Modeling Disk Traffic with Bias Methods

Chris Murray and Hao Cen

September 2006
CMU-ML-06-110

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Disk traffic modeling is useful in designing effective storage systems. One of the most difficult aspects of modeling disk trace data is understanding the underlying process which generates the trace. This work shows a novel method to model and learn the spatio-temporal locality in the trace generating process by using a conditional distribution based on recent disk accesses observed in the trace. Specifically, we present a class of models where the conditional distribution over the next disk block to access is biased towards recently-accessed disk blocks. Our method is flexible enough to be used to model any temporally-ordered finite sequence of events. We show several variants of this method and show how the method can be used not only to understand the trace-generating process but also to evaluate the performance of a storage system. Our experiments show that our method does a good job of capturing the process generating disk traces.

Keywords: Disk traces, disk traffic modeling, bias methods.

1 INTRODUCTION

Disk traffic data has been used to design effective storage system by feeding a real disk trace into trace-driven simulation systems. Collecting real trace data is labor intensive and time consuming. Using artificial trace data for trace-driven simulation systems allows researchers to generate new data by varying parameters, which would be expensive or impossible to do with real trace data. Also, being able to generate artificial trace data on the fly can save a huge amount of storage space.

Our goal is to design a model of the trace-generating process. Specifically, this model seeks to explain the ordering of accesses to different disk blocks as time evolves. This models seeks to capture not only the pure spatial or temporal aspects of the disk trace, but also the interplay between these. It does this using a small number of parameters and works in a way that allows for and captures some aspects of spatio-temporal locality. That is, this model captures the fact that real disk traces observe periods of high frequency accesses to a small number of disk blocks and that the accesses are biased towards recently-accessed blocks.

2 RELATED WORK

A number of methods for modeling disk traces have been proposed recently. Ganger [1] motivates the problem and shows the inadequacy of the Poisson arrival model due to its failure to capture spatial-temporal locality or burstiness in the trace. Fractional ARIMA is used to generate synthetic video traces. It has also been found to be unable to capture spatial-temporal locality or burstiness. [2]. Fractal Brownian motion [3], Fractal Wavelets [4], and On/Off models [5] provide interesting temporal modeling but do not capture spatial locality. Also these models require fitting a large number of parameters, so they lose the benefits of conciseness. The B-model [6] requires only one parameter to describe the entire trace, and the model fitting and trace generation algorithm run in linear time. The major shortcoming is that it fails to capture spatial locality properties of the trace. The I-model generates two-dimensional traces by multiplying marginal traces on time and space. That is, if 10% of the total requests arrive at time t and 5% of the total requests occur on address s, then $10\% \times 5\% = 0.5\%$ of the total requests have arrival time t and address s. The I-model preserves marginal distributions in the temporal domain and in the spatial domain, but it throws out information about spatial-temporal correlation, which is precisely what we want to capture.

The PQRS model [6] accurately captures spatio-temporal burstiness using only a small number of parameters, but it imposes a fractal structure on the disk trace and deals with absolute time, not system state, as the determinant of burstiness. An important method for capturing pure temporal burstiness models inter-arrival gaps in time series [7]. It uses a hierarchical Markov model to determine the inter-arrival gaps between events, and is shown to be useful in a variety of domains. Other models look at device performance relative to

other devices [8] or use a version of regression combined with a training set to predict storage device performance. [9]

3 DISK TRACES

The simplified version of a disk trace that we look at is an ordered list of tuples $[s_i, t_i]$ for $i = 1$ to L. $s_i \in S = \{1 \dots N\}$ tells which block on the disk was accessed, and $t_i \in \{1 \dots T\}$ tells the time stamp of the access (i.e. the number of milliseconds between the beginning of trace collection and the access). We aren't concerned with whether the access was a read or a write, or how many blocks the access read, and we assume the trace is given ordered in time so that $t_{i+1} \geq t_i$.

3.1 MARGINAL DISTRIBUTIONS

Given a disk trace, we observe some marginal distribution over the disk blocks $s \in \{1 \dots N\}$. We will call this the *marginal spatial distribution*. The observed marginal spatial distribution simply records, for each disk block $s \in S$, the fraction of accesses that went to block s. That is,

$$p^{\text{marg}}(s) = \frac{N(s_i = s)}{L}$$

where $N(s_i = s)$ counts, over i, the number of occurrences of the argument.

