Towards a Zero-Knowledge Model for Disk Drives

Francisco Hidrobo[1,2]
[1]Laboratorio SUMA. Facultad de Ciencias
Universidad de Los Andes
La hechicera. 5101. Mérida, Venezuela
fhidrobo@ac.upc.es

Toni Cortes
[2]Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
Jordi Girona 1-3. 08034 Barcelona, Spain
toni@ac.upc.es

Abstract

In this paper, we present a model for disk drives with zero knowledge about the modeled drive. This model is part of our proposal to design a storage system capable of extracting all potential performance and capacity available in a heterogeneous environment with as little human interaction as possible. To make the model, our system automatically learns the behavior of the drive without expecting any prior knowledge about it from the user. In order to achieve this zero-knowledge model, we have studied three approaches: linear approximation, quadratic approximation and neural networks. We have implemented and evaluated these three approaches and found that neural networks are a great mechanism to model drive behavior. This approach has errors below 10% in read operations.

1 Introduction

There are many application where the I/O performance is a bottleneck and thus many solutions have been proposed. One of the most promising one consists of configuring the storage system and data placement to maximize the storage-system performance for a specific workload. In general, this approach consists of finding the optimal configuration and data placement for the I/O system given a specific application or set of applications. Currently, these optimizations are done by experts who use their experience and intuition to make this configuration and placement. This means that only a few sites can take advantage form this kind of "optimal" placement benefits because not everybody has (or can afford) an expert to place data in the best possible way. For this reason, a tool that could perform this tuning in an automatic way would be a great step in making this technique available to a wider range of sites. Furthermore, this tool becomes even more useful if the optimal configuration and placement varies throughout the time making it more difficult to keep the right placement up to date.

Our mid-term objective is to design a storage system capable of extracting all potential performance and capacity available in a heterogeneous environment with as little human interaction as possible. We envision the system as an advanced data-placement mechanism that analyzes the workload to decide the best distribution of data among all available devices, as well as the best placement within each device. Figure 1 presents the system architecture with its modules and the relationship between them.

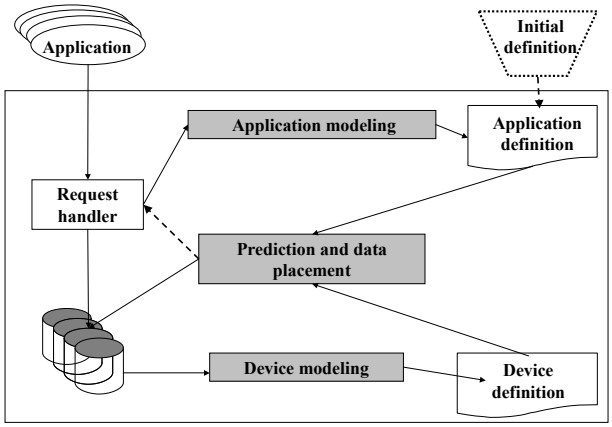

Figure 1. Block architecture for our system

The objective of this paper is to describe the *device-model* and how we plan to integrate it in the global system. We present a model for disk drives with zero knowledge, which means that it will automatically learn the behavior of the drive without any prior knowledge from the user. In order to achieve this zero-knowledge model, we have studied three different approaches: linear approximation, quadratic approximation and neural networks. We are aware that these approaches are very simple (specially the first two), but we do not need a perfect predictor, but one with reasonable accuracy but fast in giving the result.

We have implemented and evaluated these three ap-

proaches and found that neural networks are a great mechanism to model drive behavior.

The remainder of this paper is organized as follows. Section 2 describes the general operation of our system. Section 3 presents the model and details for each approach. Section 4 contains the development of our proposal, it specifies the disk drive and workloads used. Then, section 5 shows the results obtained. Section 6 discusses related work and, to finish, section 7 presents the conclusions that can be extracted from this work.

2 Autonomic storage system: a global picture

When the system is initially started, or when a new storage device is added, the model of the new drive or drives is built. The objective of this model is to be able to give an estimation of the performance a given workload will achieve for a specific data placement. To build the drive model, we plan to execute some synthetic tests, trace the behavior of the disk, and use these traces to train the model (always taking the drive as a black box). Fortunately, this model will not change during the live of the system because it is not application dependent, and thus the training time is not a critical issue.

