Dynamic data reallocation in disk arrays.

Ron Arnan [*] Eitan Bachmat [†] Tao Kai Lam [‡] Ruben Michel [§]

September 20, 2006

Abstract

We present an application which optimizes the configuration of data in a disk array. The application relies on very little input data which is readily available from many storage devices.

1 Introduction

Storage systems contain devices like disks which are mechanical. Devices which require mechanical motion are slow in comparison with fully electronic computing devices such as processors. One of the problems associated with disk drives is the interference between different workloads which share the disk. If several workloads on the same drive are concurrently active, then the disk head will shuttle (seek) back and forth between their respective locations. Such mechanical motion is very time consuming. One of the methods which can be employed to alleviate this problem in multi disk systems is *data reallocation*. The idea is to separate interfering workloads by moving one or several of the workloads to other disks, where they will cause less interference.

We present a data reallocation scheme for improving the performance of a disk array which we designed and implemented. The reallocation scheme was the first fully automated, commercially available, scheme for large disk arrays. It is completely transparent to operating systems and can handle heterogeneous disk arrays with different types of disks. It also takes the nature of various workloads into account. One of the main problems in designing such a scheme in a realistic setting is that there is insufficient information regarding the amount of interference among workloads which share a disk drive, therefore we cannot design accurate models of the interference. When accurate information is not available the most important thing is to make conservative estimates which will minimize the chances of errors. We will show how to generate provably conservative estimates for the interference. These estimates may overestimate the interference, namely, they may lead us to believe that disks with good performance (little interference) have poor performance (large interference). However, the estimates never substantially underestimate the interference, namely, they never lead us to believe that a poorly performing disk is performing well. As a result of this property we will be able to diagnose all poorly performing disks. Furthermore, if we can reallocate data so that the estimates predict good system performance, then the resulting system, after reallocation will indeed perform well. The penalty for using conservative estimates is low. From

[*]EMC Corporation, Hopkinton, MA 01748.

[†]Dept. of Computer Science, Ben Gurion University, Beer Sheva, Israel 84105, email: ebachmat@cs.bgu.ac.il

[‡]EMC Corporation, Hopkinton, MA 01748, email: tlam@emc.com

[§]EMC Corporation, Hopkinton, MA 01748.

time to time a disk with good performance is incorrectly diagnosed as having poor performance and the reallocation scheme tries to improve the performance by displacing its workloads. While this activity is wasteful, it is benign and essentially harmless.

The conservative estimates which we use are based on a very classical model of user activity, the independent reference model (IRM). These estimates use precisely the information which is readily available from the disk array and can therefore be used in practice. In addition they can be computed quickly, allowing the reallocation scheme to scale to petabyte sized systems and beyond.

The rest of the paper is organized as follows: In section 2 we provide relevant background information on the organization of some typical large disk arrays and the basic modeling of disk drives.
In section 3 we introduce the seek estimates which we will use and explain how they can be computed efficiently.
In section 4 we consider some variants of our basic estimates and the effects of various features of disk drives and arrays which are not considered in the basic estimates.
In section 5 we present the basic architecture of the reallocation scheme.
In section 6 we present the results of a controlled lab experiment.
Section 7 presents results for a real production system.

2 Storage System Description

Although the design of the reallocation scheme is quite general it was intended to operate within a certain type of storage system, EMC's Symmetrix family of disk array. The Symmetrix is typical of "high end" storage systems and shares some common design features with other systems in this category from several other vendors such as Hitachi and IBM. We briefly describe some features of this type of disk array which are relevant to the application.

Large storage systems contain many disks. They are connected in many cases to several servers which may have different operating systems and different applications running on them. All the data in the storage system is divided into fairly large units called *volumes*. Volumes of a given system may have different sizes, however, a single size (typically a few Gigabytes) is the most common case.

The volumes operate as autonomic units in the storage system. They may have different logical attributes which help manage them. Each volume holds data of a given operating system. Each volume may be accessed from a subset of ports by a subset of servers. Different volumes may also be protected in different ways, either by mirroring or parity. Physically volumes occupy a set of contiguous tracks on one or more disks. A volume is the basic unit of data for which activity statistics counters are collected and reported. The following statistical counters are available to us

- **Read requests**: The number of read I/O operations which were requested of a given volume during a given time interval.
- **Write requests**: The number of write I/O operations which were requested of a given volume.
- **Prefetch tracks**: Modern disk arrays have the ability to identify certain user access patterns and to send data from the disk to the cache in anticipation of that data being used, an operation which is called *prefetch*. The data is sent in fixed size units which are called tracks.

We can count the number of tracks which were prefetched (read) from the disk for any given volume.