To model the marginal spatial distribution, we will simply store the observed marginal spatial distribution explicitly. We could approximate the marginal spatial distribution in some manner, but since our true goal is to look at spatio-temporal locality we won't worry too much about the pure spatial marginal distribution

The other marginal distribution we can observe from the disk trace is the *temporal marginal distribution*. We don't want to model the temporal distribution as a function of absolute time t: rather we want to model the distribution of *time gaps*. We do this by differencing the disk trace time stamps

$$t_i^{gap} = t_{i+1} - t_i$$

and then transforming and discretizing the observed series of gaps into a new series of log-gaps as

$$\text{lev}_\text{i} = \lfloor \log_2(1 + \text{t}_\text{i}^{\text{gap}}) \rfloor \tag{1}$$

We call the lev_is *burstiness levels*, where a low value of lev_i indicates a period of rapid disk accesses, and a high value of lev_i indicates a period of slow disk accesses. The observed marginal distribution of burstiness levels is

$$p^{\text{temp}}(\text{lev} = \text{k}) = \frac{\text{N}(\text{lev}_\text{i} = \text{k})}{\text{L} - 1}$$

We define these marginal distributions because any good model of the disk trace generating process should approximately preserve these marginal distributions.

3.2 SPATIO-TEMPORAL LOCALITY

The most important aspect of this work is looking at the spatio-temporal locality of a disk trace. We want to capture the fact that there is some "inertia" in a disk trace. That is, fixing the marginal temporal and spatial distributions defined above, we can define some conditional distribution over the next disk block to be accessed at time t, having observed the trace up to time $t - \epsilon$, as $p^{\text{cond}}(s;t)$. The unrealistic assumption that disk blocks are drawn I.I.D. from the marginal distribution would posit that $p^{\text{cond}}(s;t) = p^{\text{marg}}(s)$. We want to capture, with as few parameters as possible, the fact that a disk block that is accessed at time t will likely be accessed again soon after time t, so we will bias this conditional distribution towards recently-accessed blocks.

To be more precise, we'll define event $R(s,t)$ to be: $R(s,t) = 1$ if disk block s was accessed "recently" before time t and $R(s,t) = 0$ otherwise. Then we model the *conditional* distribution over the next block to be accessed so that it's biased in favor of recently-accessed blocks.[1] We define β as a factor to model the extent of bias towards recently-accessed blocks, with $\beta = 1$ indicating no bias and $\beta \gg 1$ indicating that the trace is much more likely than under I.I.D. to access a recently-accessed block next. We will present several methods of biasing the conditional distribution in favor of recently-accessed blocks, and we will collectively call these recent event bias methods, where β is the *recent bias coefficient*.[2] We will show how to interpret β and how to learn it from a real disk trace.

Two of our methods model the the conditional distribution as

$$p^{\text{cond}}(s;t) = \begin{cases} c_t \times \beta \times p^{\text{marg}}(s), & \text{if R(s,t)=1} \\ c_t \times p^{\text{marg}}, & \text{if R(s,t)=0} \end{cases} \tag{2}$$

$$c_t = \left(c_t \mid \sum_{s \in S} p^{\text{cond}}(s;t) = 1\right) \tag{3}$$

That is, the fact that block s was accessed recently makes it β times as likely as under I.I.D. that it will be accessed next. c_t is just a normalizing constant. Using this conditional distribution, we can create two different recent event bias models based on two precise definitions of $R(s,t)$. The first is that $R(s,t) = R(s,t;k)$, where $R(s,t;k) = 1$ if block s is one of the k most-recently-accessed blocks, where k is a parameter called the *memory size*. This is the *size-limited* model since in this case the trace-generating process only remembers the last k unique blocks that were accessed and it biases the next accesses towards these blocks.

The second model has that $R(s,t) = R(s,t;\tau)$, where $R(s,t;\tau) = 1$ if block s was accessed within the last τ seconds. In this case τ is the trace's *memory*

[1] From here on whenever we refer to the conditional distribution, we're referring to the conditional distribution over the next block to be accessed, given the history of the trace thus far.

