# A Comparison of Alternative Techniques to Estimate Response Time for Data Placement

Shahram Ghandeharizadeh, Shan Gao
Department of Computer Science
University of Southern California
Los Angeles, CA 90089, USA
shahram,sgao@cs.usc.edu

Chris Gahagan
BMC Software Inc.
2101 CityWest Blvd.
Houston, TX 77042, USA
chris_gahagan@bmc.com

**Abstract**

Technological advances in networking, mass storage devices, processor and information technology have resulted in a variety of data services in diverse applications such as e-commerce, health-care, scientific applications, etc. While the cost of purchasing technology is becoming cheaper, the same cannot be stated about the cost of *managing* an information infrastructure. In order to reduce this cost, one needs tools that empower system administrators to explain and reason about a storage subsystem's past performance, e.g., response time. Ideally, an administrator would employ these tools to speculate on both physical organization of data and hardware changes. With a hypothetical change, one may use the previously observed response times to quantify the expected enhancements. In this study, we investigate linear regression, a M/D/1 queuing model and SEER as three alternative techniques to estimate response time. All techniques enable an administrator to speculate on changes to the placement of data and its expected impact on response time. A choice between these techniques is a tradeoff between accuracy and space/computational complexity to estimate response time. In our experimental studies, SEER provides a higher accuracy by using more storage space and computational cycles.

## 1 Introduction

An economical trend in the area of mass storage is cheaper, faster devices that provide higher capacities. The same trend does not apply to the cost of *managing* an information infrastructure [6, 7]. Management cost includes (a) the overhead of an administrative staff to maintain systems and (b) the cost associated with down-times that render a service (data) unavailable. With the fast pace of technological changes, it is unrealistic to develop a tool that fully automates the management of a computing infrastructure. Instead, it is more appropriate to develop tools that empower administrators to reason about a system's behavior, identify limitations, speculate on solutions to address these limitations, and evaluate the impact of their proposed solutions. These tools must be extensible in order to incorporate new hardware solutions as they become available.

Response time is an important performance criterion for many applications. A slow system might appear as unavailable to the end user, contributing to the down-time costs. The focus of this paper is to evaluate the alternative techniques that estimate system response time. These techniques are intended for those environments where: a) a request employs only one resource, and b) the environment gathers and stores data about user requests and system behavior over a period of time.

A simple approach to computing response time is to maintain a simulator that models the critical components of an application. The system collects a trace of when a request arrives and how much resources it consumes. When the system administrator inquires about a proposed change, the system invokes the

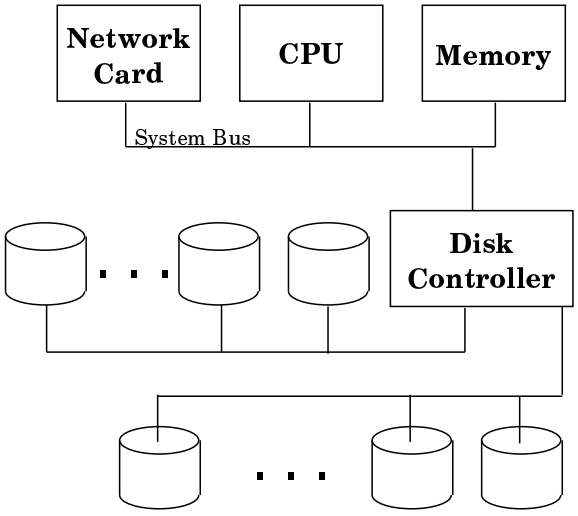



Figure 1: A multi-disk hardware platform.

simulator with the gathered traces in order to estimate the change in response time. This approach, termed Absolute, would provide a fairly accurate estimate of the anticipated response time (assuming the simulator models system components accurately). It would require some storage space for the trace data and CPU cycles to invoke the simulator. With the current technological advances, Absolute might become feasible in the near future.

The focus of this paper is on techniques to estimate response time of a system. Even with Absolute becoming feasible, these techniques continue to be of use for several reasons. First, they will always be faster than Absolute because they have a lower complexity. Second, they provide a user with an approximate view of the anticipated response time improvements. The user may then invoke Absolute on those temporal regions that are of interest. This is similar to an art critic previewing low-resolution images of many paintings. For those paintings (regions) of interest, the critic retrieves full-resolution images (invokes Absolute).

