Optimization of Mass Storage Hierarchies

Bruce Jacob

Advanced Computer Architecture Lab
EECS Department, University of Michigan
blj@umich.edu

Tech Report CSE-TR-228-95

May 12, 1994

Optimization is often a question of where one should put one's money in improving performance. As far as large storage hierarchies go, intuition suggests (and common practice supports) adding as much as is affordable of the fastest technology available. Many cache hierarchy studies have shown that this is often not the optimal approach, and we show that for mass storage hierarchies it always tends to be the wrong approach. For large data sets, as is the case for network file servers, a machine with no RAM and several gigabytes of disk performs 30% faster than a machine with no disk and a cost-equivalent amount of RAM.

This paper presents a mathematical analysis of the optimization of an I/O hierarchy, as well as trace driven simulations of a network file server in support of the analysis.

1.0 Introduction

Over the past several years, there has been a substantial increase in the speed and capacity demands placed on computer memory systems. Great strides have been made in the capacity of mass storage devices such as magnetic disk, tape, and optical media, but improvements in the speed of these devices has not kept up with the improvements in CPU speeds and semiconductor memory. Caching is used to hide the deficiency, but the widening gap between semiconductor memory and magnetic storage is making it easy to lose much performance through poor system configuration. Larger miss costs need to be offset by higher hit rates, inducing system administrators to buy more and more main memory for workstations; this is not necessarily the correct thing to do, especially in the case of file servers. In this environment, it is very important to spend one's money wisely, as the cost figures are staggering and $5,000 misplaced can result in a difference in performance of more than a factor of two.

Historically, much emphasis and attention has been paid to optimizing cache hierarchies [5][6][11][14][19], to managing large stores of data, and to having enough memory in a system [9]; however, little attention has been paid to how all the optimizations fit together. Previous studies have demonstrated that the optimal size *and* number of cache levels increase with an increasing data set size [5][6][14], and yet as our systems grow larger we continue to pour RAM into systems instead of adding to the caches on disk, or even more levels in the hierarchy.

Typically, system administrators play a reactive game, adding more RAM when the system is thrashing, and adding more disk when the users complain of too little file space. We demonstrate that this technique can cost over 30% in performance, and most system configurations tend to sit in the worst-case region.

1.1 Direction

The goal is to optimize the mass storage hierarchy under given cost constraints as well as reality constraints; for instance, DRAMs are not available in a continuum of speeds and sizes, but instead come in a few different speeds and only a few different cost-effective sizes.

Already, much research has gone to optimizing single-user workstation-class machines on a network, and the results are largely buy more RAM. The focus of this paper is instead *how to optimize machines connected to mass storage*. These few but important machines are typically file servers, with allotted budgets large enough to allow an administrator to have possibly too many configurations to choose from.

1.2 Background

Caching is used to mask the unacceptable latency of accessing slow devices. The hard disk, which was once used as the mass store, is now used as a cache to even larger tertiary storage, and the semiconductor memory becomes merely a higher-level cache. The enormity of I/O space is changing the way computer systems are being used, designed, and optimized.

There have been a number of research projects exploring this new frontier of computing. Nakagomi, et al, [13] suggested replacing the entire hierarchy with a single large, fast magneto-optical array, trading the top level access time for a much higher bandwidth. In addition, less burden would be placed on the server machine due to a reduction in the overhead of copying data back and forth between several levels of storage devices. Several papers [3][7][20] suggest the use of non-volatile RAM to improve I/O performance by providing a reliable write-buffer to reduce physical write traffic and improve response time. Finally, future devices such as holostore [17] are projected to fit nicely in between semiconductor memory and magnetic disk in the storage hierarchy. Holostore technology offers capacity that approaches the density of magnetic disks with access times closer to semiconductor memory.

Another class of research involves the reconfiguring of traditional I/O devices in novel ways, rather than adding new types. Combining devices such as magnetic disk drives into disk arrays exploits parallelism yielding an increase in bandwidth [10]. Tape striping performs a similar optimization for magnetic tape drives [8].

However, in the many papers on making large storage systems faster, there has been little mention of the enormous wealth of research done to optimize cache hierarchies. Most of the research mentioned looks at improving a single level of the hierarchy, or reducing the number of levels in the system. It is perhaps an obvious point, but one which has been largely ignored; the work done in finding optimal memory hierarchies is very applicable to the area of mass storage hierarchies.

1.3 Results

This paper presents a new twist on old models for cache hierarchies, as applicable to mass storage hierarchies. The model uses a measure of program traces that is independent of cache sizes and configurations; something lacking in many cache analysis reports. The model is used to predict hierarchy configuration behavior in the face of cost constraints, and simulations of a network file server are used to check the model.

The results are that finding an optimal configuration at some cost point is partly a function of finding the optimal configuration at a smaller cost point, i.e. the optimization problem has optimal subproblems. This is good news to system administrators; it suggests that if a certain amount of money is spent creating an optimal system configuration, that money will not be wasted when it comes time to upgrade the system (as long as the technologies involved do not change radically).

An optimal configuration at a higher cost point will not have *less* of anything than the original configuration; it will never make sense to take some memory and trade it in for disk, or vice versa.

A surprising result is that a balanced system is one with several gigabytes of disk cache and *no RAM*. Remember that this paper is concerned with server machines, which is analogous to looking at level-2 caches; the program traces seen by a L2 cache have been filtered by the L1 cache and exhibit less locality of reference, driving miss ratios up by large factors [14]. It is not surprising, then, that in order to improve performance of a server machine, size counts more than speed. A similar result is championed throughout the literature; when you haven't much money to spend, or when the locality is poor, large and cheap is better than small and fast [10][18].

A performance ratio is presented which appears in many of the calculations. It is derivable from the base equations and is independent of cache sizes, hierarchy configurations, and process traces. It is a characterization of how the technologies in the hierarchy interact, in terms of performance as well as cost.

The performance ratio is used to predict behavior and where optimal configurations lie. A very simple heuristic is presented which uses the ratio and for any size of the L1 cache produces the optimal size for the L2 cache, and vice versa. Results are calculated based upon a given workload and are specific to that workload, however that and the technology specs (cost per bit, access time) are all that the model depends upon.

The main conclusion drawn is that you can never have too much disk. As file sizes grow, as the number of users grows, as the number of files per user grows, the importance of large on-line storage will grow. Typical client-side disk caches are on the order of several to several dozen megabytes in size, so a server-side L1 RAM cache of this size or less is a waste of valuable funds. However, as most network file systems have some sort of write-through policy to guarantee consistency, writes tend to make the client-side cache look smaller, so several megabytes of RAM will not be useless.

Regardless, a fundamental result is that there should be several gigabytes more disk cache than RAM cache in a system; which is not to be confused with simply attaching disk to a system. Disk cache is to be used *entirely* as cache to the file system; it is not meant to simply contain the file system. Performance numbers indicate that a poorly configured system can be several times slower than the optimal configuration at the same cost, so the distinction is critical.

1.4 Overview of Paper

Section 2.0, *Mass Storage Hierarchies*, describes mass storage hierarchies and the technologies that comprise them, Section 3.0, *Mathematical Analysis*, presents a model of such hierarchies and analyzes the model to determine how to find optimal configurations at given cost constraints. Section 4.0, *Simulations*, describes the simulator and program traces used to check the validity of the model, while Section 5.0, *Comparison of Simulated and Analytical Results*, compares the results of the simulations with the conclusions drawn directly from the model.

2.0 Mass Storage Hierarchies

A typical mass storage hierarchy consists of semiconductor memory used as I/O buffer pools by the operating system, one or more magnetic disks, and a large tertiary storage device such as a cartridge tape or optical disk carousel. As storage requirements become greater, the use of more cost-effective storage media, such as magnetic tape and optical disks, become an integral part of the storage hierarchy. This paper assumes a system configuration similar to that described in Antonelli, et al. [2]; a large tertiary storage device housing the entire file system, with RAM and magnetic disks acting as L1 and L2 caches on top.

The following sections describe the various technologies used in storage hierarchies, including their strengths, their weaknesses, their costs and their performance numbers.

2.1 Volatile Semiconductor Memory

Volatile semiconductor memory, typically DRAM, is what is primarily used as the top level of the storage hierarchy. DRAMs can be accessed in tens to hundreds of nanoseconds (typically 80), and cost approximately $30 per megabyte [1]. DRAMS have a transfer rate of roughly 160 MB/s, derived from a latency to first access on a page of 80 ns and subsequent accesses to the same page are on the order of 25 ns [12].

DRAM is very good at improving I/O read performance but suffers from its volatility; it does little good in speeding up writes, as in order to maintain consistency and integrity writes must often pass immediately through the volatile RAM cache to some non-volatile storage (typically disk) at a lower level.

2.2 Non-Volatile Semiconductor Memory

Several types of devices fit into this category. Currently, NVRAMs have similar access times to DRAMs, but slightly higher costs per megabyte. The most common type of NVRAM is battery-backed DRAM. This configuration obviously has the same access latency of regular DRAM, but with the additional cost of the battery or uninterruptible power supply. Safe RAM [7] is implemented in this form of NVRAM.

The advantage that non-volatile memory holds over volatile RAM is that it can truly be used in place of disk. Once an item of data is in non-volatile RAM it is as safe as on disk and need not be flushed out just to ensure its integrity. The result is faster writes through the use of NVRAM [3]. Also, Flash RAM (a type of NVRAM) has cells that tend to be much smaller than DRAM cells, indicating a possibility that Flash will in the future become cheaper than DRAM.

2.3 Volume Holographic Storage

Volume holographic storage (*holostore*) is a page-oriented device that writes and reads data in an optical form. The data consists of a 2-D array of spots called a page. The storage medium for a read-write holostore subsystem can be a photo-refractive crystal. The device is called holostore is because the Fourier transform of the data spot array is a hologram. Pages are placed in the photo-refractive crystal as a 2-D array of stacks and are accessed by a light beam diffracted through the crystal at different angles.

The advantage to holostore is that the access time for data is projected to be on the order of a few microseconds, and all data on a page is retrieved simultaneously (which is not meant to imply that it all arrives into main memory or the processor cache simultaneously). The storage capacity of holostore is competitive with magnetic or optical disk; since it is a 3-D storage media, its volumetric storage density is extremely high [17]. Unfortunately, it is not yet available.