- **KB read**: The total number of kilobytes read by all read I/O to the volume. Since the number of bytes which are read in a single I/O operation may vary, this counter need not be proportional to the **read requests** counter.
- **KB written**: The total number of kilobytes written by all write requests to the volume.

A volume is also the basic unit of data which can be easily moved within the system using standard copy tools internal to the storage system. We will therefore use volumes as atomic units of data for the purposes of data reconfiguration. Reconfiguration will be achieved by moving volumes from one disk to another or exchanging locations between volumes.

Large storage systems have a large cache which serves both read and write I/O operations. When an I/O request is presented to the storage system it is first handled by the cache. For example, A read request for data which is present in the cache does not generate disk operations. When the request is a write request, it is usually written to the cache and only later to the disk. If another write request to the same data address arrives before the previous write request has been written to disk, the writes will be coalesced and only a single write will be performed. For these and other reasons such as prefetch operations, the number of I/O requests which a volume receives is not necessarily the same as the number of I/O operations which are performed at the disk level in servicing the requests. Consequently there are separate counters which collect the number of requests from the server to the volume and the number of disk operations which the various requests generated. By considering the counters which count disk operations we can bypass the effects of caching and obtain the number of disk operations directly. This is a distinct advantage over data which is gathered at the servers.

2.1 Seek time

We model the utilization of a disk with the type of available data described above. The main benefit of moving a volume from one disk to another is to lower the amount of seek time between requests to the different volumes residing on the same disk and operating concurrently. The concurrent activity of different volumes on the same disk causes the disk to shuttle back and forth between the volumes, thus creating long seeks which lower the efficiency of the disk. It is these interference transitions between different volumes that we wish to model. We first recall the basic features of disk geometry and seek times.

Disks are two dimensional storage devices consisting of several platters. Data resides on *tracks* which form concentric circles of varying radii around the center of a platter. The set of all tracks which are at the same radius on the different platters is called a *cylinder*. The location of data on a disk can be described by coordinates r, θ, h. The radial coordinate r is the cylinder number where the outermost cylinder is given the number 1 and cylinder numbers increase consecutively as we proceed inwards towards the center of the disk. The θ coordinate describes the position of the data location within a track in terms of the angle (counter clockwise) between the data location and some arbitrary fixed ray which represents the 0 angle. The third coordinate h denotes the platter on which the data resides. Switching times between platters are fairly small and will not play a role in the proceeding discussion we will thus assume that the disk consists of a single platter and apply the term track to signify the radial coordinate. The disk rotates at a constant speed in one direction (say clockwise). To get from location $A = (\theta_1, r_1)$ to location $B = (\theta_2, r_2)$ the head of the device must first perform a radial motion from the track r_1 to the track r_2. The time it takes the disk head to perform this radial motion is known as (radial) seek time. The head starts and ends with no radial velocity and must first accelerate, reach a maximal speed, and then decelerate towards the targeted track. The acceleration and deceleration processes are invariant under translation. Furthermore as the distance $|r_1 - r_2|$ grows the head spends more time at higher speeds and so the average velocity during the transition increases. Consequently the time it takes to seek from r_1 to r_2 has the form

$$d_F(r_1, r_2) = F(|r_1 - r_2|)$$

where F is a concave non decreasing function (note that the slope of F is inverse-proportional to the peak velocity during the transition).

We now consider the partition of the disk into volumes. We number the tracks (cylinders) of the disk $1, \ldots, k$ and assume that there are n volumes $V_1, \ldots, V_n$ residing on the disk. We recall that a volume occupies a contiguous set of tracks, consequently the arrangement of volumes induces a partition of the tracks $1 = r_0 < r_1 < r_2 < \ldots < r_n = k$ such that volume V_i resides on tracks in the range $r_{i-1} \leq j < r_i$. We define the middle track of volume i to be $m_i = \frac{r_{i-1}+r_i}{2}$. Since volume capacity is typically much smaller than disk capacity, the seek time between middle tracks of volumes is a good representative for the seek time between any locations in the different volumes. Following our earlier discussion we conclude that the seek time between volumes V_i and V_j is given by $t_{ij} = d(V_i, V_j) = F(|m_i - m_j|)$ where F is concave and non decreasing which depends on the specification of the disk drive.

2.2 Modeling I/O activity

Let I be a time interval. We wish to estimate the amount of time T_I, that the disk spent seeking between volumes during time interval I. We say that T_I is the *total seek time* during I. The basic information that we can obtain from the counters regarding the I/O access pattern during I can be encoded in a vector $\vec{a} = \vec{a}_I = (a_1, ..., a_n)$ which represents the volume activity levels during I, namely, a_i is the observed number of disk requests for volume i during time interval I. The total amount of activity directed at the disk is $A = \sum_i a_i$.

