Autonomic Storage System Based on Automatic Learning *

Francisco Hidrobo[1] and Toni Cortes[2]

[1] Universidad de Los Andes, Mérida 5101, Venezuela,
[2] Universitat Politécnica de Catalunya, Barcelona 08034, Spain
hidrobo@ula.ve, toni@ac.upc.es

Abstract. In this paper, we present a system capable of improving the I/O performance in an automatic way. This system is able to learn the behavior of the applications running on top and find the best data placement in the disk in order to improve the I/O performance. This system is built by three independent modules. The first one is able to learn the behavior of a workload in order to be able to reproduce its behavior later on, without a new execution. The second module is a drive modeler that is able to learn how a storage drive works taking it as a "black box". Finally, the third module generates a set of placement alternatives and uses the afore mentioned models to predict the performance each alternative will achieve. We tested the system with five benchmarks and the system was able to find better alternatives in most cases and improve the performance significantly (up to 225%). Most important, the performance predicted where always very accurate (less that 10% error).

1 Introduction

One of the main trends in the computing world is the increasing needs for I/O capacity and performance shown by applications.

Many approaches to solve this problem have been proposed in the last decades. One of the most promising consists of configuring the storage system and the placement of data 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 workload. Currently, these optimizations are usually done by experts who use their experience and intuition to make this configuration and placement. 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 objective is to design a storage system capable of extracting all potential performance and capacity available in a heterogeneous environment with as little

* 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.

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, and the best placement within each device.

The aim of this paper is to present the global design of our proposal. This design includes a disk model based on neural networks, an approach to model the workload based on reduced traces and some sample strategies to generate different placements.

2 Autonomic storage system: a global picture

In order to place the work in the right context, we will first give a global description of how the whole system works, and then we will describe in detail each of the parts that build it.

In the starting point, when the system is new, we have a set of disks attached to the system. The first thing we have to do is to model them. This model should have two main properties. First it should be able to predict the performance of a given workload, without having to run it. Second, it should treat the disk as a "black box".

Once we have all disks modeled, and thus we can predict the performance of any possible workload, we start learning the workload behavior. This step mainly consists of tracing the requests done to the disk and keeping them in file-system internal data structures.

Periodically, which could be once a day, once a week, or any other period depending on the needs, the system uses the workload behavior learned to generate different placement alternatives that may (or may not) improve the performance of the applications running on the system. As these new placements cannot be implemented and tested, we will use the disk models to predict the performance each new placement would achieve.

After the performance of all proposed placements is predicted, we pick the best one and compare it with the performance of the current workload (which has been learned at the same time as the workload). If the new placement is better that 10% the performance of the current one, then we take the effort of moving the blocks to implement the new placement and thus improve the performance of the applications using these disks.

It is also important to see, that whenever a new disk is added, it has to be modeled and then it will be used by the generator of placement alternatives to place blocks in it.

Once described the module as a whole, it is important to notice that the objective of this paper is to test that all modules are possible and that the final system works well. Our current version is not integrated in the file system, but it is implemented as separated modules that work off-line, but placing them in the kernel and running them on-line it is just an implementation problem that lies as future work.

3 Disk Model

The main objective of this module, and also the main difference compared to other approaches, is to design a model that has no previous knowledge about the drive to model.

In our system, 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:

$$St \approx M(R)$$

where: R is the input vector with components:

R_{Addr}: address of the first-requested block,

R_{Jump}: difference (in blocks) between this request and the previous one.

R_{Size}: request size.

St is the request service time.

Our experience has shown us that it is better to have one model for each operation (Read and Write). Thus, we will use one model for each request type.

In [1], we studied different approaches using several applications and drives, and we found that neural network are the best mechanism to model drives.

Neural Networks have a high capacity for function approximation, and this is exactly our objective. 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.