In order to have a focused description of the alternative techniques, and without loss of generality, assume a multi-disk computer as our hardware platform, see Figure 1. We assume the relations that constitute the database are horizontally declustered [5, 4, 1] into fragments, with one or more fragments assigned to a single disk. We consider requests at the granularity of block references that retrieve a single block from a fragment. Thus, a request references a single disk drive. One objective of this study is to observe the past response times and reason about changes that would improve these observed response times. With this objective, we break time into intervals, termed time slices. For each time slice, we store enough information to: a) reconstruct the observed response time, and b) reason about changes to the placement of fragments and how it would impact the observed response time. For example, for a given time slice, we strive to answer the following hypothetical question: "How would the observed response time change if fragments $f_1$ and $f_2$ were assigned to disk number 6?" Obviously, we want to provide an accurate response with minimal storage, bandwidth, and computational resources.

In Sections 2, 3, and 4, we describe linear regression, queuing models, and SEER as alternative techniques for our objective. In each section, we present experimental results from a 9 disk configuration that services requests based on a trace of requests gathered from an Oracle database management system. The pattern of request arrival is shown in Figure 2. In this figure, the x-axis consists of time slices where each time slice is six minutes long. Clearly, this arrival pattern does **not** correspond to a Poisson distribution. In the reported experiments, the requested block size is fixed, resulting in a 6 millisecond service time. With these experiments, linear regression stores the following for each time slice: a) a pair of values, i.e., (load,

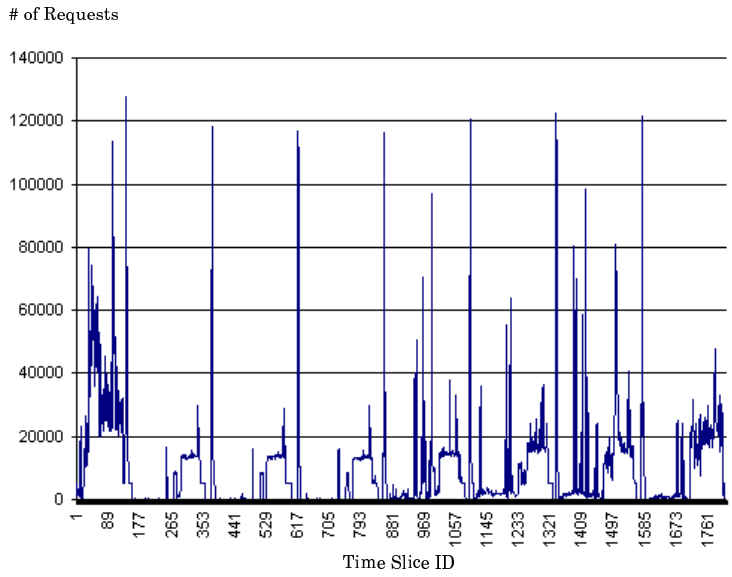



Figure 2: Number of requests as a function of time

average response time), and b) a vector of $N$ values pertaining to the load imposed by each fragment during a time slice. It computes an equation that can estimate response time for a) each time slice, and b) a hypothetical data placement. The queuing models store a similar data set with one difference. Instead of a pair of values, it only stores system utilization per time slice. SEER is a new technique that stores significantly more data, an $N \times N$ matrix. Its computational complexity is also higher. However, it provides a higher degree of accuracy. The current technological trends justify the use of SEER. We conclude with brief remarks in Section 5.

## 2 Linear Regression

A simple statistical approach to estimate response time is linear regression. With this approach, we record the response time and system load for each time slice. This produces a scatter plot of response time as a function of system load, see Figure 3. Thus, there might be several y-values for each tick on the x-axis. The load can be quantified in different ways. We quantify it as the number of bytes retrieved during a time slice.

Linear regression inputs the points on the plot of Figure 3 to compute a function for response time:

$$RT = a + (b \times load) \tag{1}$$

where a is an offset relative to the y-axis and b is the slope of the line specified by the linear function. Each point has a certain distance from the line specified by equation 1. This might be a positive or negative value. It is positive when its y-value exceeds that of the point predicted by Equation 1. It is zero if it matches the point predicted by Equation 1. Otherwise, it is a negative value. The following equations for $a$ and $b$ ensure that the sum of these positive and negative values is zero:

$$b = \frac{N \times \sum(RT \times load) - (\sum load \times \sum RT)}{N \sum load^2 - (\sum load)^2} \tag{2}$$