We need to estimate T_I using this available data. To explain the difficulty in making such estimates consider the simplest case of a disk with two volumes. We further assume that the seek time between the volumes has been normalized to be 1. Assume that the observed activity vector during some time interval is $(5000, 5000)$, that is , there were 5000 I/O requests to each volume during the time interval. We can consider two different scenarios which are consistent with the observed statistics. In the first scenario the first 5000 requests were to volume 1, followed by 5000 requests to volume 2. In this case there was only a single seek between a request to volume 1 and a request to volume 2, hence, $T = 1$. In the second scenario odd numbered requests go to volume 1, while even numbered requests go to volume 2. In this case every transition between requests requires a seek between volumes and therefore $T = 10,000$. We see that the data available to us is simply insufficient to generate an accurate estimate of T since there is too much uncertainty. In the absence of accurate models we will settle for the next best thing, namely, a conservative estimate. We will generate a simple estimate which is never too far from the largest possible seek time T which is consistent with the input data. We note that the largest possible seek time also depends on the activity vector and not just on the total activity A. For instance consider the vector $(8000, 2000)$. The number of I/O is still $A = 10,000$ however, the maximal value of T which is consistent with our information is $T = 2000$. To estimate T_I we use the independent reference model (IRM) which is a simple probabilistic model whose input is the activity vector $\vec{a}$. In the IRM we assume that I/O requests are generated independently of each other. We further assume that each volume has a probability p_i of being accessed in a given I/O request. The probability p_i reflects the relative popularity of volume i in comparison with other volumes on the disk. To be compatible with the observed activity vector we set the request probability for state i to be $p_i = a_i/A$. It is well known, [13], that for this model the expected seek time between successive I/O requests is

$$E = \sum_{i,j} p_i p_j t_{i,j} = \frac{1}{A^2} \sum_{i,j} a_i a_j t_{i,j} \tag{1}$$

To obtain the formula, we note that the probability of a request landing in volume i is p_i. Since requests are independent, the probability of the next landing in volume j in p_j, therefore the probability of a seek from volume i to volume j is $p_i p_j$. The time spent on seeking from volume i to volume j is t_{ij}. Summing over all pairs of volumes yields the formula. Since there were A requests during time interval I the total expected total seek time during I is given by $T(IRM) = AE = \frac{1}{A} \sum_{i,j} a_i a_j t_{i,j}$ We will refer to this estimate as the IRM estimate.

The IRM estimate has two nice properties. For proofs of these properties we refer the reader to [3].

1) The IRM estimate always satisfies $T_I(IRM) \geq \frac{1}{2} T_I$, hence the actual total seek time during I is never more than twice the IRM estimated seek time.

2) Assume that we subdivide the interval I into two subintervals I_1 and I_2, with activity vectors $\vec{a}_{I_1}, \vec{a}_{I_2}$ respectively. Let $T_{I_1}(IRM)$ and $T_{I_2}(IRM)$ be the IRM estimates computed separately for I_1 and I_2, then

$$T_I(IRM) \geq T_{I_1}(IRM) + T_{I_2}(IRM)$$

The first property says that $T(IRM)$ cannot grossly under estimate the total seek. This is an important property since it implies that a disk with a low estimate of total seek time actually experiences only a small amount of seek activity between volumes. We conclude that if the seek estimate of the disk is low then moving volumes from the disk to other disks will not improve performance significantly. We also use this property when assessing disks as targets for receiving additional active volumes. We add the activity of the volume which we intend to add to the disk to the activity vector. If the IRM estimate remains small then the addition of the volume to the disk will not significantly increase the total seek time and hence it will not appreciably degrade the performance of the other volumes on the disk. Consequently, a low estimate for the what if scenario of adding a volume to a disk says that it is safe to move the added volume to the disk.

2.3 Using the Model to Optimize the Configuration

We would like to use the IRM estimates to move volumes between disks, with the goal of lowering seek times. The basic idea is to sample the activity from during time intervals of fixed duration S and to apply the IRM estimate to each interval separately. We can see from property 2 that if the sampling frequency is too low the IRM estimates will produce unnecessarily large seek estimates.

