SMARTMIG: Risk-Modulated Proactive Data Migration for Maximizing Storage System Utility

Li Yin
Univ. of California, Berkeley
yinli@eecs.berkeley.edu

Sandeep Uttamchandani
IBM Almaden Research Center
sandeepu@us.ibm.com

Randy Katz
Univ. of California, Berkeley
randy@eecs.berkeley.edu

Abstract

The goal of storage management is to maximize the overall utility of the storage system by continuously tuning the amount of resources allocated to multiple independent competing applications. Due to variations in access characteristics, service level objectives, and exception events such as failures and load surges, there is a need to invoke corrective actions such as data migration to modify the resources allocated to a given application. There is a significant body of research on automated data migration – their focus has primarily been on optimizing for the current system load without considering load forecasts; the scheduling of the migration operation today is currently heuristic and coarse-grained; finally, there is a need to factor in prediction inaccuracies and migration data-size (referred to as risks) in the decision-making.

This paper proposes SMARTMIG*: a framework for optimizing the storage utility by proactively scheduling data migration using time-series forecasts.* SMARTMIG *generates several plans for* what *data to migrate,* where *to migrate,* how *to migrate (i.e., the migration speed), and* when *to migrate. These plans are generated using constraint optimization, and their selection is modulated by risk analysis of the prediction accuracy and the migration overheads. For the experimental evaluation of* SMARTMIG*, we developed a detailed storage system simulator, and analyzed the quality of migration decisions made in different scenarios. Our results show that for a significant percentage of scenarios,* SMARTMIG *in a automated fashion minimizes the utility loss by 80% compared to no action invocation.*

1. Introduction

Growing consolidation of storage systems necessitates resource sharing among multiple competing applications with different access characteristics and *Service Level Objectives*(SLOs). The goal of storage management is to maximize the overall *utility*, by intelligently allocating the available storage resources among the applications based on their priorities and usage characteristics – the resource allocation decision is not a one-time task but rather needs to be continuously optimized for changes in the application's IO characteristics, changes in SLOs, occurrence of exception events (such as failures and load surges). Changing the resource allocation at run-time is accomplished by invoking corrective actions such as throttling, migration, replication and hardware provisioning. The domain of this paper is using data migration as a corrective storage management action.

Automated invocation of data migration is an area of ongoing research – it requires deciding *what* data-set to migrate, *where* to migrate, *when* to migrate, and *how* to migrate (the migration speed). There are several research projects [6, 18, 12] and commercial tools [2] that assist in deciding what data-set to migrate, where to migrate and the migration speed. However, existing techniques have the following limitations. First, decision-making is typically optimized for improving only the current system state rather than taking into account the forecasted trends in system load and workload usage. This limits the invocation of migration until after the SLOs are violated, rather than proactively preventing the violation from happening. Second, migration has been traditionally treated as a background task that is invoked at night when the system is lightly loaded. However, in utility-based computing, migration can be scheduled as a foreground task to correct resource bottlenecks for high priority applications. This necessitates techniques to decide *when* to schedule migration. Third, migration incurs the cost of moving Tera-bytes of data that might take several days in the real-world. This decision should be made carefully while taking into account the inaccuracies in load forecasts and component models (referred to as *risks*). In contrast to existing techniques, the goal of a migration planner is not to select a migration option that maximizes the overall utility, but rather maximizes utility with minimal *risk*.

This paper proposes SMARTMIG: a framework that makes decisions on *what* data-set to migrate, *where* to mi-

grate, *how* to migrate, and *when* to migrate such that the overall system utility is maximized. SMARTMIG uses a combination of optimization, planning, and risk evaluation schemes – the optimization phase decides the *what* and *where* by formulating it as a constraint optimization problem with the objective to maximize the overall system utility for a given provisioning window. The output of the optimization is not just a single solution but rather the $top-K$ options. For each of these options, the planning phase decides the *when* and *how* – it may be possible that there may not be a feasible when-how combination for all the top-K options. Finally, the short-listed migration plans (what, where, when, how) are analyzed for the level of risk involved versus the expected benefit. The plan with maximum utility and minimum risk is selected for invocation.

The key contributions of this paper are:

1. A migration scheme that takes into account not just the current system state but also the forecasted workload trends.

2. A constraint-based formulation of migration that is based on maximizing storage system utility for a configurable lookahead window.

3. A scheme for risk analysis that considers the migration overheads and the accuracy of future forecasting and selects the low risk, high benefit migration option.

We validate SMARTMIG by implementing it as a part of a file-system simulator – the models used in the simulator are derived from an actual enterprise class storage controller. Decisions in different scenarios are evaluated by comparing them with the "ideal" migration option (assuming perfect future knowledge), as well as when no corrective action is invoked.

The rest of the paper is organized as follows: Section 2 gives details of the related work. Section 3 describes the input modules of SMARTMIG. Section 4 gives details of the migration algorithm, followed by the experimental section in Section 5. The paper concludes and enumerates future work in Section 6.

2. Related Work

Several schemes have been proposed to determine the new data placement configuration (*what* and *where* decisions) or the migration speed (*how*). This section briefly reviews the existing techniques to automate data migration. The term workload refers to the streams of IO requests from the applications.

The traditional approach of selecting what to migrate (referred to as *migration candidates*) is based on *data temperature* defined as the quotient of the load (heat) and data size. Migration candidates are ranked in the descending order of temperature and the one with the highest temperature will be migrated to the least loaded component. The procedure is repeated till the system converges to a balanced state [11, 29]. Another scheme [23] for deciding migration candidates takes into account the periodic load pattern and performs an elaborate cost/benefit analysis before migration is invoked.

Recently, several migration techniques have been proposed as a part of the iterative observe-analyze-act loop. These algorithms use models to determine the component's ability to support workloads, and selects migration candidates and targets such that workload requirements can be satisfied. Hippodrome [6] is a storage system configuration tool that automatically adapts to changing workload demands without human intervention. It is structured as an iterative loop of analyzing workloads to determine their requirements, and creating a better storage system configuration to meet those goals. Based on the new storage system design, Hippodrome triggers migration to move data accordingly. Hippodrome uses a greedy approach to create a sequential plan for the migration such that the amount of scratch space (which is used to reduce the amount of data that needs to be moved) required is minimized. As a follow-on work, Anderson et. al. [5] performed an experimental study on offline migration algorithms which can create the migration plan for Hippodrome. They proposed migration algorithms with and without space and performance constraints. These were tested using different types of multi-graphs. Their results show that all algorithms actually perform much better in practice than the theoretical boundary. However, in schemes described above, migration is invoked either for load-balancing purposes or for initial system configuration. In this paper, our framework SMARTMIG is designed to take advantage of the well-known time-series forecasting techniques and maximize storage utility in a proactive fashion.