$$a = \frac{\sum RT - b \times \sum load}{N} \tag{3}$$

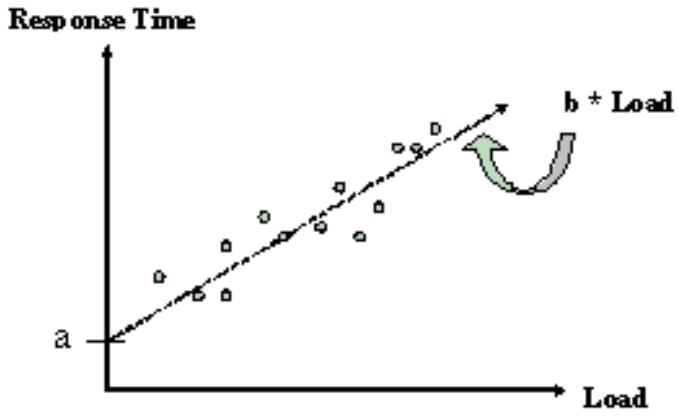
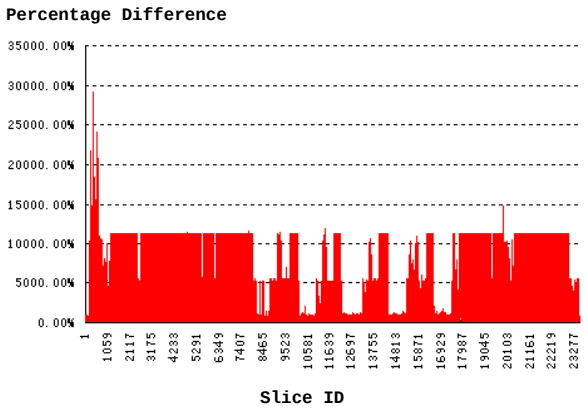
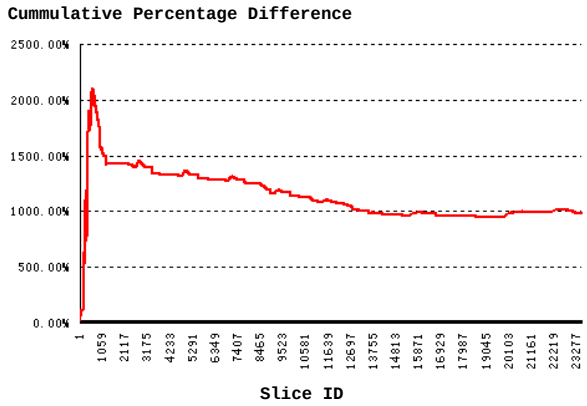



Figure 3: A scatter-plot of response time as a function of load.




4a. Percentage error for each time slice



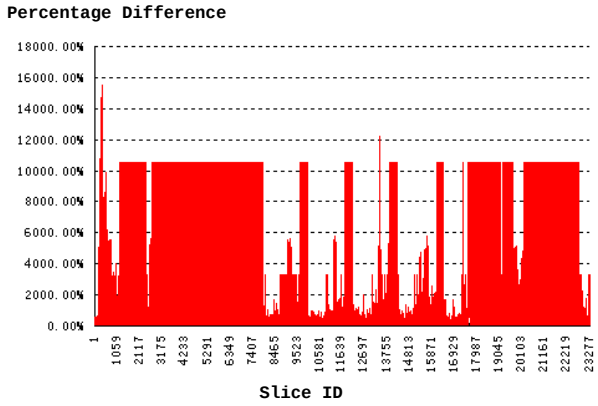

4b. Cumulative percentage error

Figure 4: Linear regression with fixed-length time slices.

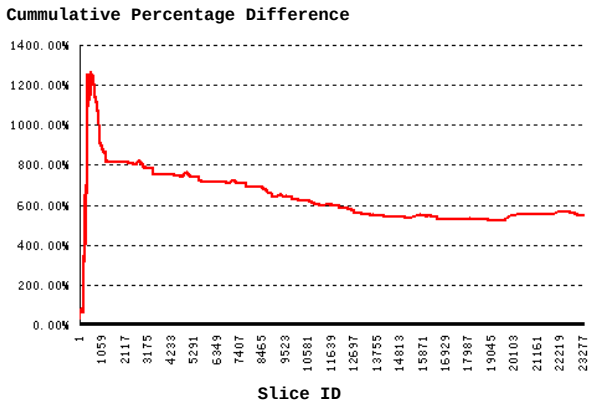
N denotes the total number of samples.

In order to reason about the placement of data, the system can store the number of bytes retrieved from a fragment per time slice. This is the load imposed by that fragment on a given disk during a time slice. With a hypothetical assignment of fragments to a disk, we sum the total load attributed by these fragments and use Equation 1 to estimate an average system response time.

