigcup C_i$. Mark c_i as visited.
4. Continue *Step 3* until there is no more counter in S to be visited.
5. Output S as the workload signature.

In *Step 2* and *Step 3*, we find out the set of counters determining a specific counter using classification and regression tree (CART). A CART is a decision tree which performs classification and regression. M-LISP forms a working sample set of sampled data. Let the counter c_i needs to be

modeled and the counter set C_i to be determined. The values of c_i in the working sample set along with all values of all counters (including IOPS and Latency) in the working sample set are passed to the CART building function. The CART outputs the decision tree model and the set of counters C_i which determine c_i using the data in the working sample set.

Once the CART for a counter to be modeled, is constructed, counters selected at all intermediate nodes of the CART collectively represent the set of counters which determine the predictor variable or the counter to be modeled (as in *Step 3* and *Step 4* of the algorithm). The workload signature set, thus identified, represents a minimal transitive closure of the counters such that any counter within the set is dependent on at least one other counter within the set and no other counter outside the set. Therefore, in the process of prediction of the performance of a storage system in terms of latency vs. IOPS curve, we first predict the counter values and then predict the latency in terms of the new counter values.

CART is constructed in a top-down manner by iterative partitioning of the available data. At the root node all data (i.e., all samples of the working sample set) are available. The data is split into two sets based on a certain counter (say ctr_i) value (numeric value of the counter) such that all data with the counter value less than certain threshold (say, θ_i) are in one set and the rest are in another set. Therefore two child nodes are created for the root node. For new sample if $ctr_i < \theta_i$ then the sample goes to the left child node, otherwise goes to the right child node. The counter ctr_i and threshold θ_i are selected automatically such that certain reduction of error is maximized. The reduction of error is the difference between the error at the parent node and the sum of errors at the child nodes. The error is the error measured with respect to the counter to be modeled (predictor variable) which is *Latency* in *Step 2* and c_i in *Step 3*. The error with respect to a predictor variable or a counter (e.g. root mean squared error or RMS error) is the sum of squared differences between the sample counter values and the average value for that counter over all samples present at that node. Therefore, if the threshold (θ_i) is changed then distribution of the samples in the child nodes changes and therefore the reduction of error changes. Such a process of partitioning the data and creating two child nodes is performed recursively at each node – the most relevant counter and the appropriate threshold at each node are determined such that the reduction of error is maximized every time [2]. The process of recursive partitioning at a node is stopped if the error is less than a certain threshold called the 'tolerance' parameter. Formally, the tolerance at a node is the ratio of the maximum reduction in the error due to partitioning of the data at that node and the error in that node. Evidently if the tolerance is large then partitioning is stopped early and the depth of the tree becomes small. On the other hand, for a small tolerance, a large CART is created. The 'tolerance' parameter can be user-defined. Alternatively, 'tolerance' can be decided by the method of cross-validation (either 10-fold cross-validation or leave-one-out cross-validation).

3.3 Extrapolation using Kriging

Algorithm 1, that is shown below, describes the complete extrapolation algorithm. k is the number of observed samples having IOPS range from 0 to X and the requirement is to predict Latency from IOPS range X to $X + \Delta X$. This is done incrementally by predicting one sample at a time. For each new sample IOPS are determined as previous sample IOPS plus δX and all the signature counters are extrapolated using Kriging with IOPS and previous sample counters as input and finally Latency for the new sample is extrapolated using modified Kriging as IOPS and signature counter values as input. For example, we have observed up to 1000 IOPS, and the provisioning request queries the performance at 1500 IOPS. Therefore, $\Delta X = 500$ and let $\delta X = 50$. In such case, we can first generate a synthetic sample at 1050 IOPS based on the observed samples. Next we generate a synthetic sample at 1100 IOPS based on the observed samples and the synthetic sample at 1050 IOPS. Next we generate the synthetic sample at 1150 IOPS based on the observed samples and the synthetic samples at 1050 and 1100 IOPS. We continue this process until we generate the synthetic sample at 1500 IOPS. At every time we obtain the relevant observed samples from the relevant CARTs. Note that, as discussed before, we extrapolate each counter in the workload signature during the generation of synthetic samples. Training data for Kriging is obtained using *getTrainingData*() procedure which is explained in the sequel.

Kiriging in its original form is used for interpolation but we have modified Kriging for extrapolation. Kriging also outputs confidence band around predicted latency which says with 95% confidence, actual latency will not go above upper latency and lower latency for each IOPS value.