Once we have all storage devices modeled, the system is used normally by applications. During this normal execution, the workload modeler keeps track of the accesses done to all the devices and it builds a model of what is happening in the system. This model should be good enough to be able to regenerate a trace of an execution with the same characteristics than the real execution. How this model works is out of the scope of this paper.

Once we have gathered enough information about the accesses to the storage system to have a clear picture of what is going on, we start studying what are the important files/blocks, access patterns, etc. With this information, the data-placement module decides which possible changes make sense to improve the performance of the storage system (i.e. place data sequentially, interleave related files, put special blocks in the fastest disks, or faster zones in the same disk, etc.) Hopefully, we will have many different data placements that may, improve the actual placement. Then, we feed the workload information and the new placements into the drive model to decide the performance each new placement would achieve. Afterward, we compare all possibilities and decide which one is better and whether the new placement is worth the effort of making the changes.

2.1 Requirements for the drive model

From the previous description, we can extract the following requirements.

- The drive model has to be able to predict the behavior of the device with no prior knowledge about it. It has to be able to learn the behavior by its own.
- The error observed between the model and the reality has to be reasonable, but it is more important to know an upper bound of the error that can be used as a threshold. Knowing the potential error, we can make decisions that guarantee that at least, the new placement is not going to be worst than the actual one. Our intuition tells us that we can accept errors up to a 10% because it normally does not make sense to move data for improvements smaller than 10%.
- The time needed to get the performance for a given workload with a given data placement has to be as fast as possible. The faster it is, the larger the number of possible configurations we will be able to test, and thus the probability to find the best configuration will also increase.
- The training time for the model is not important because it is done only once in the life of the device.
- It is also important to notice that the model does not need to have an internal representation from where to extract information. Decisions about data placement will be taken using the workload information and the intelligence put in the data placement module. The drive model is only needed to get a performance value for a given configuration, but no to explore why this performance is achieved.

3 Model approaches

The main objective, and also the main difference compared to other approaches, is to design a model that has no previous knowledge about the drive to model. This will allow us to use mathematical tools to implement it and, we will be able to apply this approach over a wide variety of storage systems with minimal (hopefully none) additional effort.

3.1 Workload representation in the model

As we have already mentioned, our model uses traces for both its training stage and its prediction functionality. In the first case it is the trace of a synthetic application designed to learn as much as possible form the drive, while in the second case it is a trace that represents the workload we want to measure. In both cases, a trace is make by a significant set of requests.

To represent each request, we use the following data:

1. request type (Read or Write) (**Op**),

2. address of the first-requested block (**Addr**),

3. difference (in blocks) between this request and the previous one (**Jump**). We do not use the difference in cylinders because that would mean a knowledge about the internal design of the drive.

4. request size (**Size**).

With this data we define a vector R, with the following elements: $R.Op$, $R.Addr$, $R.Jump$, and $R.Size$.

We have also evaluated other parameters such as time between requests, larger history, etc., but no improvement has been observed. This lack of improvement is either because the extra parameters do not add significant extra information to the chosen parameters or because they add noise into the model and a good training becomes much more difficult.

In addition, each request has a service time (St). This is normal in the case of the training trace because we need this time to learn the performance of the different requests. In the case where the trace represents a workload to predict, this service time is what we want know, although in this paper we will have it just to be able to compare the real execution of an application and the performance predicted by the drive model.

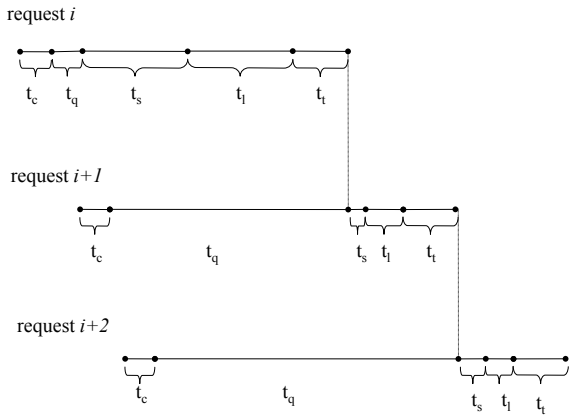

Figure 2. I/O service time separation

The service time (St) can be expressed by the equation:

$$St = t_c + t_q + t_s + t_l + t_t \quad (1)$$

where:
t_c = command overhead time,
t_q = queue time,
t_s = seek time,
t_l = latency,
t_t = time to transfer data.