Figures 4 shows the accuracy of linear regression with fixed-length time slices. In these figures, we used a simulator to measure both the observed response time and system load for each time slice. We stored this information in a database management system. Next, we computed the $b$ and $a$ values using Equations 2 and 3. Using these, we invoked a program that employs linear regression to estimate the response time for each time slice (with the observed load). Figure 4.a show the percentage error between the observed and estimated values for each time slice. In these figures, the first 9 ticks on the x-axis correspond to the percentage error for each of the 9 disks for the first time slice of Figure 2. Hence, there are 9 times as many ticks on the x-axis of these figures when compared with Figure 2. Figure 4.b show the cumulative average percentage difference as a function of time. This value is computed as a function of the x-axis by computing the average for the first x values. For example, with x-value equal to 200, the reported y value is the percentage error observed for the first 200 observed x-values. In essence, the error in one time slice is




5a. Percentage error for each time slice




5b. Cumulative percentage error

Figure 5: Linear regression with variable-length time slices.

carried over to the computation of the percentage error for the next time slice.

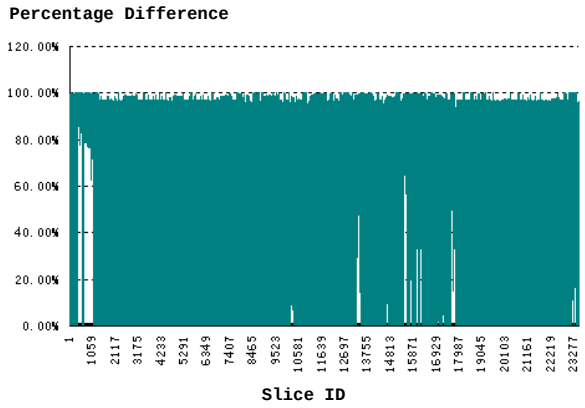
One factor contributing to the high percentage error in Figure 4 is the fixed-length time slices. This is because requests may arrive in one time slice (say $S_1$) and finish during the subsequent time slice (say $S_2$). With our methodology, the load and response time of these requests are attributed to $S_2$. We adjusted our methodology by using variable-length time slices. With this methodology, a time slice does not end until the system is idle, i.e., there are no active requests in the system. Thus, both the load and observed response times are attributed to a single time slice. Figure 5 shows the percentage error with this methodology. In these figures, the minimum time slice is six minutes long. These results demonstrate that the accuracy of linear regression is improved using variable length time slices. In these experiments, the maximum time slice is 995848 milliseconds (16.59 minutes) long.

The obtained results demonstrate that our use of linear regression is inaccurate for estimating response time. In particular, our measure of load fails to capture the bursty nature of request arrivals, see Figure 2. To illustrate, assume that the service time of each request is 1 time unit and consider the following two scenarios. In the first, twelve requests arrive one time unit apart during one time slice. In the second, the same twelve requests arrive at the beginning of a time slice. Both time slices report the same observed load. However, while the average response time in the first case is 1 time unit, the average response time in the second scenario is 6.5 time units. The accuracy of linear regression might be improved if response time is described in terms of both the imposed load and burstiness of requests.

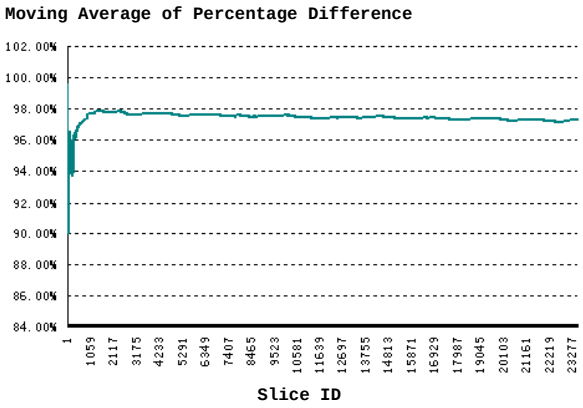
# 3 Queuing Models

There is an existing body of work from queuing theory [3] and operation research to estimate response time. M/G/N is Kendall's notation for representing a queuing model. The first part represents the input process to the model, the second is the service distribution, and the third is the number of servers. For example, a M/M/1 model has a Poisson arrival rate (M is an abbreviation for Markovian), exponential service time, and consists of one server. A M/D/1 is similar except that its service time is deterministic. The Pollaczek-Khinchin mean formula states that the mean response time of a one server system that employs a first-come-first-serve (FCFS) policy is:

$$(1 + \frac{1 + C_v^2}{2} \times \frac{\rho}{1 - \rho}) \times E[S]$$




6a. Percentage error for each time slice




6b. Cumulative percentage error

Figure 6: M/D/1 with fix-length time slices.

