Improving SimPoint accuracy for small simulation budgets with EDCM clustering

Joshua Johnston[1] and Greg Hamerly[2]

[1] Department of Computer Science, Texas A&M University, TAMU 3112
College Station, TX 77843-3112
joshua.johnston@neo.tamu.edu **
[2] Computer Science Department, Baylor University
Waco, TX 76798
greg_hamerly@baylor.edu

Abstract. Detailed processor simulation is extremely costly on large benchmark suites, where each program may run for billions of instructions and take months of simulation time. We can obtain good approximate answers in less time using limited simulation, but deciding which regions to simulate is a difficult problem. SimPoint is one approach for choosing simulation regions, based on the k-means clustering algorithm. We propose using an alternative clustering model based on a mixture of exponential Dirichlet compound multinomial (EDCM) models. This method outperforms k-means in performance prediction accuracy when simulation budgets are limited. The EDCM mixture can cluster high-dimension frequency vector data directly, without dimension reduction, and trains quickly.

1 Introduction

When designing computer processors, it's important to tune them for performance, power usage, size, etc. Software simulation is a widely used method for evaluation. However, detailed performance evaluation on a simulator is extremely slow. For example, it can take many months of real CPU time to evaluate a full set of SPEC CPU benchmarks for one processor design. Worse, the number of designs to evaluate grows exponentially with the number of parameters involved, so full, detailed simulation is prohibitively slow.

The architecture community has investigated several ways of reducing simulation time, with some dramatic improvements. While it may take years of CPU time to simulate a full benchmark suite, a good approximation might be obtained in hours by limited simulation. Most of these approaches save time by simulating only some parts of a benchmark and ignoring other parts. SimPoint [1] uses k-means data clustering to identify patterns of behavior in a benchmark program's execution, and then uses detailed simulation on a sample of each behavior to obtain a representative picture of the whole program's execution.

** Work done primarily while at Baylor University.

The k-means algorithm works well for this application, but it has difficulty operating on high-dimension data, like benchmark traces represented as basic block vectors. Often, such programs will have more than 100,000 basic blocks (dimensions). The difficulties of learning are due partly to the cost of processing large amounts of data, and partly to the so-called 'curse of dimensionality,' which requires the number of examples to grow exponentially with the dimension in order to densely populate the input space.

SimPoint overcomes this difficulty by using random projections to reduce the data's dimensionality prior to clustering. However, it would be preferable to use a clustering algorithm which could operate on the original data. Because of this, we turn to methods that are more successful in high dimensions because they consider dimensions to be independent (the naïve assumption) and do not compute distances or joint probabilities. One example is the multinomial distribution, often used in text clustering. Previous work [2, 3] has used mixtures of multinomials for clustering program traces, but with little success. The multinomial model does not do well at modeling 'bursty' behaviors [4], or those behaviors which are rare but occur repeatedly once they do occur (as we expect in program execution traces).

In this paper, we use the exponential Dirichlet compound multinomial (EDCM) for automatically identifying program phases, in place of k-means. The EDCM is a new [5], efficient approximation of the Dirichlet compound multinomial (aka the multivariate Pólya distribution). Both are related to the multinomial. The EDCM has been used for clustering text documents, but not for clustering program executions. An EDCM mixture is efficient to train and can cluster in the original space of inputs (more than 100,000 dimensions in our experiments) without difficulty. To our knowledge, this is the first time that basic-block vector program execution traces have been clustered in their original dimension.

For limited simulation budgets, EDCM clustering provides improved prediction error rates and lower prediction error variance compared with k-means. This is important because small simulation budgets are where structured program analysis is most useful, as opposed to techniques based on regular or random sampling, like SMARTS [6]. Ultimately, this means less simulation time for the same performance prediction accuracy.

2 Background and related work

We begin by discussing the need to reduce simulation time in large workloads. We then discuss the data representation and learning problems for clustering benchmarks. A brief review of related work then leads to the motivations for our current work.