Algorithm 1 Extrapolation

Input: S : Signature with k samples, and C_i, $0 \leq i \leq m$: counters in signature set S.
C_{ij} denote value of counter i at sample j
Output: Latency values for next $\frac{\Delta X}{\delta X}$ Samples.

```
Next Sample index z = k + 1
Initialize next sample C_iz = 0, ∀C_i ∈ S
for iops = X + δX; iops ≤ X + ΔX ; iops = iops + δX do
    iops_z = iops
    for Every C_i ∈ S do
        Test = [iops_z , C_j(z−1) [∀ C_j ∈ S, and j ≠ i] , Latency]
        Train = getTrainingData( C_i , [ iops_z , C_j(z−1) [∀C_j ∈ S and j ≠ i], Latency] )
        C_iz = modifiedKriging( Test ,Train )
    end for
    Test = [ iops_z , C_jz [∀C_j ∈ S, and j ≠ i] ]
    Train = getTrainingData( Latency , [ iops_z , C_jz [∀C_j ∈ S] ] )
    [latency_z, confidence]
        = modifiedKrigingConfidence( Test , Train )
    Increment sample index z = z + 1
end for
```

Algorithm 2 getTrainingData

```
Tree = Load( Counter.T )
Nodes = findNode( Tree ,vector)
Return ( All samples corresponding to Nodes )
```

[2]The recursive partitioning of the data and process of creating child nodes generates a binary tree

3.3.1 Selecting train sets using CART

Training set is required at every step while extrapolating using Kriging. Very naive method of providing training set is to give the entire available data, but when number of samples n is large training can become time consuming and cannot be done in *real time*. Our framework adopts a selective method of choosing samples which are very close to current sample value of the counter using CART. The following steps are executed in this regard:

1. Generate Cart tree for every counter $C_i.T$ $C_i \in S$ using rest counters in S
2. Retain
   (a) Tree $C_i.T \; \forall \; C_i \in S$
   (b) Node Sample Mapping $\forall \; C_i \in S$

For each counter we generate the CART tree using rest of the counters as input. CART tree basically divides the space into small regions and each region corresponds to a node in CART tree. Entire data of k samples is divided into smaller sets for each counter. We retain the CART tree and also the node sample mapping for each counter in the signature.

Algorithm for getting training data is described below as Algorithm 2. While the training set is requested for a given counter and sample, we choose IOPS from the given sample counter values from previous samples, and then we traverse the CART tree using these, and finally arrive to a leaf node from where we return all the samples corresponding to this resulting node.

3.3.2 Kriging for extrapolation

We use ordinary Kriging for the purpose of prediction. Usually ordinary Kriging is used for interpolation in the geostatistics domain. We used the same with for the purpose of extrapolation in the multi-dimensional space by relaxing the constraints of the ordinary Kriging. In Kriging, the functional value for an unobserved sample is represented by a weighted linear combination of those for the observed samples. The weights are derived from the variogram model which essentially estimates the variation in the observation as a function of the distance. Let x denote the sample and $f(x)$ represents the predicted variable or the function to be predicted. Let $(x_i, f(x_i))$ be the i^{th} observation and there are n such observations. Then the predicted functional value $\hat{f(x)}$ for any sample x is represented as

$$\hat{f}(x) = \sum_i w_i f(x_i) \tag{1}$$

where w_i is the weight. First the experimental variogram is obtained by observing the variance g ($g_{ij} = 0.5 * (f(x_i) - f(x_j))^2$) with respect to the distance between the samples d ($d_{ij} = \|x_i - x_j\|$). From the experimental variogram, a model variogram function $\hat{g}$ is estimated using polynomial regression such that the error $\sum_{i,j,i<j} \|g_{ij} - \hat{g}(d_{ij})\|$ is minimized. The weight coefficients for any new sample x_p are then expressed as

$$\mathbf{G}\mathbf{w} = \mathbf{h}_p \tag{2}$$

where $\mathbf{G} = [\hat{g}(d_{ij}]$ is the model variogram matrix and $\mathbf{h}_p = [\hat{g}(d_{ip})]$ is a vector of variances with respect to the new sample x_p. The variances are computed using the model variogram $\hat{g}(.)$. In ordinary Kriging, the weights are found under the constraint $\sum_i w_i = 1$. Since this specific constraint makes the linear system over-constrained, certain slack variable is introduced.

Since we are doing extrapolation, we relax the constraint $\sum_i w_i = 1$. We find an optimum solution for equation (2) with a constraint $w_i \geq 0$ for all i under a quadratic optimization framework. Note that we cannot use the constraint $\sum_i w_i = 1$ since we are not operating in the interpolation region. In the extrapolation region, this constraint may not hold true. Once we obtain the weights, we estimate the variance of prediction as

$$var(p) = \mathbf{w}^{'} \mathbf{h}_p \tag{3}$$

Once we obtain the variance in estimation, we approximate the local distribution of the predicted variable as a Gaussian distribution, and find out the 95% confidence interval as $2\sqrt{var(p)}$, and the prediction band within 95% confidence is given as $\hat{f}(x_p) \pm 2\sqrt{var(p)}$.