We use a feed-forward neural net with the following configuration: 3 neurons in input layer, 25 neurons in the hidden layer, and one neuron in output layer (service time). To resolve the problem, we use a Levenberg-Marquardt back-propagation algorithm.

As we can see, this approach behaves as a "black box" and does not need any previous knowledge about the device, it can learn the behavior by running a synthetic trace. This learning process was described in a previous paper [1].

4 Workload model

We focused the workload model on files because it allows us to focus the effort toward the most important (i.e. used) files of the application to reduce the size of the model and the complexity of the data-placement module.

Our file model is based on the segments accessed and their relationship. We define a segment as a set of blocs requested in a single disk operation. Please keep in mind that our model works at disk-drive level and thus segments are not necessarily what users request, but what the operating system (or file system) sends to the disk drive after being filtered by the buffer cache and some possible reordenation done during the process. Thus, our file model stores, for each segment, the following information:

- The first logical block requested. Using logical blocks makes our model independent of the physical location of the block.

- The total number of requests made to this direction. Used in order to keep some information about the importance of different segments.
- Request type (read/write). This value is expressed as read probability.
- The average size of the requests made to this segment.
- The list of possible predecessor requests. This list contains information about the requests that preceded it in the trace. Each entry in the list has:
  - the file id. This field is used for two reasons. The first one is to identify the physical location of the last accessed block. Second, it is used to find relationships among different files.
  - the last logical block of the previous request. This field is used to find the number of blocks the disk has "jumped" from one request to the other.
  - the number of times that this element preceded the request. This field is used to know the percentage of times this sequence occurred.

As we can see, this model fulfills all our requirements. It can be generated on-line because new requests can be added very easily. No physical information is kept, and thus it is block-location independent. It does not keep any time information and thus it is device independent. It does not grow unnecessarily but it has enough information (even inter-file relationships) for the data placement module to take decisions. Finally, building a trace with the information required by the drive model previously described (initial block, R/W, jump and size of the request) is trivial.

5 Prediction and data placement

This module implements its functionality through three independent steps: generation of placement alternatives, evaluation of the generated alternatives, and application of the best placement found.

5.1 Generation of placement alternatives

In order to generate a placement, we first decide a logical order in which blocks should be placed and then we decide in which disk blocks we map them. Finally, we may refine the solution to take into account specific drive characteristics.

Logical ordering The main idea is to encourage spatial locality. This approach can reduce the seek distance and can increase the probability that the disk read-ahead improves the general performance. We can follow two different approaches:

a) **Based on the access pattern**. We use the access pattern represented in the file model, to find the probabilistic access sequence. We can generate a directed graph $G(V, E)$, where the vertex set (V) represents the accessed blocks. The set of edges (E) is defined as follows: E_{ij} belongs to E if block i is acceded after block j. Furthermore, each edge E_{ij} has a weight (edge weight), which represents the number of times that block j is preceded by i.
In order to generate the logical ordering, we eliminate edges, taking only the

edge that has the largest weight, until each node only has one input edge and one output edge, except the first and the last one.

b) **Based on sequential structure**. We directly use the logical order of the file regardless of the real accesses. This means that we use the pointers to data blocks to make the list.

Physical placement Once we have the logical order, we need to find a spot (or spots) in the disk to place them sequentially according to the logical order. To do it, we have several alternatives regarding the sets of disk blocks we consider to relocate the file. The three studied alternatives are presented here:

a **All disk blocks**. It uses all disk blocks except those that contain metadata such as superblock copies. This option may move blocks from other files.
b **File-used blocks**. It uses all blocks used by the defined as important files.
c **File-used blocks plus free blocks**.

Refinement: special drive conditions Depending on the drive type, it may have some special conditions that could be taken into account to refine the physical placement. In this work we have only tested one that deals with the different density of disk zones.