2.1 Detailed architectural simulation

Computer architecture uses simulation to determine the performance of new processor designs. These simulations are on detailed *software* implementations

of the processor design, occurring well before the fabrication of a real chip. One type of study a designer may perform is a design space exploration, trying many variants of a processor design. There are many variables that must be fine-tuned, such as cache size, branch predictor type, etc. Thus, one general design may lead to hundreds of variants, all of which need to be compared via simulation. Simulation produces a set of summary statistics over a program's execution, like the cycles per instruction (CPI) rate, or the branch prediction error rate.

Detailed processor simulation is slow. The SPEC CPU2000 suite of benchmarks has 36 program/input pairs, and executes a total of approximately 4 trillion instructions. At a rate of 500 million instructions per hour for the detailed mode (`sim-outorder`) of the SimpleScalar simulator [7], a detailed execution of the entire set of SPEC CPU2000 benchmarks would require almost a year of CPU time. And this must be done for *each* combination of parameters in a design-space exploration, compounding the problem.

Since many programs have repetitive behaviors, it is excessive to simulate a program's entire execution. A solution is to simulate small portions that will be representative of the entire execution. Of course, choosing *which* portions to use is an important problem. A simple technique is to simulate a small contiguous region of each program, starting at the beginning or some fixed offset (e.g. after one billion executed instructions). However, the simulated interval may not be representative of the whole program. By sampling throughout the program, the results are much better. This has led to approaches such as SimPoint and SMARTS [6], among others.

2.2 Representation and learning

Sherwood et al. [8] identified that a key to predicting performance of a program is the code it is executing. They developed a basic block vector (BBV) representation for program traces, in which a program's execution is divided into non-overlapping time intervals. The intervals can be fixed-length (e.g. every 100 million executed instructions), or variable-length (e.g. on the boundary of every nth procedure call). Before detailed simulation, each interval is profiled using a fast simulation model to determine which basic blocks execute during that interval. A basic block is an atomic unit of program execution, having one entry and one exit. Each interval has its own BBV, which starts with all zero values. When the interval is profiled, if basic block d executes, the dth entry in the BBV is incremented by the number of instructions in that basic block. Each program typically has thousands of basic blocks (e.g. the programs in SPEC CPU2000 range from 2,039 to 103,308). In a machine learning context, the number of basic blocks is the dimensionality, and each BBV is an example.

The learning problem is then to identify program behaviors, sample a subset of program intervals which are representative of these behaviors, and compute a set of weights indicating the proportion of execution each behavior represents. Simulating only the sample and using the weights to combine the results gives an estimate of the performance of the whole program. Researchers can simulate a disjoint sample of intervals by either fast-forwarding (using faster, non-detailed

simulation) between detailed simulation intervals, or by checkpointing (saving the state of the system) right before the detailed simulation regions.

2.3 SimPoint and *k*-means

The SimPoint project [1] proposed using data clustering on the BBVs of a program's execution to identify its key behaviors. After dividing the intervals into k clusters, each cluster contains BBVs with similar execution patterns, regardless of when they execute (temporal information is not used in clustering). SimPoint then chooses k intervals ('simulation points'), one from each cluster, to be simulated in detail. Associated with each simulation point is a weight which corresponds to the size of its cluster. After performance statistics have been gathered on these simulation points with detailed simulation, a weighted combination of their statistics gives an estimate of the performance of the entire program.

SimPoint employs the k-means algorithm for clustering. The advantages of this algorithm are that it is easily implemented and efficient. Various optimizations are also available because of the hard-assignment property of the algorithm (each example belongs completely to exactly one cluster). However, k-means makes several limiting assumptions. For example, each cluster has the same spherical covariance. Also, k-means suffers somewhat from the curse of dimensionality since it uses distances for comparing examples, and, in high dimension, most examples appear to be far apart.

To combat the difficulties of high dimension, SimPoint uses random linear projection [9] prior to clustering. This technique uses a small number of randomly-chosen vectors onto which the original BBVs are linearly projected. Previous work showed that 15 dimensions are sufficient to obtain good results with SimPoint [10]. Projection also dramatically reduces the run time of k-means, as the projected BBVs are much smaller than the original vectors.