2.4 Magnetic Disk

Magnetic disks are most commonly used for on-line storage. Access latencies for magnetic disks make them barely fast enough to be considered “on-line,” and they are still far too slow to communicate directly with a processor. Disk access times are typically in the tens of milliseconds, and for this reason magnetic disks would typically be used as a second-level cache (below the I/O Buffer Memory) in the storage hierarchy. Most installations actually use magnetic disks not as a cache to the file system but to store the entire file system itself. As suggested in [2] and in talks given by Antonelli this tends to be an increasing waste of precious resources as file systems get larger.

Although the access time for magnetic disks have been improving at a very slow rate, capacity continues to grow rapidly. Striping data among several disks in a disk array, such as in RAID [10] can improve the bandwidth of magnetic disk devices, but the single-request latency remains high for small requests. Typical bandwidth numbers for disk are around 10 MB/s, average latencies are around 10 ms, and for large drives the cost is $0.50 per MB [1].

2.5 Magnetic Tape

Traditionally, magnetic tape has been used as a backup file system not very well integrated with the rest of the devices in the hierarchy. The large storage capacity and low price-per-bit makes this medium very attractive for tertiary storage as user file spaces continue to grow. Since tape is a serial device, the latency for accessing data on a tape can be very long due to the time to serially search through the tape for the correct data. In addition, writes may only be done in an append-only fashion.

If properly placed into a storage hierarchy, however, magnetic tape can be effectively used. Assuming the higher levels of the hierarchy service most of the file accesses, the average access time for accessing data in a storage system backed by magnetic tape can remain reasonable. Similar to magnetic disks, tape striping has been proposed to improve the band-width of magnetic tape devices [8].

Bandwidth numbers for cartridge tape drives range from 100 KB/s to 50 MB/s, average seek times are around 10 sec to 40 sec, depending upon the technology and capacity of the cartridge [8].

2.6 Optical and Magneto-Optical Disk

Slow to become mainstream for a variety of reasons, optical disks nevertheless have the potential to replace magnetic tape as the tertiary storage media of choice. Optical disks have a price-per-bit similar to magnetic tape and allow efficient, random access to large volumes of data and could very well prove to be the more effective of the two [15]. In fact, Nakagomi, et al. [13] suggest replacing everything from the magnetic disk array on down to the tape backup devices with one large (terabyte-sized) magneto-optical device.

Optical disk bandwidth figures range from less than 1 MB/s to just over 10 MB/s, average seek times range from 60 ms for smaller drives to 7 seconds for large jukeboxes [16].

3.0 Mathematical Analysis

Quite a bit has been published on the optimization of cache hierarchies. Chow showed that the optimum number of cache levels scales with the logarithm of the capacity of the cache hierarchy [5][6]. Rege and Garcia-Molina demonstrated that it is often better to have more of a slower device than less of a faster device [9][18]. Welch showed that the optimal size of each level must be proportional to the amount of time spent servicing requests out of that level [19].

This paper builds upon previous cache studies with one major twist. Previous studies have been able to find solutions for optimal hierarchy configurations, but the results contained dependencies upon the cache configuration; the number of levels or the sizes of the levels or the hit rates of the levels. This paper presents a model for a cache hierarchy that is dependent upon the characterization of the technologies that make up the hierarchy as well as a characterization of the application workloads placed upon the hierarchy and nothing else. This makes it very easy to find closed form solutions to the optimization problem.

3.1 Welch’s Analysis

Welch took a model of a memory hierarchy from Chow [5] and concluded that if you have a fixed budget and a known set of probabilities of hitting any given level of a memory hierarchy, you can find the optimal appropriation of funds to the various hierarchy levels, so that the average effective access time is minimized. This optimal appropriation leads to such a *balanced* memory hierarchy when the proportion of money spent at a given level is equal to the proportion of time spent at that level servicing requests [19].

Formally, if every hierarchy level i has probability P_i of being accessed, every level has a total cost B_i (equal to cost/byte times capacity), and the time to access hierarchy level i is t_i, then for every fixed total system cost $S = \sum B_i$ the average time per access $T_{avg} = \sum B_i t_i$ is minimized when

$$\frac{P_i t_i}{T_{avg}} = \frac{B_i}{S}$$

which is when the proportion of dollars spent at each level is equal to the proportion of time spent at each level. We would like to be able to rearrange this to get

$$\frac{S}{T_{avg}} = \frac{B_i}{P_i t_i}$$

which would suggest that to minimize the average access time T_{avg}, the fraction S/T_{avg} should be conserved across all levels of the hierarchy, for a given system cost S. If this were true, then to maintain a balanced memory hierarchy when you increase the capacity of the hierarchy, the amount spent *at each level* needs to increase, proportional to the technology's performance and inversely proportional to the amount of time spent at the level. Assuming that the probability distributions do not change by an enormous amount for small changes in S, the simple result is that one should add a little to every level in the hierarchy when spending money on upgrading.

The problem is that the probability distributions *do* change; each P_i is a function of the memory hierarchy configuration and of the workload placed upon it, which is why the variable shows up in the results; you cannot get rid of it. The way Welch's theorem stands, you can know when you have reached an optimal configuration, but solving for one gets a little tricky.

3.2 Present Work: The System Model

The problems with previous analytical cache studies are that in order to make the analysis tractable, they often make a number of assumptions which make the analyses less realistic. These assumptions include

- the availability of a continuum of device technologies,
- that the fault probabilities (miss ratios) of the individual hierarchy levels obey a power function,
- that the cost per bit of a technology is derivable from its speed, and
- that every technology obeys the same cost-speed relation.

This analysis depends only upon the cost per bit and access times of the technologies that make up the hierarchies, and a characterization of program traces. The assumptions are that cold start and compulsory misses can be ignored for the moment (compulsory misses in a network file server on a scale of months are constant, and so disappear when finding the minimum), and so can write behavior. This last assumption is a rather large one, and is the subject of our ongoing research.

3.2.1 Stack Distance Curves

In this analysis, we make use of two related characterizations of program traces; the stack distance curves (the cumulative reference curve and its derivative). They give an insight into the locality of a program trace by measuring the LRU stack distance between successive references to the same item; i.e. if an LRU stack were being maintained (as it would in the case of a cache), how far down the item would be on its next reference. This gives a good indication of how well any given trace would perform with a cache of a specified size; if you know that 80% of all requests are within 64K of their previous reference, then a 64K cache would have a hit rate of 80% on that trace. This example makes use of the first curve, the cumulative reference curve; it plots the cumulative number of references against the stack depth (it becomes a cumulative probability curve if it is normalized by the number of references).

The second graph is the derivative of the first; it effectively plots the change hit rate as a function of the cache size. At small values of x (cache size), small changes make large differences, and further out, larger changes in cache sizes are necessary to account for the same difference in hit rate. The result is the number of references at a given stack depth (alternately the probability of each stack depth, if normalized) plotted against the stack depth in bytes, and the area under the curve represents the number of references that are hits to a given cache size.

We expect the graphs to look something like the following:

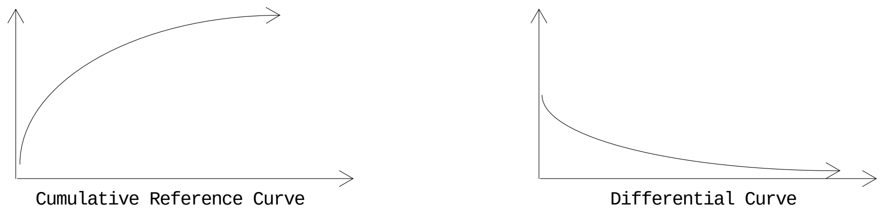


FIGURE 1. **The two stack distance curves**: the cumulative probability curve and its differential, the byte distance probability curve. Each plots stack distance against number of references, or, if normalized by the total number of references, the probability of each reference. If the curves are normalized, the cumulative probability tops out at a value of one, meaning that the area under the differential curve is defined to be 1.

In this paper, we will use normalized graphs so that the y-coordinate can be interpreted as a probability per reference, and the area under the differential can be defined to be 1.

3.2.2 The Analytical System Model

The stack distance curves are useful in the following way: the differential curve represents the number of references at any given stack depth, so a cache hierarchy can be modeled in the following manner. The L1 cache has a size of s_{L1} and a time to reference of t_{L1}. It is hit on every reference (whether the reference is a *hit* or not) and so each reference requires at least time t_{L1}. The *total* number of accesses is the area under the curve, so the total time spent in the L1 cache is given by

$$t_{L1}\int_0^\infty p(x)dx$$

where $p(x)$ is the differential.

The number of *hits* to the L1 cache is equal to the area under the curve from 0 to s_{L1}, so the number of misses is the rest of the area. This, then, is the number of requests that will be seen by the L2 cache. If the size and access time of the L2 cache are defined as s_{L2} and t_{L2}, the total time spent by the L2 cache will be

$$t_{L2}\int_{s_{L1}}^\infty p(x)dx$$

Since we have normalized the graphs to represent probability, the area from s_{L1} to infinity is the L1 miss *ratio* rather than the miss *count*, and the equations become a measure of time per reference, instead of the total running time of the entire trace. The average time per reference of the whole hierarchy is the sum of the average times spent in each of the hierarchy levels; the general time formula is given by

$$T_{avg} = t_{L1}\int_{0}^{\infty} p(x)dx + t_{L2}\int_{s_{L1}}^{\infty} p(x)dx + t_{L3}\int_{s_{L2}}^{\infty} p(x)dx + \ldots + t_{Ln}\int_{s_{Ln-1}}^{\infty} p(x)dx \qquad \text{(EQ 1)}$$

which is very like Chow's formula (also used by Welch)

$$T_{avg} = \sum_{i} H_i t_i$$

where the time to access is the sum of the access times of each level scaled by the hit rates which are (equal to the miss rates of the next highest level) [5][6][19].