[2] An event, in this case, is an access to a disk block, but our family of methods could be applied to any ordered sequence of events.

duration, and this method of modeling the conditional distribution is the *time-limited* model since in this case the trace generating process only remembers recently accessed blocks for a certain amount of time.[3]

We also use a third method to bias accesses towards recently-accessed blocks which we call the *hit rate matching* model. Using this method the conditional distribution depends on a parameter h which gives the probability that the next block accessed will be a recent block. Mathematically, under the hit rate matching method the conditional distribution is

$$p^{\text{cond}}(s;t) = \begin{cases} \text{Access a recent block,} & \text{with probability h} \\ \text{Access a non-recent block,} & \text{with probability (1-h)} \end{cases} \quad (4)$$

and once a decision is made to draw a recent or a non-recent block, that draw is made according to the marginal distribution $p^{\text{marg}}(s)$. Here the definition of recently accessed is the same as in the size-limited method: the last k blocks that were accessed are marked as recently accessed, and the rest are marked as not recently accessed. Though it's not used explicitly in the conditional distribution, we show in the next section how to get a numerical estimate the extent of recent event bias in the hit rate matching model, and since this parameter has the same interpretation as β in the other models, we will also refer to it as β for the hit rate matching model.

Intuitively, the hit-rate-matching model implies a constant probability to draw a recent block, while the size-limited and time-limited models imply that the probability to draw a recent block depends on the distribution $p^{\text{marg}}(s)$. We will refer to a model, with its memory parameter, as a model specification. That is, a model specification might be "A size-limited model with a memory size of 32." A model specification with a short memory (small k or τ) will have bias towards very recently-accessed blocks, while a model specification with a longer memory (larger k or τ) will also bias accesses towards less recently accessed blocks. Under all of the above variants the interpretation of β is roughly the same. It measures the factor by which the conditional distribution is more likely than I.I.D. to access a recent block. In our experiments we observe very high β factors in the disk traces, indicating very high bias towards recently-accessed blocks and the inadequacy of the I.I.D assumption. Our method of analyzing traces works in other domains (network traces for example) and observing different β factors in different domains or on different traces could tell us that qualitatively different behavior is generating the traces.

4 LEARNING THE SPATIO-TEMPORAL LOCALITY IN A TRACE

Given a model specification and a real disk trace, we want to be able to estimate the β coefficient (or h value for the hit-rate matching method) to understand the trace-generating process.

[3]We will show in a later section how k or τ can be chosen by likelihood maximization

4.1 LIKELIHOOD MAXIMIZATION METHOD

The way we estimate β for the size-limited and time-limited models is by likelihood maximization. That is, given a trace, a value of β, and a model specification we can compute L, the total likelihood of the ordering of disk block accesses in the trace. We do this by computing

$$L = \prod_{i=1}^{N} p(s_i \mid s_{i-1} \dots s_1) \tag{5}$$

When we run through the trace to compute this likelihood L, at any point in time we know which blocks s have been recently accessed since we know the model specification. We can define

$$\Gamma = (s \in S \mid R(s) = 1), \qquad \gamma = \sum_{s \in \Gamma} p^{\text{marg}}(s)$$

$$\Phi = (s \in S \mid R(s) = 0), \qquad \phi = \sum_{s \in \Phi} p^{\text{marg}}(s)$$

That is, all recently-accessed disk blocks are in Γ and the rest are in Φ. The *marginal* probability to access a block in Γ or Φ, respectively, is γ and ϕ. Since recently-accessed disk blocks are β times more likely to be accessed next than they would be under the marginal distribution, the conditional probability for the next access to be to some recently-accessed block is $\beta \times \gamma$[4] under the size-limited and time-limited models.

Since we can compute the conditional likelihood of an access given β using equations 2,3, and 4, we can compute the total likelihood of the trace with equation 5. Thus we can compute the β value that maximizes this total likelihood: this is the MLE (maximum likelihood estimate) β. The simple algorithm we use to do this computes the likelihood of the trace for several β models and hones in on the β which best explains the trace, though any other optimization method could be used.