Table 2: Configuration of storage systems used.

Storage System	Processors	Primary Memory	Storage system interconnect	Size of Volume
A	4 Intel Xeon	3 GB	FC host adapter	600 GB
B	4 AMD Opteron	16 GB	SAS host adapter	800 GB

4. EXPERIMENTAL SETUP

The goal of our experiment section is to show that our algorithm is robust and that it can predict both in the interpolation and extrapolation regions for varying 1) workloads, 2) system loads and 3) system configurations. Thus, we have created four scenarios described in Table 3 to test the robustness of M-LISP. Systems A and B listed in this scenario table are described in Table 2. Here is a description of the different scenarios:

1. **Experiment 1:** In this experiment we ran workload WL1 on Volume 1 on Storage Server A with 100% read and 100% random IOs. We used a micro-benchmark that is similar to IOTest or IOMeter.
2. **Experiment 2:** This is same setup as in Experiment 1 but we ran WL2 which is 50% read and 50% random IOs. The purpose of this experiment was to see whether M-LISP predicts correctly for different types of workloads.
3. **Experiment 3:** This setup is same as experiment 1, but we ran a background replication job in addition to WL1. The purpose of this experiment was to see whether M-LISP prediction can handle the case when other workloads are running on the system.
4. **Experiment 4:** In this experiment we ran a macro-benchmark TPC-W on Storage Server B. The purpose of this experiment was to see whether a) M-LISP could predict for a storage system with different CPU, memory and disk configuration and b) whether it could predict for a macro-benchmark.

For all the experiments we stressed the system until the disks got saturated and we made predictions both in the unsaturated and in the saturated regions.

Table 3: Experiment Scenarios

Name	Workload	Storage System	Bottleneck Resource	Read %	Rand %	I/O Size	Clients	Think Time	Dataset Size	RAID Stripe size	Additional Load
WL1	I/O load generator	A	Disk I/O	100	100	4KB	5–150	0	40GB	10+2	—
WL2	I/O load generator	A	Disk I/O	50	50	4KB	5–150	0	40GB	10+2	—
WL3	I/O load generator	A	Disk I/O	100	100	4KB	5–150	0	40GB	10+2	Replication
WL4	TPC-W	B	Disk I/O / CPU	Mixed	Mixed	3.32KB (median)	3–30 EBs	10ms	70GB DB + 50GB Static	8+2	Micro I/O load generator

We implemented M-LISP in Matlab environment (version 7.8) with 4GB RAM. We used the in-built CART functionality and polynomial regression functions of the Matlab. For computing the model variogram from the experimental variogram, we used cubic polynomial. For approximating the latency vs. IOPS and the counter values vs. IOPS, we used cubic polynomial. In the CART, we have chosen a tolerance parameter of 0.1 and did not use the cross-validation method to determine the size of the tree. Cross-validation can provide better results in terms of interpolating the function. However, we limit the depth of the tree in order have variability of the samples in the leaf node. If all samples in a leaf node have almost same value then according to the algorithm, the extrapolated region will be flat, and latency will exhibit a constant value. A tolerance parameter of 0.1 indicates that there is 10% deviation of the target variable from the mean, i.e., there is a maximum 20% variation of the samples. We consider 20% deviation as good enough to capture the rising nature of the curve. If we increase the tolerance i.e., the captured variance in a leaf node then the extrapolated curve rises very quickly and largely deviates from the actual data. Note that this rise of the curve occurs due to relaxation of the constraints in Kriging.

5. RESULTS

5.1 Performance

For all workloads described in the scenarios Table 3, we computed the extrapolated curve for 21 different equally spaced points. We then interpolate the intermediate behavior by linear segments. For example, if we observe IOPS up to 1000, and predict latency for an IOPS ranging up to 1800 then we choose $\delta X = 40$ (which translates into predicting at 21 different points including the point where IOPS = 1000). It is possible to measure the error in prediction at all these points and take the average error. However, from the perspective of storage provisioning, the error at IOPS = 1000 may not convey any extra information. We therefore, report the results at four different points separately which shows the behavior closer to the observed region as well as farther from the region. Along with the errors, as reported in Table 4, we provide separate graphs showing the predicted latency and the actual observed latency in the extrapolated region for the various experiments. We computed the percentage error, E, as

$$E = 100 * \frac{abs(predictedLatency - observedLatency)}{observedLatency} \quad (4)$$

The observed latency can vary over a range for a given IOPS. We therefore, measure the average latency for a range of given IOPS. For example, for an $IOPS = 1000$, the observed latency is the average of all latencies observed for IOPS in the range of 950 to 1050.