3.2.3 Defining the Tradeoffs

In this paper, we are interested in hierarchies consisting of RAM, Disk, and Tertiary Storage, and will concern ourselves with the following instance of Equation 1:

$$T_{avg} = t_{RAM}\int_{0}^{\infty} p(x)dx + t_{Disk}\int_{s_{RAM}}^{\infty} p(x)dx + t_{Jukebox}\int_{s_{Disk}}^{\infty} p(x)dx$$

Since the probability integral from 0 to infinity is defined to be 1, we get

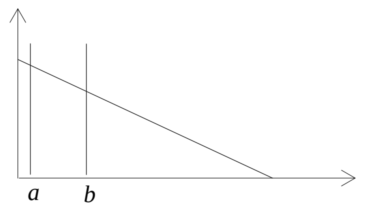
$$T_{avg} = t_{RAM} + t_{Disk}\int_{s_{RAM}}^{\infty} p(x)dx + t_{Jukebox}\int_{s_{Disk}}^{\infty} p(x)dx \qquad \text{(EQ 2)}$$

Note the absence of any reference to the size of the tertiary storage. This is as it should be, since if the bottom level of the hierarchy did not contain everything referenced, it would be considered a cache to the lower level. The time to reference the tertiary storage does show up in the formula, scaled by the miss rate of the Disk level, which also makes intuitive sense.

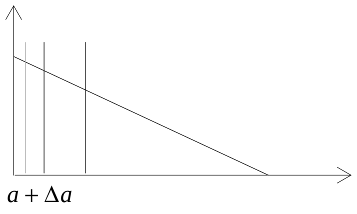
We wish to know the behavior of the optimal configuration given a fixed cost constraint. We are only concerned with the cost of the cache system, so if the total system budget is given by B and the costs per bit of RAM and disk are given by c_{RAM} and c_{Disk}, we have

$$B = c_{RAM}s_{RAM} + c_{Disk}s_{Disk} \qquad \text{(EQ 3)}$$

We can use these equations to find where the tradeoff points in the configuration lie. Assume for the moment that as the system budget increases, the optimal configuration will not rearrange the previous configuration; that as more money is allotted to the system it will not become advantageous to throw away some amount of disk for RAM or vice versa (this will be shown to be the case later). At any point, we can add a cost-equivalent amount of RAM or disk, and all we need to do is determine which addition will make the entire system faster. Let us say that we have an arbitrary differential curve for $p(x)$ that looks like the following:



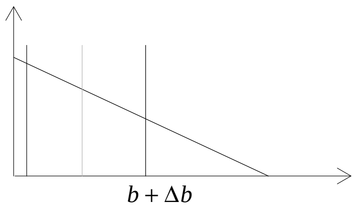
where $a = s_{RAM}$ and $b = s_{Disk}$. Then the expected average time per access is given by Equation 2. At this point, we can add an amount Δa of RAM to the system,



and obtain a savings in time per access by reducing the number of accesses that miss in the first level of the hierarchy. This savings is equal to the time to access the Disk level multiplied by the number of accesses saved; the area under the curve gained in RAM by adding Δa.

$$T_{avgsaved} = t_{Disk} \int_{a}^{a+\Delta a} p(x)dx \qquad \text{(EQ 4)}$$

Alternately, we can add a cost-equivalent amount of disk to the system, Δb, where $\Delta a c_{RAM} = \Delta b c_{Disk}$



and obtain a savings in the average access time of

$$T_{avgsaved} = t_{Jukebox} \int_{b}^{b+\Delta b} p(x)dx \qquad \text{(EQ 5)}$$

The amount of disk or RAM one can add to the system, the shape of the probability curves, and the existing hierarchy setup are all completely independent of each other. Therefore the optimization question is whether the horizontal change of adding a chunk of disk buys more accesses (scaled by the time to access tertiary storage) than adding a sliver of RAM (scaled by the time to access disk). The area under the disk portion of the curve can afford to be much smaller than the area under the RAM portion of the curve, as it is scaled by an access time that is typically 100 times larger than the access time scaling the area under the RAM portion of the curve, and the tradeoff is equal when the ratio of the areas is equal to the ratio of the access times. Since the sizes of RAM lie in the steep portion of the curve, while the sizes of disk are out where the curve probably flattens out, one can easily see this being the case, and so the answer to the optimization question is certainly not obvious.

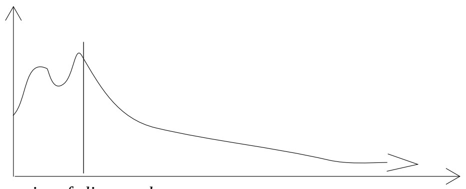
3.3 Present Work: The Analysis

In this section the following topics are covered; Section 3.3.1, *Finding Optimal Configurations*, analyzes the equations from the previous section, and Section 3.3.2, *The Crossover Point*, determines where the first dollar should go in and at what point it makes sense to add the first bit of RAM or disk. Finally, Section 3.3.3, *Modeling the Differential as an Exponential*, and Section 3.3.4, *Modeling the Differential as a Polynomial*, assume the shape of the probability curve follows some well-defined functions and the sections predict where the interesting things happen.

3.3.1 Finding Optimal Configurations

We can now find the optimal configurations for all values of the system budget. In the following sections, let s_{RAM}, s_{Disk}, and $s_{Jukebox}$ represent the sizes of the levels in the hierarchy (units = MB); let t_{RAM}, t_{Disk}, and $t_{Jukebox}$ represent the access time for those technologies (units = sec), and let c_{RAM} and c_{Disk} represent the costs of those technologies (units = dollars/MB). B represents the total system budget, in dollars.

Let us assume that the differential curve has no local minima (which is intuitively realistic for average workloads; remember that we are intent on optimizing a server, so even if the original full trace has a curve that looks like the following:


size of client cache

we only see the portion of the traces which reaches the server; that portion which spills out of the client-side cache). The spillover portion is represented by the tail of the curve to the right of the vertical line in the graph. This has some interesting implications concerning the size of client-side caches; for example if the client's cache is too small and the portion of the I/O requests that spills out to the server has a locality curve with local maxima and minima, then the server could believe there to be global minimums in access time where in fact there are only local minimums.

We have an optimal configuration when T_{avg} is minimized. Using Equation 2 and Equation 3 and remembering that

$$\frac{d}{da}\left(\int_a^b f(x)dx\right) = -f(a)$$

we allow s_{RAM} to vary with each given system budget and find minimums when $\frac{\partial T_{avg}}{\partial s_{RAM}} = 0$;

$$\frac{\partial T_{avg}}{\partial s_{RAM}} = -t_{Disk}p(s_{RAM}) + t_{Jukebox} \cdot \frac{c_{RAM}}{c_{Disk}} \cdot p\left(\frac{B - s_{RAM}c_{RAM}}{c_{Disk}}\right)$$

giving solutions where

$$\frac{p(s_{RAM})}{p(s_{Disk})} = \frac{t_{Jukebox}c_{RAM}}{t_{Disk}c_{Disk}} \qquad \textbf{(EQ 6)}$$

The ratio on the right hand side turns out to be a recurring theme, so we assign it a label of Ψ, the performance ratio.

$$\Psi = \frac{t_{Jukebox}c_{RAM}}{t_{Disk}c_{Disk}} \qquad \textbf{(EQ 7)}$$

The ratio of the values of the differential at s_{RAM} and s_{Disk} is a measure of the difference in hit rates; each value represents the number of references that would hit in the cache if the cache were increased by a small (essentially zero) amount. The performance ratio gets larger as disk performance increases relative to tertiary storage, and as the price of disk decreases relative to the price of RAM. We find the optimal configuration point where the number of reference hits gained by increasing the amount of RAM by some small amount, compared to the number of reference hits gained by increasing the amount of disk by some small amount, is equal to the performance ratio. Ψ is a measure of the effectiveness of the disk cache level; it represents the time saved by adding more disk to the system (by taking the time away from the tertiary storage system) and it represents the cost-effectiveness of disk as compared to RAM.

Equation 6 is therefore a method of finding the optimal solution; given a probability curve that represents the expected workload, for any value of s_{RAM} we can find the corresponding optimal amount of disk that should be in the system, and for any value of s_{Disk}, we can find the corresponding optimal amount of RAM that should be in the system. We shall see that it is not quite this simple because at small values of disk the optimal amount of RAM is often a negative value, but this obviously is easily overcome.

3.3.2 The Crossover Point

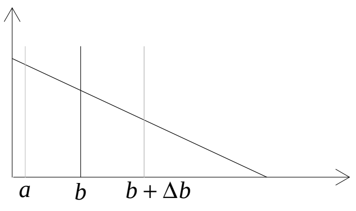
One of the simplest questions to ask at this point is *where should the first dollar go?*

It is not quite as simple as comparing Equation 4 to Equation 5, as we also need to account for the addition of a new level in the hierarchy (at dollar $0, the only thing in the system is the tertiary storage). The amount of time saved per reference by adding *anything* to the system is equal to the time to reference the jukebox multiplied by the expected number of references that will be hits in the new level added (those that will not have to go to the tertiary storage), with the additional expense of having to access the new technology on every reference. We shall see that the first dollar will always be spent upon disk. This agrees with results from [10], [18], and [19]; when the amount of money to spend on the cache is small, capacity counts for more than speed.

Assuming for the moment that RAM will not be part of the optimal configuration until system budget numbers get large, another question that we can ask is *where is the crossover point?* We will answer the question, and in doing so, find at an alternate way of arriving at (and explaining) Equation 6.

At what cost point does it makes sense to add the first dollar of RAM; at what system budget is the performance gained by adding a dollar of RAM equal to the performance gained by adding a dollar of disk? The crossover point is the cost point at which it makes just as much sense to add the next bit of RAM as it does to add the next bit of disk.

Consider the following scenario:



We have a system with only disk in it; the disk has size b and for the same amount of money we can add an amount a of RAM or an amount Δb of disk. We wish to find the point at which the area under the curve from 0 to a is equal to the area under the curve from b to Δb, where the area from 0 to a is scaled by the time to reference the disk level, and the area from b to Δb is scaled by the time to reference the tertiary storage level.