There is an interesting body of research on choosing the migration speed. Aqueduct [18] is an on-line migration algorithm based on control-theory. It adjusts the migration speed such that the front-end users (applications) receive a guarantee on the upper-bound of the latency. The latency bound is also estimated based on performance models. QoSMig[12] proposes an adaptive rate-control scheme for migration. It leverages the traffic shaping ability from Sleds [10] and proposes an on-line admission control algorithm. In order to enable a unified framework for migration and I/O requests, QoSMig assigns a "reward value" to each regular and migration request. Particularly, the reward for migration requests is based on the migration utility analysis and the expected distribution of arrival time of I/O requests. The rate control is enforced through an admission control algorithm that admits requests with maximum system re-

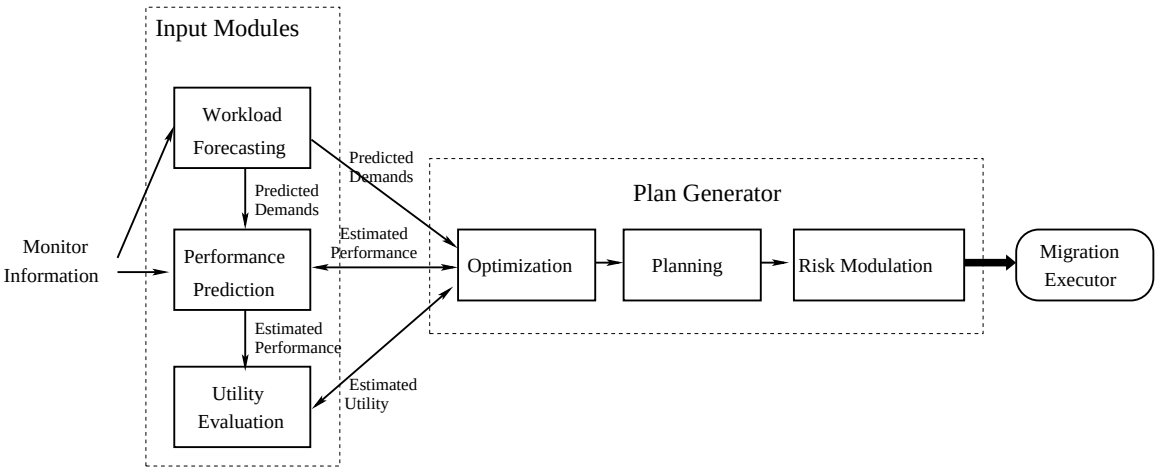

Figure 1. Architecture of SMARTMIG

ward. Both Aqueduct and QoSMig operate in an on-line manner - feedback loops and scheduling mechanisms are only useful when the migration is underway and they can not determine the migration speed before the migration is invoked. The ability to predict possible impact of various migration speeds is very useful for planning a migration beforehand. For this purpose, SMARTMIG applies a model-based approach to explore the design space and choose the "optimal" migration speed. However, models can never be accurate. A more practical scheme is to use a model-based approach to estimate the possible impact on the system and decide a rough range of migration speeds. Once the migration is invoked, on-line schemes like Aqueduct or QoSMig can be plugged in to provide safer and more precise speed control.

As mentioned earlier, since migration today is invoked reactively, the problem of *when* has been ignored by most schemes: they either assume migration will be invoked when the system is lightly loaded or manually invoked by the administrator whenever appropriate. SMARTMIG proposes a complete migration planning algorithm which will decide migration candidates and targets and migration speed as well as the best migration start time.

SMARTMIG applies several techniques ranging from time-series forecasting [27] and performance modeling [28, 4, 3, 8, 26] to constraint optimization [22, 19, 1] to generate a migration plan. Extensive research has been done in each individual area. SMARTMIG leverages these well-studied techniques in the domain of storage systems.

3. SMARTMIG Input Modules

The architecture of SMARTMIG is shown in figure 1. It takes forecasted workload demands, predicted component performance and utility value as input and generates a migration plan accordingly. The decision making is triggered both *reactively* (after SLOs are violated) or *proactively* (based on forecasted growth in workloads).

The rest of this section describes details of time-series forecasting, performance prediction and metrics for storage system utility. The details of decision making are covered in Section 4.

3.1. Time-series Forecasting

The forecasting of future workload demands is based on extracting patterns and trends from historical data. There are several well-known techniques for time series analysis such as ARIMA [27, 9], Neural Network [7], etc. The general form of a time-series function is as follows:

$$y_{t+h} = g(X_t, \theta) + \varepsilon_{t+h} \quad (1)$$

where: y_t is the variable(s) vector to be forecasted. t is the time when the forecast is made. X_t is the predictor variable, which usually includes the observed and lagged values of y_t until time t. θ is the vector of parameter of the function g and ε_{t+h} is the prediction error.

In our experiments, we use the ARIMA time-series algorithm to perform time-series analysis on HP's Cello99 trace [20] and the results are shown in Section 5.

3.2. Performance Prediction

The goal of performance prediction is to estimate storage component performance for any given workload demands and system settings. SMARTMIG uses the most commonly used performance metrics: throughput ($Thru$) and latency (Lat) as examples in the rest of the discussion.

There are several techniques for making performance predictions; these range along the spectrum of white-box and black-box approaches. White-box approaches

[26, 15, 21] establish equations using device specific information based on expert knowledge. Simulation based approaches [13, 30, 24] measure the performance of a configuration using a storage system simulator. Black-box approaches [14, 4, 28] require minimum expert input and device specific information and predict performance based on past historical information.

Because SMARTMIG needs to explore a large candidate space in a short time, simulation based approaches are not feasible due to their long prediction overhead. Both white-box and black-box approaches can be used in SMARTMIG. In the real-world, since the device specific information and expert input is often difficult to get, black-box techniques like table-based solutions[4] and regression models [28, 14] are more desirable for SMARTMIG. Specifically, in our experiments, we use models generated using regression techniques (results in Section 5).

3.3. Utility Evaluation

The concept of "utility" is introduced to evaluate the user's perception of 'satisfaction'. There are different ways to define the utility functions. In SMARTMIG, utility function associates workload performance with a utility value, which quantifies the user's degree of satisfaction. There are several techniques to define a utility function – we enumerate a few of them:

- *Provided by the administrators*. In some cases, the administrators may not have enough knowledge to define a good utility function. They typically use trial-and-error, trying multiple versions of the utility function to get the desired level of service.
- *Defined based on the SLOs and priorities.* Equation 2 gives one way of defining.

$$UF(Thru, Lat) = \begin{cases} 0 & \text{if } Lat > SLO_{lat} \\ \frac{Pri * min(Thru, SLO_{thru})}{SLO_{thru}} & \text{otherwise} \end{cases} \quad (2)$$

- *Defined based on price and SLOs.* The dollar amount is associated with the level of service received, e.g., \$1000/GB if the latency is less than 10ms, otherwise, \$100/GB.