4.2 SIMULATION METHOD

When we use the hit-rate-matching model of trace generation, β doesn't appear explicitly in the conditional distribution but h does. We can learn the value for h from a trace as

$$h^{\text{obs}} = \frac{\sum_{i=1}^{L} R(s_i, t_i)}{L}$$

[4]Of course this only works for $\beta \times \gamma \ll 1$, which means that memory (size/duration) shouldn't ever be too big compared to the total number of blocks. e.g. On a disk with 1,000,000 blocks, this method would encounter a problem if we defined the last 600,000 blocks accesses to be "recently-accessed" and estimated that $\beta = 20$

The numerator is just the total number of accesses that went to a recently-accessed block, so h^{obs} is the fraction of accesses in the trace which went to a recently-accessed block.[5]

Though β doesn't appear in the conditional distribution for the hit rate matching method, we can *define* β in such a way that it has the same interpretation and thus provides some comparability with the β factors from other methods. Under the hit rate matching method, to get an estimate of β we'll first calculate the "hit rate" h^{obs} observed in the trace, and then estimate what h^{IID} would be under the hypothesis that blocks are drawn I.I.D. from the marginal distribution $p^{\text{marg}}(s)$. We can estimate this by drawing some large number L' of blocks I.I.D. from the marginal distribution to generate an artificial trace, and then computing for this artificial trace

$$h^{\text{IID}} \approx \frac{\sum_{i=1}^{L} R(s_i)}{L'}$$

Finally, our estimate of β is

$$\beta = \frac{h^{\text{obs}}}{h^{\text{IID}}}$$

which is the factor by which the observed "hit rate" h^{obs} is higher in the real trace versus an I.I.D. trace on the same marginal distribution. Note that, as before, $\beta = 1$ implies no spatio-temporal locality, and $\beta = x$ means that the real trace is, in some sense, x times more likely than under I.I.D. to access a recently-accessed block.

4.3 MULTIPLE BETA FACTORS

We expect a single factor β associated with only a single k or τ that defines the spatio-temporal locality to do a reasonably good job of modeling the behavior of a trace. However, we expect that the true trace generating process may have by several levels of spatio-temporal locality. For example, it may be the case that accesses are strongly biased towards the most recently accessed 2 disk blocks, and weakly biased towards the most-recently-accessed 10 disk blocks.

To clarify with an example, imagine a system using a disk with 100 blocks, where marginally each block is accessed about 1% of the time. Further, imagine that usually 5 processes run at any given time, and each process has its own 2 disk blocks to which it writes. This trace is best modeled with two β coefficients associated with two different k values: one coefficient β_2 strongly biases the trace such that the next access is very likely to be to one of the two most-recently-accessed blocks,[6] but another smaller coefficient β_{10} slightly biases the trace such that the next access is more likely to go to one of the 10 most recently accessed blocks. To address this we propose defining some number w

[5] This h^{obs} can be interpreted as a hit rate on a cache of size k which always kicks out the least-recently-used block, hence we named this method the hit rate matching method.

[6] Since one process will likely get to write to its two blocks several times before another process can run

of different β_i coefficients, each associated with a different memory size k_i or memory duration τ_i. Some optimization techniques (like gradient descent with random restart) could be used to learn the MLE β factors from the real trace.

4.4 MODELING TEMPORAL BURSTINESS

We model temporal burstiness by using a Markov model [7] [10]. Specifically, in our experiments we used first-, second-, and third-order Markov models to describe the distribution of inter-arrival gaps given by lev_i (equation 1). In a first order Markov model, the distribution over the next burstiness level ($\text{lev}_{\text{i}+1}$) depends only on the current burstiness level (lev_i). In a third-order Markov model, the distribution over the next burstiness level ($\text{lev}_{\text{i}+1}$) depends on the three most recent burstiness levels ($\text{lev}_\text{i}, \text{lev}_{\text{i}-1}, \text{lev}_{\text{i}-2}$). We learn these transition distributions in either case by counting. For example, in the second-order Markov model,

$$p(\text{lev}_{\text{i}+1} = \text{a} \mid \text{lev}_\text{i} = \text{b} \wedge \text{lev}_{\text{i}-1} = \text{c}) = \frac{\text{N}(\text{lev}_\text{j} = \text{a} \wedge \text{lev}_{\text{j}-1} = \text{b} \wedge \text{lev}_{\text{j}-2} = \text{c})}{\text{N}(\text{lev}_{\text{j}-1} = \text{b} \wedge \text{lev}_{\text{j}-2} = \text{c})}$$