We have the following relations:

$$T_{savedbyaddingRAM} = t_{Disk} \int_{0}^{s_{RAM}} p(x)dx - t_{RAM} \quad \text{(EQ 8)}$$

$$T_{savedbyaddingDisk} = t_{Jukebox} \int_{s_{Disk}}^{s_{Disk}+\Delta Disk} p(x)dx \qquad \text{(EQ 9)}$$

$$c_{RAM}s_{RAM} = c_{Disk}\Delta Disk \qquad \text{(EQ 10)}$$

We find the crossover point when Equation 8 and Equation 9 are equal, and if we let the sizes of each incremental amount to add to the system approach zero, we find a solution when

$$\lim_{s_{RAM} \to 0}\left(t_{Disk}\int_0^{s_{RAM}} p(x)dx - t_{RAM}\right) = \lim_{\Delta disk \to 0}\left(t_{Jukebox}\int_{s_{Disk}}^{s_{Disk}+\Delta Disk} p(x)dx\right)$$

This yields

$$\lim_{s_{RAM} \to 0}(t_{Disk}p(s_{RAM})\Delta Disk - t_{RAM}) = \lim_{\Delta Disk \to 0}(t_{Jukebox}p(\Delta Disk)\Delta Disk) \qquad \text{(EQ 11)}$$

Since $t_{RAM} \ll t_{Disk}$, we lose the t_{RAM} term and approximate this as

$$\lim_{\Delta Disk,\, s_{RAM} \to 0}\left(\frac{p(s_{RAM})}{p(s_{Disk})}\right) = \lim_{\Delta Disk,\, s_{RAM} \to 0}\left(\frac{s_{Disk}}{s_{RAM}} \cdot \frac{t_{Jukebox}}{t_{Disk}}\right) \qquad \text{(EQ 12)}$$

Using Equation 10, this yields something that looks very similar to Equation 6:

$$\frac{p(0)}{p(s_{Disk})} = \frac{t_{Jukebox}c_{RAM}}{t_{Disk}c_{Disk}} \qquad \text{(EQ 13)}$$

The right hand side is the performance ratio Ψ, and the relation says that the crossover point occurs farther out when the effectiveness of disk is high, and closer in when the effectiveness of disk is low.

In general, every optimal point is a kind of crossover point; it is a point at which (as the amounts approach zero) it makes as much sense to add an amount of RAM as it does to add an amount of disk to the system. If it were otherwise, the solution would not be optimal; if there were an advantage to adding zero RAM over adding zero disk, then the optimal solution at that cost point would instead have more RAM and less disk in it.

This being the case, we can think of the entire crossover point discussion as a method for finding *any* optimal point after the crossover point. The starting value for RAM would be s_{RAM} instead of 0, and the $-t_{RAM}$ term would not appear in Equation 7 (which is fine, since the term was dispensed with in Equation 12). The result would be an alternate derivation of Equation 6.

3.3.3 Modeling the Differential as an Exponential

If we assume that $p(x)$ looks like $\alpha e^{-\alpha x}$ for some α, we have

$$s_{Disk} - s_{RAM} = \frac{\ln(\Psi)}{\alpha} \qquad \text{(EQ 14)}$$

Combining with Equation 3, we conclude that for that optimal configurations at each budget B, the sizes of RAM and disk are given by

$$s_{RAM} = \frac{B}{c_{RAM} + c_{Disk}} - \left(\frac{c_{Disk}}{c_{RAM} + c_{Disk}} \cdot \frac{\ln(\Psi)}{\alpha} \right) \qquad \text{(EQ 15)}$$

and

$$s_{Disk} = \frac{B}{c_{RAM} + c_{Disk}} + \left(\frac{c_{RAM}}{c_{RAM} + c_{Disk}} \cdot \frac{\ln(\Psi)}{\alpha} \right) \qquad \text{(EQ 16)}$$

This tells us several things.

1. The amount of disk for the optimal configuration can never be less than the amount of RAM, in particular there can be no 0-disk system besides the initial state (where there is nothing but the tertiary storage).
2. The sizes in megabytes of RAM and disk (at least at large budget values) increase at the same rate, which actually suggests that at some point any difference between the two amounts will be meaningless (remember that this section assumes the differential is of the form $\alpha e^{-\alpha x}$).
3. The disparity between the amounts of RAM and disk gets larger as Ψ increases; as the cost of RAM increases relative to disk and as the speed of disk increases relative to the speed of tertiary devices, the gap between the sizes of the levels must also grow. Alternatively, as these ratios decrease, the disparity between RAM and disk shrinks; disk starts to buy you less when its cost approaches that of RAM or its speed approaches that of tertiary devices.
4. The amounts of disk and RAM are monotonically increasing as functions of the system budget. This means that a hill-climbing approach can in fact be used to find optimal points; it also means that whatever money you invest in obtaining an optimal configuration will not be wasted should you decide to invest more in the system at a later point.

The equations are a reminder that some mathematical solutions may lie outside the range of valid states; we cannot have negative values of RAM, as is suggested by the equations at small budget values. If the equations suggest values for s_{RAM} that are outside the allowable range, we can always pick one of the boundary conditions (the boundary condition closest to the theoretical minimum). Since the equations will never suggest a value for s_{RAM} which is greater than B/c_{RAM} (the maximum amount of RAM you could buy if you spent the entire budget on RAM), we only need to worry about values less than zero and use zero RAM for these configurations. This happens when Equation 15 yields a negative value; when

$$\frac{B\alpha}{c_{Disk}} < \ln(\Psi)$$

B/c_{Disk} is the amount of disk you can buy if you spend the entire budget on it. We have valid minimum solutions when this amount is greater than or equal to the right hand side, implying that the budget needs to grow as the cost of disk improves in relation to RAM; if RAM gets more expensive, then the budget point at which it makes sense to add the first megabyte of RAM increases. Similarly, as the performance of disk increases relative to tertiary storage, its effectiveness at speeding up the system access time (by taking some of the burden off the tertiary storage) increases, and this also drives up the initial cost point where it makes sense to add RAM to the system.

There is also the α term scaling the cost point downward as it gets larger; this makes sense, as a large value of α makes the probability curve steeper near the y-axis, allowing a smaller amount of RAM to cover a larger number of references.

As α gets large (in comparison to 1) the curve is very steep and lies close to the y-axis, decaying rapidly toward zero. This would suggest a program trace with a high degree of locality. As α approaches 0, the curve looks more and more like the constant function $y = 1$; a curveless curve, suggesting a program trace with virtually no locality at all. A value of a bit less than 1 for α yields a moderately decaying function without an enormously high y-intercept; this suggests a program trace showing a moderate degree of locality.

If we assume for the moment that $\alpha = 0.01$, we find the crossover point (the point at which it makes sense to add RAM to the system) when $B = 100c_{Disk}\ln(\Psi)$. If we assume that RAM costs about 60 times as much as disk and that magnetic disk devices are 100 times faster than tertiary storage, the crossover point is where the system has about 800 megabytes of disk. From here on, we add a megabyte of RAM for every megabyte of disk.

3.3.4 Modeling the Differential as a Polynomial

If we assume instead that the differential curve $p(x)$ looks like $\frac{\alpha - 1}{(x+1)^{\alpha}}$ for some α, we find minimums in T_{avg} when

$$\left(\frac{s_{Disk}+1}{s_{RAM}+1}\right)^{\alpha} = \Psi \tag{EQ 17}$$

where Ψ is the performance ratio from Equation 7:

$$\Psi = \frac{t_{Jukebox} c_{RAM}}{t_{Disk} c_{Disk}}$$

This gives us

$$\frac{s_{Disk}+1}{s_{RAM}+1} = \pm(\Psi)^{1/\alpha} \tag{EQ 18}$$

but since a negative value for the sizes of RAM and disk makes no sense, we only need to worry about the positive version of $\Psi^{1/\alpha}$. Combining with Equation 3, the sizes of RAM and disk for optimal configurations at each budget B are then given by

$$s_{RAM} = \frac{B}{c_{RAM} + c_{Disk}\Psi^{1/\alpha}} - \frac{c_{Disk}(\Psi^{1/\alpha} - 1)}{c_{RAM} + c_{Disk}\Psi^{1/\alpha}} \tag{EQ 19}$$

and

$$s_{Disk} = \frac{B\Psi^{1/\alpha}}{c_{RAM} + c_{Disk}\Psi^{1/\alpha}} + \frac{c_{RAM}(\Psi^{1/\alpha} - 1)}{c_{RAM} + c_{Disk}\Psi^{1/\alpha}} \tag{EQ 20}$$

This tells us several things.

1. Just as in the previous section where the function was assumed to be exponential, the amount of disk for the optimal configuration can never be less than the amount of RAM, and there can be no 0-disk system besides the initial state.
2. Unlike the results for the exponential, the sizes in megabytes of RAM and disk *do not* increase at the same rate, but instead maintain a constant ratio at all times. If we let $\alpha = 1$, then the ratio is around 6000; there should always be about 6 gigabytes of disk for every megabyte of RAM. A value of 1 yields a curve that is symmetric about the line $y = x$; it is a compromise between having most of the area under the curve between 0 and 1 (large values of α) and having the area spread nearly evenly across all values of x (when α approaches 0).
3. Similar to the results for the exponential, the ratio of the amount of disk to the amount of RAM gets larger as Ψ increases; as the cost of RAM increases relative to disk and as the speed of disk increases relative to the speed of tertiary devices, disk looks like a winning technology and it makes sense to have more of it in the system. As these performance and cost ratios decrease, the ratio of amounts of disk to RAM also decreases; disk starts to buy you less when its cost approaches that of RAM or its speed approaches that of tertiary devices.
4. The amounts of disk and RAM are monotonically increasing as functions of the system budget. This result is identical to the conclusions drawn from the exponential curve.

There is also the α term scaling the ratio in Equation 15 downward as α gets larger; this makes sense, as a large value for α makes the probability curve steeper near the y-axis and decays to zero very rapidly, allowing a smaller amount of RAM to cover a larger number of references. On the other hand, a small value for α $(0 < \alpha < 1)$ makes the function decay very slowly, with very little area under the curve near the y-axis; this makes it difficult for a small amount of memory to cover a substantial amount of references, and so large values of cheap storage become necessary.

Ψ will always be greater than 1, so we do not need to worry about Equation 20 giving us invalid solutions for s_{Disk}. However, Equation 15 can have solutions that lie outside the valid range. This occurs when

$$\frac{B}{c_{Disk}} < \Psi^{1/\alpha} - 1$$

B/c_{Disk} is the amount of disk one could buy if the entire budget is spent on disk. The inequality suggests the same conclusion that was drawn from the exponential model; that the crossover point scales with the effectiveness of the disk system. The crossover point is where the minimum solution for s_{RAM}goes from being negative to positive; when the amount of disk you can buy with the budget is greater than or equal to the α root of Ψ. If we assume for the moment that $\alpha = 1$, that RAM costs about 60 times as much as disk and that magnetic disk devices are 100 times faster than tertiary storage, the crossover point is when $B/c_{Disk} = \Psi - 1$; when the amount of disk in the system reaches roughly 6 gigabytes. Unlike the exponential example, the polynomial does not have identical slopes for the addition of RAM and disk to the system; here the amount added will always be in the same ratio but will favor disk.