Comparison with BASIL.

In order to assess the accuracy of M-LISP, we test its predictions with the predictions of BASIL black-box modeling technique that was recently proposed by Gulati et al. [4]. BASIL estimates the latency based on OIO (outstanding IO), IO size, read percentage, and random percentage. Latency is expressed as a scaled product of these four factors, each factor is added with a constant. The constants are estimated from pair of observations, and then median is considered. The scaling constant of the product is also estimated by linear regression. As mentioned, BASIL does not extrapolate. In order to predict the latency in the extrapolated region, we have to know the OIO, IO size, read percentage, and random percentage in the extrapolated region. In our experiments, as in Table 3, for workloads WL1 to WL3, we have fixed IO size, constant read percentage and constant random percentage. Therefore, latency is a linear function of OIO added with a constant.

If OIO were available in the extrapolated zone then we can obtain the extrapolated latency. We approximate the OIO as the product of IOPS and latency in the extrapolated zone. Note that, this is not real extrapolation, however, we present the BASIL curves as references to our extrapolation. Latency, according to BASIL, is just a first order polynomial of OIO with constant IO size, read percentage and random percentage. We find out this first order polynomial using regression based on the observed points. We then apply this constant using the known values of latency and IOPS in the extrapolated region, and find out the latency in the unobserved zone as if the OIO in the unobserved zone is available. We observe that BASIL is able to capture the trend in the extrapolated region. However, BASIL is not performing any extrapolation. We computed the predicted latency in BASIL using OIO, and OIO is computed using the observed latency at the same point. In real-life scenario, the OIO will not be known for an extrapolated region. We present the BASIL curves as references to show that the curve in the extrapolation region predicted by M-LISP also closely follow the BASIL curve.

Experiment 1.

In the first experiment (workload WL1), we saturated the disk using only disk reads without any other workload. We have 100% read with no sequential access i.e., 100% random

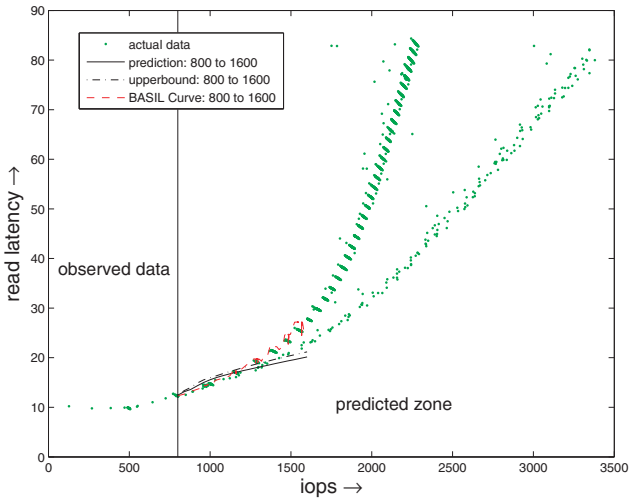

Figure 3: Extrapolation behavior for workload WL1 as in Table 3. The corresponding BASIL curve is also shown separately. The system observes up to 800 IOPS and predicts from 800 to 1600 IOPS. The latencies are in milliseconds.

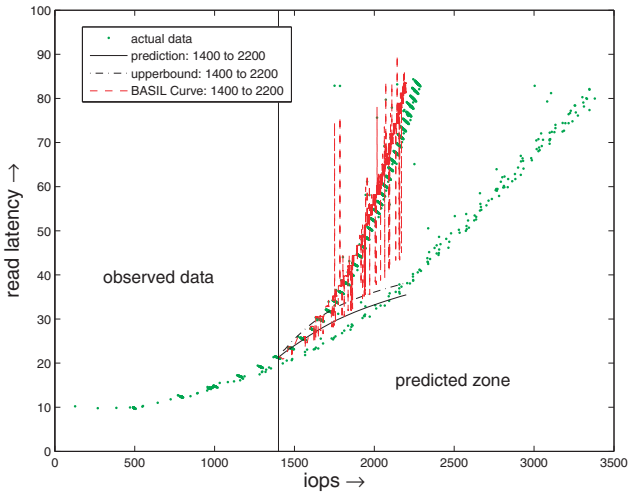

Figure 4: Extrapolation behavior for workload WL1 as in Table 3. The corresponding BASIL curve is also shown separately. The system observes up to 1400 IOPS and predicts from 1400 to 2200 IOPS. The latencies are in milliseconds.