For example, consider two volumes A and B, where A is active during the day time while B is active during the night time. By sampling the activities once every 24 hours, the IRM will 'assume' that A and B are concurrent and interfere with each other, while if we sample twice, once during the day and once during the night we get a better estimate which is much lower. The down side of frequent sampling is the burden it creates on the storage system. It also increases the number of data points and thus the complexity of computing the seek estimate. based on data from real production systems which considers the pace in which the activity of volumes in a disk array changes it seems that choosing S to be in the range of 10-30 minutes is a reasonable choice which leads to a sufficiently accurate model and maintains a low computational complexity for the algorithm. The sampling period S can also be made a dynamic variable which depends on the observed behavior of the system. Having chosen a sampling period we also need to choose the number of sampling periods W which are to be taken into account when making a decision. When considering an exchange we will use data from the last W sampling periods for creating the IRM estimates. Define three time periods. The sampling period S, the exchange period E and the window period W. The sampling period S determines how often we should sample the disk related activity counters of volumes to obtain a dynamic IRM estimate. The window size W should typically be measured in days to consider the typically cyclic activity patterns of a storage system. Finaly we need to choose how often we perform a location change in the system exchange, E. Since moving volumes is fairly expensive, a reasonable value for E should be measured in hours and even longer once the performance of the system stabilizes.

Given S, E, W, every E sampling periods we compute the dynamic IRM estimate for the last W sampling periods for each disk. The disks are then sorted according to the estimates. The effect of moving a volume from a disk D_i with a large set-up time estimate to a disk D_j with a small set-up time estimate is then computed again using the IRM but assuming a volume exchange. If there is a substantial improvement in the estimated set-up time of D_i without a large increase in the set-up time of D_j then the move is executed.

2.4 Fast Computation of the IRM estimate

The computation of the IRM estimate seems to require $O(n^2)$ operations. Recall that the numbers $t_{i,j}$ have the form $t_{i,j} = F|m_i - m_j|$ for some concave function F. It turns out to that for some forms of the function F which are very realistic the computation requires only $O(n)$ operations. One such example is that of linear functions where $F(x) = cx$. Let $S_k = \sum_{j=1}^{k} a_j$ and $S^k = \sum_{j=k}^{h} a_j$. We have the following formula of Wong [13]

$$T(IRM, \vec{a}, t_{i,j}) = \frac{1}{a} \sum_{i,j} a_i a_j c|m_i - m_j| = \frac{1}{a} \sum_{k=1}^{n-1} c(m_k - m_{k-1}) S_k S^k$$

The formula on the right requires $3n$ additions and $2n$ multiplications.

A second case is functions of the form $F(x) = c(1 - b^x)$ where $c > 0$ and $0 < b < 1$. The following lemma provides us with a fast computation for the IRM estimate

Lemma 1. *Let $e_1, \ldots, e_{h-1}$ be arbitrary numbers. Define M_{ij} to be a symmetric matrix such that for $j > i$ $M_{ij} = e_i e_{i+1} \ldots, e_{j-1}$ and $M_{ii} = 1$ for all i. Let $\vec{v} = (v_1, \ldots, v_h)$ and $\vec{w} = (w_1, \ldots, w_h)$ be a pair of vectors. Define recursively the vectors $\vec{q} = (q_1, \ldots, q_h)$ and $\vec{s} = (s_1, \ldots, s_h)$ by $q_1 = w_1$, $q_{k+1} = e_k q_k + w_{k+1}$ and $s_1 = v_1$, $s_{k+1} = e_k s_k + v_{k+1}$, then*

$$\vec{v}M\vec{w}^t = \sum_{i=1}^{n} v_i q_i + \sum_{i=1}^{n} w_i s_i - \sum_{i=1}^{n} v_i w_i$$

Proof. Consider the upper triangular matrix given by $U_{ij} = M_{ij}$ for $j \geq i$ and $U_{ij} = 0$ otherwise. Obviously $M = U + U^t - I$ where I denotes the identity. We claim that $\vec{s} = \vec{v}U$, which is easily verified once we note that for $j + 1 > i$ $M_{i,j+1} = e_j Mi, j$. Similarly $\vec{q} = U^t \vec{w}^t$ and the formula follows from the decomposition of M. □

The matrix $M_{i,j} = b^{|m_i - m_j|}$ satisfies the condition in the lemma with $e_k = b^{m_{k+1} - m_k}$. Let $\vec{a}$ be the activity vector. Setting $\vec{v} = \vec{w} = \vec{a}$ we see that $3n$ additions and $2n$ multiplications suffice for the computation of $\vec{a}M\vec{a}$. Let J be the all 1 $h \times h$ matrix, then $T(IRM, \vec{a}, d_F) = (c\vec{a}(J - M)\vec{a}^t)/a = ca - \vec{a}M\vec{a}^t/a$. We conclude that $2n$ multiplications and $4n$ additions are needed to compute S, the additional a additions are needed for the computation of a.

Either type of function F can be used to approximate seek times in disk drives to great accuracy. If the minimal seek distance $\min_i \; m_i - m_{i+1}$ is greater or equal to the threshold distance for which the disk arm reaches maximal speed then $t_{i,j} = \alpha + \beta|m_i - m_j|$. α can be computed as $\alpha = t_{i+2,i+1} + t_{i+1,i} - t_{i+2,i}$. Subtracting α we are reduced to a linear seek function which clearly satisfies Wong's additive condition. The second type of metric is useful in approximating short seeks.