Hot blocks. If the disk has a faster area, we can try to place the most used blocks in this area. We will divide the previous list in two sections. One, as big as the fast zone, with the most used blocks. A second one, with the rest of the file blocks. Both lists will be ordered as explained before. This refinement looses locality, but gains bandwidth, a trade off that has to be evaluated. To simplify things, we will only apply this refinement when all disk blocks are used.

5.2 Evaluation of alternatives

For each proposed placement, the system generates a new allocation map and takes the file model for each "important" file to create an estimate trace for these files. Then, the disk model is used to predict the performance.

5.3 Application of the proposed placement

Here, the module applies the placement proposed by the "winning alternative". In order to take some action we demanded that the placement improves the performance, at least, 10%. We decided to use 10% as threshold, because smaller gains may not be enough for the trouble of data movement and because our predictions have a potential error of up to 10%.

6 Experiments and results

In this section, we are going to show how our global system works. In order to show it, we are going to use different workloads and to run the workload in a file system artificially aged according to [2] (We also executed the tests in a new file system; but for space reasons will not be presented)

6.1 Workloads

Synthetic benchmarks

1 **SSA**: Simple Sequential Access. The application reads one file sequentially. In each read it requests 64KB. It will show us the behavior in the simplest case: accessing a single file (but no necessarily using regular intervals).
2 **FSA**: Fixed Stride Access. In this case, we used more complex pattern access than sequential. It uses a stride of 512KB and reads 8KB in each requests.
3 **MSA**: Multiple Sequential Access. Here, the application reads simultaneous three files. It requests a segment of the first file, then a segment of second one and a segment of third file. In each read it requests 64KB. These operations are repeated until reading the files completely. It will allow us to observe what happens when several files are involved, but with a reproducible behavior.

The inter-arrival time was uniformly distributed (0 and 100 ms) in all case. The files used were 300Mbytes each and we called them: $file_1$, $file_2$ and $file_3$.

Sequence of access to WEB pages In this case, we took a personal WEB page generated by a standard tool. This information is organized in many directories and files. In the test, we accessed a sequence of pages, where each requested page implied the access to a set of related files, which are normally small in size. Therefore, in this application the interrelation between files is more important that the access pattern within each file.

TPC-H benchmark Finally, we have tested the TPC-H benchmark, which is a decision support benchmark for Databases. We created a database of 1 Gbytes and used *Postgres SQL 7.2.1* for LINUX. We ran all queries, except queries number 7, 9, 20 and 21 because these queries contain functions not supported by our database manager system.

6.2 Results

Following the procedure described in the section 5.1, we created 10 possible placements for each list of important files. Table 1 presents all possible combinations and the names we have given them.

Table 1. Possible placements implemented

		Based on **A**ccess **P**attern	Based on **S**equential **S**tructure
All **d**isk blocks		APAD	SSAD
	Hot blocks (all file blocks **f**it)	APAD_HF	SSAD_HF
	Hot blocks (**n**ot all file blocks **f**it)	APAD_HNF	SSAD_HNF
Used **b**locks		APUB	SSUB
Used+**f**ree blocks		APUF	SSUF

Synthetic Workload on an aged file system If we apply the SSAD placement alternative for the SSA benchmark created on an aged file system, we could improve the performance in 44.5% as Figure 1 shows. In this experiment it is important to observe that, although the access pattern should be sequential, APAD does not achieve as good performance as SSAD. The reason behind this unexpected behavior, is that some requests were reordered when the access pattern was learned, but the real access is sequential. If more repetitions had been done to learn the pattern, as will probably be in a real system, this reordenation will only appear in the pattern if they occur very frequently and not just once.

For the FSA benchmark, Figure 2, the improvement could reach up to 225% with APAD placement.