access. The IO size is constant at 4KB. We increase the number of clients from 5 to 150 to get the saturation curve. As we increase the number of clients, the IOPS increases. We modeled the system level *read latency* against the total IOPS. Note that, as we mentioned before, any volume level latency or other counters can be modeled using M-LISP. In our results we show the behavior for read latency only. We first consider 0-800 IOPS as our observed zone and we predict beyond 800 IOPS. Figure 3 illustrates the behavior of M-LISP in predicting in the extrapolated zone. From the figure we observe that M-LISP is able to predict correctly till 1200 IOPS which is 50% of the maximum observed IOPS. We are able to predict till 50% of the maximum observed IOPS because the latency vs. IOPS curve increases linearly in that region. Next we consider up to 1400 IOPS as our observed zone, and then predict up to 2200 IOPS. Figure 4 illustrates the performance of prediction by M-LISP in the extrapolated region. We observe that the prediction is fairly close to the mean observed latency till 1800 IOPS which is close to 30% of the maximum observed IOPS. Around 1800 IOPS we approach the knee of the curve and the latency rises quickly. We also observe that the upper bound, as computed from the 95% confidence band by M-LISP (Equation 3), fairly follow the actual latency vs. IOPS curve till 1800 IOPS.

We show the percentage error over the extrapolated region for different observed zones in Figure 5. We observe that the extrapolation error is low when the extrapolation region is within 20% of the maximum observed IOPS in general. In this experiment, when the observed IOPS is within 800, the error is low till 1400 IOPS in the extrapolation region and then it increases. The error decreases as we increase the observed region. However, for a region which is closer to the observed zone, the error does not reduce much. As we move farther away from the observed zone, we find the effect of error reduction with the expansion of the observed zone. This kind of behavior makes M-LISP flexible to perform in-

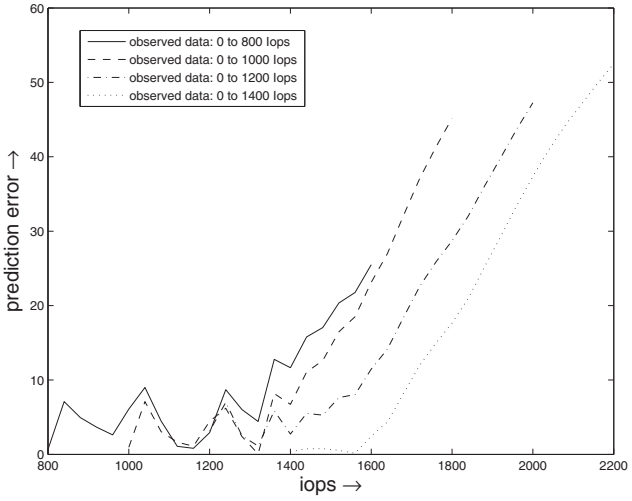

Figure 5: The change in the percentage error with different observed regions for workload WL1.

cremental provisioning in a live environment. For example, the user is using the system within 800 IOPS and the provisioning advisor recommends that he can increase IOPS by another 200 IOPS with a certain increase in latency. Once the system being used for 1000 IOPS, the system collects the data (as shown in Figure 1) and cleans the data for further provisioning request. Once the provisioning request is received, the system again computes and recommends the latency increase for an extra 200 IOPS. Every time the system gathers additional data in the increased observed zone, it refines the estimate in the extrapolated region and predicts with higher accuracy.

Experiment 2.

In our second experiment (workload WL2), we have mix-

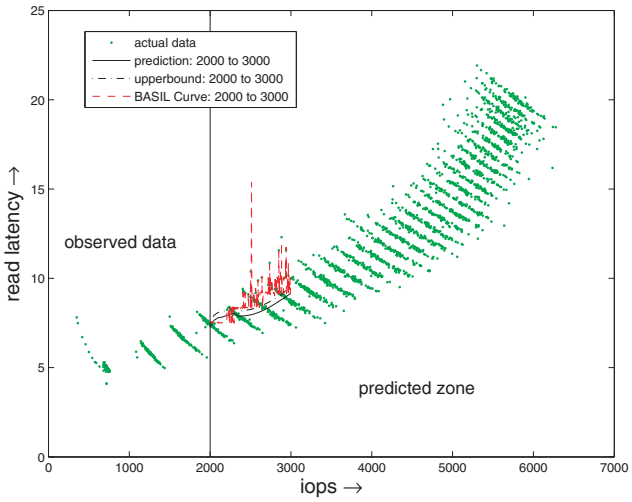

Figure 6: Extrapolation behavior for workload WL2 as in Table 3. The corresponding BASIL curve is also shown separately. The system observes up to 2000 IOPS and predicts from 2000 to 3000 IOPS. The latencies are in milliseconds.