Algorithm 1 Synthetic Trace Generating Algorithm

```
1: for s ∈ S do
2:    non-recent.Add(s)
3: end for
4: time=0
5: for i = 1 ... L' do
6:    time ← time + MarkovGapDist.DrawGap()
7:    if Rand(0,1) < β×recent.TotalWeight() then
8:       block = recent.Draw()
9:       recent.MoveToFront(block,time)
10:   else
11:      block = non-recent.Draw()
12:      non-recent.Remove(block)
13:      recent.Push(block,time)
14:   end if
15:   WriteAccess(block, time)
16:   if recent.ContainsStaleBlocks(time) then
17:      Move all stale blocks from recent to non-recent
18:   end if
19: end for
```

4.5 GENERATING ARTIFICIAL TRACES

Given a model specification, and having learned the associated parameter β or h, the marginal distribution $p^{\text{obs}}(s)$, and the parameters of the Markov model

to represent temporal burstiness, we can quickly create synthetic traces that could be used to stress test a storage system. We'll use several data structures.

- Two AVL trees *recent* and *non-recent*. The first tree represents disk blocks that were recently accessed and is initially empty and the other represents disk blocks that were not recently accessed and it initially contains all disk blocks. In both AVL trees, each node represents a single disk block s and each node has an associated probability weight $p^{\text{marg}}(s)$. The AVL trees are also modified so that each node knows the total probability weight in its subtree, allowing us to draw precisely from the marginal distribution $p^{\text{marg}}(s)$ within an AVL tree in $O(\log(N))$ time.

- One priority queue q: this is either size limited (with max size k) or time limited (with maximum staleness τ). The recently accessed nodes are in this priority queue, and when a node becomes non-recent it is removed from the priority queue and moved from the AVL tree *recent* to the AVL tree *non-recent*

A description is shown in algorithm 1.[7] Initially all blocks are not recently accessed. For each simulated accesses, we choose the next block to access using β to bias our draw towards recently-accessed blocks. Then we randomly draw a time gap from the learned Markov distribution. The new block we drew is marked as recent, and then any blocks stored in **recent** that just became non-recent are moved into the **non-recent** priority queue.[8] The algorithm runs in $O(\log(N) \times L')$, with the $\log(N)$ coming from the fact that **recent** and **non-recent** both store the marginal distribution $p^{marg}(s)$ in an AVL tree, so inserting and removing blocks costs $O(\log(N))$. To implement trace-generation for the hit rate matching model, we would change line 7 in algorithm 1 to "**if**$(Rand(0,1) < h)$**then**".

5 DISK TRACE EXPERIMENTS

The ultimate goal, of course, is to model the process that generates disk traces for the performance evaluation of storage systems. We use HP Cello traces to learn our model parameters, and use these for benchmarks. In the traces we aggregate disk blocks using an aggregation factor of 32 such that

$$s' \leftarrow \lfloor \frac{s}{32} \rfloor$$

[7] Readers may worry that line 8 will try to draw from an empty AVL tree. However, line 8 is only executed if the condition on line 7 is true, and if **recent** is empty it will have 0 total weight and thus assure that the condition on line 7 is false.

[8] Specifically, if we're using the size-limited model, only the k recently-accessed blocks can be marked as recent at any given time, so an access to a non-recent block will kick out at most one block from **recent**. Using the time-limited model, the update of the time may cause some recent blocks to not have been accessed for τ seconds and they will be removed from **recent**.