2.4 Related work

Other researchers have used various profiling techniques to categorize program behavior or stability. Early work by Denning and Schwartz identified that programs often have repetitive usage patterns for memory areas, called working sets [11]. Balasubramonian et al. [12] used hardware counters to collect statistics about performance over time, such as branch miss rates, CPI, etc. Looking at these statistics allowed them to identify changes in behavior so caches might be dynamically reconfigured. Dhodapkar and Smith [13–15] pointed out a relationship between program behaviors and instruction working sets, and that changes in the former can be correlated to changes in the latter.

Clustering models other than k-means did not fare as well as what we present here. A mixture of multinomial models, proposed by Sanghai et al. [2], did not significantly improve on the k-means algorithm, as shown by Hamerly et al. [3]. The multinomial mixture had difficulty in high dimension, did not improve on the predicted error rates, and random projection is not as straightforward to use for multinomials as for k-means. Other experiments we have performed, which

we don't report here, showed that general Gaussian mixture models (which are similar to k-means models) also do not improve performance prediction significantly over k-means. The IDDCA approach [16] and others offer some improved performance in clustering efficiency over k-means by using a single-pass training phase, instead of multiple passes like k-means. However, the focus of this paper is in high accuracy in performance prediction for small simulation budgets, so we are less concerned about the cost of clustering. Thus we compare our method in this paper with k-means-based SimPoint, as it is widely used, efficient, and makes accurate predictions.

2.5 Motivation for our approach

We wish to perform a SimPoint-style clustering analysis on basic block vectors and sample cluster representatives for the purposes of simulation. However, we wish to do this without the use of random projections and with a statistical model that is better at representing bursty program behavior [4]—those behaviors which are rare but occur repeatedly once they do occur.

While SimPoint operates well using k-means for clustering and random linear projections to make the problem tractable, we want to learn a clustering of the original data without having to use dimension reduction. There are three reasons for this. First, a pair of true, well-separated clusters could collapse together in the projected space. If this happens, the two unrelated true clusters cannot be separated by a clustering algorithm. This problem is unlikely to occur if the projected space is sufficiently large [9, 17], but it is still a difficulty. Second, if we want to analyze the original unprojected data, we must map back to the original space after clustering in the projected space. While not difficult, it requires extra computation and effort. Third, random projection adds nondeterminism, which makes it more difficult to perform repeatable research. Further, different projections may lead to different clusterings without a clear method to choose among them.

Existing approaches (e.g. k-means or multinomial mixtures) also do not account for bursty data. Many programs have this property, as they often spend time in loops over the same code regions. If a piece of code executes once within a time interval, we observe that it will be likely to execute again (perhaps many times) in that interval. Figure 1 shows how program execution data is bursty, for rare basic blocks as well as common basic blocks. Compare this figure to Figure 1 in [4], as evidence that program execution exhibits bursty behavior, like word occurrences in text documents. The EDCM is a probability distribution that is good at modeling such bursty behavior.

3 Methodology

In this section, we describe using the EDCM mixture model to automatically classify workload execution behaviors. The EDCM mixture operates on the same BBV data that SimPoint analyzes, but in the original dimension rather than