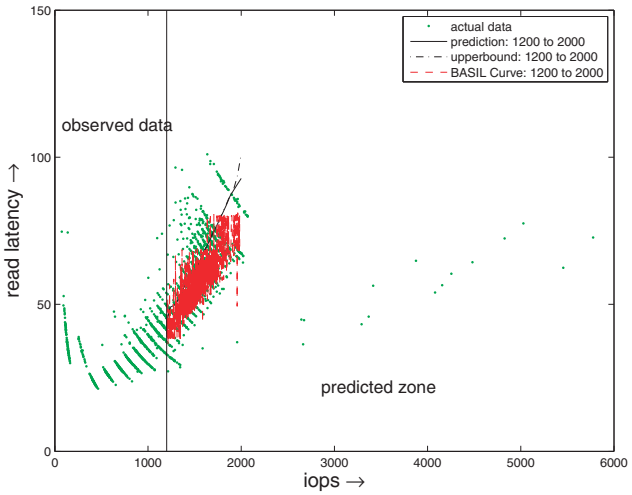

Figure 7: Extrapolation behavior for workload WL3 as in Table 3. The corresponding BASIL curve is also shown separately. The system observes up to 1200 IOPS and predicts from 1200 to 2000 IOPS. The latencies are in milliseconds.

ture of read and write operations (50% read) with 50% randomness. Here also we increase the number of clients from 5 to 150. We again model the system level *read latency* against the IOPS which is increased by increasing the number of clients. We observe till 2000 IOPS and model the behavior and predict the latency beyond 2000 IOPS till 3000 IOPS. In Figure 6, we observe that prediction closely follows the mean observed latency till 3000 IOPS itself. The upper bound of prediction, in this case, is very close to the actual prediction.

Experiment 3.

In our next experiment (workload WL3), we saturated the disk using read only operations with an additional replication workload. The system configuration is the same as our previous two experiments with 100% read and 100% random workload. Here also we increase the number of clients from 5 to 150. First we observe up to 1200 IOPS and predict the *read latency* from 1200 IOPS to 2000 IOPS. Figure 7 illustrates the prediction where we observe that the prediction almost exactly follow the actual mean latency in the extrapolated region. Here the upper bound closely follows the actual prediction and the two curves are almost indistinguishable. Here M-LISP is able to accurately predict for more than 50% of the maximum observed IOPS.

Experiment 4.

Finally we validate the performance of M-LISP on another storage system with macro-benchmark TPC-W data (workload WL4). It is mixed workload with varying read percentage and varying random-sequential ratio depending on the workload. We vary the number of emulated browser of the e-commerce server from 3 to 30 with 10ms think time. As the number of emulated browsers increases, the IOPS increases and we model the behavior of volume level *read latency* against the total IOPS. In this specific validation scenario, we do not provide the BASIL curve because the

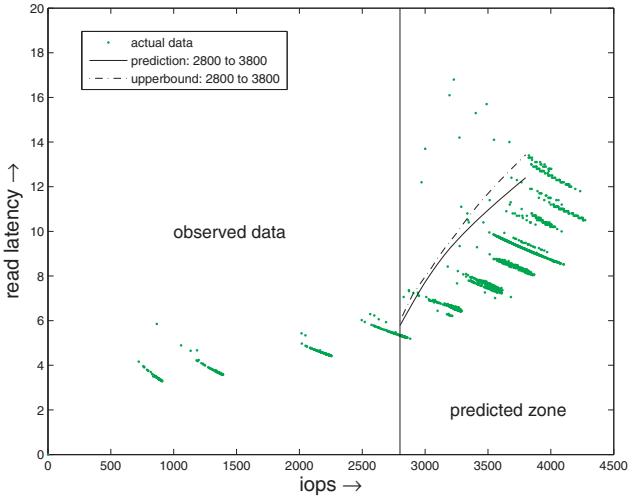

Figure 8: Extrapolation behavior for workload WL4 as in Table 3. The system observes up to 2800 IOPS and predicts from 2800 to 3800 IOPS. The latencies are in milliseconds.

IO size in this case is not constant. It is possible to get the IO size as a ratio of the *data rate* and IOPS, however, in the extrapolated region the *data rate* is also unknown. Therefore, BASIL curve in the extrapolated region will depend on the *read latency* and *data rate* as if these are known a priori. Since BASIL is not at all for extrapolation, and using two unknown variables from the actual data makes the case too artificial, we omit the BASIL curve in this scenario. We observe and model the system performance till 2800 IOPS and predict the volume level *read latency* for 2800 to 3800 IOPS. Figure 8 illustrates the fact that the prediction is conservative and actual latency is less than the predicted latency in the extrapolated region.

Table 4: The observed latencies and the predicted latencies for different IOPS and different observed zones. A '–' indicates that actual observed data were not available at that point. The latencies are in milliseconds.