2.5 Some Useful Variants

One of the chief concerns when using the IRM is that the model does not take into account many common features of workloads. We consider below various features of user access patterns and of disk drives which play a role in analyzing the expected performance of I/O requests. We describe some variations to the basic IRM estimate which can take into account some of these features to improve the decision making process in the data reconfiguration problem. We also explain why some other features do not play a substantial role in our application.

2.5.1 Sequential I/O and disk zoning

Random I/O requests behave differently than sequential I/O requests. Sequential I/O tends to be performed very efficiently by the storage system. The storage system usually identifies sequential accesses and orders the disk to prefetch more data from the sequential stream even before the server requests it, thus increasing the efficiency of the transfer process and reducing the number of seeks. As a result sequential requests tend to display much stronger locality of reference behavior than random requests.

In addition the outer radii in modern disk drives are denser and therefore have higher transfer rate than the inner radii. This phenomenon is often termed *disk zoning.* Disk zoning can be utilized to improve the performance of volumes with large data transfers which are typical of sequential activity. The amount of time a disk spends transferring data is the number of bytes read and written from/to the volume times the transfer rate in the radial zone on which the volume resides. As noted previously, Counters specifying the amount of transferred bytes per volume are commonly available.

Disks perform sequential activity very efficiently. In addition, sequential requests tend to follow in fast succession and therefore lead to less seeks. We can account for the phenomenon of sequential activity in our model by putting a different weight on sequential activity. If there were $b_{i,k}$ non sequential I/O to volume i during the sample period k and $c_{i,k}$ sequential I/O we can apply the IRM estimate to the activity vector $a_{i,k} = b_{i,k} + (1/4)c_{i,k}$. The weight of 1/4 takes the efficiency and seek reduction associated with sequential activity into account and lowers the effect of sequential activity on interference. Experiments indicate that the decisions that the application makes, based on the model estimates are stable with respect to the choice of the constant, thus a choice of 1/2 or 1/6 instead of 1/4 will not affect the qualitative nature of the results by much. From a theoretical point of view, the addition of weights is equivalent to passing from an IRM to a Partial Markov model (PMM). We refer the reader to [3] for further results in this direction.

In addition, as noted above we can perform internal disk reorganization in order to place sequentially active volumes at the outer edge of the disk, improving bandwidth for sequential applications.

2.5.2 Rotational latency and internal volume seeks

We have defined the seek time between requests to the same volume to be zero. In reality these short intra volume seeks do contribute to the utilization of the disk. In practice they can often be modeled by the parameter α in Wong's formula, so the formula $t_{i,j} = \alpha + \beta|m_i - m_j|$ holds also for $i = j$. These additions to the IRM have also been considered in the context of static data placement on a single device in [10]. In addition each disk movement from one location to another has a rotational latency component.

These terms do not play a significant role play a significant role in identifying disks with interference which may benefit from data reconfiguration, since they do not directly involve interactions between volumes. They simply add a linear term in the total activity A. In the process of moving data from one disk to another these terms will simply transfer to the new disk. They do improve the utilization estimates.

2.5.3 Correlation and phasing

In many cases the activities of different volumes will be correlated. this typically occurs when the data residing on different volumes is accessed by the same application and is rather common. IRM estimates over successive sampling periods will suitably capture this phenomenon. When two volumes are simultaneously active they contribute a cross term to the IRM estimate, while volumes with similar levels of activity but which are not correlated will often be active at separate time periods, an occurrence which does not contribute to the IRM estimate.

Phasing [4] refers to a more fine grained relation between workloads on different volumes. Sometimes two volumes seem to be active simultaneously in a certain time interval. Upon closer inspection which can only be done using a trace of the user's accesses it is seen that the application alternates rapidly (on the order of seconds or less) between using one volume and the other. Stated otherwise, the volumes are not active simultaneously, they only appear to be so when activity is measured on a large time scale. When two volumes have such phased behavior they can reside simultaneously on the same disk since they do not interfere with each other. On the other hand the corresponding IRM estimate may be high leading to the false conclusion that the volumes should be separated. This is an example in which the IRM will strongly overestimate a problem, however, as we noted before this will in almost all cases lead to unnecessary data shuffling, but not to bad performance in the long run.

Phasing gives an example of a situation in which better utilization of storage resources may be possible if extensive real time trace data becomes available as input.