3.3.5 Conclusions

The following table summarizes the results of the curve fitting expedition:

TABLE 1. Comparison of modeling the differential as an exponential and a polynomial curve.

Approximation Function for p(x)	**Crossover Point for $\alpha = 1$**	**Optimal Size of RAM (only valid after crossover)**
$\alpha e^{-\alpha x}$	$B = c_{RAM}\ln(\Psi)$, $B \cong 256$	$s_{RAM} = \frac{B}{c_{RAM} + c_{Disk}} - \left(\frac{c_{Disk}}{c_{RAM} + c_{Disk}} \cdot \frac{\ln(\Psi)}{\alpha}\right)$
$\frac{\alpha - 1}{(x+1)^{\alpha}}$	$B = c_{Disk}(\Psi - 1)$, $B \cong 3000$	$s_{RAM} = \frac{B}{c_{RAM} + c_{Disk}\Psi^{1/\alpha}} - \frac{c_{Disk}(\Psi^{1/\alpha} - 1)}{c_{RAM} + c_{Disk}\Psi^{1/\alpha}}$

A few things seem to be independent of our choice for a well-defined approximation of the locality curve:

- the amount of disk in the optimal solution will never be less than the amount of RAM,
- the amount of disk will never be zero,
- the disparity between disk and RAM increases as Ψ increases
  (whether it be in a ratio between the two or in a constant difference), and, of course,
- the ratio of the values of the locality function at the size of RAM and disk are equal to the performance ratio.

A few things seem at least at first glance to be very dependent upon our choice of approximations, and therefore weaken the model somewhat:

- the rate at which RAM and Disk levels grow
  (parallel lines in exponential approximation, diverging lines in polynomial approximation), and

- the effect the performance ratio has upon the crossover point

  (for the exponential curve, the crossover point increases with the log of Ψ; for the polynomial curve, the crossover point is proportional to Ψ).

It seems to be a fairly robust model, although seemingly small changes in the modeling of the locality curve produce large changes at the other end.

4.0 Simulations

In order to check the validity of the model, we present a trace-driven comparison of the effectiveness of a number of different I/O hierarchies, given a set of cost values. We decided that the minimum increment to spend on an upgrade would be $256, which should buy roughly 1/2 GB of disk space or 8 MB of RAM.

4.1 Workload Description

The data that was used for the workload in the trace-driven hierarchy simulations was collected by CITI via their logging AFS server [4]. The only data that the server sees are those accesses not serviced from the client's local cache.

We use approximately one month's worth of trace data captured from a server named "marge" from April 14, 1992 through May 8, 1992. The full traces represent over 20 million records of several different types of AFS server requests. The commands that we are interested in are *fetchdata, storedata, removefile, createfile, removedir*, and *makedir*. The rest of the commands do not actually read and write data; they are used to synchronize the local cache with the server, in the case of *fetchstatus*, etc.

The environment in which the traces were captured was the University of Michigan Institutional File System (IFS). The IFS consists of AFS servers and clients running on various platforms scattered across the U-M campus. The types of applications span a wide range, including general-purpose computing such as e-mail and text-editing as well as software archives and traditional engineering type jobs. The file sizes range from very small (less than one KByte) to fairly large (several megabytes). We believe that the traces accurately portray typical file server activity.

4.2 Simulator Description

Our simulator is a combination of a number of similar modules that implement objects such as memory, disk drives, and automated tertiary storage devices such as MO jukeboxes or cartridge tape autoloaders, and a skeleton frame that connects object modules together. The object modules follow the same interface so they are completely interchangeable and new modules can be implemented and inserted into the hierarchy with minimal effort.

TABLE 2. Summary of specs used for simulator

Technology	**Capacity**	**Block Size**	**Cost per MB**	**Latency**	**Bandwidth**
DRAM file buffers	8MB-64MB, in increments of 8MB	8 KBytes	$32.00 ($256 buys 8MB)	negligible	160 MB/sec
Magnetic Disk	512MB-6GB, in increments of 512MB	16 KBytes	$0.50 ($256 buys 0.5GB)	10 ms, plus a rotational delay of 5 ms	10 MB/sec
Tertiary Storage	100 GB	64 KBytes	$0.001	1 sec	1 MB/sec

The object modules are responsible for implementing the various caches and keeping track of the usage statistics. At each level, if the item requested is not present it is requested from the next level down. The cost per megabyte, transfer time, and latency figures are the same as those described in Section 2.0.

The running times reported by the simulator are an upper bound, in that the time taken at a given level depends upon the number of hits as well as the number of misses; if there is a miss at a level, time is taken to fill and prefetch; it is not the case that misses are handled entirely as a hit at the next lower level.

In order to simplify implementation, writes were treated similarly to reads in a read-modify-write manner. Future studies will deal more with this issue. Przybylski ignores writes altogether in his cache analysis [14], but it is not the case that writes can be simply dispensed with in an I/O hierarchy, as the cost must be accounted for somewhere.

4.2.1 Memory Object

The memory module keeps lists of blocks of user data in a roughly LRU-type set associative cache configuration. For all accesses, the time spent is equal to the number of blocks requested divided by the transfer rate.

4.2.2 Disk Object

The disk module assumes a fixed number of cylinder groups, with an essentially infinite number of platters - no matter how large the disk is defined to be, it is still considered one physical disk drive; the size of each cylinder group (the number of blocks allowed in each group's linked list) is determined by the disk's capacity. The group in which a given file is placed is determined by a hashing function based upon the user number and file number, so that a user's files tend to be in or near the same cylinder group.

The latency is the approximate time to move the "head" of the disk to the requested cylinder group from the group of the previous request. It is calculated to be the distance in cylinder groups times the given latency, divided by the number of groups. The total transfer time is the latency plus the number of blocks requested divided by the given transfer rate.

4.2.3 Tertiary Storage Object

The tertiary storage module is an approximation of a cartridge tape autoloader or magneto-optical jukebox. Each storage unit is defined to hold 5 GB of data, and the module can read several units at once. The total transfer time is calculated similarly to the disk object in that each reader remembers where the last request was and calculates latency by multiplying distance by the given latency divided by the number of regions. The total transfer time is this latency plus the number of blocks requested divided by the given transfer rate.

The tertiary storage module is always the lowest level of the storage hierarchy, thus a data access never misses there. The only reason this module is instantiated in the hierarchy is to compute miss times when the data is not present in the next higher level of the hierarchy.

```
Config: 48 MB bufcache -> 100 GB jukebox
- buffercache: 133.48 secs, 2090254 hits, 943517 misses, 49152 KB => $1536.00
- jukebox: 57371.71 secs, 943517 hits, 0 misses, 0 MB
Total Time: 57505 secs spent Total Cost: $1536

Config: 40 MB bufcache -> 512 MB disk drive -> 100 GB jukebox
- buffercache: 134.17 secs, 1380374 hits, 1653397 misses, 40960 KB => $1280.00
- diskdrive: 12372.24 secs, 958016 hits, 695381 misses, 524288 KB => $256.00
- jukebox: 47970.83 secs, 695381 hits, 0 misses, 0 MB
Total Time: 60477 secs spent Total Cost: $1536

Config: 32 MB bufcache -> 1024 MB disk drive -> 100 GB jukebox
- buffercache: 134.83 secs, 1356336 hits, 1677435 misses, 32768 KB => $1024.00
- diskdrive: 12560.64 secs, 1139927 hits, 537508 misses, 1048576 KB => $512.00
- jukebox: 42409.36 secs, 537508 hits, 0 misses, 0 MB
Total Time: 55104 secs spent Total Cost: $1536

Config: 24 MB bufcache -> 1536 MB disk drive -> 100 GB jukebox
- buffercache: 135.58 secs, 1330944 hits, 1702827 misses, 24576 KB => $768.00
- diskdrive: 12761.52 secs, 1265990 hits, 436837 misses, 1569520 KB => $768.00
- jukebox: 37913.68 secs, 436837 hits, 0 misses, 0 MB
Total Time: 50810 secs spent Total Cost: $1536

Config: 16 MB bufcache -> 2048 MB disk drive -> 100 GB jukebox
- buffercache: 136.47 secs, 1300941 hits, 1732830 misses, 16384 KB => $512.00
- diskdrive: 12999.23 secs, 1355938 hits, 376892 misses, 1973072 KB => $1024.00
- jukebox: 34317.64 secs, 376892 hits, 0 misses, 0 MB
Total Time: 47453 secs spent Total Cost: $1536

Config: 8 MB bufcache -> 2560 MB disk drive -> 100 GB jukebox
- buffercache: 139.73 secs, 1206190 hits, 1827581 misses, 8192 KB => $256.00
- diskdrive: 13753.91 secs, 1483685 hits, 343896 misses, 2311552 KB => $1280.00
- jukebox: 31880.61 secs, 343896 hits, 0 misses, 0 MB
Total Time: 45774 secs spent Total Cost: $1536

Config: 3072 MB disk drive -> 100 GB tape drive
- diskdrive: 15315.49 secs, 1712675 hits, 320448 misses, 2609408 KB => $1536.00
- jukebox: 30345.12 secs, 320448 hits, 0 misses, 0 MB
Total Time: 45660 secs spent Total Cost: $1536
```

FIGURE 2. An example of the simulator output, for the cost point of $1536, which can be partitioned 7 ways between Disk and RAM.

4.3 Results of Simulations

Figure 2 is an example of what the simulator produces as output.

The Total Cost of a configuration is the sum of the costs of the buffer cache and the disk drives. It does not include the cost of the tertiary device, as that appears in every single configuration.

The capacity *used* at each level is in the level's *KB* field; this is the capacity currently being used by the simulation. It is less than or equal to the total defined capacity due to the fact that we divide the capacity into a number of equal-sized blocks and if the blocksize does not divide evenly into the capacity we waste the space. Typical block sizes were high (around 8 KB or 16 KB) in order to provide realistic numbers (AFS chunking is in 64K blocks), as well as make the simulations faster.