Workload	Observed Region	Extrapolated Region				
	(I) $0 \leq IOPS \leq I$	(I^1) $IOPS$	$OL(I^1)$ Observed Latency at $IOPS = I^1$	$PL(I^1)$ Predicted Latency at $IOPS = I^1$	Percentage Error	$UL(I^1)$ Upper bound of Latency at $IOPS = I^1$
WL1	800	1000	14.66	15.55	6.08	15.96
		1200	16.87	17.36	2.92	17.96
		1400	21.26	18.78	11.65	19.66
		1600	27.05	20.15	25.49	21.17
	1000	1200	16.86	17.61	4.43	17.91
		1400	21.26	19.83	6.72	21.94
		1600	27.05	20.77	23.12	24.24
		1800	36.86	20.21	45.16	24.20
	1200	1400	21.26	20.68	2.74	22.01
		1600	27.05	23.95	11.47	26.09
		1800	36.86	26.27	28.72	28.49
		2000	53.03	27.97	47.26	30.14
WL2	2000	2400	8.03	7.89	1.87	8.19
		2600	8.41	8.09	3.74	8.39
		2800	8.49	8.56	0.81	8.86
		3000	8.63	9.17	6.30	9.69
	3000	3400	9.74	11.32	16.30	11.66
		3600	10.28	12.30	19.59	13.19
		3800	10.66	13.12	23.12	14.30
		4000	11.03	13.86	25.73	15.19
WL3	1200	1400	53.64	56.53	5.38	56.83
		1600	65.91	67.61	2.57	67.91
		1800	73.07	80.92	10.74	81.22
		2000	71.35	92.81	30.07	101.15
	1400	1600	65.91	61.94	6.03	62.23
		1800	73.07	70.27	3.82	70.57
		2000	71.35	79.22	11.03	79.52
		2200	-	88.72	-	89.03
WL4	3000	3400	7.918	9.615	21.43	10.014
		3600	8.465	10.751	27.00	11.524
		3800	9.075	11.704	28.96	12.791
		4000	10.329	12.571	21.71	13.818
	3200	3600	8.465	9.973	17.82	10.771
		3800	9.075	11.130	22.63	12.098
		4000	10.329	12.181	17.93	13.280
		4200	11.051	13.181	19.27	14.390
	3400	3800	9.075	10.570	16.46	11.596
		4000	10.329	11.803	14.27	13.082
		4200	11.051	12.892	16.66	14.295
		4400	-	13.901	-	15.372

Table 5: The computational time in seconds for different workloads with different number of measured counters and for different number of observed samples. The number of extrapolated points is 21 across all workloads.

Workload	No. of counters	No. of extrapolated points	No. of observed samples	Elapsed time
WL1	5039	21	188	220.07
WL2	5042	21	265	185.27
WL3	1695	21	541	2465.09
WL4	20	20	846	1945.03

6. RESULTS ANALYSIS

The previous section provided experiment results for varying workload, system and load setups that showed M-LISP's robustness and the ability to predict in the extrapolation region. In this section, we provide further detailed analysis on the results for experiments 1 and 3. A detailed technical report contains analysis for all the experiments. We analyze the results with respect to 1) when to trigger model building 2) execution time of the algorithm and 3) Automatic selection of system counters (workload signature) by M-LISP for creating the model.

When to build Model.

As can be seen in Table 4, for WL1 when the model was trained in the observed region up to 800 IOPs, it is able to predict with reasonable accuracy up to 1400 IOPs. The prediction percentage error goes above 20 percent for 1600 IOPs. However, when the model building observed region is at 1200 IOPs, the prediction error for 1600 IOPs comes down to around 11 percent. Thus, continuous periodic model building can be triggered to ensure that the modeling building observed region keeps up with the increased system load in order to ensure low (value can be specified as policy input) prediction error rates. The frequency of model building can be adjusted depending upon how quickly the load on the system is increasing in comparison to the previous observed region when the model was built. We also observe that the actual latency is almost always below the upper bound predicted M-LISP. Thus the upper bound can definitely be used for the incremental provioning purpose.

Computation Time.

We designed M-LISP to operate in live systems so that models can be created in reasonable amount of time for provisioning and model correction purposes. These models can be built either on a storage controller or on an external management server. As can be seen in Table 4, the model creation time is a function of the number of extrapolated points, number of observed samples, and number of counters. Thus, the creation time varies from case to case, but in general, it is in the order of tens of minutes.

Workload Signature.