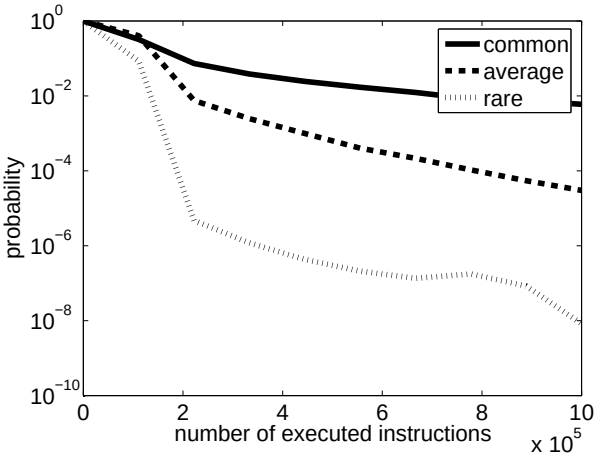

Fig. 1. The probability of seeing the given number of executed instructions within one interval for basic blocks which are common (top 5% of basic blocks, in terms of occurrence frequency), average (next 10%), and rare (remaining 85%). This graph shows an average over the SPEC CPU2000 suite of benchmarks. The curves tend to level off and form long tails, which indicate burstiness. A non-bursty source's curve should decrease exponentially.

having to rely on random linear projections to reduce the data's dimension. We discuss basic clustering questions that arise from this application, such as how we train EDCM mixtures, how we choose cluster representatives, and how one might select the number of clusters that should be used.

3.1 The EDCM and its relatives

The EDCM is an approximation of the DCM (Dirichlet compound multinomial), which is a relative of the multinomial probability distribution. The EDCM possesses several useful properties which allow it to perform well in our domain with high dimension and bursty code execution behaviors. For what follows, we discuss single instances of each type of distribution, which is one cluster in a clustering model. For clustering, we use a mixture of k EDCM distributions.

Multinomial A multinomial distribution defines a probability for a vector x, which contains the counts of occurrences of D mutually exclusive events. For a fixed multinomial distribution, each feature x_d $(1 \leq d \leq D)$ has probability θ_d of occurring (with constraints $\sum_d \theta_d = 1$ and $0 \leq \theta_d \leq 1$). We observe n events $(n = \sum_d x_d)$ and compute the joint probability of those events using

$$Pr(x|\theta) = \frac{n!}{\prod_{d=1}^{D} x_d!} \prod_{d=1}^{D} \theta_d^{x_d}. \tag{1}$$

As x_d increases, the $P(x|\theta)$ decreases (because usually $\theta_d < 1$). In bursty phenomena like programs, however, we expect that if a basic block is executed once, then it is likely to be executed again soon due to temporal locality. Thus, we do not want to penalize repeated events by reducing probabilities as severely as the multinomial distribution does.

Dirichlet compound multinomial The Dirichlet compound multinomial (DCM) is a variant of the multinomial model [18]. Its construction allows it to model bursty behavior.The DCM can be considered a distribution over multinomial distributions. The probability of generating a vector x (e.g. a basic block vector) using the DCM with parameter vector α is

$$Pr(x|\alpha) = \frac{n!}{\prod_{d=1}^{D} x_d!} \frac{\Gamma(s)}{\Gamma(s+n)} \prod_{d=1}^{D} \frac{\Gamma(x_d + \alpha_d)}{\Gamma(\alpha_d)}, \tag{2}$$

with $n = \sum_d x_d$ and $s = \sum_d \alpha_d$. Note that unlike the multinomial's θ_d, the DCM parameters (α_d) and EDCM parameters (β_d, discussed next) are not required to sum to one.

Exponential Dirichlet compound multinomial The EDCM distribution is an approximation of the DCM that brings it into the family of exponential distributions. Exponential distributions are useful because they are easy to evaluate and their parameters are simple to estimate. Elkan [5] proposed the EDCM, noting that the vector representation of a text document is likely to be sparse, because most documents contain only a small subset of the entire vocabulary. Basic block vectors are also sparse; on average, over 86% of the entries are zero (for the data we consider). Elkan shows a DCM approximation can be made if $\alpha \ll 1$, which we find to be true for the majority of basic blocks in BBV data. The EDCM is formulated as

$$Pr(x|\beta) = \frac{n!}{\prod_{d:x_d \geq 1}^{D} x_d} \frac{\Gamma(s)}{\Gamma(s+n)} \prod_{d:x_d \geq 1}^{D} \beta_d. \tag{3}$$

Here, β is used to distinguish the parameters of the EDCM from α, the parameters of the DCM. Similar with Equation 2, $n = \sum_d x_d$ and $s = \sum_d \beta_d$.