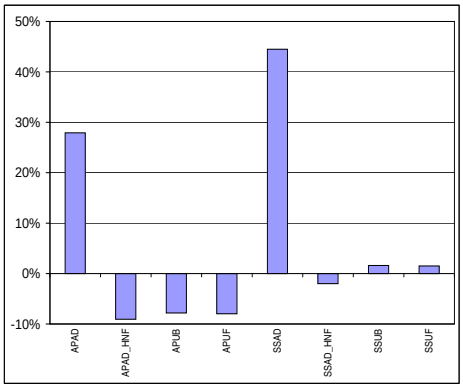

Fig. 1. SSA on on file system aged

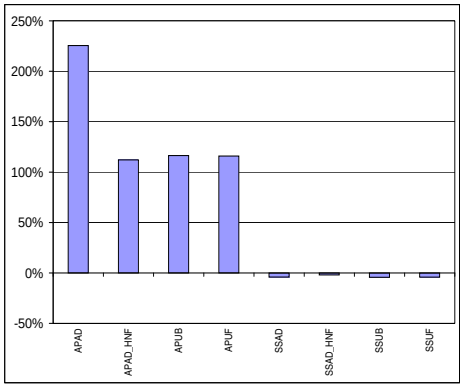

Fig. 2. FSA on file system aged

With regard to MSA benchmark, Figure 3 shows that the best placement, APAD, could improvement the performance over 100% for three files.

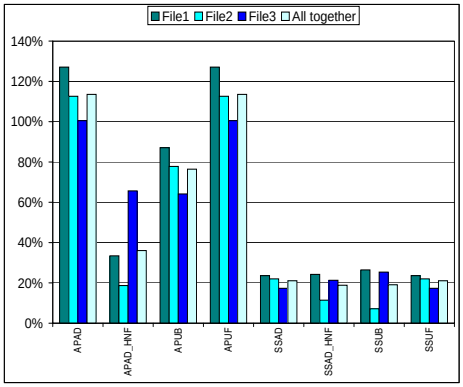

Fig. 3. MSA on file system aged

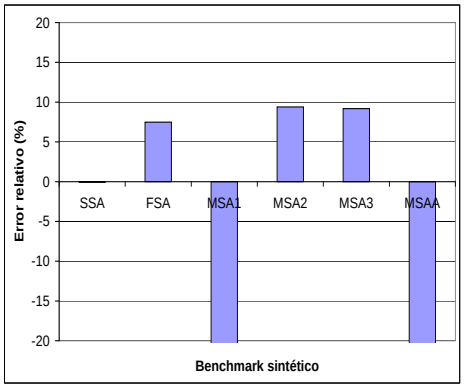

Fig. 4. Best placement (Relative error)

When we applied the winning placement and measured the real performance (Figure 4), we can observe that the relative errors were again very small, but for file1 of MSA where the relative error is over 100%, which makes the global error for the MSAA benchmark grow unexpectedly.

The problem in MSA benchmark is that the buffer cache policies generate a different behavior when the blocks are remapped. Therefore, the access pattern learned and used to make the prediction is very different from the one obtained once the new placement has been applied. To prove this fact, we executed MSA benchmark turning off the read-ahead mechanism in the buffer cache. In this new experiment, the relative error was, once again, below 10% for all files. Obviously, the proposal is not to eliminate the read-ahead in the buffer cache to obtain reliable results; rather the idea is to develop a integrated approach between our system and the file system to guarantee that such changes of behavior will not be produced or will be taken into account by the prediction system.

Sequence of access to pages WEB on a file system aged As we can see in figure 5, the values show us that the alternatives based on the access pattern could improve the performance near to 100%. The reason for this behavior is that file were spread throughout all the disk while they need to be close.