The number of hits plus the number of misses is not necessarily the same from configuration to configuration; this is to be expected when the shape of the hierarchy changes. When the simulator misses at a given level, it brings in the entire page from the lower level that the data resides upon; if, for example, the page size of the buffer cache is 4KB and the page size of the disk is 16KB, then every miss in the buffer cache brings in 16KB, or 4 buffer cache pages. These prefetches are tagged as hits in the simulator, whether the pages are later referenced or not, and so the totals may not be the same across different configurations. This will be fixed in future versions of the simulator.

Note that for every cost point, a number of different configurations are run, and the optimal one is taken to represent that cost point; in the figures, the configuration that shows up is the optimal of a large number of different configurations. At

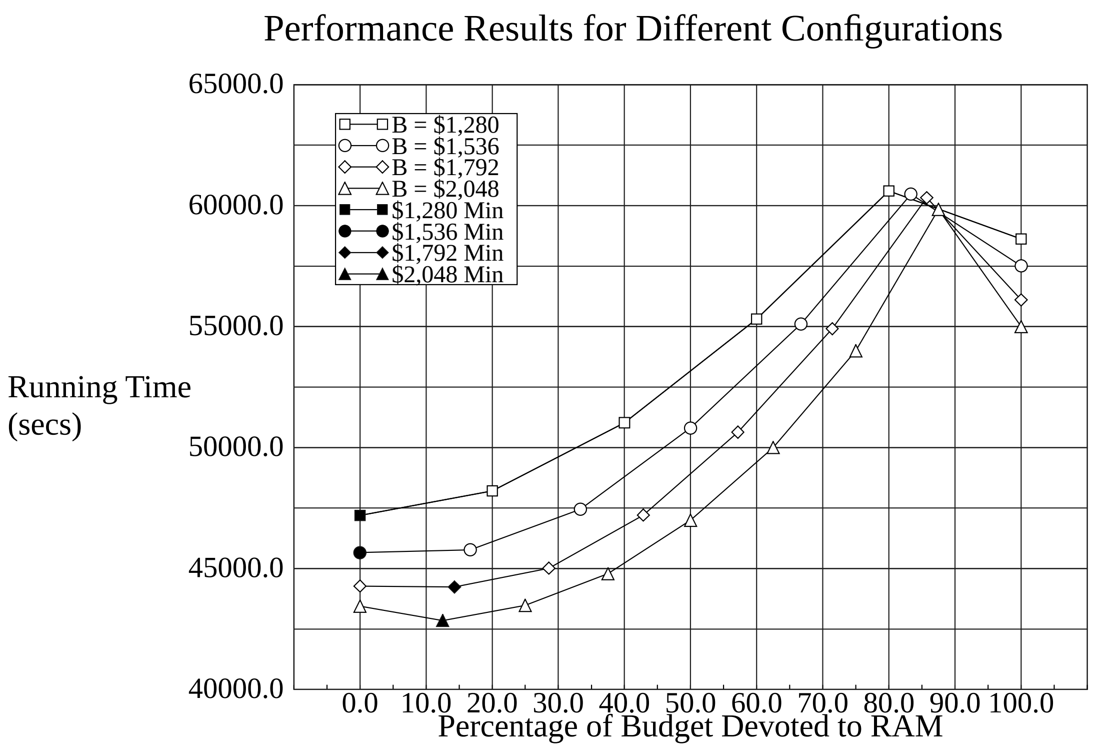


FIGURE 3. The configuration curves for four different budgets ($1280, $1536, $1792, and $2048). Each point represents a run of the simulator; its x-coordinate is the proportion of the total budget devoted to RAM and its y-coordinate is the total running time of the configuration. The fastest running times for each budget are highlighted by darkened polygons.

small cost points, every possible combination was simulated, but as the cost points grew larger the number of 10-hour simulations grew too large to continue this practice, and so only four or five different configurations are compared.

Figure 3 shows the performance variations over a number of configuration costs. Each of the curves corresponds to one running of a simulation, as in Figure 2 above.

It is interesting to note that the worst case happens when there is all RAM except for one 1/2 GB disk cache, and the performance does not seem to get any better as more space is added to the L1 RAM cache. The optimum point moves from the boundary condition (zero MB RAM) to 8MB at cost point $1536. This is the crossover point, and at larger budget values the minimum should keep moving to the right.

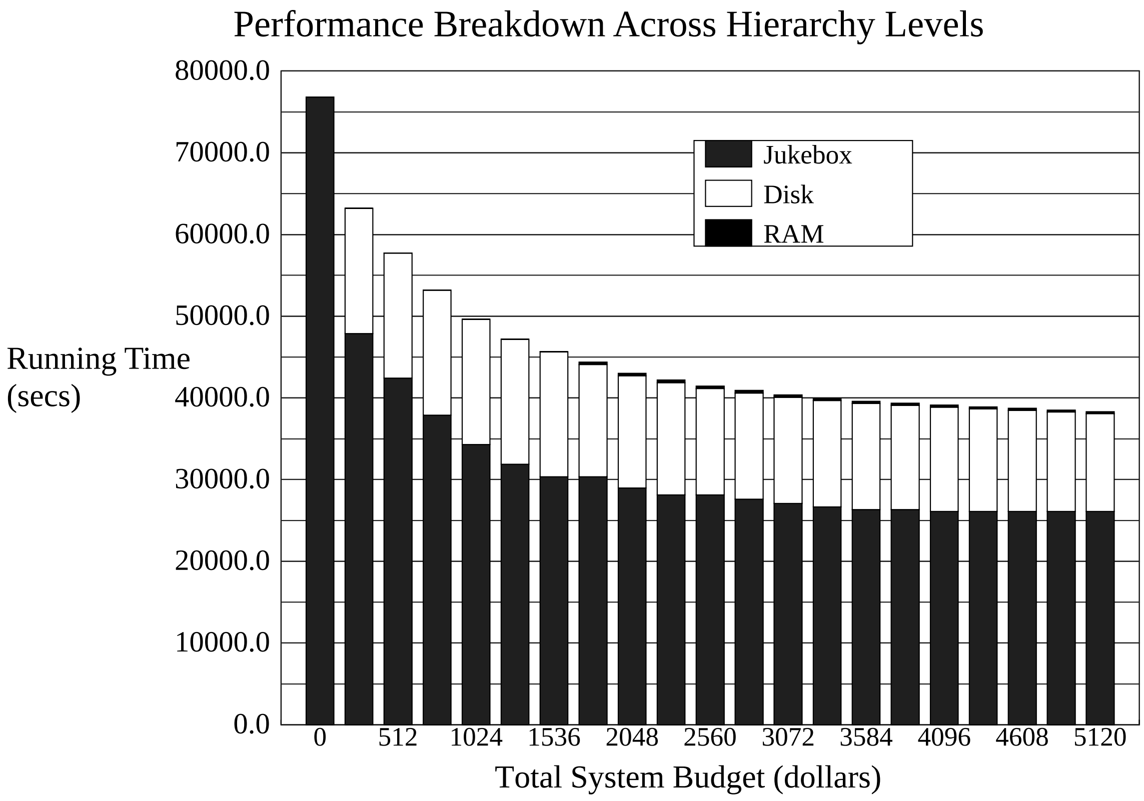


FIGURE 4. The total running times of optimal configurations for different system costs. The running times are broken down by technology; for instance at cost point $256, the total system running time is decreased roughly %15 by adding 512MB of disk, which now accounts for roughly one fourth of the total time. The running time of any given level decreases whenever an amount is added to the level immediately above. The slivers that represent RAM time are difficult to see, but can be detected because of this behavior; for instance, between cost points $1536 and $1792 the time for Jukebox does not decrease, but the time for Disk does. Here is where the first 8MB of RAM is added.

Figure 4 shows the effect of adding more and more cost to the hierarchy, and shows within each cost point the performance break-downs of RAM, Disk, and tertiary storage.

This shows the expected asymptotic behavior of adding memory to the hierarchy. It is interesting to point out that the running time drops by almost a factor of two when 3 gigabytes of disk are added to the system, at a cost of about $1,500. From then on, three times that dollar amount worth of RAM and disk is added to the system for a performance increase of only another 10-15%.