Based on utility functions for each workload, the overall storage system utility value is defined as:

$$\begin{aligned} U_{sys} &= \sum_{j=1}^{N} U_j \\ &= \sum_{j=1}^{N} UF_j(Thru_j, lat_j) \end{aligned} \quad (3)$$

Where N is the number of workloads in the system, U_j, UF_j, $Thru_j$ and Lat_j are the utility value, utility function, throughput and latency for workload j respectively.

4. Migration Plan Generator

The decision making procedure consists of three phases: *Optimization phase*, *Planning phase* and *Risk Modulation phase*. This section covers the details of each of these phases.

4.1. Overview of the Decision-making algorithm

SMARTMIG's design goal is to maximize user's satisfaction for the given optimization window T, which is equivalent to minimizing system **U**tility **L**oss UL_{sys}, defined as follows:

$$\begin{aligned} UL_{sys} &= U_{max} - U_{sys} \\ &= \sum_{j=1}^{N} U_{max_j} - \sum_{j=1}^{N} U_j \\ &= \sum_{j=1}^{N} UF_j(D_j, SLO_{j_{lat}}) \\ &\quad - \sum_{j=1}^{N} UF_j(Thru_j, Lat_j) \end{aligned} \quad (4)$$

Where D_j is the demand of workload j and $(Thru_j, Lat_j)$ is the achieved performance of workload j. U_{max_j} is the "ideal" utility value if all of workload j's requests can meet the SLO goal and U_{max} is the maximum system utility value.

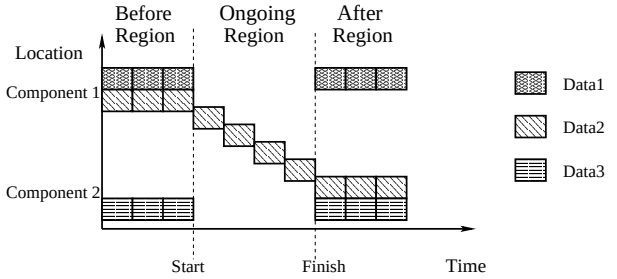

Figure 2. Migration Regions

The migration operation is partitioned into three regions: *Before* migration happens, when the migration process is *Ongoing* and *After* migration finishes (shown in Figure 2). The motivation of the partition is because each region behaves differently and is affected by different migration parameters. For a given optimization time window T, SMARTMIG aims to choose *what*, *where*, *how*, *when* that

can minimize the overall system utility loss over three regions, shown in Equation 5.

$$UL(T) = UL_{Before} + UL_{Ongoing} + UL_{After} \quad (5)$$

In order to achieve minimum overall utility loss, each parameter has to be carefully selected. For example: *what* and *where* will affect the *After* utility, *how* will change the *Ongoing* behavior and *when* will affect the region boundary. In addition, these parameters are not independent and will affect each other. For example, solutions leading to maximum *After* utility may not lead to optimal *Ongoing* utility because the best placement configuration may involve larger data movement, and introduce more utility loss during the migration operation. Therefore, the ideal optimal solution should consider the interaction among four parameters and examine all possible combinations. However, full combination scanning introduces a very high complexity: even a simple version of the sub-problem *what* and *where* is an NP-complete bin-packing problem. SMARTMIG trades the optimality to reduce the complexity. It breaks the decision procedure into three phases and uses the final goal of optimizing the system utility to guide the design of each phase. The three phases are:

- *Optimization Phase*: Finds the top-K answers for *what* to migrate and *where* to migrate.
- *Planning Phase*: Finds the best *when* and *how* options for each of the top-K $< what, where >$ pairs.
- *Risk Modulation Phase*: Evaluates the risk associated with each migration plan and selects the one leading to maximum utility and minimum risk.

In each phase, with the assistance of time series prediction, performance models and utility evaluation, SMARTMIG can estimate the decision impact on the system and explore the design space very quickly. In the rest of the section, we will discuss the design details for each phase.

4.2. The Optimization Phase: *What* and *Where*

The *what* and *where* decide how the migration operation wants to alternate the resource allocation such that the system can operate in a better state (higher utility value). It reflects the "permanent" effect of migration and the final goal of finding the best data placement plan that minimizes system utility loss. This problem is a data placement problem and can be formulated as a classical constraint optimization problem (shown in Table 1).

A traditional data placement problem can be reduced to an NP-complete bin-packing problem by mapping the components as bins and workload resource requirements as objects. For SMARTMIG, the objective function of optimizing the system utility makes the problem even more complicated because: (1) "Size of bin" is not fixed because workloads are interleaved and affect each other's performance. As a result, the capability of each component (bin size) is not static and is changing as the set of workloads running on it changed. (2) "Size of object" (utility value) is not static. The utility value is a function of received performance and varies with the change of data placement. To reduce the complexity, SMARTMIG applies the classic greedy technique to find the approximated optimal solution (shown in the flow chart 3).

Variable	s_{ij}
Minimize	$\sum_{i=1,j=1}^{N,M} s_{ij} UF_i(Perf_{ij})$
Subject to	$\sum_{j=1}^{M} s_{ij} = 1$ for all i=1 to n
	$S_{ij} = 1$ if data i is placed on component j
	$S_{ij} = 0$ otherwise
	Where s_{ij} is the optimization variable and reflects if data i will be placed on component j or not. UF_i is the utility function of workload i and $Perf_{ij}$ is the predicted performance of workload i on component j. Please notice that the $Perf_{ij}$ is related to the data placement configuration.

Table 1. Constraint optimization of *what* and *where*

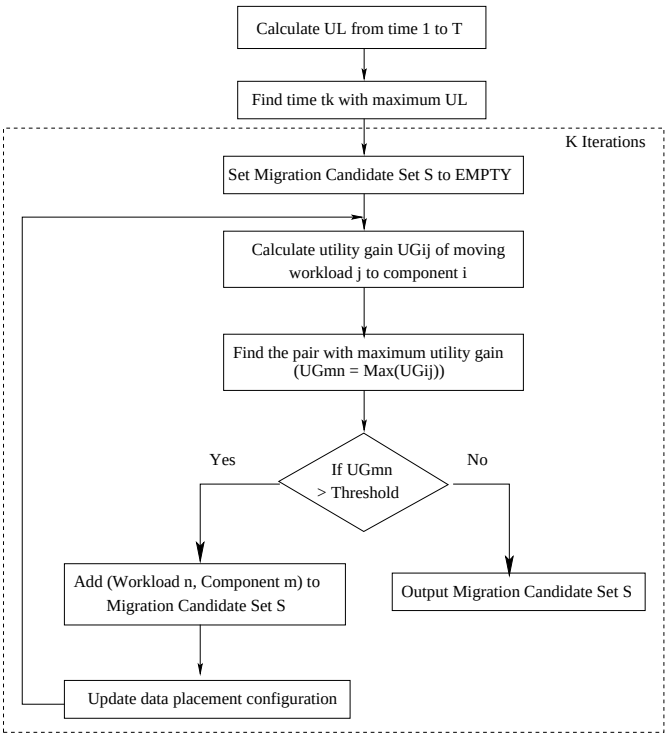

Figure 3. Flow Chart of Optimization Phase

In the flow chart, t_k is the time when maximum utility loss happens in the given optimization window T. For a different T value, maximum utility loss may happen at a different time (t_k) and therefore, the *what* and *where* decisions are optimized for different system settings. In each step of the greedy operation, for every ($workload_j$, $component_i$) pair, the utility gain UG_{ij} of moving workload j to component i is estimated. Here utility gain UG is defined as the utility difference of the new data placement from the old one. The pair leading to the maximum utility gain (shown as ($workload_n$, $component_m$) in the figure) is put into the migration candidate set. The greedy procedure repeats until the utility gain of moving any workload is marginal–less than some *threshold*, where the *threshold* reflects the trade-off between the convergence speed and the quality of the solution.