M-LISP, as mentioned before, automatically extracts the workload signature for computing the latency in the extrapolated region. The workload signature is minimal transitive closure such that every counter is dependent on at least one counter in the set and no other counter outside the set. By examining the behavior of these counters, M-LISP is able to predict the behavior of latency. System-wide NFS IOPS and latency are the quantities we are modeling, and thus, they are common to both WL1 and WL3. The other counters (shown in Table 5) are automatically selected by M-LISP and they are different for both WL1 and WL3. Note that, in both the above cases, only one volume had an active workload. Hence, the counters corresponding to other volumes did not appear in the signature. We observed that in certain few cases the signature set is expanded or changed if the observation set changes. For example, for low IOPS where the latency is almost constant, even a smaller change in the latency is reflected into selection of certain counters. On the other hand when IOPS increases and latency changes sharply, these sensitive counters are masked by the counters which significantly influence the latency. These results show that M-LISP's ability to dynamically select the right set of counters is an important facet for our live learning framework.

Table 6: Counters used for workload signatures

Counter	WL1	WL3
System-wide NFS IOPS	X	X
System-wide avg. NFS read latency	X	X
Two bins of IOPS histogram	X (400-500, 500-600)	-
Rate of data received over the network	-	X
One bin of the histogram of rate of network data sent	X (2-3MB)	-
Amount of CPU time spent processing interrupts	X	-
NFS read latency for the active volume	-	X

7. SUMMARY AND CONCLUSIONS

In this paper, we have taken a black-box approach to design an incremental storage provisioning system based on machine learning. Since this is a black-box model, M-LISP is workload and system configuration agnostic. Also unlike the other black-box approaches [7, 4], M-LISP does not preselect any counter, rather it automatically extracts a workload signature depending on the workload pattern and the region of observation. As opposed to the existing black-box approaches [7, 4, 16, 17], M-LISP is able to perform extrapolation, and provide the user an advice about the performance of the system for certain extra amount of workload. New unconstrained Kriging has been used in the relevant leaf nodes of CART to perform extrapolation. This kind of modeling is also new from the machine learning perspective. We used Kriging to obtain a confidence band associated with the prediction. Instead of Kriging, incremental support vector regression [15, 13] or other online adaptive learning techniques [1] can also be used for the same purpose if the confidence band is not required. However, the comparative performance of these alternative learning algorithms with the present version of M-LISP needs to be judged as a future study. Proactively understanding the impact of a different workload on an existing running workload is a future extension to M-LISP that we are currently pursuing.

8. REFERENCES

[1] J. Basak. Online adaptive decision trees: Pattern classification and function approximation. *Neural*

Computation, 18:2062–2101, 2006.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1983.

[4] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar. Basil: Automated io load balancing across storage devices. In *Proc. 8th USENIX Conf File and Storage Technologies (FAST)*, pages 169–182, 2010.

[5] A. Gulati, G. Shanmuganathan, I. Ahmad, C. A. Waldspurger, and M. Uysal. Pesto: Online storage performance management in virtualized datacenters. In *Proc. 2nd ACM Symp. Cloud Computing (SOCC '11)*.

[6] A. Merchant and P. Yu. Analytic modeling of clustered raid with mapping based on nearly random permutation. *IEEE Trans Computing*, 45:367–373, 1996.

[7] M. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. Ganger. Modeling the relative fitness of storage. In *Proc. Int Conf Measurements and Modeling of Computer Systems, SIGMETRICS 2004*, pages 37–48, 2004.

[8] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, USA, 2006.

[9] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27:17–29, 1994.

[10] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409Ű435, 1989.

[11] E. Shriver, A. Merchant, and J. Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *Proceedings of Sigmetrics '98*, pages 182–191, 1998.

[12] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. Technical Report NC-TR-98-030, NeuroCOLT, Royal Holloway College, University of London, UK, 1998.

[13] N. A. Syed, H. Liu, and K. K. Sung. Incremental learning with support vector machines. In *Proceedings of the Workshop on Support Vector Machines at the International Joint Conference on Artificial Intelligence (IJCAI-99)*, San Mateo, 1999. Morgan Kauffmann.

[14] M. Uysal, G. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. In *Proc. MASCOTS*, pages 183–192, 2001.

[15] V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 281–287. MIT Press, Cambridge, MA, USA, 1997.

[16] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage device performance prediction with cart models. In *Proc. Int Conf Measurements and Modeling of Computer Systems, SIGMETRICS 2004*, pages 412–413, 2004.

[17] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage device performance prediction with cart models. Technical Report CMU-PDL-04-103, Parallel Data Laboratory, Carnegie Mellon University, Pittsburgh, USA, 2004.

[18] C. K. I. Williams. Prediction with gaussian processes: From linear regression to linear prediction and beyond. In *Learning and Inference in Graphical Models*, pages 599–621. Kluwer, 1998.