Burstiness An event is considered bursty if, when it occurs once, it is likely to occur many times within a short time [19, 20]. Madsen et al. [4] performed analysis on a corpus of text documents and defined three different groups of words: common, average, and rare. The rare words comprised 89% of the vocabulary but only 8% of all the word occurrences in the text documents. These occurrences are very important, though, because they serve as markers to help identify document content. In Figure 1, we show a similar analysis for basic blocks that illustrate that they too have bursty behavior, due to temporal locality. For example, if a benchmark contains a tight loop, the basic blocks within the loop will occur many times within the interval that contains that loop.

3.2 EM learning for a mixture of EDCMs

Before this section, we used x_d to denote the dth element of the single vector x. From here on, our discussion focuses on a collection of example vectors. We introduce a second subscript and use x_i to denote the ith vector in a collection X, and x_{id} to denote the dth element of that vector.

To cluster BBVs, we use a mixture of k EDCM distributions. The mixture trains on the BBVs, and after training each EDCM represents a probability density for one cluster, or phase. Each cluster has its own set of β parameters for the EDCM as well as a prior probability for the cluster (where the sum of the priors is one). The learning goal is to estimate a suitable set of parameters for the β vector of each cluster and the prior probabilities for the clusters.

We use the standard expectation-maximization (EM) algorithm for learning an EDCM mixture for a fixed number of clusters, k. We start with a random set of initial β values and a set of uniform priors (a randomized M-step). Each E-step calculates the membership probability $P(c_j|x_i)$ for the jth cluster c_j and example x_i, according to Bayes' rule. Each M-step chooses the cluster priors and β values that maximize the likelihood of the model based on the current expected memberships. The training proceeds with alternating E and M steps until the models converge. The interested reader is referred to Elkan's paper [5] for details of the algorithm.

Since the EM algorithm can get stuck at local optima which are far from the global optimum, it is customary to use several randomized restarts of the algorithm. Each restart uses a different random initialization, and is allowed to run to convergence. Among the different models learned, we choose the one having the best likelihood.

3.3 Choosing simulation points

After performing clustering, we must select a representative (simulation point) from each cluster to simulate in detail. We examine several different methods of choosing simulation points from our final clusterings. Let the notation s_j denote the interval chosen as the simulation point for cluster c_j.

Mode (highest probability) Because the EDCM is a probability distribution, each example x_i is assigned a probability $Pr(x_i|c_j)$ by cluster c_j. Choose a representative for cluster c_j by finding the example x_i with the highest probability in that cluster: $s_j = \arg\max_{x_i \in c_j} Pr(x_i|\beta_j)$.

Smallest L_p distance Compute the centroid, $\bar{x}_j$, of each cluster c_j as the mean of the examples within it: $\bar{x}_j = \frac{1}{|c_j|} \sum_{x_i \in c_j} x_i$. Choose x_i with the smallest L_p distance to $\bar{x}_j$: $s_j = \arg\min_{x_i \in c_j} \left(\sum_{d=1}^{D} |x_{id} - \bar{x}_{jd}|^p \right)^{1/p}$. For $p = 2$, this is called the straight-line (or Euclidean) distance; for $p = 1$, this is the called the Manhattan distance.

4 Experimental evaluation

We use EDCM mixtures to cluster the SPEC CPU2000 benchmark suite. Each benchmark's execution has been split into fixed-length intervals of 100 million instructions each. We rely on the work of others—each of these benchmarks has been simulated in its entirety [1] on the SimpleScalar simulator [7]—for the true metrics of performance on a per-interval basis. The simulated CPU uses the Alpha Instruction Set Architecture, and each binary is compiled with full optimizations. For full details on the simulated CPU, see [10].

To evaluate the accuracy of our approach, we compare our method's predicted performance in terms of weighted average CPI (cycles per instruction) with the true average CPI as obtained from full simulation. We compare the prediction performance of the EDCM, implemented in MATLAB, with the current version of k-means implemented in SimPoint. We choose simulation points for EDCM using the L_1 distance to cluster means. SimPoint selects simulation points using L_2 distance to the cluster means.