The *Optimization phase* does not return a single solution, but rather the $top-K$ options – the greedy procedure is repeated K times, such that after each iteration, the workload (dataset) with the minimum $UG/SIZE$ ratio is blocked from being considered in the remaining iterations. Intuition is to eliminate the low benefit (UG) and high cost ($SIZE$) candidates.

4.3. The Planning Phase: *How* and *When*

For each solution returned by the *Optimization phase*, the *Planning phase* generates the detailed plan for when to start migration and the corresponding migration speed. The *how* decision is relatively straight-forward as the migration speed at time t_i is only affected by the workload demands and the system setting at t_i. However, the *when* decision is more complicated and decisions on *what*, *where* and *how* will all affect the final decision on *when*. For example, if the utility value of the new data placement is much higher than the old one, it is desirable to start migration as early as possible. On the other hand, if the data takes very long to migrate and introduces high utility loss in the procedure, a more preferred solution is to wait until the system is lightly loaded. Therefore, the migration start time can only be determined after other decisions have been made. In the rest of this section, we describe the *how* aspect first and then the algorithm for *when*.

***how*: Finding Migration Speed**

The objective of migration speed is also to minimize system utility loss. Specifically, the migration process will (1) introduce extra utility loss because it will compete with application workloads for the already limited resources. But at the same time (2) it will benefit the system once the migration process is finished. Because of the bi-directional impact of the migration process, it is very difficult to quantify the exact impact on the utility loss for different migration speeds. SMARTMIG settles for the approximated solution based on the intuition of migrating more data when the system is lightly loaded and less when the system is busy; it chooses a migration speed that is proportional to the system's spare resource. In addition, SMARTMIG also makes suggestions on the resource allocation plan for other workloads using a greedy approach such that the limited resources are allocated to higher priority workloads, resulting in a better utility value. By correlating migration speed with system utilization status, and throttling low priority workloads, SMARTMIG minimizes the extra utility loss due to the migration procedure. The *how* decision procedure is summarized as follows:

1. Estimate system utilization: $Sys\_Utilization = \frac{TotalLoad}{MaximumLoad}$.
2. Set migration speed as $MigSpeed = (1 - Sys\_Utilization) * p * MAX\_SPEED$, where p is a number between 0 to 1 and is changing according to which spectrum of utilization the system is operating in. The heavier the system is loaded, the smaller the p is. The MAX_SPEED is the maximum migration sending rate allowed by the system and it reflects how aggressively the migration can perform.
3. Search the optimal resource allocation plan (sending rate) for other workloads with the goal of maximizing system utility. The greedy procedure stops when no utility benefit can be gained by increasing the sending rate of any workload, or all workloads have all their requests satisfied.

Using the algorithm described, the migration speed for time 0 to T can be determined, represented as $MigSpeed_{t_i}$. We are now ready to determine the migration start time t^*

***when*: Choosing Migration Start Time t^***

The decision on *when* is not always straight-forward and needs to consider several parameters. For example, for the same system state, the migration should be invoked immediately if the system load has an expected growth trend, but should be delayed if very soon the system will be lightly loaded. Similarly, if the migration dataset can finish very quickly and if the new configuration can introduce a high utility gain, the migration operation should start immediately, but on the other hand, if it takes longer to finish, it may be preferable to delay it until the system is lightly loaded. As we will show in the experimental section, for the same system settings and for different migration data sets, the best start time changes accordingly. In general, the *when* decision is related to the answers to *what*, *where* and *how* and also to the future state of the system. SMARTMIG selects the best start time t^* by considering the details

of the migration option, the performance and utility information of the system, and the future workload trend.

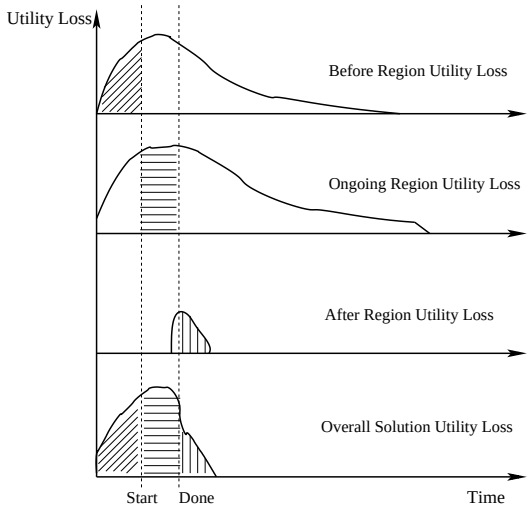

Figure 4. Overall Solution Utility Loss

Figure 4 illustrates how to derive the overall system utility loss. The x axis is the time and y axis is the utility loss. The *Before Region Utility Loss* curve plots the "expected" utility loss under the old data placement configuration at each time in the optimization window. The *Ongoing Region Utility Loss* gives information on the expected utility loss if migration is going on and the *After Region Utility Loss* shows the utility loss under the new data placement setting. The overall utility loss for a migration option can be derived by using the *Before* curve before migration starts, the *Ongoing* curve during migration and the *After* when it is done (shown in the *Overall Solution Utility Loss* curve in the figure). For example, for a given migration solution starting from t_i and ending at $t_i + m_{ti}$, where m_{ti} is the lead-time if migration starts at t_i, the overall utility loss is to use *Before* utility loss for $[0, t_i - 1]$, *Ongoing* for $[t_i, t_i + m_{ti}]$ and *After* for $[t_i + m_{ti} + 1, T]$ (shown in Equation 6).

$$\begin{aligned} UL(T,t_i) &= UL_{Before} + UL_{Ongoing} + UL_{After} \\ &= \sum_{l=1}^{i-1} UL_{tl_{sys}} + \sum_{l=i}^{i+m_{ti}} UL^1_{tl_{sys}} + \sum_{i+m_{ti}+1}^{T} UL^2_{tl_{sys}} \end{aligned} \quad (6)$$

where UL, UL^1 and UL^2 are utility loss for the Before, Ongoing and After region respectively.