2.5.4 Other features

There are several other features of I/O requests streams and disk drives which affect performance. We may mention burstiness, spatio temporal correlations, data alignment, disk scheduling and disk caching among others. We will not present a description of these features, suffice to say that their various effects on performance are invariant under data reconfiguration. By invariance we mean that the features will have essentially the same effect regardless of the location of the data to which they pertain, in particular we do not need to consider or model them in an application whose purpose is data reconfiguration.

One partial exception to this rule is disk scheduling. Disk scheduling may be positively affected by the outcome of a data reconfiguration application since disks tend to do a better job of scheduling requests when the requests are localized in a restricted radial zone of the disk.

3 Experimental Results

In this section we test whether our models are capable of capturing disk interference. As noted above we have implemented our design in a commercial product, the Symoptimizer, we present the results of a controlled experiment which was performed during the design stage in 1998, and a much more recent example in a production environment, where Symoptimizer was activated.

3.1 The controlled experiment

We designed a disk reconfiguration software module which uses only the IRM estimates 1 without any of the variants described in 2.5. We tested the module in the controlled experiment described below.

A trace of the activity of an Oracle database was captured in 1998. The trace is about 4.5 hours long. It describes decision support queries which were presented to the database by multiple users. There were 28 queries which were requested, sometimes with multiplicity, by a total of 56 users. The database belonged to a large medical claims insurer.

We placed the database on a Symmetrix 4 system with 50 18GB drives. The data was mirrored (two copies of each data item for fault tolerance purposes), so the total data capacity was about 450 GB. On each drive there were 8 equal capacity volumes, these volumes were the basic units which the algorithm could swap. The basic operation of the algorithm was to exchange the locations of two volumes. We considered two base configurations. In the first which will be referred to as the *Basic* configuration, files and index tables of the database were essentially randomly placed. In some cases this resulted in different portions of the same index table ending up on the same physical disk drive. In the second configuration which we call the *Basic striped configuration* the configuration was carefully planned by a database expert using knowledge about the structure of the database and well known rules of thumb regarding data base configurations. Files were striped across several disks to improve parallelism and different parts of tables were placed on different drives, again to guarantee that they do not interfere with each other's activity.

We ran the traced queries on a SUN Ultrasparc 6000 with 10, 250MHZ processors, with the storage system in one of the two base configurations. The algorithm sampled the activity of the volumes every 15 minutes. At the end of the run the algorithm suggested some swaps and the trace was run again. This was repeated several times to see the change in performance.

the table below summarizes the performance results after several swaps have been performed on one of the two basic configurations. The query column represents the query number, the multiplicity column represents the number of times the query was executed during the trace run, the basic column represents the running time of the average running time of the query in the first basic configuration, the striped column represents the average running time in the well planned (striped) basic configuration, basic+20 column present running times after 20 swaps were performed starting with the basic configuration and the striped+15 column shows the running time after 15 swaps starting from the striped configuration.

query	mult	basic	Stripped	b+20	s+15
1	27	273	403	229	271
2	41	47	54	45	51
3	34	562	712	576	549
4	36	2	2	2	2
5	40	84	92	42	49
6	45	29	32	30	30
7	33	85	51	47	50
8	35	25	23	25	24
9	38	1	1	1	1
10	29	1	1	1	1
11	27	3	3	3	3
12	35	120	115	93	91
13	33	1	1	1	1
14	43	12	9	7	8
15	37	13	14	13	13
40	1	3874	2192	2779	2013
41	1	3911	2681	2944	2352
42	1	4226	2187	2680	1888
43	1	1035	1083	1015	1007
46	1	6268	4444	3818	3742
47	1	473	303	631	331
56	2	1397	1738	1170	1356
57	1	1692	1192	1081	947
58	2	1545	1673	1337	1248
60	2	587	749	736	788
61	1	5090	4114	3013	3439
65	3	998	763	669	732
66	2	950	794	686	802

Table 1: Experiment results

3.2 Discussion of results

We first consider the comparison between the basic configuration and the (basic) striped configuration. We observe that while striping vastly improved the performance of some very long queries such as queries number 40,41,42,46 and 61 (group A) it also seriously slowed down queries number 1 and 3 (group B) which are of intermediate size but have high multiplicity. As a result the overall average query time remained essentially unchanged when the striping took place. We may try to explain this phenomenon as follows. Some queries or applications have large I/O bandwidth requirements. When the data of such applications is striped the data can be brought to the host server in parallel from several disks, thus speeding data transfer rates and improving performance. On the other hand there are serial applications and queries where response time matters more since the calculation is being held until the I/O completes. Such serial applications do not benefit from striping, in fact striping tends to increase the number of large seeks the disk has to perform, since jobs which were previously handled by separate disks are now requested concurrently from the same disk. In other words, spreading the activity of volumes also spreads their contention. This increase will also be reflected in the set-up time estimates. It is plausible to assume that queries in group A are of the first type, namely, more parallel and sequential, while those in group B are more serial and less sequential. We make these statements with reservations since we have not analyzed the queries or their traces. In fact one of the advantages of models like the IRM is that they are oblivious to such analysis and hence do not require any knowledge of the applications involved.