Figure 2 shows an example with three requests. Here, the request $i+1$ waits and goes to the disk when the request i has been finished; in same way, the request $i+2$ must wait until the $i+1$ finish.

In the same way as Ruemmler and Wilkes did in [11], we have eliminated the queue time in the service time (St) for three reasons. First, our model is aimed at estimating the performance as function of block distribution and we assume that the block distribution only affects the seek and latency time. Secondly, the controller wait time depends on the disk service time and the inter-arrival time of the requests, this last time is application depended; therefore, if we can reduce the first one, we will improve the wait time. Finally, we obtain a better prediction because we do not have a potential cumulative error from previous prediction, if the waiting time were taken into account, an incorrect estimation would imply incremental error for next requests.

Therefore, if we assume that the command overhead time is constant then it can be ignored because changes in the placement will not affect it, the service time is $t_s+t_l+t_t$ and it is modeled like a "black box"

Finally, with all this information we can build a list of requests where each line is a pair (R, St).

Our experience has shown us that it is better two have one model for each operation (Read and write). For this reason, we will use one model for each request type. Thus, we have a set of vector (*trace*) with 3 components(**Addr, Jump, Size**), one for read operations and another for write operations.

3.2 General approach

We propose a general model based on a mathematical function. We assume that we can find a function (M) that approximates the service time St for each request. Thus, our general model can be expressed by:

$$St \approx M(R) \quad (2)$$

where:
R is the input vector ($R.Addr$, $R.Jump$, $R.Size$),
St is the request service time.

Our objective is then, to try to make an implementation of the function M that models St reasonably well. Once built the model, it could be used to find the response time for each request done by any application.

3.3 Performance metric

In order to compare different data placement algorithms we need to define a metric, that will be the output of the drive model. We have decided to use **throughput**

(Kbytes/sec) as our metric because it is a common metric to represent I/O system performance. Then, we defined two throughput values (read and write):

$$T_r = \frac{\sum_{j=1}^{m} R_j.Size}{\sum_{j=1}^{m} St_j}, \text{ if the request is read}$$

$$T_w = \frac{\sum_{j=1}^{n} R_j.Size}{\sum_{j=1}^{n} St_j}, \text{ if the request is write}$$

where:
$R_j.Size$ is the size for request j,
St_j is the service time for request j,
m is the number of read requests,
n is the number of write requests.

T_r is the throughput for read operations and T_w is the throughput for write operations.

3.4 Linear model

The most simple way to implement a function such as M is using a linear approximation. When using a linear model, we assume that the disk drive model, represented by equation 2, can be express by:

$$M(R) = A \cdot R \tag{3}$$

where:
$R = \{R.Addr, R.Jump, R.Size\}$ is the parameter array,
$A = \{a_1, a_2, a_3\}$ is the values array that approximates the linear equation.

To find the values for A that model the behavior of the disk, we have used a multiple linear regression using least squares.

3.5 Quadratic model

As the linear approximation might not be flexible enough, we have also studied a Non linear approximation of M. We add the second degree terms to the linear approach to take advantage of its higher capacity. To implement this approach we use equation 4.

$$M = A \times R + R \times B \times R^T \tag{4}$$

where:
$R = \{R.Addr, R.Jump, R.Size\}$ is the parameter array,
$A = \{a_1, a_2, a_3\}$ are the values array for linear component,
$B =$ an upper triangular matrix with term b_{ij}.

For this approach, we need to compute A and B. We resolve this problem with a nonlinear least-squares data fitting by the Gauss-Newton method.