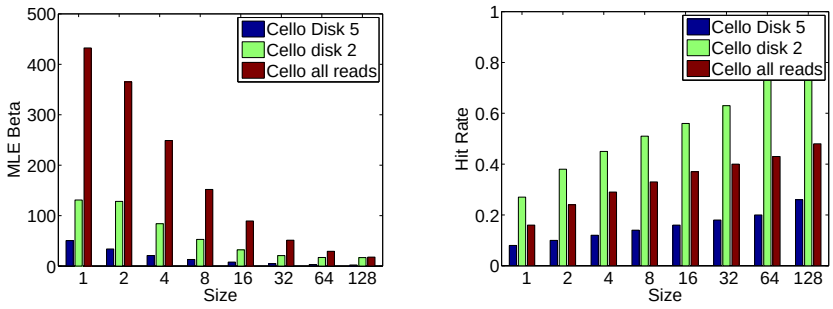

Figure 1: The left plot shows MLE β values for three disk traces. Values of β are shown for different memory sizes k under the size-limited model. The right plot shows "hit rates" h^{obs} for three traces under the hit-rate-matching model.

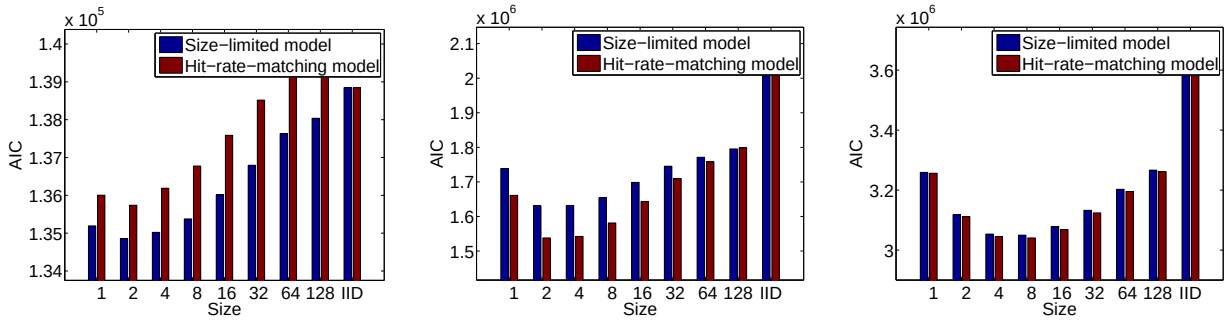

Figure 2: Shows AIC values for various memory sizes k under the size-limited and hit rate matching models. The left-most plot is the cello disk 5 trace, the middle is the cello disk 2 trace, and the right most plot is the aggregate reads trace. AIC values are computed using the MLE beta estimate. The right-most AIC value is computed under the I.I.D. assumption where $\beta = 1$. Lower AIC values indicate a model that better explains the data. In all cases, the bias models explain the data better than the marginal distribution, and the best model over the three traces uses a relatively small memory size ... between 2 and 8.

and use s', rather than s, in all our experiments. This keeps the representation of the spatial marginal distribution smaller and models the fact that disk reads will read in an entire cache line, not just a single block. We model temporal burstiness with first-, second-, and third-order Markov models. To evaluate our model we use two methods. First, to evaluate how well the β bias coefficients explain the disk block access ordering in the trace, we use the the Akaike Information Criterion

$$AIC = 2k - 2\text{LL}$$

where LL is the log-likelihood of the disk block access ordering of the trace and k is the number of parameters we estimated (two if we use a single β and k or τ).[9] The AIC statistic penalizes models with a large number of parameters, and a *lower* value of the AIC statistic indicates a model that better explains the disk block access ordering in the trace. We also created a very simple disk simulator to test the effectiveness of our Markov model at capturing the temporal burstiness of the trace, and we use measures of request wait time and queue length in real versus synthetic trace to do the comparison.

Our first goal is to look at the extent of spatio-temporal locality in the traces. We can see in figure 1 that all traces have very strong bias towards recently-accessed blocks. Indeed we see from the MLE β estimates that, for the trace taken on disk two and the aggregate trace of disk reads, the next access under the conditional distribution can be over 100 times more likely than I.I.D. to be to the most-recently-accessed block. Clearly the traces studied are not I.I.D. Unsurprisingly, the MLE β values decrease as the memory size increases. This makes sense because, under I.I.D. the chance of drawing the single most recently-accessed block is tiny since the traces typically access thousands of different blocks, so even a small hit rate on the most-recently-accessed block corresponds to a very high β. However, under I.I.D. the chance to draw one of the most recent 1024 blocks, for example, is already reasonably high, so even with a very high bias towards the most recently accessed 1024 blocks, the β value cannot be very high because probabilities can't be more than 1. The hit rates in figure 1 are surprisingly high for small memory sizes: the trace on disk 2 re-accessed the most-recently-accessed block almost 30% of the time, leading to this trace's high β.