E[S] is the expected service time, $\rho$ is the utilization of the server, and $C_v^2$ is the squared coefficient of variation on service time. $C_v^2$ is defined as:

$$C_v^2 = \frac{V[S]}{E[S]^2}$$

where V[S] is the variance of the service time.

With a M/D/1 queuing model, the service time is constant. Hence, V[S] equals zero, and $C_v^2$ in turn is zero. For this model, the average response time is:

$$(1 + \frac{1}{2} \times \frac{\rho}{1 - \rho}) \times E[S]$$

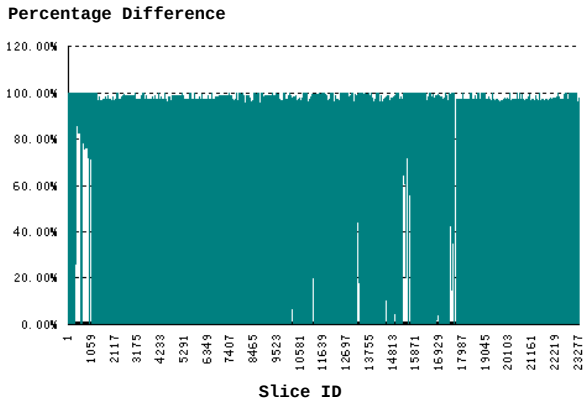
With a M/M/1 queue, the service time is exponentially distributed. Hence, the variance in service time, V[S], equals the square of the expected service time, $E[S]^2$. This means $C_v^2$ equals 1. Thus, the estimated response time is:

$$\frac{E[S]}{1 - \rho}$$

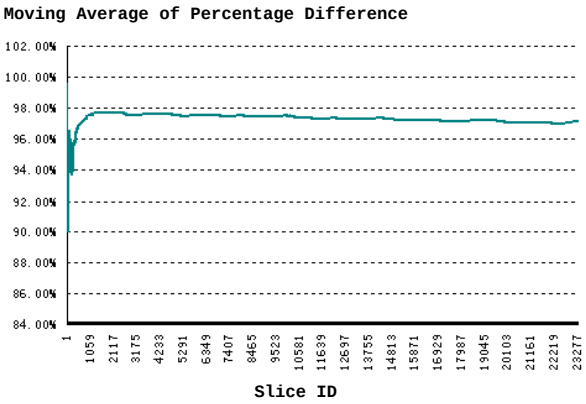
The above estimation of response time assumes a Poisson arrival rate. A departure from this assumption changes average wait time and queue lengths.

For each time slice, the system may store the utilization of the server, the service time of the disk and its variance in order to estimate the response time. For example, if the utilization of the disk drive is 40% in a time slice and its service time is deterministic and fixed at 6 milliseconds then the expected response time is 8 milliseconds. Moreover, it may store a vector that maintains the utilization attributed to each fragment per time slice. Based on this, it can reason about placement of fragments and compute an answer to the hypothetical question posed in Section 1.

Figures 6 shows the accuracy of Equation 3 to estimate response time. Similar to the discussion of Section 2, we ran our simulator with a fixed service time of 6 milliseconds and 6 minute (fix-length) time slices, and stored the observed response time and utilization. Utilization is the percentage of time the system is busy during a time slice. If T is the duration of a time slice and the system is busy for B time units then utilization is $\frac{B}{T}$. Next, we used Equation 3 to estimate response time for each time slice (using the observed utilization) and compare it with the observed response time. Figure 6.a shows the percentage error for each time slice. Figure 6.b shows the cumulative percentage error.




7a. Percentage error for each time slice




7b. Cumulative percentage error

Figure 7: M/D/1 with variable-length time slices.

One factor contributed to the observed inaccuracies is the arrival pattern of requests with the trace; it does not correspond to a Poisson distribution. Another contributing factor is the fix-length time slices. Similar to the discussion of Section 2, we configured the system to use variable length time slices with a minimum length of 6 minutes. Figure 7 shows the percentage error with this experimental methodology. In our experiments, the variable length time slice had no impact on the observed percentage error. Generally speaking, one must use M/M/1 model when the service time is not deterministic. In this case, if the distribution of observed service time is not exponential then one might observe a higher percentage of error.