3.6 Neural Network model

Neural Networks have a high capacity for function approximation [8], and this is exactly our objective. However, to find the best network for a specific problem requires a lot of probes. We have tested a simple architecture based on feed-forward network to resolve the function approximation problem because it has been proved to be a simple and effective approach [6].

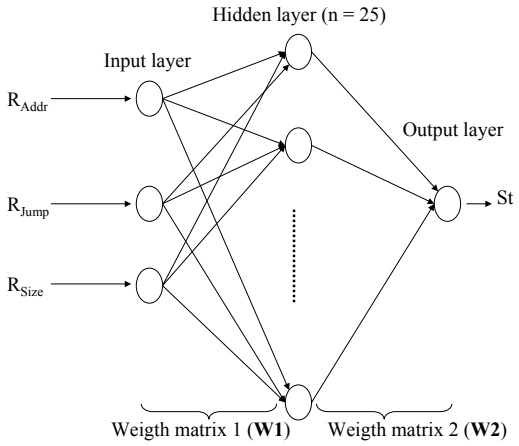

Figure 3. Neural Network Architecture

Figure 3 shows our neural network architecture, it is a feed-forward neural net with following configuration:

- 3 neurons in input layer (one for each component of input vector).
- 25 neurons in the hidden layer using hyperbolic tangent sigmoid transfer function.
- one neuron in output layer (service time) using linear transfer function.

To resolve the problem, we use a Levenberg-Marquardt back-propagation algorithm [10]. This algorithms gives as result the weights matrices W_1 and W_2. These matrices are used, with the transfer function in each layer, to calculate the output. Thus, the service time is approximated by:

$$M = TF_2((TF_1(R \times W_1)) \times W_2) \tag{5}$$

where:
TF_1 is the transfer function for neurons in the hidden layer,
TF_2 is the transfer function for output.

Table 1. Disk comparison

Label	SCSI	IDE	Pendrive
Name	DEC-RZ29B	MAXTOR-4D040H2	PenDrive
Interface	SCSI	IDE-AT	USB
Sectors per track	144 to 252	58 to 118	n/a
Size	4.3 GB	40 GB	128 MB
Rotational speed	7200 RPM	5400 RPM	n/a
Cache (KB)	1024	2048	none
Cache segments	Dynamic	Dynamic	n/a

4 Methodology

To check the validity of our proposals, we have run some benchmarks on different disks and the models obtained have been compared to the real performance obtained.

4.1 Disks

We have chosen 3 disks with significantly different characteristics to check the potential of our model. Table 1 has the basic information about them.

4.2 Applications used

To compare the proposed models we have run three applications that cover a wide range of environments: Linux kernel compilation, SPECWEB99, and TPC-H (scaled to each disk). A description of these applications follows:

4.2.1 Linux kernel compilation

This benchmark has been used in many research projects to measure global system performance [4, 9]. Furthermore, it is a task that is run many times per day in the LINUX development world. To create a new kernel we make the follows steps:

- We do an extensive cleaning and remove the configuration file (*make mrproper*).
- We use the previous settings as the default values (*make oldconfig*).
- We make sure that all necessary files for the make are placed in the correct place and update dependencies (*make dep*).
- We clean up old object files left from previous runs (*make clean*).
- We build the kernel. (*make bzImage*).
- We compile all kernel modules not specified in the kernel configuration file (*make modules*).

4.2.2 SPECWEB99

This benchmark is widely used to evaluate the performance of World Wide Web Servers and was developed by Standard Performance Evaluation Corporation (SPEC) [14].

We used the *wafgen99* utility provided with the SPECweb99 kit to create the workload file_set on the server. This created a *file_set* directory tree. We used a configuration with one client, we left all default values, for example:

- Percent of overall dynamic requests (30 %) (*DYNAMIC_CONTENT=0.3*).
- Percent of dynamic requests that are posts (16 %) (*DYNAMIC_POST=0.16*).
- Percent of dynamic requests that are Custom Ad Rotation GETS (42 %) (*DYNAMIC_CAD_GET=.42*).
- Percent of dynamic requests that are GETs calling the CGI code (5 %) (*DYNAMIC_CGI_GET=.005*).
- Number of files each in each class (*MAX_FILE=8*).