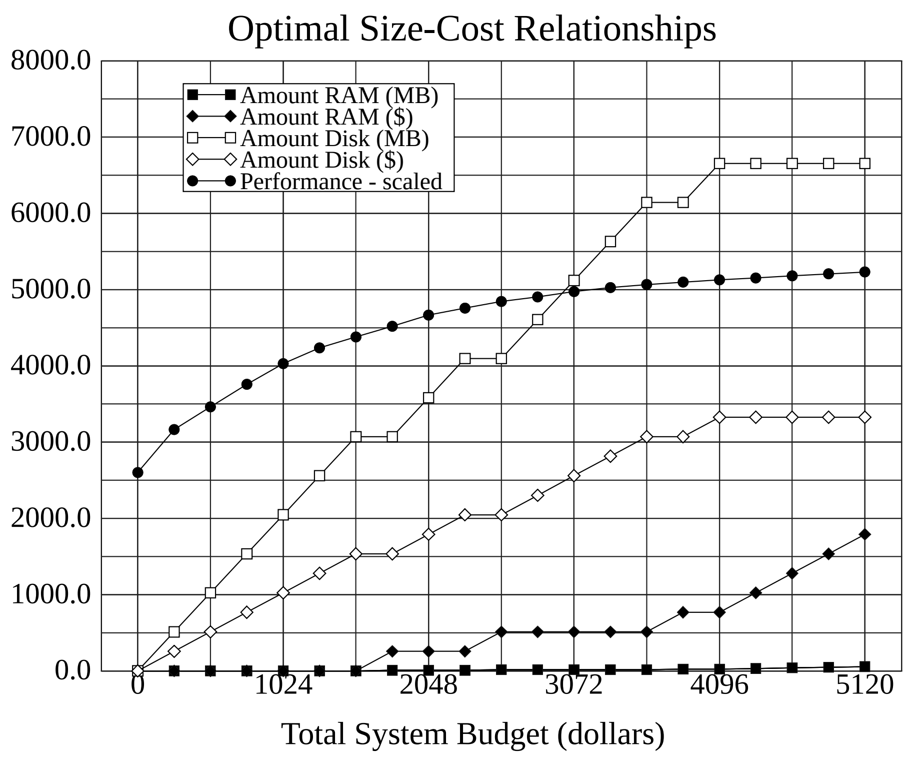


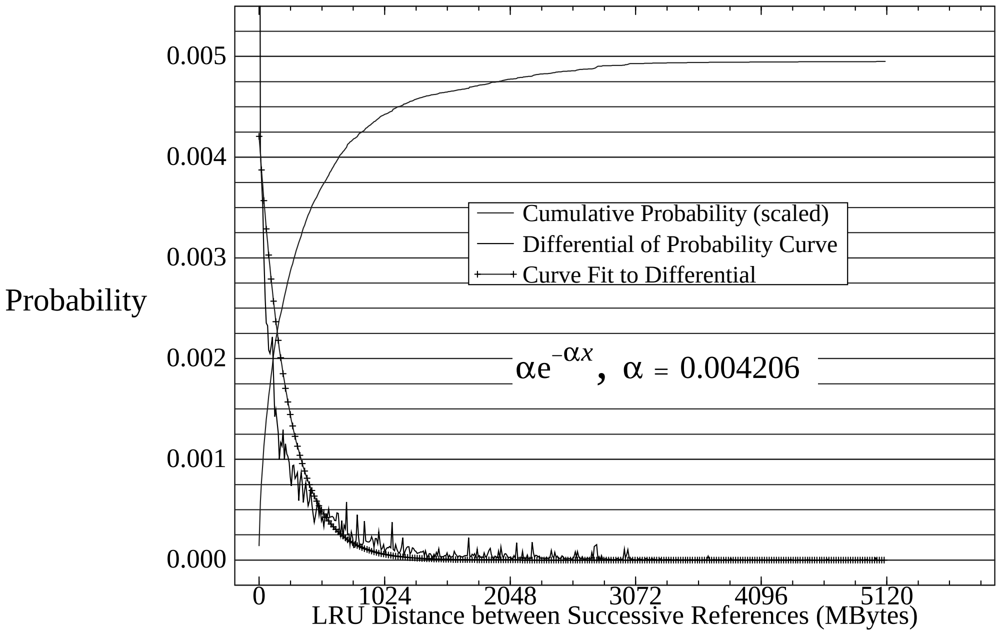
FIGURE 5. **The sizes and costs of RAM and Disk for the optimal configurations**. Note that the MBytes RAM curve is a constant multiple of the Cost RAM curve; the Disk curves are similarly related. Thus the MBytes RAM curve which is lost down near the x-axis is shaped exactly like the Cost RAM curve higher up. This is where the linear nature of the Size/Cost relation is evident; while it still makes sense to add disk to the system (while the size of the disk is still less than the size of the data set in the traces) the optimal amount of RAM and Disk both increase linearly as a function of System Cost. As soon as there is enough disk to cover the data in the traces, no more disk needs be added to the system and all future increments in system budget go toward RAM. It is very important to note that this curve contains the effect of all cold start misses as well as all compulsory misses; nothing has been removed; the performance is very pessimistic.

Figure 5 plots the RAM costs of the optimal configurations for each cost point, as well as the scaled performance of the system (proportional to 1 over the running time). The step-wise nature of the cost and size curves is due to our restriction that we only allow additions of 8MB RAM or 512MB disk at a time.

At the start, there is a question of whether to add disk or RAM. Since the client AFS caches have filtered out much of the available locality before it gets to the server, a small amount (8MB) of RAM will not achieve a high enough hit rate to offset the fact that we would be going to tape often. The half-gigabyte disk drive, however, will show a much better hit rate, and this serves to offset the disk's much lower access time. The crossover point is seen to be somewhere between $512 an $1536.

The graph demonstrates an effect not predicted by the analysis; at a point, the disk curves top out and no longer increase, while the RAM curves increase rapidly since every dollar after that point is spent on RAM. This is due to the fact that the traces are finite and so cannot be approximated by a real function that asymptotes to zero; here, the traces actually *reach* zero and as a result once the disk size increases to the value where the differential curve hits the x-axis, there are no more references that can be used to reduce the running time of the tertiary device (by turning them into hits at the disk level).

We believe that this is what is happening around the $4,000 cost point; from here on it seems that there is no more benefit to adding any more disk. This would suggest that the effective working set size of the AFS traces is somewhere around six gigabytes; for larger working sets, the point would be appropriately pushed off to the right.



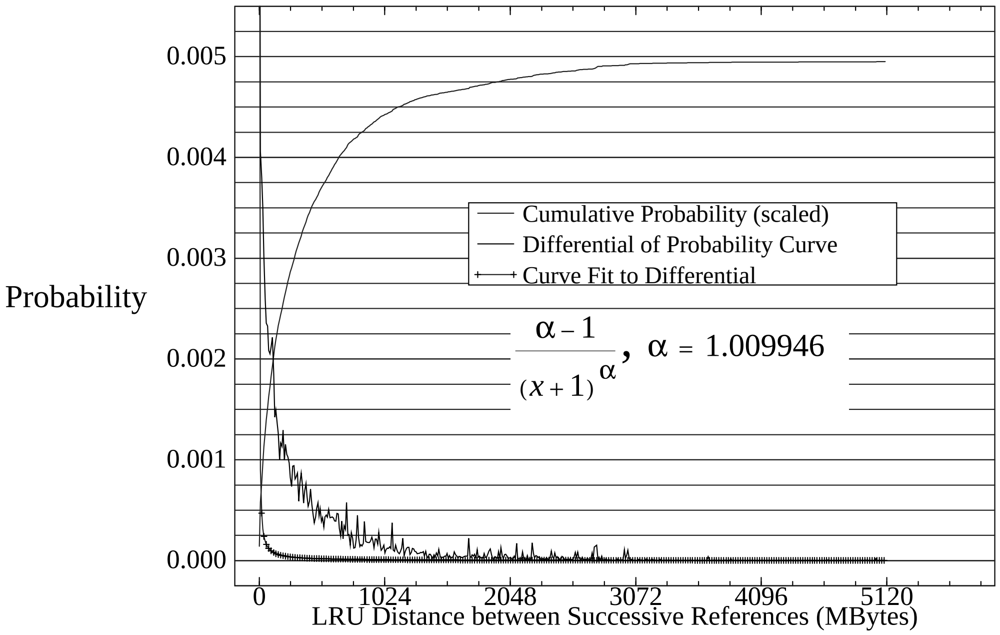


FIGURE 6. Fitting curves to the observed differentials. The graphs plot the locality behavior observed in the simulator. The first graph is fit with an exponential function, the second with a polynomial. The fitted curves are used for the values of α that they produce, in order to verify the analysis.

5.0 Comparison of Simulated and Analytical Results

The traces used in Section 4.0 were analyzed for their locality behavior, producing the curves shown in Figure 6. The analysis produced the cumulative probability curve and its differential, and the differential was fit by the two functions used in Section 3.3.1.

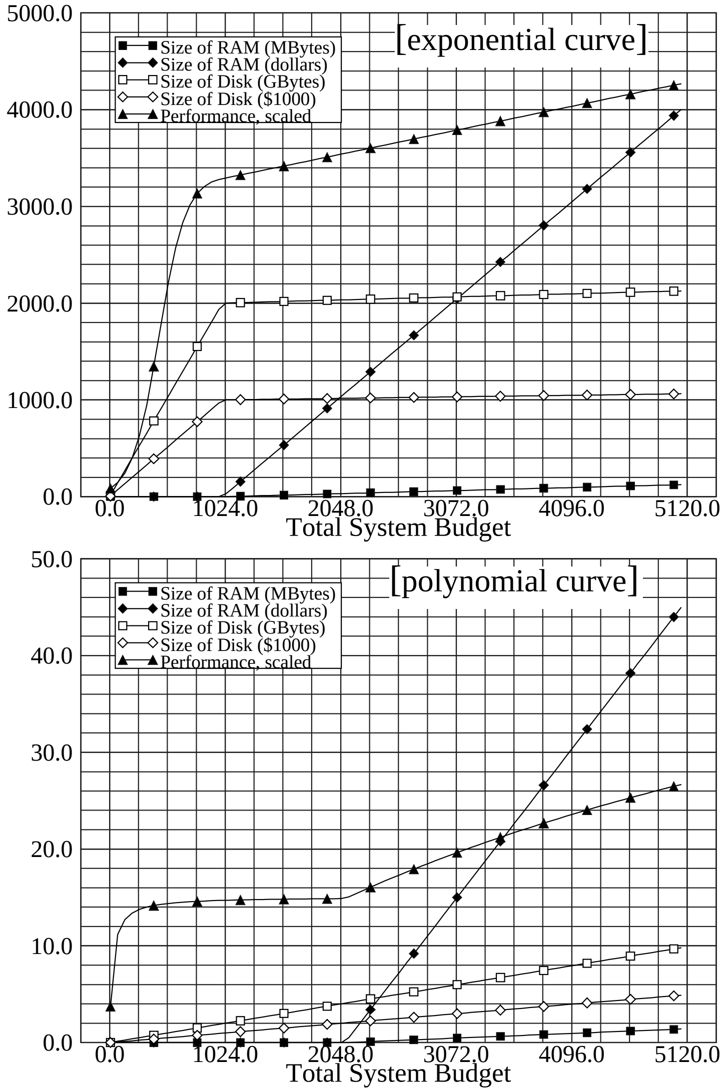


FIGURE 7. Calculated values for optimal configurations as a function of cost. Using the measured values for α, these curves were produced, showing the expected crossover points, ratios between RAM and Disk cache sizes, and the expected performance curves for each budget.

As a result, we can now graph the equations from Section 3.3.1 using the measured values for the α constants. For the exponential curve, we calculate the sizes and costs depicted in Figure 7, and plot the scaled performance figures as well. Note that the slopes of the MBytes RAM and MBytes Disk curves are parallel after the crossover point at around $1000. This crossover point agrees well with the simulations (see Figure 5), but the slopes of the lines do not. This is likely due to the fact that the exponential curve has the bulk of its area near the y-axis, and decays to zero very rapidly, giving the higher levels in the hierarchy the advantage, as it makes it easier to capture many references in a small amount of storage.