The underlying assumption behind SimPoint is that basic block vectors with similar behavior will accordingly exhibit similar performance metrics (e.g. CPI). An important point is that we do not use the CPI metrics for clustering; we only use the basic block vectors.

4.1 Prediction accuracy and model complexity

Our first question is whether or not EDCM mixture models find clusterings that are suitable for predicting simulation metrics. We compare EDCM mixtures to SimPoint (using k-means) for different fixed simulation budgets. We run each method with different numbers of clusters (values of k), and each value of k represents a simulation budget for which we compare performance prediction of the two methods.

A primary concern in processor design is the amount of time spent in detailed simulation, which is directly proportional to the number of clusters (and therefore number of selected simulation points) a program's execution is broken into, and the size of each simulation point. This leads the practitioner to prefer fewer simulation points. Therefore, we examine the results of clustering and prediction using EDCM mixture models and k-means for values of k in the range of 2 to 15. For each value of k, we cluster all 36 program/input pairs in the SPEC CPU2000 suite of benchmarks, generate k simulation points, and calculate the predicted average CPI of the program/input pair using these selected intervals of program execution.

We make several points of comparison between the prediction errors of EDCM mixture models and k-means. The first comparison is the average amount of prediction error made per value of k, where the average is taken across the 36 different program/input pairs in the benchmark suite. The bars in Figure 2 show that for smaller values of k, the EDCM provides better accuracy at predicting average CPI values. Above a certain value of k (about $k \geq 8$), the two methods both provide less than 2% prediction error, on average. Figure 2 also

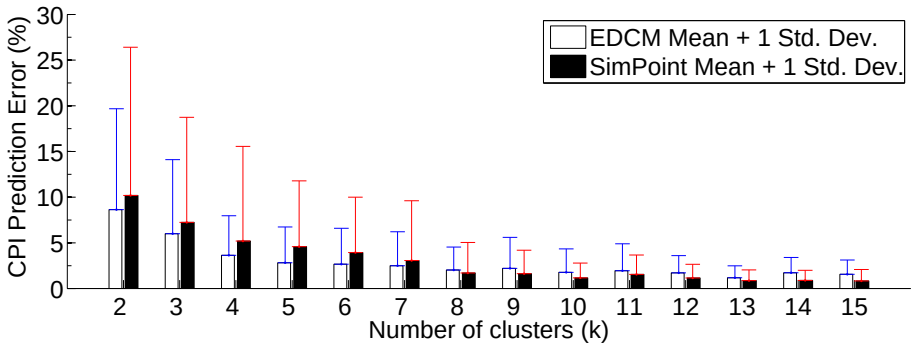

Fig. 2. This plot shows the mean CPI prediction error, plus one standard deviation (STD), of EDCM and k-means clustering for k=2 to 15 clusters. Each bar represents the average and standard deviation of the errors across all 36 SPEC CPU2000 benchmarks. It's clear that for small simulation budgets (i.e. small k), EDCM clustering gives better prediction performance (lower prediction error). It also gives more consistent prediction performance, with a smaller standard deviation for the smaller values for k.

shows the standard deviation, as errorbars, of the predicted CPI values across all program/input pairs per value of k. Again, we see that for smaller values of k, EDCM mixture models provide predictions that have smaller standard deviations than k-means. This is significant because it means that the EDCM is doing a more consistent job at making accurate predictions.

Figure 3 compares the maximum prediction error per value of k for both clustering methods. This graph shows that when EDCM mixture models do perform poorly (on particularly difficult program/input pairs where small values of k may not be enough to capture the range of program behaviors), they still outperform k-means clustering. In fact, we see that as k increases, the EDCM's maximum CPI prediction error drops much more quickly than does the same measure for k-means. Once both methods have enough clusters (k is large enough) to capture subtle differences in program behavior, the maximum error for both is consistently low.