Figure 2 shows the AIC values under the size-limited and the hit-rate-matching models. Recalling that a lower AIC value implies a model that explains the data better, we see that neither model clearly prevails, since the trace on disk 5 has lower AIC values for the hit-rate-matching model and the trace on disk 2 has lower AIC values for the size-limited model. But it is clear that shorter memory sizes/durations do a better job of explaining the trace and in all cases the bias models are far better than the I.I.D. model. Thus the traces are better modeled as having a stronger bias towards a few most-recently-accessed blocks

[9] Our experiments are looking at different ways to model the spatio-temporal locality, not the pure marginal $p^{marg}(s)$, so in computing the AIC we won't penalize for the $N-1$ parameters we learned to describe the marginal $p^{marg}(s)$. Even if we did, this term would appear in all models we consider so it wouldn't affect the choice of which is best.

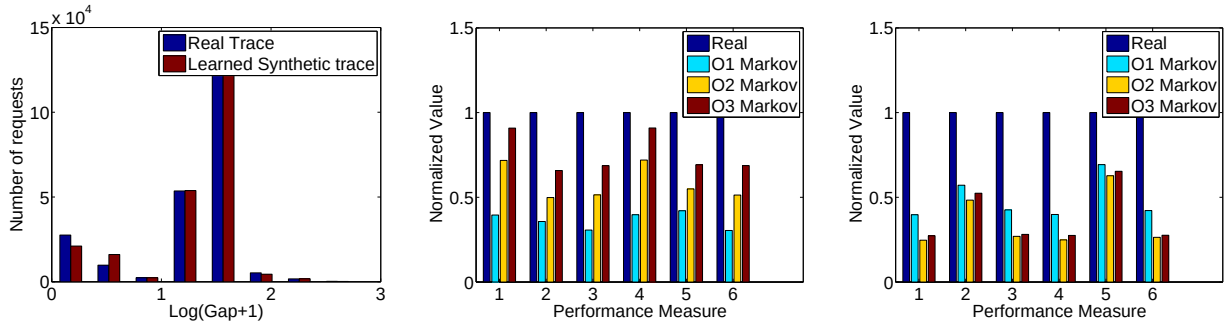

Figure 3: The left figure shows a log histogram of time gaps between requests in a real cello trace and a synthetic trace. The right two figures show disk simulator performance measures for a real trace and synthetic traces. The measures are (1) Maximum Queue Length (2) Average Queue Length (3) Std. Dev. of Queue Lengths (4) Maximum request wait time (5) Average Request Wait Time (6) Std. Dev. of request wait times. These measures capture the effectiveness of the Markov models at capturing pure temporal burstiness.

rather than as having a weaker bias towards a larger number of recently-accessed blocks.

Next we examine how well the pure Markov models capture the temporal burstiness of real disk traces. The left graph in figure 3 shows that the Markov models we use do a very good job of reproducing in a synthetic trace the distribution of inter-arrival gaps from a real trace. But the disk simulator results shown in the center and right of figure 3 show that some aspects of temporal burstiness aren't captured by the Markov models, since even the synthetic traces created by third-order Markov models have different simulated disk performance from the real traces.[10]

Finally we looked at how the synthetic traces generated using the various bias models combined with the Markov temporal model compare to the real trace. The recent event bias models only affect which disk block is drawn (not the time stamp). So the primary difference in disk performance between using the bias models with the Markov temporal model versus using the Markov temporal model and drawing blocks I.I.D. is that the bias model should generate more realistic disk cache hit ratios. So in figure 4 we compare the disk cache hit rates on a simple 40-block LRU cache across various bias models for several traces. In each trace shown in the figure, the left bar corresponds to the real trace, the next bar is an I.I.D. synthetic trace, and the other 6 bars are synthetic traces using various bias models. Clearly the I.I.D model does a terrible job of matching the real disk cache hit rate, while the size-limited and hit-rate-matching models generally do a very good job of matching it. The time-limited model's performance is more variable, but clearly much better that I.I.D. It

[10]These experiments purely looked at the ability of the Markov models to capture temporal burstiness, and did not utilize the bias aspects of the model.