When compared with linear regression, the queuing model appears more accurate because its x-axis has a much smaller scale. One reason for this is that equation 3 always under-estimates the average response time. Thus, the percentage error is always below 100%. Of course, the parameters of linear regression, $a$ and $b$, could also be adjusted to always underestimate response time (by using the minimum y-value observed for each x-value of the scatter plot in Figure 3).

# 4 SEER

One approach to estimate response time with 100% accuracy is to store the average wait time and service time per time slice. (We assume variable-length time slices where a time slice has a minimum duration and ends only when the system is idle.) However, this cannot estimate the expected response time when the placement of fragments across the disks is modified.

In this section, we describe SEER, a new approach to estimate the average response time. Our experimental results demonstrate that this technique is more accurate than the other two alternatives. This new approach stores the following per time slice: a) the average service time for requests that reference a fragment, S($f_i$), b) the total number of requests referenced during that time slice, NumReq, and c) an SEER matrix that is used to estimate the wait-time between the requests that reference fragments. SEER is also a measure of burstiness for request arrivals. It is a $N \times N$ matrix where $N$ is the number of fragments. Assuming that the fragments in the system are uniquely numberd as $f_1$ to $f_N$, the value $f_{i,j}$ in the matrix denotes SEER($f_i$, $f_j$). These values are initialized to zero at the beginning of a time slice. When a request that references fragment $f_j$ arrives, it has a service time, $S_{f_j}$, and an arrival time, $T_{arr,f_j}$. The sum of these two values is the departure time of $f_j$, $T_{depart,f_j} = T_{arr,f_j} + S_{f_j}$. The same holds true for each request referencing fragment $f_i$. (Note that we do not consider the wait time in queues.) For two requests that reference $f_i$ and $f_j$, if $T_{arr,f_i} \leq T_{depart,f_j} \leq T_{depart,f_i}$ then SEER($f_i$, $f_j$) is a positive value and equal

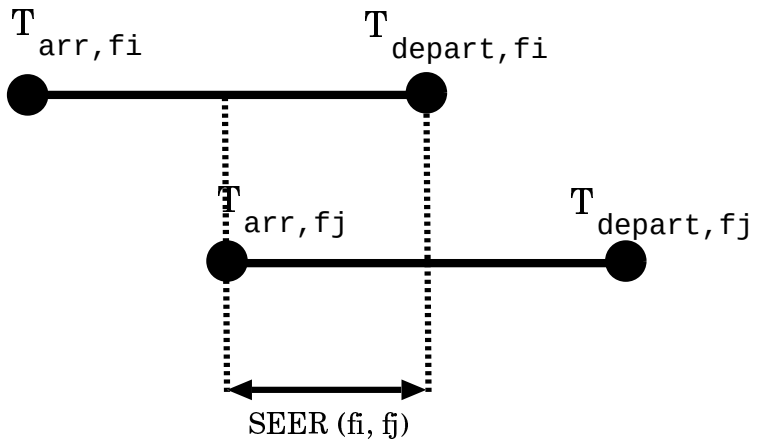



Figure 8: SEER between two requests referencing fragments $f_i$ and $f_j$.

to $T_{depart,f_j} - T_{arr,f_i}$, see Figure 8. Otherwise, it is zero. This value is added to SEER($f_i$, $f_j$) in the matrix. Note that SEER($f_i$,$f_j$) is defined for all fragments, i.e., there is a value for two fragments that reside on different disks. This information is useful because it enables the system to hypothesize about scenarios when these two fragments are assigned to the same disk.

The computation of the SEER matrix is as follows. For each fragment, the system maintains an arrival time and departure time. Upon the arrival of a request referencing fragment $f_i$, the system checks to see if the arrival time of this request is greater than the current departure time for $f_i$. If this is the case then the departure time of $f_i$ is set to the departure time of $f_i$. Otherwise, the system (a) computes the SEER between fragment $f_i$ and itself[1], and (b) the departure time of $f_i$ is incremented with the service time of the new request. Next, it computes the temporal overlap between this fragment and every other fragment with a departure time greater than the arrival time of this request. The obtained value is added to SEER($f_i$, $f_j$).