4.2.3 TPC-H

It is a benchmark for decision support databases. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. It has been developed by the Transaction Processing Performance Council (TPC) [16].

We created a database of 1 Gbytes and used *Postgres SQL 7.2.1* for LINUX. We ran all queries, excepting queries number 7, 9, 20 and 21 because these queries contain not supported functions in our Database manager system. We did not use refresh function because we are not interested in global system performance and do not want to bring results in this field.

4.3 Training trace

As we have already mentioned, in order to train our models, we need a training trace. To obtain this trace we need to design a synthetic application that covers all (or at least

most of) the important situations such as consecutive requests, long jumps, short jumps, accesses in inner and outer parts of the disk, small and large inter-arrival times, etc.

We made a synthetic application to create the training trace. This application was developed at user-level with the following characteristics:

- 130000 independent requests,
- 4/5 Reads and 1/5 Writes,
- size of requests: 80% is between 4KB and 128KB, 20% is between 128KB and 2MB,
- 20% of sequential requests and 80% uniformly distributed,
- inter-arrival time is uniformly distributed between 0 and 100 ms. in order to have in the trace all possible inter-arrival times (longer than 100 ms will not affect the I/O performance). It is important to notice that this intervals does not have to represent real behavior, but to have a good range of possibilities for the model to be trained.

5 Results

To see the behavior of the different approaches, we present a set of graphs where the the percentage of relative error (Equation 6) is shown.

$$R_e = |T_{op}(Model) - T_{op}(Real)| * 100/T_{op}(Real) \quad (6)$$

where:
op is read or write,
$T_{op}(R)$ is the real throughput,
$T_{op}(M)$ is the throughput predicted by the model.

5.1 Read operations

Figure 4 shows the results for read operations in each disk. We can observe that the neural network is able to predict the throughput much better than both the linear and quadratic approximations. Furthermore, there are some cases where the error obtained by these two simpler models is higher than 30%. Finally, we can also observe the the results obtained by the neural-network model have the most uniform behavior for all benchmarks. This difference in behavior can be explained because both, the linear and the quadratic models, are too simple to model the real disk behavior. The only exception to this rule is for the pen drive, where all models are very good. This is because read behavior is quite linear, and thus easy to model.

It is also interesting to notice that all the errors observed and neural network are smaller than 10% (actually 8%), which seems to be a very reasonable value for our purposes.

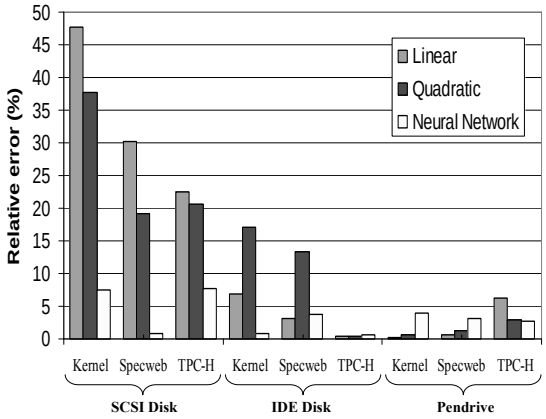

Figure 4. Relative error for reads

5.2 Write operations

Figure 5 presents the same experiments as for read operations, but for write requests. We observe that high errors are achieved no matter which of the three models is used. This high errors in the prediction are specially important for the KERNEL compilation and the SEPECWEB because they have many write operations that are immediate reported and we cannot know when they really end. On the other hand, for the TPC-H benchmarks, the results are reasonably good because the immediate report feature is not an important one in for this application. This happens because, in this benchmark, write requests are big and thus cannot be immediately reported by the disk.