Whatever the real situation may be the IRM did a good job of improving the performance of nearly all queries including those of type A and B. The only exceptions being queries 47 and 60 starting from the basic non striped configuration. These queries also got worse starting from the striped configuration but to a lesser extent. Notice though that these queries have very small multiplicity, 1 and 2 respectively and are relatively short among such queries. The fact that the IRM estimate can be used to improve the performance of queries of type A and type B shows that the IRM is sensitive to both bandwidth and response time, while striping is more restricted to bandwidth improvement. The IRM estimate scales linearly with activity and is thus sensitive to activity levels in the disk with the goal of load balancing. the IRM estimate is also sensitive to seek distances and the amount of interference between volumes, with the goal of improving response time through seek minimization. The end result is a balanced improvement which takes both performance metrics into account.

3.3 Production system

The second system we consider is a production system. The data to be presented below shows the effect of data reconfiguration on a production disk array in the period between June 2004 and December 2004. The system had 78, 146GB Seagate LPX disk drives, for a total capacity of $10,905,523MB$ which are nearly $11TB$. The total utilized capacity was $10,872,350MB$ resulting in a capacity utilization of 99.7 % . The system was mirrored and divided into 675 equal size data volumes. The application running on the system was a SAP oracle database. Data was striped across the system by human experts.

In the following figures we see the average read miss time for the various disks in the system. Figure 3.3 shows the performance of the system disks before the data reallocation application was turned on in July 2004. The storage system samples read requests which were misses and therefore had to be handled by the disk. The average response time over the sample is taken for each disk drive. The results are sorted in decreasing order and plotted in the graph. As can be seen, the response time of the worst performing disk is about 10.5 milliseconds. The median response time is about 6.5 milliseconds.

Figure 3.3 shows the same graph after about 10 weeks. Reallocation was performed very con-

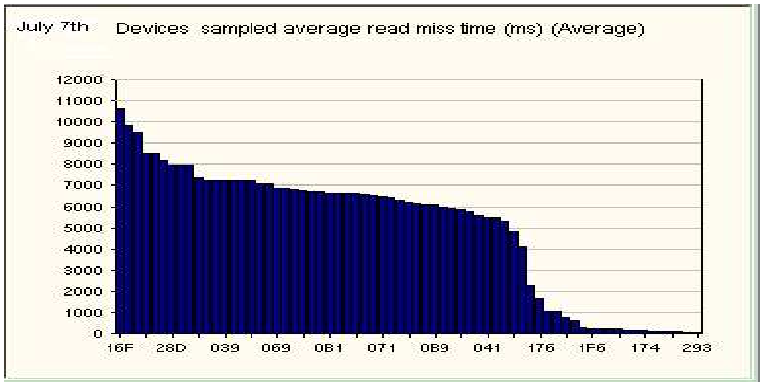

Figure 1: Read miss response time before data reallocation.

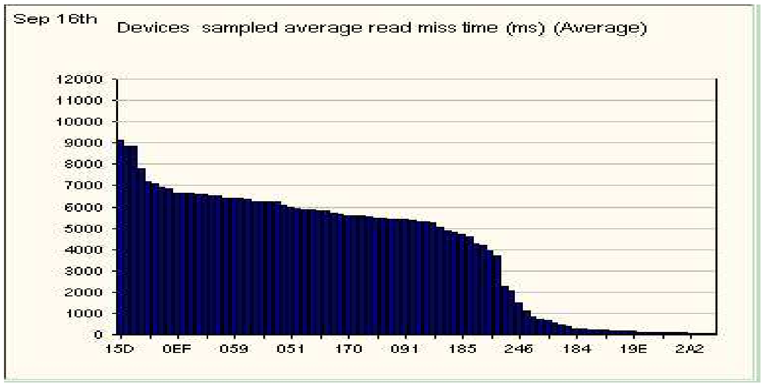

Figure 2: Read miss response time during data reallocation.

servatively. It was done only on weekends and only small amounts of data were moved each time. This very cautious approach was natural given the mission critical nature of the data. We see that the response time of the worst performing disk has improved to about 9 milliseconds, while the median performance dropped to about 5 milliseconds.

The third figure 3.3 shows the situation after about 20 weeks. The performance of the worst disk dropped to about 8 milliseconds. In fact, all but one disk show response time of 7 milliseconds or less. The median response time dropped to 4 milliseconds. These number suggest a $25-30\%$ improvement over the initial configuration.