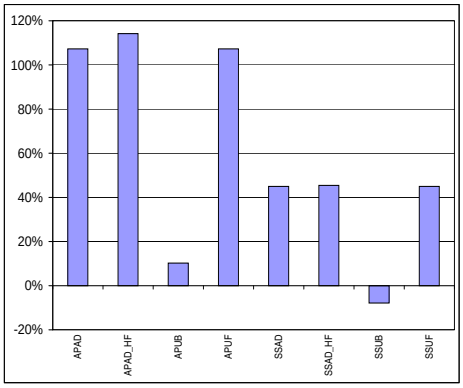

Fig. 5. Sequence of access to WEB pages

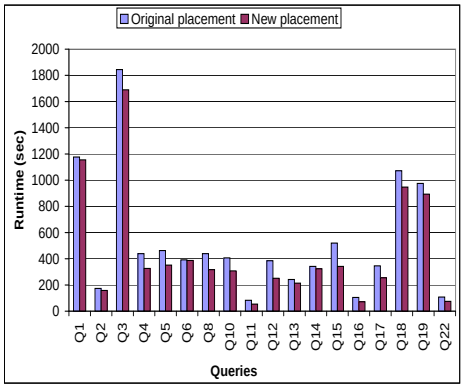

Fig. 6. Runtime reduction for TPC-H

TPC-H on a file system aged The best placement alternative for the TPC-H on an aged file system is SSAD, and could improve the performance in 142% for access to partsupp table, 73 % for order table, 32% for lineitem table and 39% if we take all tables together.

To give a final idea of the potential improvements, we computed the real runtime improvement for each query and then all queries together. Figure 6 shows the runtime (in seconds) obtained for each query with the original placement and when the placement SSAD was applied. The global reduction is 14.64%.

In a real system, we would not first learn the behavior, then move data, and finally reexecute the application. The normal behavior is that we learn during some time, move the data, and then continue the execution (not repeating exactly what we learned). To test how our system would behave in such an environment we tested WEB and TPC-H benchmarks. In the first case we learned from one sequence of pages (one possible session) and tested with a different one (another session by another user). In the TPC-H we learned from half of the queries and tested the other half.

The results showed that the performance improvement obtained with the new placements and the "not learned" part of application were similar to the ones presented in the previous sections. This means that the system is able to do accurate predictions and it can be used for autonomic computing.

7 Related Work

The first kind of storage-drive 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 [3] and the one done by Ganger et al. [4, 5]. 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 [6] Both, simulations and analytical models need prior knowledge of the disks, which is not always easy to find, while our neural network approach does not. Finally, there is also another group of proposals that treat the drive as a "black box" and learn the behavior after a training period [7, 8] but they either need huge data strutres or work at a very high level (not request level as we need).

With regard to application model, some works have been proposed in characterizations and modeling of I/O access pattern [9–14]. Most of these studies were made at the file system level. Gómez and Santonja [15] developed an approach to analyze and model disk access pattern but cannot be learned and used on-line. In some aspect our workload model is similar to the presented by Madhyastha and Reed in [16], and by Oly in [17]. They used Hidden Markov Model (HMM) for classification and prediction of I/O access patterns. Basically, They use the model to predict the next action for the file system or to propose an allocation strategy according to the access pattern classification.

Finally, some work has been done on block replacement to improve I/O performance [18, 19], but are not integrated in an autonomic system as ours.

Regarding our global system, there are some tools similar to our system such as MINERVA [20] and Hippodrome [21] development in the HP Laboratories. However, they target their work to the configuration of RAID systems.

8 Conclusions

In this paper, we have presented an autonomic storage system based on automatic learning of previous behavior. This system uses a modular design to explore different placement possibilities and decides which one is better and whether the new placement is worth the effort of making the changes.

In addition, to achieve the autonomic storage system, we have proposed novel mechanisms to model disks and applications that may be of use to the research community in different environments.

The tests done showed that the performance prediction is reliable. Furthermore, we have also shown that this prediction can be used to improve the performance significantly when the data is not correctly placed.