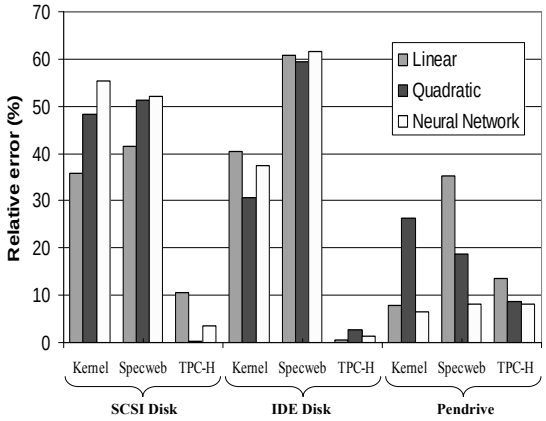

Figure 5. Relative error for writes

Summarizing, the prediction for write operations, if the

disk performs many immediate reports, is not very good. We will see in the related work that this problem also appears in previous work but has not been a big problem for those models to be usable.

5.3 Results with write cache disabled

To verify the immediate report effect we took measures with write cache disabled in SCSI and IDE disks (Pendrive does not have this characteristic) and created new training traces for this case. This test makes sense for two reasons. First, many SCSI disks have this write cache deactivated by default (for fault tolerance issues). And second, we want to verify the reason for the bad models obtained in the previous subsection.

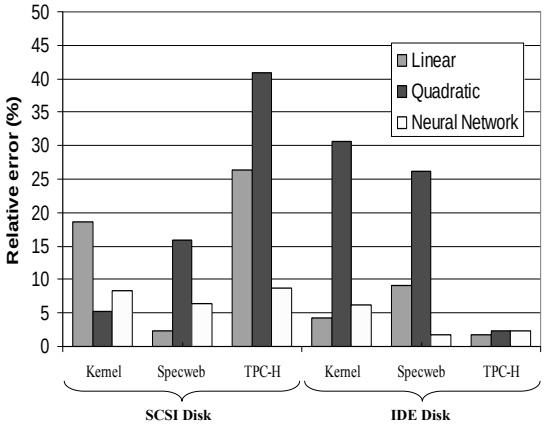

Figure 6. Reads with write cache disabled

We repeat the experiments with write cache off. Figure 6 shows the results for read operations when the write cache is off. Here, we observed similar results as when the cache was on. We know that the immediate report modify the service time for read operation, but these changes are caught by the model with no problem.

Figure 7 shows the results for write operations with write cache off. In these case, we observe a very similar behavior than with read operations.

With these experiments, we can conclude that immediate reports are the reason for the high prediction errors in the write requests and that, should it be deactivated, neural networks would be as good to predict writes as they are to predict read requests.

5.4 Approaches running cost

In addition, we measured the elapsed time to estimate the performance in each approach, we also included measure

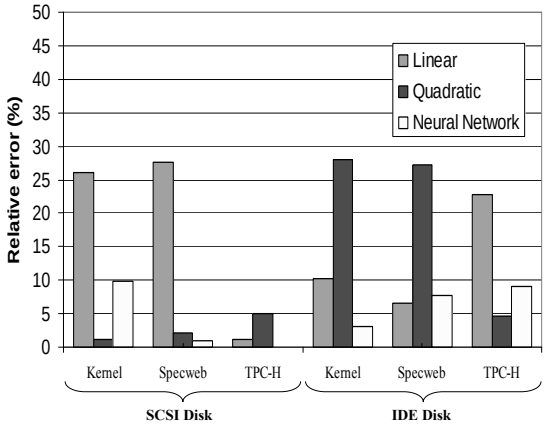

Figure 7. Writes with write cache disabled

with simulation system presented in [5]. Table 2 shows the results for three approaches and the simulation system.

Table 2. Time to estimate the performance in biggest trace

Approach	Time
Linear approximation	50 ms
Quadratic approximation	500 ms
Neural Network	12.4 s
Simulation	332.83 s

As the time needed to estimate the performance depends on the number of requests in the trace we have used the biggest trace to compare the times needed by each model to predict the performance of the trace.

The slowest of the three proposed models is the neural network. It takes around 12.4 seconds to estimate the performance of the mentioned trace. The linear approach takes 50 ms and the quadratic one 500 ms. The neural network approach takes more time because it needs to make two matrix products and to apply the transfer function on each value generated by each neuron, whereas the other approaches, linear and quadratic, make simpler matrix products. However, the time for neural network approach is very short compared to the time used by the simulation system, which takes 332.83 second to finish the same task.