Knowing parameters of *what*, *where* and *how*, the UL, UL^1 and UL^2 can all be easily calculated: for each time point l, $UL_{tl_{sys}}$ can be calculated by estimating workloads performance for the old data location settings and the utility value can be derived based on the performance. Similarly, for $UL^1_{tl_{sys}}$, we can estimate the performance by considering workload demands and migration speed at time t_l and calculate the utility value accordingly. $UL^2_{tl_{sys}}$ is calculated under the new data location setting. In addition, the migration lead-time m_{ti} can be estimated based on $MigSpeed_{t_l}$ and migration size $TotalMigSize$:

$$m_{ti} = min(\text{k such that } \textstyle\sum_{j=t_i}^{k} MigSize_{tj} \geq TotalMigSize) \quad (7)$$

where $TotalMigSize$ is the total size of the migration candidates and $MigSize_{tj} = MigSpeed_{tj} * Len_{tj}$ is the data moved in interval t_j with Len_{tj} as the interval duration.

Clearly, the best start time t^* is the t_i that leads to minimum $UL(T,t_i)$ and can be found by scanning the whole optimization window.

In summary, the *Planning phase* determines the migration speed along with the best start time t^* for each of the K *what*, *where* options from the *Optimization phase*. The K (*what*, *where*, *how*, *when*) solutions with the corresponding overall system utility loss UL_{M_k} are then analyzed in the *Risk Modulation Phase* to select a low risk, high benefit option.

4.4. Risk Modulation Phase

The goal of SMARTMIG is to assist the administrator in finding a good migration option. The *Optimization phase* and *Planning phase* aim to find options leading to lower utility loss (higher utility). But because time series forecast can have errors and migration operations are not cost-free (they will consume system resources to move data around), there is a risk associated with each migration option. The goal of the *Risk Modulation phase* is to modulate the risk of each migration option and return the one with low risk and high benefit.

Risk captures the probability that the utility improvement of action invocation will be lost (in the future system-states) as a result of volatility in the workload time-series functions e.g., the demand for W_1 was expected to be $10K$ IOPS after 1 month, but it turns out to be $5K$. Additionally, the formulation of risk should take into account the loss in utility as a result of making the wrong decision e.g., moving data at 11am in a weekday morning *(during high system utilization)* has a higher risk compared to moving it at 9pm on a weekend (during low system utilization) – the utility lost due to a wrong decision is higher in the former case than the latter. Similarly, the impact of the wrong decision is dependent on the amount of data moved.

There are several techniques for measuring risk – actions for assigning storage resources among workloads are analogous to portfolio management in which funds are allocated to various company stocks. In economics and finance, the *Value at risk*, or VaR [16], is a popular technique used to estimate the probability of portfolio losses based on

the statistical analysis of historical price trends and volatilities in trend prediction. In the context of SMARTMIG, *VaR* represents the probability, with a 95% confidence, that the workload system will not grow in the future (optimization window T), making the migration invocation unnecessary.

$$VaR(95\%\, confidence) = -1.65\sigma \times \sqrt{T} \tag{8}$$

where:
σ = Standard deviation of the time-series request-rate predictions.

The risk value $RF(M_k)$ of migration solution k is calculated as follows:

$$RF(M_k) = -(1+\alpha_{M_k}) * VaR \tag{9}$$

where α reflects the risk factor of a migration option and is defined as follows:

$$\alpha_{M_k} = \frac{bytes\_moved_{M_k}}{total\_bytes\_on\_source} * Sys\_Utilization_{M_k} \tag{10}$$

where $Sys\_Utilization$ is the system utilization when the action is invoked and is estimated similarly as in Section 4. Intuitively, a higher system utilization or larger migration data set will lead to a larger α_{M_k} and therefore, a larger $RF(M_k)$ value, which reflects the fact that the migration operation has higher risk and is less preferred.

For the K solution output by the *Planning phase*, the *Risk modulation phase* will calculate the risk value $RF(M_k)$ for each of them and scale the overall utility loss (shown in Equation 11). The goal of the scaling is to balance the benefit (UL_{M_k}) and the risk ($RF(M_k)$) of a migration option.

$$UL^*_{M_k} = (1 + RF(M_k)) \times UL_{M_k} \tag{11}$$

After the scaling, only the option with minimum $UL^*_{M_k}$ remains. Lastly, because migration always involves cost to move data around, the *Risk Modulation phase* uses a threshold to filter migration operations with marginal benefit ($UL^*_{M_k} > UL_{threshold}$) and returns a zero solution if the remaining migration option cannot bring the system enough benefit to justify the risk.

5. Experimental Results

The goal of the experimental evaluation is to evaluate SMARTMIG's ability to generate feasible and efficient migration plans. The experimental evaluation is divided into three parts: First, understanding the accuracy of time-series forecasting and component performance models – the analysis uses data collected from real-world systems. Second, a sanity check of SMARTMIG by testing its ability to make decisions for different combinations of input parameter values – this evaluation uses storage system simulator. Third, an efficiency test of SMARTMIG i.e., evaluating the runtime complexity by stressing SMARTMIG with different numbers of workloads and components. We also evaluate the sensitivity of the migration plans to model-errors.

5.1. Input Information Quality Examination

SMARTMIG gets as input the time-series forecast and component performance models – this information is used for decision making, and affects the accuracy of the migration plans.

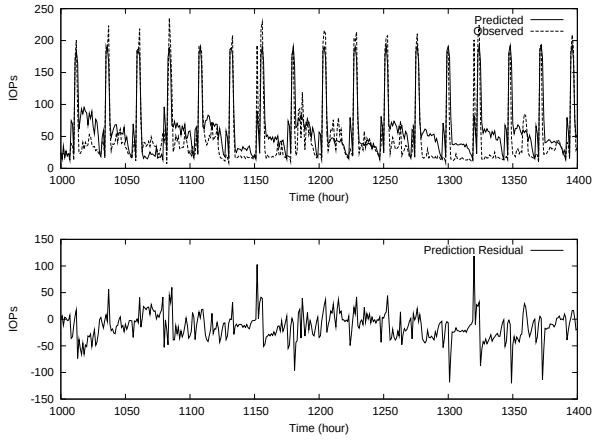

Figure 5. Results of ARIMA based forecasting of the Cello99 traces– the models are trained with the historical information of 1000 hours a) Forecasting workload patterns for the next 400 hours (b) Prediction residual for 400 hour forecasting

For the time-series forecasting, we choose a random 2-month interval (Nov.1 to Dec.30) in HP's Cello99 trace [20] – the average system load is sampled on a per-hour basis - the first 41 days are treated as training data and the remaining days as testing data. The best-fit ARIMA model for the data is ARMA(4,3) and SARMA(4,3). As seen in the forecasting results in Figure 5, more than 60% of the residuals are less than 30 IOPS off the real value and more than 80% are within a 50 IOPS difference.