The results from using the polynomial curve are also shown in Figure 7. The shapes of the curves look similar to the curves produced by the simulator, but the scale is off; the amount of disk in the system greater than the amount of RAM in the system by a factor of almost 10,000, while the simulated curves fit within the same scale. The performance numbers are more realistic; for instance, they do not start as near to zero in the initial configuration.

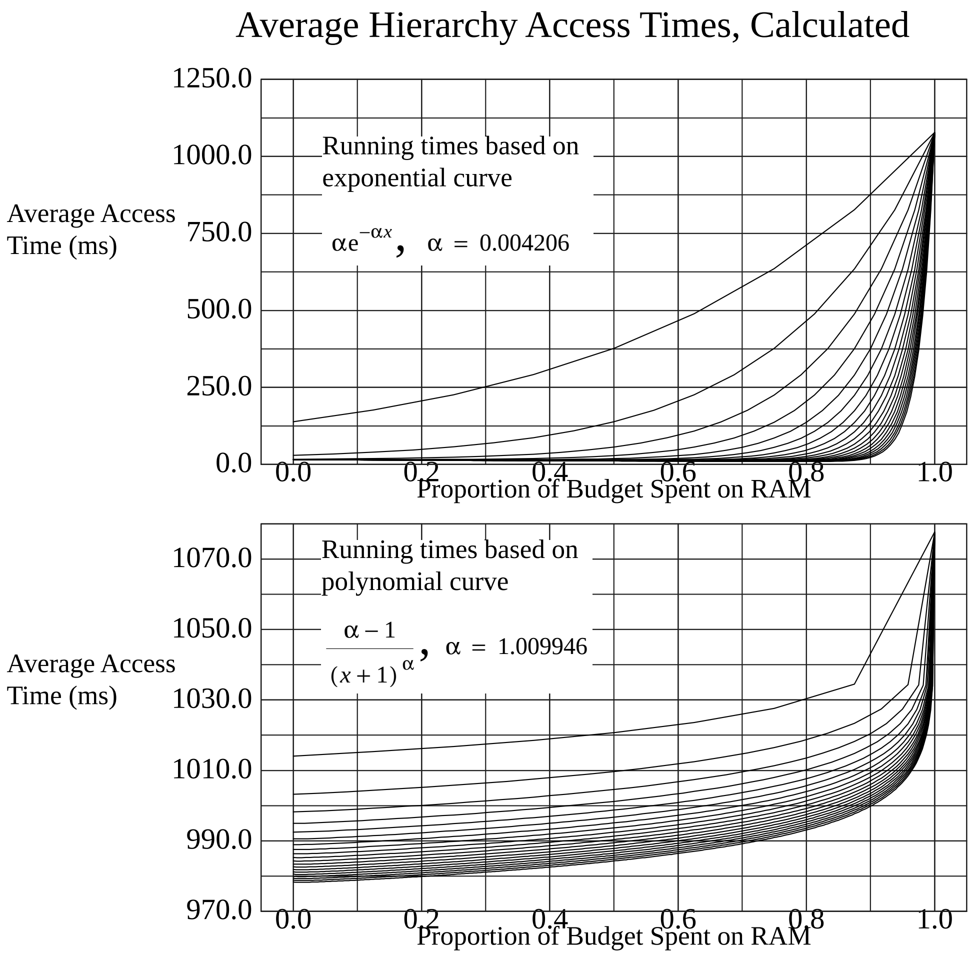


FIGURE 8. Access times for different configurations of system budget, based upon the observed values for α in the simulations. These graphs are similar to the one shown in Figure 3.

We can also plot the analytical curves for the expected running times of all the possible configurations for each budget. These curves are analogous to Figure 3, and are shown in Figure 8.

Each line in the graphs is derived from a single cost point; the curve represents the different ways of appropriating the budget between RAM and disk. The lines toward the top of the graphs represent smaller system budgets; the lines toward the bottom represent larger ones. The budgets range from \$256 to \$5120, in increments of \$256.

The graph based upon the polynomial is very interesting; at first glance, it appears to indicate that for every budget the optimal configuration has no RAM whatsoever. However, remember that in Section 3.3.1 and Figure 7 the polynomial approximation had almost 4 orders of magnitude more disk then RAM in the system. The curves in the polynomial-based graph of Figure 8 *do* in fact have local minimums once the system budget gets large enough; they are simply very hard to see as they are quite close to the y-axis.

The graph based upon the exponential curve appears to have either no minima on the individual curves or minima when the proportion of RAM is zero. Actually, the curves quickly start bottoming out at a non-zero RAM value, and by the time the last cost curve is reached (\$5120), the minima are almost at the lower right corner.

There are a number of inaccuracies in the graphs, but the overall trends are represented. It seems that much of the error can be attributed to two things. First, the fitted curves were not especially good fits; new curves are being plotted as this is being written. Second, and maybe more importantly, the simulator made a number of assumptions that the analytical model did not. For instance, the model ignores the effects of cold start and compulsory misses. The simulator did not, and the graphs demonstrate this (the performance curves are pretty bad). Also, the model ignores writes for the moment; this allows for a cleaner set of equations, but causes problems when the simulated results (which do *not* ignore writes, but rather treat them like reads) are compared against the calculated ones.

6.0 Conclusions and Future Work

We have demonstrated the comparative effectiveness of adding disk buffers or RAM buffers to a given system. The stack distance curves are an invaluable tool for doing experimental cache work; they are independent of the cache size and configuration and yet can be used to predict cache performance figures, as they are differentiable.

We have found a closed form solution for determining the optimal configuration of a two-level hierarchy. Since we have restricted focus to dealing with machines connected to mass storage devices, we are typically dealing with servers, whose file request streams exhibit a lower degree of locality than most people are used to thinking about. This arrangement yields a number of surprising results, like the fact that disk is far more important to systems than most people believe; a system with a few gigabytes of disk acting solely as a cache for the file system will perform better than a system with a cost-equivalent amount of RAM instead.

In the derivation of the solution for optimality, we present the performance ratio; a characterization of the effectiveness of the disk level. The ratio is used in calculating the optimal configuration for a given workload, scaling upward the point at which RAM should be added to the system, as the value of the ratio increases.

As far as future work goes, there are a number of items to work upon. Since the hierarchy model does not consider writes, this needs to be added and its effect upon the results needs to be measured. Along the same lines, the simulator needs to be rewritten to ignore compulsory and cold start misses in order to make the comparisons fair.

There have been a number of studies that suggest the *number* of levels in the hierarchy must grow as the size of the hierarchy grows. This makes intuitive sense, and should be investigated.

Acknowledgments

The present work is based upon the work done by myself and Seth Silverman for a class project in the Winter of 1994 (Peter Chen's EECS 598). I wrote the simulator, Seth ran the simulations and made the resulting graphs, and Eric Sprangle pointed us in the right direction to begin with.

Enormous thanks go to the men of CITI: Peter Honeyman, Saar Blumson, and Dan Muntz, for graciously allowing us to use their AFS traces. Without their data, most of this work would not have been done.

References

1. Advertisements, *Computer Shopper*, April 1994.
2. Antonelli, C.J. and Honeyman, P., "Integrating Mass Storage and File Systems," *Twelfth IEEE Symposium on Mass Storage Systems*, pp. 133-138, April 1993.
3. Baker, M., Asami, S., Deprit, E., Ousterhout, J. and Seltzer, M., "Non-Volatile Memory for Fast Reliable File Systems," *Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS-V), pp. 20-22, October 1992.

4. Blumson, S., Honeyman, P., Ragland, T. E. and Stolarchuk, M. T., “AFS Server Logging,” CITI Technical Report 93-10, November 1993.

5. Chow, C. K., “On Optimization of Storage Hierarchies,” *IBM Journal of Research and Development*, Vol. 18, No. 3, May 1974.

6. Chow, C. K., “Determination of Cache’s Capacity and its Matching Storage Hierarchy,” *IEEE Transactions on Computers*, Vol C-25, No. 2, February 1976.

7. Copeland, G., Keller, T., Krishnamurthy, R. and Smith, M., “The Case for Safe RAM,” *Proceedings of the Fifteenth International Conference on Very Large Databases*, pp. 327-335, August 1989.

8. Drapeau, A. L. and Katz, R. H., “Striped Tape Arrays,” *Twelfth IEEE Symposium on Mass Storage Systems*, pp. 257-265, April, 1993.

9. Garcia-Molina, H., Park, A., and Rogers, L., “Performance Through Memory,” Proceedings of the 1987 ACM SIGMETRICS Conference, pp. 122-131, May, 1987.

10. Katz R. H., Gibson, G. A. and Patterson, D. A., “Disk System Architectures for High Performance Computing,” *Proceedings of the IEEE*, pp. 1842-1858, December 1989.

11. MacDonald, J. E. and Sigworth, K. L., “Storage Hierarchy Optimization Procedure,” *IBM Journal of Research and Development*, Vol. 19, No. 2, March 1975.

12. Mitsubishi Electric, *Data Book: Semiconductor Memories/RAM*, 1990.

13. Nakagomi, T., Holzbach, M., Van Meter, R. and Ranade, S., “Re-Defining the Storage Hierarchy: An Ultra-Fast Magneto-Optical Disk Drive,” *Twelfth IEEE Symposium on Mass Storage Systems*, April 1993.

14. Przybylski, S. A., *Cache and Memory Hierarchy Design: a Performance-Directed Approach*, Morgan Kaufmann, San Mateo, CA, 1990.

15. Quinlan, S., “A Cached WORM File System,” *Software Practice and Experience*, pp. 1289-1299, December 1991.

16. Ranade, S., *Mass Storage Technologies*, Meckler Publishig, Westport, CT, 1991.

17. Redfield, R. and Willenbring, J., “Holostore Technology for Higher Levels of Memory Hierarchy,” *Eleventh IEEE Symposium on Mass Storage Systems*, pp. 155-159, October 1991.

18. Rege, S. L., “Cost, Performance, and Size Tradeoffs for Different Levels in a Memory Hierarchy,” *Computer*, Vol. 9, No. 4, April 1976.

19. Welch, T., “Memory Hierarchy Configuration Analysis,” *IEEE Transactions on Computers*, Vol. C-27, No. 5, May 1978.

20. Wu, M. and Zwaeneopoel, W., “eNVy: A Non-Volatile, Main Memory Storage System,” March, 1994.