Thus we see that the EDCM mixture is consistently better at clustering program phase behavior and predicting CPI performance for small simulation budgets (small values of k). For large simulation budgets, most methods of selecting simulation regions will work well, because a larger sample affords more opportunities for the sample to represent the program as a whole. However, when simulation budgets are limited and small, we still want to obtain the most accurate predictions possible, and this is where the EDCM mixture is the clear winner.

There are several reasons to expect that the EDCM mixture is a better model for program behavior for small samples. Frist, the SimPoint method with k-means operates on a reduced-dimension version of the original frequency vector

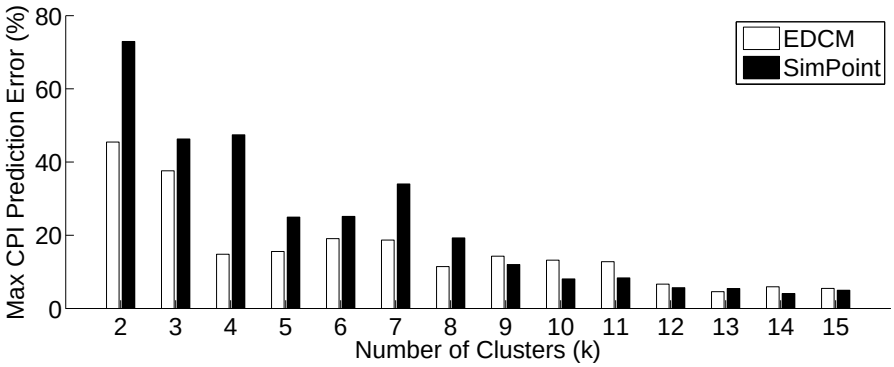

Fig. 3. This plot shows the *maximum* CPI prediction error of EDCM and k-means clustering for k=2 to 15 clusters. Each bar represents the maximum CPI prediction error across all 36 SPEC CPU2000 benchmarks. For small k, EDCM gives lower maximum errors, making its results more useful than k-means for evaluating performance across all benchmarks.

data. Dimension reduction can cause some clusters which were naturally far apart to overlap when projected. While this will happen infrequently, the EDCM avoids this problem altogether, since it operates on the original data.

Second, the EDCM mixture incorporates a prior probability for each cluster, meaning that some clusters can be larger and others smaller (with respect to the percentage of the number of frequency vectors in each cluster). The k-means algorithm uses uniform prior probabilities (which are implicit in the algorithm), meaning that k-means clusters tend to have the same size. This difference allows EDCM to be more flexible than k-means at fitting the naturally occurring clusters when those clusters have varying sizes. Figure 4 shows how the cluster sizes vary for both clustering methods. We report this as the standard deviation of cluster sizes averaged over all benchmarks for k from 2 to 15. Cluster sizes are measured as percentages of the total execution, so if three clusters contain 500, 300, and 200 frequency vectors, then their sizes are reported as 0.5, 0.3, and 0.2, respectively. We can see that EDCM has consistently higher standard deviations, indicating that it is finding clusters whose sizes vary more than the clusters found by k-means, whose clusters tend to be more uniformly sized.

Third, the basic model of a cluster in the k-means algorithm is a spherical Gaussian, where the variance of the Gaussian is the same across all clusters (these attributes are also implicit in the algorithm). While the spherical Gaussian is a reasonable model that is widely used for clustering, it is not as flexible or appropriate as the EDCM which explicitly models the counts in frequency vectors.