For the component model, we apply the regression tree based algorithm GUIDE [17] and model the latency as a function of workload features that include IO sending rate, read-write ratio, random-sequential ratio, request size and footprint size. Training data is generated by varying the features of individual workloads. The quality of the latency model derived using 200 training points with 5 interleaving file-system flows is shown in Figure 6. The observed latency has a mean of 0.26 seconds; the correspond-

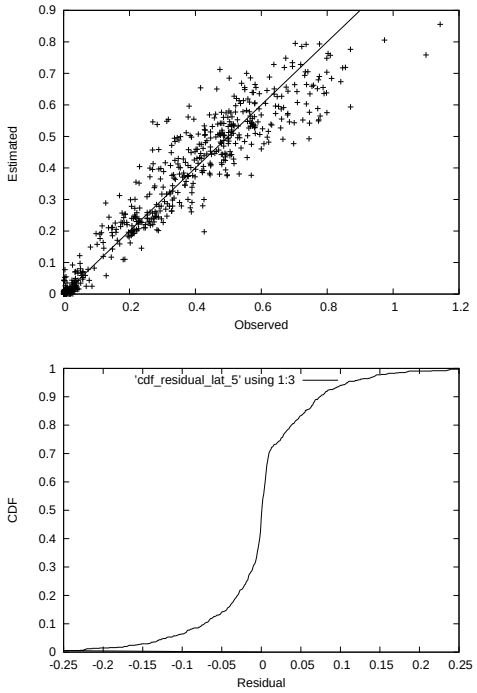

Figure 6. Results of GUIDE based component model – the models are trained with 200 sampling points with 5 fbws interleaving in the system a). Observed latency vs. Predicted latency. The x=y line is plotted together b). CDF of performance prediction residual

ing residual has a mean of 0.04 seconds with nearly 90% of the residual less than 0.1 seconds.

Although these results are by no means complete, they show that the techniques for time-series forecasting and machine learning lead to reasonable workload forecasting and component model results respectively.

5.2. Sanity Check and Efficiency Test

We implemented SMARTMIG as a part of a file-system system simulator. The simulator takes as input the original system state (i.e., old data placement configuration), workload forecast, component performance models, utility functions and SLOs, and generates as output the migration decisions. The scenarios for testing were generated using permutations of the following configuration parameters:

- *Initial data placement*: we intentionally create an unbalanced system (60% of the workloads will go to one component and the other half are distributed to the rest of the components randomly). The rationale behind this design is to make the migration operation necessary.
- *Workload features:* The sending rate and footprint size of each workload are generated using a Gaussian mixture distribution – with a high probability, the sending rate (or footprint-size) is generated using a normal distribution with a lower mean value and with a low probability, it is generated using another normal distribution with a larger mean value. The reason for using the Gaussian mixture distribution is to mimic the real-world system behavior: a small number of applications contribute a majority of the system load and access the majority of data.
- *Workload trending*: To mimic workload changes in real systems, we changed the access rates of the workloads over time. In our experiments, 30% of workloads were increased, and another 30% were decreased. In particular, the increasing step size is generated using a random distribution with a mean of 1/10 of the original load and the decreasing step size is randomly distributed with mean of 1/20 of the original load.

We vary the number of workloads in the system from 10 to 100 and generate 10 scenarios automatically for each of them. Out of the 100 scenarios, 14 of them did not call SMARTMIG because the initial setting did not cause any utility loss. The remaining 86 scenarios experienced utility loss to various degrees, ranging from 0.7% to 55% of the maximum system utility. We did not generate scenarios with more serious utility loss because migration may not be the correct solution in that situation (for example, new hardware should be requested). Figure 7 plots the Cumulative Distribution Function (CDF) of the percentage of utility loss (defined as the $\frac{Max\_Utility - No\_Action\_Utility}{Max\_Utility}$). As shown in the figure, more than 55% of the scenarios experienced more than 10% utility loss, out of which around 30% had a higher than 20% utility loss.

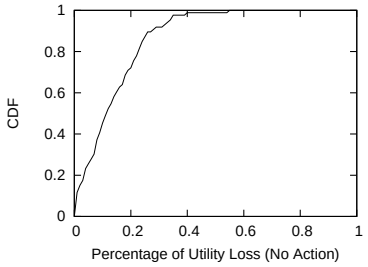

Figure 7. CDF of percentage of overall utility loss without migration operation

5.2.1. Sanity Check Test 1: Working of Individual Piece

We randomly pick one scenario from the 86 cases and examine the working of each phase. The selected scenario had 20 workloads distributed on 4 components. The initial data placement caused a 7.8% utility loss. SMARTMIG was called with a 14 day optimization window. The top 5 solutions returned are summarized in Table 2.

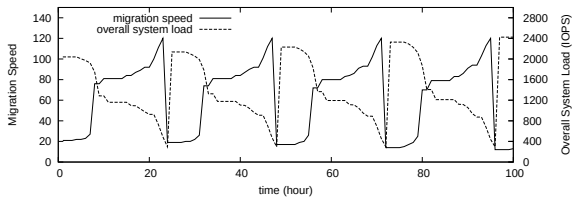

Figure 8. Migration Speed Vs. Overall System Load. X axes is the time index with the left Y axes as the migration speed and the right Y axes as the overall system load

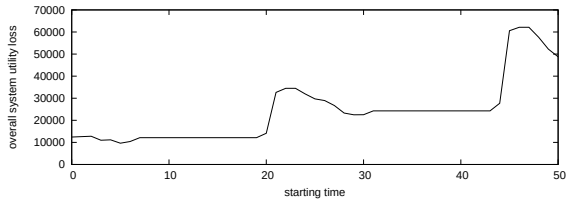

Figure 9. Overall utility loss for various migration start times

The table shows the *Optimization phase* found five sets of (*what*, *where*) decisions. For each of them, the *Planning phase* selected different start times, and the *Risk Modulation phase* assigned a risk value accordingly. Solution 1 is finally selected because it leads to minimum utility loss. The risk analysis assumes the standard deviation (σ) for a one day forecast is 10% of the real value. In addition, Figure 8 shows the detailed decision on the migration speed – it changes according to the load in the system. The start time selection is shown in Figure 9 (only the first the 50 hours are plotted for better visibility). The x axis is the start time and the y axis shows the corresponding overall system utility if migration starts at time x. As shown in the figure, the minimum overall system utility is achieved if migration starts at 5 (when the system is lightly-loaded for the first time).

Test 2: Impact of Optimization Window *T*

SMARTMIG is designed to find the "best" migration option for a given optimization window T. For different Ts, SMARTMIG may return different migration parameters which fits better for the given T. Using the same settings as in the previous test, we changed the optimization window from 14 days to 7 days. SMARTMIG returned different *what* and *where* answers. The five solutions returned are shown in Table 3.