References

1. Hidrobo, F., Cortes, T.: Towards a Zero-Knowledge Model for Disk Drives. In: Proceedings of the AMS, Seattle, WA, USA, IEEE Computer Society Press (2003)
2. Smith, K.A., Seltzer, M.I.: File System Aging Increasing the Relevance of File System Benchmark . In: Proceedings of SIGMETRICS. (1997)
3. Ruemmler, C., Wilkes, J.: An introduction to disk drive modeling. IEEE Computer **27** (1994) 17–28
4. Ganger, G., Worthington, B., Patt, Y.: The DiskSim Simulation Environment (Version 2.0). http://www.ece.cmu.edu/~ganger/disksim/ (2004)
5. Schindler, J., Ganger, G.R.: Automated Disk Drive Characterization. In: Proceedings of SIGMETRICS, Santa Clara, CA, USA, ACM Press (2000) 112–113
6. Shriver, E., Merchant, A., Wilkes, J.: An analytic behavior model for disk drives with readahead caches and requests reordering. In: SIGMETRICS, Madison, Wisconsin, USA, ACM Press (1998) 182–191
7. Thornock, N.C., Tu, X.H., kelly Flanagan, J.: A STOCHASTIC DISK I/O SIMULATION TECHNIQUE. In: Proceedings of Winter Simulation Conference, Atlanta, GA, USA, ACM Press (1997) 1079–1086
8. Anderson, E.: Simple table-based modeling of storage devices. Technical Report HPL-SSP-2001-04, HP Laboratories (2001) http://www.hpl.hp.com/SSP/papers/.
9. Kotz, D., Nieuwejaar, N.: Dynamic File-Access Characteristics of a Production Parallel Scientific Workload. In: Proceedings of Supercomputing, Washington, DC, USA, IEEE Computer Society Press (1994) 640–649
10. Kroeger, T.M., Long, D.D.: The Case for Efficient File Access Pattern Modeling. In: Proceedings of HotOS, AZ, USA, IEEE Computer Society (1999) 14–19
11. Ware, P.P., Jr., T.W.P., Nelson, B.L.: Modeling File-system Input Traces via a Two-level Arrival Process. In: Proceedings of the 28th Conference on Winter Simulation, Coronado, California, United States, ACM Press (1996) 1230–1237
12. Ware, P.P., Thomas W. Page, J.: Automatic Modeling of File System Workloads Using Two-Level Arrival Processes. ACM TOMACS **8** (1998) 305–330
13. Ruemmler, C., Wilkes, J.: *UNIX Disk Access Patterns.* In: Proceedings of Winter USENIX Conference, San Diego, CA, USA (1993) 405–420
14. Ganger, G.R.: Generating Representative Synthetic Workloads. An Unsolved Problem. In: Proceedings of 21st International Computer Measurement Group Conference, Nashville, TN, USA, Computer Measurement Group (1995) 1263–1269
15. Gómez, M.E., Santoja, V.: A new approach in the analysis and modeling of disk access pattern. In: Proceedings of ISPASS, Austin, Texas, IEEE Press (2000)
16. Madhyastha, T., Reed, D.: Input/Output Access Pattern Classification Using Hidden Markov Models. In: IOPADS, San Jose, CA, USA, ACM Press (1997)
17. Oly, J.: Markov Model Prediction of I/O requests for Scientific Applications. Master's project, University of Illinois at Urbana-Champaign (2000)
18. Vongsathorn, P., Carson, S.: A system for adaptive disk rearrangement. Software - Practice and Experience **20** (1990) 225–242
19. Akyürek, S., Salem, K.: Adaptive Block Rearrangement. ACM TOCS **13** (1995) 89–121
20. Alvarez, G., Borowsky, E., Go, S., Romer, T., Becker-Szendy, R., Golding, R., Merchant, A., Spasojevic, M., Veitch, A., Wilkes, J.: *MINERVA: an automated resource provisioning tool for large-scale storage systems.* TOCS **19** (2001) 483
21. Anderson, E., Hobbs, M., Keeton, K., Spence, S., Uysal, M., Veitch, A.: Hippodrome: running circles around storage administration. In: Proceedings of FAST, Monterey, CA, USA, USENIX, Berkeley, CA. (2002) 175–188