These numbers show that neural networks are fast enough (specially if compared with simulation time) and their accuracy is also good enough (as we have seen above).

6 Related Work

Modeling storage-drive behavior has been an important issue for the last decade. These models have been used for many different reasons, and thus they have different functionality and characteristics. In this section we will do a fast overview of the most important work done on storage-drive modeling. We will also compare this work with our proposal and describe the differences in either functionality or accuracy.

The first kind of models we find in the bibliography are based on simulation techniques. In this group we found specially interesting the proposals done by Ruemmler and Wilkes [11] and the one done by Ganger et al. [7]. The first one proposed a simulator based on the mechanical behavior of disks. The second example is a simulator where the needed parameters are not all mechanical, but also the behavior that can be observed from the exterior of the drive. To extract these parameters, they also proposed an automatic tool named DIXTRAC [12]. In both cases, the simulator had to be aware of the way disk drives work and thus different behaviors always imply changes in the code to implement this new functionality. It is true that simulators usually achieve much accurate results, but the penalty of not treating the drive as a black box and the time they take to simulate (30 times slower) is too high for our purposes.

Another possibility is the usage of analytical models where the input is not a trace (as in the previous group), but a characterization of the load (such as average inter-request time, percentage of read and writes, etc.). In this group, we will also present two very significant projects. The first one was done by Shriver et al. [13] and proposed a model for disk drives. This model needs some previous knowledge about the disk such as seek time, transfer time, cache size, and cache transfer bandwidth among others. This information has to be found by the administrator in order to model a given disk. In addition, there are issues that are not taken into account that may modify the behavior such as different cylinder densities or segmented caches. Regarding the relative error (were reads and writes are not separated), they achieve errors that depend on the disk utilization. For instance, if the disk is utilized a 50%, the relative error is around 20% (depending on the load). Our model is able to achieve good results with no previous knowledge, which is not the case for this model. In addition, our read results are better than the ones presented in this paper, and thus we imagine that they would achieve similar errors that ours should they divide read and write errors. The second example in this group was proposed by Uysal et al. [17] and targeted for disk arrays. This model also needs information from the disk array such as cache characteristics (which is normally very difficult to find). In their tests, only synthetic workloads were used, which means that the real behaviors of real application may not be represented. The relative error they present is up to 40% in read operations and up to 45% in write operations. As we can see these errors are higher that ours, but on the other hand, its has been used in Minerva [1], which is a similar tool than the global one we propose, but with a different objective. They do not relocated disk blocks, but try to find the best configuration for the disk array.

Finally, there is also a group of proposals that treat the drive as a black box and learn the behavior after a training period. The first example in this group is the work done by Thornock et al. [15]. In this work they build a set of tables that represent the function that return the service time for a request in terms of seek distance, queue size, type of operation, and request size. The problem with this proposal is that the tables they propose end up being very large (specially for large disks) and thus may become intractable in real cases. In addition, they trained the model with a subset of trace they use to test the model. This is not reasonable because the model becomes workload depended. It is true they achieved very small errors, but when we use part of the trace to train our model we also achieve excellent results (errors below 1%). Finally, Anderson [2] also proposed a models based on a table. This table returns the throughput for a whole workload as a function of the average-request size, distribution of operation type, sequentiality of request and average queue length. This means that they have a cell in the table for each possible workload. This is too high level for our needs because we want to have detailed performance depending on the distribution of the blocks. Regarding the errors, they achieve error between 10% and 20% depending on the size of the matrix. As they work with workloads, distinguishing reads and writes does not make sense, but their error contain the average of both. Furthermore, this model is being used in Hippodrome [3] which is also a system to configure the storage system. Finally, this approach is very time consuming as they have to compute the Euclidean distance between the new workload and all the ones in the table.

7 Conclusions

In this paper we present a new way to model disk drives. The objective of this model is to become an essential part of a self-managed storage system, and has been thought to work with the other modules in the system.

In this line we have evaluated tree possibilities: linear approximation, quadratic approximation and neural networks. We have concluded that neural networks fulfill all our requirements because they learn from previous behavior, they are fast enough to use and they achieve errors below 10%.