#	Migration Candidates and Targets	Size (GB)	Scaled Utility Loss
1	(1: 0→2)	14	11308
2	(6: 0→1)	12	87850
3	(3: 0→2) (5: 0→2)	17	11501
4	(4: 0→2) (5:0→2)	140	42282
5	(5: 0→2) (13:0→3)	16	11436

Table 3. Solutions Returned By SMARTMIG with a 7 Day Optimization Window

After risk modulation, solution 1 will be selected and workload 1 will be migrated from 0 to 2, which is different from the decision in the previous test.

Test 3: Impact of Utility Configuration

SMARTMIG aims to optimize system utility and the utility functions for each workload will affect the solution returned by SMARTMIG. In this test, we stay with the same settings as in Test 1 and change the utility functions of three workloads. The results returned by SMARTMIG are given in Table 4:

#	Migration Candidates and Targets	Size (GB)	Scaled Utility Loss
1	(17: 0→2) (3: 0→2) (1: 0→2)	30	2949
2	(17: 0→2) (3: 0→2)	16	23700
3	(17: 0→2) (4: 0→2) (5: 0→3)	144	15813
4	(17: 0→2) (5: 0→2) (13: 0→3)	20	2796
5	(5: 0→2) (17:0→2)	9	16200

Table 4. Five Solutions Returned By SMARTMIG With Different Utility Configuration

After risk modulation, solution 4 is selected, which is different from the one selected for the original system settings and optimization window. These tests demonstrate SMARTMIG's ability to optimize the migration decisions for different values of input parameters and system configuration.

5.2.2. Efficiency Test Test 1: Percentage of Utility Loss Elimination

#	Migration Candidates and Targets	Size (GB)	Utility Loss	Start Time (hour)	Scaled Utility Loss
1	(5: 0→2) (1: 0→2)	19	10408	5	11629
2	(5: 0→2) (3: 0→3) (4: 0→1)	152	22552	4	43715
3	(5: 0→2) (3: 0→3) (13: 0→3)	28	10408	1	12208
4	(5: 0→2) (13:0→3) (14: 0→2)	118	22552	5	38981
5	(5: 0→2) (13:0→3) (17: 0→2)	20	10408	4	11694

Table 2. Solutions Returned By SMARTMIG with a 14 Day Optimization Window

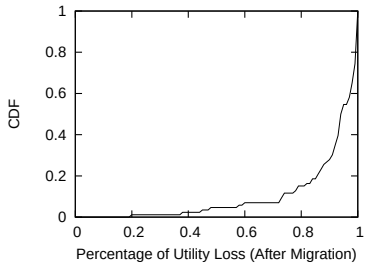

Figure 10. CDF of percentage of overall utility loss savings with SMARTMIG

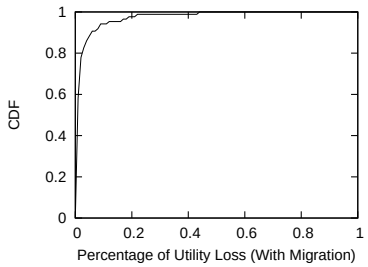

Figure 11. CDF of percentage of overall utility loss with SMARTMIG

For each of the 86 scenarios, SMARTMIG makes decisions in three phases and returns the one with the minimum scaled utility loss. In order to understand how much the system is saving by invoking the migration operation returned by SMARTMIG, we measure the percentage of loss in utility saved by SMARTMIG, defined as $\frac{No\_Action\_Utility\_Loss - Utility\_Loss\_With\_Migration}{No\_Action\_Utility\_Loss}$. From the CDF curve (Figure 10), we observe that SMARTMIG successfully eliminates 80% of the utility loss in more than 80% of the cases and for the rest, it saves about 60% in another 12% of the cases. The final percentage of utility loss is plotted in Figure 11. For more than 90% of the cases, the storage system exhibits less than 0.7% utiltiy loss, which is a big improvement compared to Figure 7.

Test 2: Computational Overhead of SMARTMIG

Figure 12 plots the computational overhead (time taken) of SMARTMIG. We ran the simulator on a Linux machine

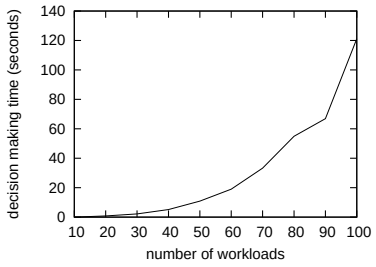

Figure 12. Computation Overhead of SMARTMIG

with a Pentium 4, 2.66GHZ CPU and 512MB of memory. We varied the number of workloads in the system and recorded the time SMARTMIG took to generate a migration plan. Ten cases are generated for each workload number setting and the average time overhead is plotted in the figure. The curve grows exponentially with the number of workloads. The majority of the overhead is from the migration speed determination phase, where SMARTMIG greedily searches for the optimal resource allocation for each workload. If SMARTMIG does not try to optimize the other workload's sending rate, the exponential effect will be gone. However, the system may experience a higher utility loss due to un-regulated resource competition.

Test 3: Sensitivity Test of Performance Model Errors

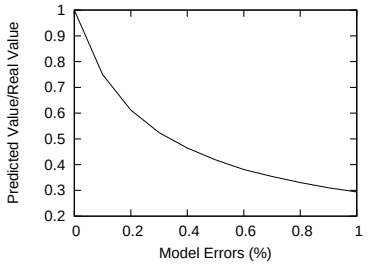

Figure 13. Impact of Model Errors on the accuracy of predicted utility loss

Our sanity check experiments were based on the assumption that perfect component models are available. However, in reality, this is not always true. To measure

the sensitivity of the migration decisions to the model errors, we generated a set of synthetic component models that SMARTMIG uses for decision-making. The latency calculated using these models is labeled as the "predicted latency" and the corresponding utility loss as the "predicted utility loss". For the exact same settings, the "real latency" is simulated using the synthetic model and offset by a random error following a normal distribution. Because the residual normally grows with the real value, we generated a random scaling factor rather than the absolute error. For example, if the latency is 10ms and the random error is 0.2, the real latency is 10*(1+0.2) = 12ms. After this, the returned "real latency" is used to calculate the "real utility loss" for the decision. By doing this, we have full control of the model error rate and the sensitivity test is possible.