The value of SEER defines an optimistic wait time between requests referencing two different fragments. For a time slice, if one wants to speculate on the average response time with $K$ fragments, numbered $f_1$ to $f_K$, assigned to the same disk drive then we add the observed average service time for the requests referencing these K fragments, $S_{avg}(K)$, to the wait time expected between these K fragments, $W_{avg}(K)$. These quantities are defined as follows:

$$S_{avg}(K) = \sum_{i=1}^{K} S_{avg}(f_i)$$

Where $S_{avg}(f_i)$ is the average service time observed for requests that reference fragment $f_i$. The wait time between these $K$ fragments is defined as:

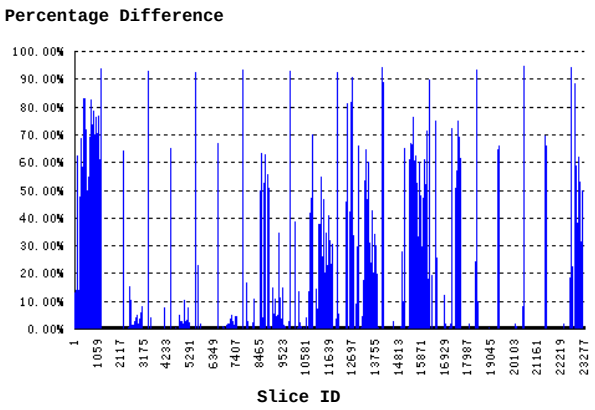
$$W_{avg}(K) = \frac{\sum_{i=1}^{K-1}\sum_{j=i+1}^{K} SEER(f_i,f_j) + \sum_{i=1}^{K} SEER(f_i,f_i) + \sum_{i=2}^{K}\sum_{j=1}^{i-1} SEER(f_i,f_j)}{NumReq}$$

The self overlap, SEER($f_i$, $f_i$), defines how long the requests that reference the same fragment wait for one another in the system.

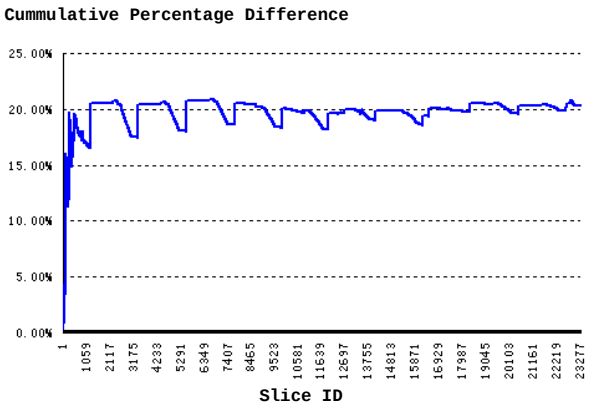
Figure 9 shows the percentage difference between the predicted and observed response times for variable length time slices[2]. The obtained results are a significant improvement when compared with linear regression and M/D/1 queuing models. Similar to the M/D/1 queuing model, SEER under-estimates the average

[1]This is the self temporal overlap. See the next paragraph for further discussion.

[2]With fix-length time slices, the x-axis of Figure 9.a would have a scale that ranges in value from 0 to 1600%. We eliminated these results because the previous sections motivated the importance of using variable length time slices.




9a. Percentage error for each time slice




9b. Cumulative percentage error

Figure 9: SEER with variable-length time slices.

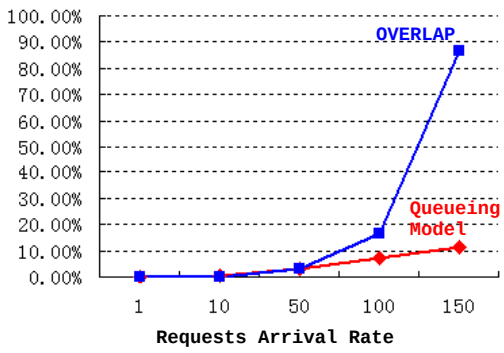
response time of a request. This is because it does not consider the wait-time in queues to service requests. To illustrate assume that three requests ($r_a$, $r_b$ and $r_c$) reference different fragments ($f_a$, $f_b$ and $f_c$) assigned to a single disk. Assume a) the service time of each request is 1 time unit, 2) $r_a$ and $r_b$ arrive at time $T_1$, and 3) $r_c$ arrvies at time $T_1 + 1$. Because of queueing delays $r_c$ might overlap either $r_a$ or $r_b$. However, our current model ignores such delays. Hence, it sets SEER($f_b$,$f_c$) to zero because it employs only service time to estimate how long these two requests are in the system. This explains why the percentage error is higher when the system load is higher, compare Figures 9a and 2.