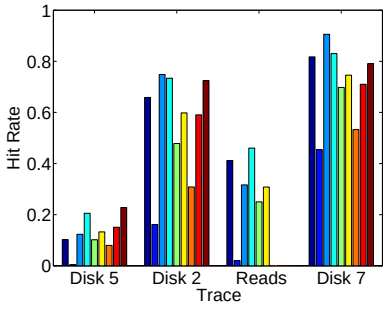

Figure 4: Cache hit rates on a cache of size 40 for various traces. Within a group, the leftmost bar (1) is the hit rate on the real trace. The next bar (2) is on a synthetic trace using the same marginal distribution but drawing I.I.D. The next six bars are hit rates for the following models: (3,4) Size-limited with with $k = 2$ and $k = 64$ (5-6) Hit rate matching with $k = 2$ and $k = 64$ (7-9) time-limited with $\tau = 8, 64, 1024$.

seems that the time-limited model of medium duration ($\tau = 64$ milliseconds) performs closest to the real traces. While at first it may seem odd that the hit rate matching model doesn't reproduce the exact same disk hit rate as the real trace, remember that the hit rate matching model matches the real traces hit rate for some memory size k, which is not necessarily equal to the size of the disk's cache. In the experiments of figure 4 we tried hit rate matching models with $k = 2$ and $k = 64$ and came reasonably close to the real hit rate on a disk cache with memory size 40.

6 CONCLUSION

We presented a method with a family of variants that models the process that generates disk traces, specifically the extent of bias towards recently-accessed blocks in the condition distribution. We showed how β factors model the extent of this bias and how to efficiently estimate the β factor from a trace and generate a synthetic trace with a given β factor. Lastly we looked at real disk traces and found that the bias models do a good job of explaining the access ordering in the trace and that the β factors in real traces can be quite large, indicating strong bias towards recently accessed blocks.

The β factors can tell storage system engineers some important information about the load placed on disks, and the trace generating procedure can be used to stress-test storage systems over a wide array of scenarios with high and low β factors.

Some further research would be to look more closely at using optimization techniques to simultaneously calculate multiple MLE β factors on a given trace, rather than being restricted to one. Also, it would be interesting to more tightly couple the extent of recent event bias with the level temporal burstiness.

7 ACKNOWLEDGEMENTS

The authors wish to thank Christos Faloutsos for many helpful discussions and data, Michael Mesnier for discussions and insights, and Geoff Gordon for support, ideas, and paper comments.

References

[1] G. R. Ganger. Generating representative synthetic workloads: An unsolved problem. In *Proceedings of the Computer Management Group (CMG) Conference*, pages 1263–1269, 1995.

[2] Mark W. Garrett and Walter Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *SIGCOMM*, pages 269–280, 1994.

[3] Will E. Leland, Murad S. Taqq, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. In Deepinder P. Sidhu, editor, *ACM SIGCOMM*, pages 183–193, San Francisco, California, 1993.

[4] R. H. Riedi, M. S. Crouse, V. J. Ribeiro, and R. G. Baraniuk. A multifractal wavelet model with application to network traffic. *IEEE Transactions on Information Theory*, 45(4):992–1018, 1999.

[5] R. Riedi and J. Vehel. Multifractal Properties of TCP Traffic: a Numerical Study. In *IEEE Transactions on Networking*, October 1997.

[6] M. Wang, A. Ailamaki, and C. Faloutsos. Capturing the spatiotemporal behavior of real traffic data, 2002.

[7] J. Kleinberg. Bursty and hierarchical structure in streams, 2002.

[8] Michael Mesnier Intel. Modeling the relative fitness of storage devices.

[9] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. Ganger. Storage device performance prediction with cart models, 2004.

[10] Rabiner Lawrence R. A tutorial on hidden markov mmodels and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.