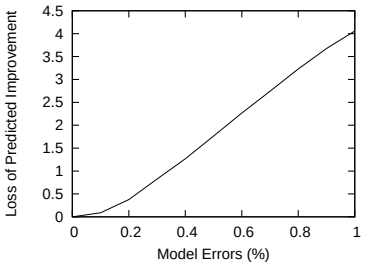

Figure 14. Impact of model errors on the percentage of predicted utility saving

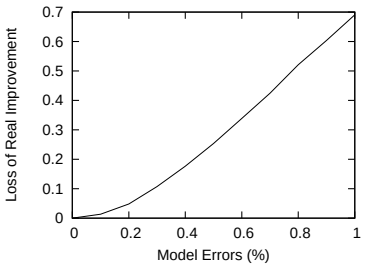

Figure 15. Impact of model errors on the percentage of real utility saving

Figure 13 is the ratio of the *Predict_Utility_Loss* and *Real_Utility_Loss*; it captures the accuracy of the predicted results. As shown in the figure, the accuracy of the predicted value drops quickly with a higher error rate. With an error rate of 0.2, the predicted value only captures about 60% of the real value. In addition, Figure 14 plots the loss of saving percentage, which is defined as $\frac{Real\_Utility\_Loss - Predict\_Utility\_Loss}{Predict\_No\_Action\_Utility\_Loss}$. Similarly, we found for a model error rate of 0.2, nearly 37% of the savings is gone. However, if we measure the loss of the real saving percentage, which is defined as $\frac{Real\_Utility\_Loss - Predict\_Utility\_Loss}{Real\_No\_Action\_Utility\_Loss}$, the mismatch percentage is much smaller (shown in Figure 15). The reason is that the real system's utility loss can never be correctly predicted and therefore, the under-estimation of the utility loss without any migration in the real system balances out part of the under-estimation of the utility loss with any migration.

These three figures suggest that if the model error rate is less than 20%, decisions made based on them have reasonable accuracy and can provide useful information. Otherwise, decisions returned by SMARTMIG need to be carefully considered before the action is invoked. Fortunately, as shown in Figure 6, the performance model derived using the real system has a 0.04 second average residual for the real latency with a mean of 0.26 seconds. That is about a 15% relative error. Many previous studies also demonstrate a well-constructed model can predict system performance with reasonable accuracy [28]. Therefore, it is very possible that we can derive a model with less than a 20% error rate.

6. Conclusion and Future Work

This paper describes SMARTMIG – a proactive data migration framework that uses both the current and forecasted system states for making migration decisions. Additionally, SMARTMIG selects migration options where the improvement in system utility is proportional to the risk involved. A proof-of-concept implementation for SMARTMIG demonstrates the feasibility of the formulation and its sensitivity to model accuracy. As ongoing and future work, we are working on the following: First, implementing SMARTMIG as a part of GPFS [25]: a commercial high-performance file-system. Second, as the experimental results show a strong correlation between model accuracy and precision of the migration decisions, we are exploring a two-pronged approach to better model accuracy and less sensitive migration formulation. Third, we are developing pruning techniques to reduce the computation overheads of the optimization, making SMARTMIG scalable in large data-center and scientific deployments.

Acknowledgment

We want to thank the reviewers for their comments.

References

[1] Glpk (gnu) linear programming kit. http://www.gnu.org/software/glpk/glpk.html.

[2] IBM TotalStorage. http://www-1.ibm.com/servers/storage.

[3] G. Alvarez, K. Keeton, E. Riedel, and M. Uysal. Characterizing data-intensive workloads on modern disk arrays.

4th. Workshop on Computer Architecture Evaluation using Commercial Workloads, Jan. 2001.

[4] E. Anderson. Simple table-based modeling of storage devices. Technical Report HPL-SSP-2001-4, HP Laboratories, July 2001.

[5] E. Anderson, J. Hall, J. D. Hartline, M. Hobbs, A. R. Karlin, J. Saia, R. Swaminathan, and J. Wilkes. An experimental study of data migration algorithms. In *WAE'01: Proceedings of the 5th International Workshop on Algorithm Engineering*, pages 145–158, London, UK, 2001.

[6] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running circles around storage administration. *Proceedings of Conference on File and Storage Technologies (FAST)*, pages 175–188, Jan. 2002.

[7] M. E. Azoff. Neural network time series forecasting of financial markets. 1994.

[8] E. Borowsky, R. Golding, P. Jacobson, A. Merchant, L. Schreier, M. Spasojevic, and J. Wilkes. Capacity planning with phased workloads. *Proceedings of the first international workshop on Software and performance*, pages 199–207, 1998.

[9] P. J. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting*. Springer, 2002.

[10] D. Chambliss, G. A. Alvarez, P. Pandey, D. Jadav, J. Xu, R. Menon, and T. Lee. Performance virtualization for large-scale storage systems. *Proceedings of the 22nd Symposium on Reliable Distributed Systems*, pages 109–118, Oct. 2003.

[11] G. Copeland, W. Alexander, E. Boughter, and T.W.Keller. Data placement in budda. 1988.

[12] K. Dasgupta, S. Ghosal, R. Jain, U. Sharma, and A. Verma. QoSMig: Adaptive rate-controlled migration of bulk data in storage systems. In *Proceedings of IEEE International Conference on Data Engineering 2005*, Apr. 2005.

[13] G. R. Ganger, Y. N. Worthington, and B. L. A. Patt. The DiskSim simulation environment version 1.0 reference manual. Technical Report CSE-TR-358-98, 27 1998.

[14] F. Hidrobo and T. Cortes. Towards a zero-knowledge model for disk drives. In *AMS '03: Proceedings of the fifth annual international workshop on Action Middleware Services*, pages 122–130, June 2003.

[15] E. K. Lee and R. H. Katz. An analytic performance model of disk arrays. *SIGMETRICS Perform. Eval. Rev.*, 21(1):98–109, 1993.

[16] T. J. Linsmeier and N. D. Pearson. Risk measurement: An introduction to value at risk. 1996.

[17] W.-Y. Loh. Regression trees with unbiased variable selection and interaction detection. 12:361–386, 2002.

[18] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: online data migration with performance guarantees. *Proceedings of Conference on File and Storage Technologies (FAST)*, pages 175–188, Jan. 2002.

[19] Z. Michalewicz and D. B. Fogel. *How to Solve it: Modern Heuristics*. Springer, 2004.

[20] C. Ruemmler and J. Wilkes. A trace-driven analysis of disk working set sizes. Technical Report HPL–OSR–93–23, Palo Alto, CA, USA, May 1993.

[21] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *Computer*, 27(3):17–28, 1994.

[22] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, 2003.

[23] P. Scheduermann, G. Weikum, and P.Zabback. Adaptive load balancing in disk arrays. 1993.

[24] J. Schindler and G. Ganger. Automated disk drive characterization, 1999.

[25] F. Schmuck and R. Haskin. Gpfs: A shared disk file system for large computing clusters, 2002.

[26] E. Shriver, A. Merchant, and J. Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 182–191, New York, NY, USA, 1998. ACM Press.

[27] N. Tran and D. A. Reed. ARIMA time series modeling and forecasting for adaptive i/o prefetching. *Proceedings of the 15th international conference on Supercomputing*, pages 473–485, 2001.

[28] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage device performance prediction with CART models. *SIGMETRICS Perform. Eval. Rev.*, 32(1):412–413, 2004.

[29] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback. Self-tuning database technology and information services: From wishful thinking to viable engineering. Aug. 2002.

[30] J. Wilkes. The pantheon storage-system simulator. Technical Report HPL-SSP-95-14, HP Laboratories, december 1995.