We can also report that the actual performance of the application considerably improved.

4 Related Work

Dynamic data placement has been considered in [14]. In that work the load on a disk is simply A the activity of the disk, ignoring disk geometry completely. The Hippodrome project of HP labs and it's predecessor, Minerva, are concerned with data placement and reconfiguration and are presented in [2, 12]. The models in this project are based on the work which is presented in [1, 4]. The project assumes the existence of traces of I/O activity, which are still rare. It also faces the challenge that the traces are host based and thus black box models of the storage systems are

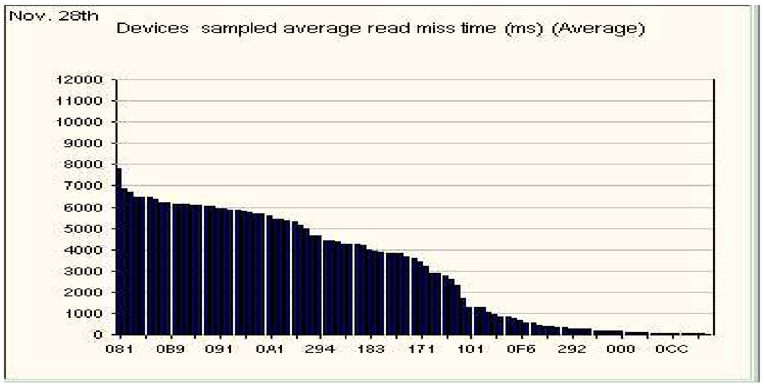

Figure 3: Read miss response time after data reallocation.

needed in order to figure out actual disk activity. As a result the model is based on very large experimentally based tables of numbers specifying the performance of a particular storage system on a specific workload with a few parameters, [1]

The application described in the current paper was made commercially available in 1999. It is the first commercial implementation of a storage based data reconfiguration software module and is also described in [5]. It was followed by another commercial implementation about a year later [6].

Acknowledgments: Special thanks go to past and present members of the Symoptimizer team including, Noam Zakai, Victoria Dubrovsky, Hui Wang and Dave DeRoches. they helped plan the application, and develop it into a commercial product. We thank them for the constant exchange of ideas which greatly influenced the paper. Special thanks also go to Sanjay Narahari who performed the controlled experiments.

References

[1] Anderson E. Simple table based modeling of storage devices, *Technical report HPL-SSP-2001-4 HP Labs,* 2001.

[2] Anderson E., Hobbs M., Keeton K., Spence S., Uysal M. and Veitch A. Hippodrome: Running circles around storage administration, *Conf. on file and storage technologies (FAST02),* 175-188, Monterey CA, 2002.

[3] Bachmat E., Lam T.K. and Magen A., A rigorous analysis for set-up time models - a metric perspective, *Proceedings of COCOON 2006*, Taipe, Taiwan, 2006. Journal version submitted.

[4] Borowsky E., Golding R., Jacobson P., Merchant A., Schreier L., Spasojevic M. and Wilkes J. Capacity planning with phased workloads, *1st workshop on software and performance (WOSP98),* 199-207, Santa Fe, 1998.

[5] EMC Corporation, *EMC Control Center product description guide.*

[6] Hitachi data systems, *Hitachi Cruise Control TM user's guide*, (MK-92RD106).

[7] Ruemmler C. and Wilkes J., An introduction to disk drive modeling, *IEEE Computer 27,* 17–28, 1994.

[8] Shriver E., Performance modeling for realistic storage devices, *PhD Thesis,* New York University, 1997.

[9] Shriver E., Merchant A. and Wilkes J. An analytic behavior model for disk drives with read ahead caches and request reordering, *Proc. Of SIGMETRICS,* 182-191, 1998.

[10] Triantafillou P., Christodoulakis S. and Georgiadis C., Optimal data placement on disks: A comprehensive solution for different technologies, *IEEE Transactions on Knowledge and Data Engineering 12,* 324-330, 2000.

[11] Triantafillou P., Christodoulakis S. and Georgiadis C., A comprehensive analytical performance model for disk storage device technologies under random workloads, *IEEE Transactions on Knowledge and Data Engineering 14,* 140-155, 2002.

[12] Wilkes J., Traveling to Rome: QoS Specifications for Automated Storage System Management, in *Proc. of the intl. workshop on Quality of Service, IWQoS,*, 75-91, 2001.

[13] Wong C. K. *Algorithmic Studies in mass storage systems*, computer science press, 1983.

[14] Zabbak P., Weikum G. and Scheuermann P. Dynamic file allocation in disk arrays. In *SIGMOD conference,* 406-415, 1991.