## 5 Discussions and Future Research Directions

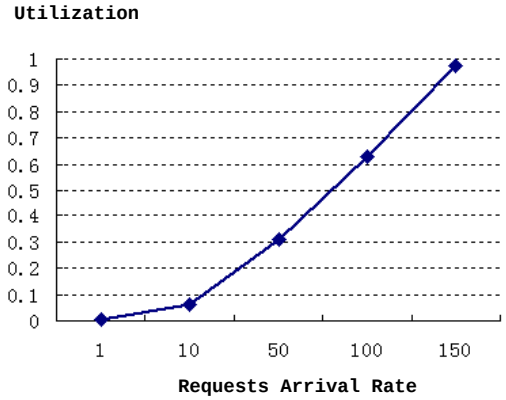
This paper details three alternative techniques to estimate response time and reason about the placement of data across multiple disks. We employed a trace driven evaluation study to compare these techniques with one another. SEER is more accurate when compared with linear regression and M/D/1 queuing model because it maintains state information pertaining to burstiness of requests. Note that SEER outperforms the queuing model because pattern of request arrivals is not Poisson. With an environment consisting of a single server and a Poisson arrival rate, the queuing model and overlap provide the same accuracy with low system loads (utilization less than 20%), see Figure 10. With high system utilization, the M/D/1 queuing model is significantly more accurate than SEER.

We also analyzed the percentage of error for different system utilizations using the trace driven study, see Figure 11. In this figure, the y-axis is the percentage error for each observed system utilization. With a low system utilization, the percentage error is so high for linear regression that it eclipses the difference between the queuing models and SEER. Hence, we show linear regression starting at 60% utilization. In our experiments, linear regression predicts response time with a high accuracy when the system load is high (more than 80%). SEER predicts response time with a high accuracy with a low system load because it ignores queuing delays. While its percentage error increases with a higher system load, it is more accurate than queuing model.

An immediate future research direction of this effort is to analyze extensions to SEER to further increase its accuracy. In addition, this study assumed a deterministic response time for all block requests, i.e., 6 milliseconds. While this might be reasonable for database applications because they reference the storage subsystem with fixed block sizes, it does not hold true for all data intensive applications, e.g., continuous media servers [2]. We intend to investigate extensions of this study in support of variable length blocks. A




10a. Percentage error




10b. Utilization

Figure 10: Comparison of M/D/1 and SEER using a single-server with Poisson arrival rate.

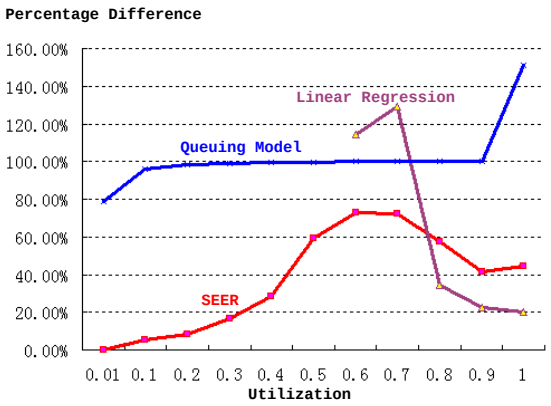



Figure 11: A comparison of all three techniques.

long term objective is to develop a re-configurable simulator that collects traces and reconfigures itself to dynamically model its target environment. This simulator should be extensible to include new solutions. It should empower a system administrator to reason about the system's past behavior and estimate enhancements with proposed hardware changes, e.g., purchase of newer disk models, replacement disks, etc.

## References


[1] S. Ghandeharizadeh and D. DeWitt. A Multiuser Performance Aanalysis of Alternative Declustering Strategies. In *Proceedings of the 6th IEEE Data Engineering Conference*, 1990.

[2] S. Ghandeharizadeh and R. Muntz. Design and Implementation of Scalable Continuous Media Servers. *Parallel Computing*, pages 91–122, 1998.

[3] L. Kleinrock. *Queuing Systems.* Wiley Interscience, 1975.

[4] M. Livny, S. Khoshafian, and H. Boral. Multi-disk management algorithms. In *Proceedings of the 1987 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 69–77, 1987.

[5] D. Ries and R. Epstein. Evaluation of Distribution Criteria for Distributed Database Systems. Technical Report UCB/ERL Technical Report M78/22, UC Berkeley, May 1978.

[6] A. Veitch, E. Riedel, S. Towers, and J. Wilkes. Towards global storage management and data placement. Technical Report HPL-SSP-2001-1, Hewlett Packard Laboratories, March 2001.

[7] Gerhard Weikum. The Web in 2010: Challenges and Opportunities for Database Research. In *Informatics*, pages 1–23, 2001.