Finally, we have also detected a limitation in our model based on neural networks. The prediction for write opera-

tions, if the disk performs many immediate reports, is not very good. Nevertheless, the results obtained even for write requests is similar to the ones obtained by similar researches done by other groups, and we should not forget that the behavior of our model for read operations is much better that previous proposals. Furthermore, there are many cases where the important issue is to improve read operations because they dominate the performance. For these cases our model will be an excellent approach, and for the rest, it will be as good as what has already been proposed.

7.1 Other possible usages for the model

It is important to notice that this model has been specially designed to be part of the autonomic system presented in the introduction, and this is enough reason for its existence. Nevertheless, it can be used in other areas (possibly with some modifications and improvements).

Another possible usage of the neural-network model is to define the performance of a given storage device. If manufacturers (or researchers) published the neural model for the different storage devices as part of the specification, potential buyers could trace their applications, and feed them in the models for different storage devices to see which one is better for them before they actually buy the storage device.

8 Acknowledgments

This work was supported in part by a grant from FONACIT (Venezuela) which is gratefully acknowledged, by the Ministry of Science and Technology (Spain), and by FEDER funds of the European Union under grants TIC2001-0995-C02-01. We also acknowledge the European Center for Parallelism of Barcelona (CEPBA) for supplying the computing resources.

References

[1] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. *MINERVA: an automated resource provisioning tool for large-scale storage systems. ACM Transcations on Computer Systems*, 19(4):483–518, Nov 2001.

[2] E. Anderson. Simple table-based modeling of storage devices. Technical Report HPL-SSP-2001-04, HP Laboratories, Jul 2001. http://www.hpl.hp.com/SSP/papers/.

[3] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: running circles around storage administration. In *Conference on File and Storage Technology (FAST'02)*, pages 175–188. USENIX, Berkeley, CA., January 2002.

[4] A. D. Balsa. GNU/Linux Benchmarking Practical Aspects. Linux Gazette, Nov 1997. http://www.linuxgazette.com/issue23/bench2.html.

[5] T. Cortes and J. Labarta. *HRAID: A Flexible Storage-System Simulator*. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 772–778. CSREA Press, June 1999.

[6] T. L. fine. *Feedforward Neural Network Methodology*. Springer-Verlag, New York, 1999.

[7] G. Ganger, B. Worthington, and Y. Patt. The DiskSim Simulation Environment (Version 2.0). http://www.ece.cmu.edu/~ganger/disksim/.

[8] M. H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, 1995.

[9] W. Henning. Linux Kernel Compilation Bechmark. Linux Gazette, Sep 1998. http://www.linuxgazette.com/issue32/henning2.html.

[10] D. W. Marquardt. An Algorithms for the Least-Squares Estimation of Nonlinear Parameters. *SIAM Jounal of Applied Mathematics*, 11(2):431–441, June 1963.

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

[12] J. Schindler and G. R. Ganger. Automated Disk Drive Characterization. In *Proceedings of the Sigmetrics 2000*, pages 109–126. ACM Press, June 2000.

[13] E. Shriver, A. Merchant, and J. Wilkes. An analytic behavior model for disk drives with readahead caches and requests reordering. In *Proceedings of the joint international conference on Measurement and modeling of computer systems (SIGMETRICS'98)*, pages 182–191, Madison, Wisconsin, June 1998. ACM Press.

[14] Standard Performance Evaluation Corporation (SPEC). SPECweb99 Benchmark. http://www.spec.org/osg/web99/.

[15] N. C. Thornock, X.-H. Tu, and J. kelly Flanagan. A STOCHASTIC DISK I/O SIMULATION TECHNIQUE. In *Proceedings of the 1997 Winter Simulation Conference*, pages 1079–1086, Atlanta, GA, USA, Dec. 1997. ACM Press.

[16] Transaction Processing Performance Council. TPC-H. http://www.tpc.org/tpch/.

[17] M. Uysal, G. A. Alvarez, and A. Merchant. A Modular, Analytical throghput Model for Modern Disk Array. In *Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS-2001)*, Cincinnati, Ohio, Aug. 2001.