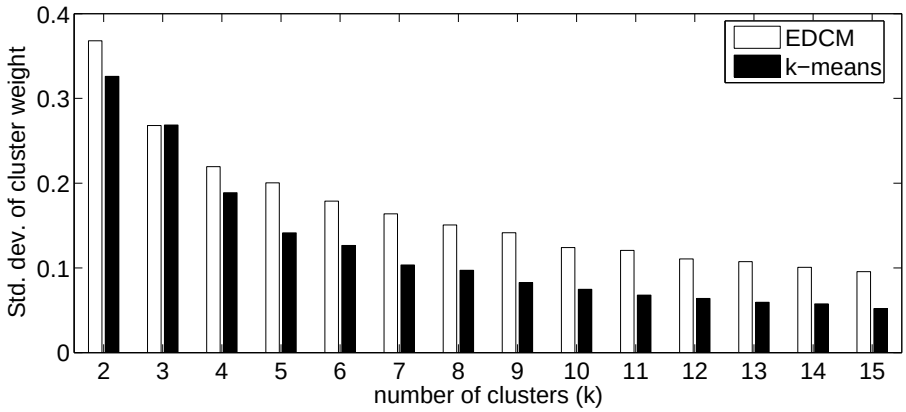

Fig. 4. This plot shows the standard deviation of the size of each cluster for EDCM and k-means. Higher bars indicate more variation in cluster size, while lower bars indicate more uniform cluster sizes. The EDCM mixture, which explicitly incorporates a prior probability for each cluster, can recognize clusters of more varying sizes. Each bar represents the standard deviation over all benchmarks for each k. Each cluster size is measured as the number of vectors in the cluster divided by the number of vectors in the whole benchmark.

4.2 Choosing representatives

Once we have a clustering of basic block vectors (i.e. program execution intervals) with an EDCM mixture, we need to then select a single interval as the representative, or simulation point, for that cluster. SimPoint uses the interval whose basic block vector is closest (in Euclidean, or L_2 distance) to the k-means center, which is the geometric mean of the vectors in the cluster.

We investigate several different methods for choosing cluster representatives from the clusterings induced by the EDCM mixture model. Using the methods outlined in Section 3.3, we found that using the L_1 (Manhattan) distance to choose simulation points resulted in prediction with the lowest percent error.

It's not surprising that L_1 distance outperforms L_2 distance. BBVs have a large number of dimensions, and others have shown that in high dimension, for the L_p distance metric, lower values for p give more meaningful results than higher p [21].

We initially suspected that using probabilities for selecting simulation points would yield good results, but it did not. We discovered that certain basic blocks exhibit very bursty behavior, such as a single basic block occurring up to 10^7 times in one interval. The EDCM does not penalize bursty behavior as much as (e.g.) the multinomial. So while the EDCM can represent bursty data well, this causes problems when asking for the most probable vector from the distribution.

4.3 Time and space complexity

The time and space complexity of the EM algorithm for training an EDCM mixture is linear in the number of clusters, the number of basic block vectors, and the dimension of the data. All these are comparable to the linearity of k-means for the same attributes. Both algorithms will also depend on the number of required iterations (but a reasonable average-case bound on this is an open question in machine learning [22]).

A direct comparison of the running times of EDCM and k-means on this application should consider the amount of raw data that each is processing. Since it does not reduce dimension, the EDCM mixture is operating on approximately 1,500 times more data than does k-means with its projected datasets. This figure is adjusting for the fact that some basic blocks never execute and are pruned by our EDCM implementation.

In our experiments, running the EDCM to cluster the 36 benchmarks in the SPEC CPU2000 suite, with 5 random restarts per k ($2 \leq k \leq 15$) takes about 50 hours on a 3 GHz Intel Pentium 4 computer with 2 GiB of RAM. Running k-means with the same configuration takes 771 seconds. Both approaches use single-threaded (non-parallel) code, but clustering separate benchmarks is easily parallelized by clustering one benchmark per available processor.

There are several reasons for the difference in running time. First, SimPoint is highly optimized for the application at hand, and is written in C++, while our EDCM mixture is written in MATLAB and is currently not optimized (we plan to provide an optimized version in future work). Second, as we mentioned, EDCM is actually processing 1,500 times more data than k-means. When viewed in this light, the EDCM mixture is actually quite fast. Finally, it's not necessary to cluster for all values of k if the desired k is known in advance. This will cut clustering time dramatically.

We emphasize that architecture researchers are willing to spend this time clustering benchmarks, since the later simulations will usually take a much larger proportion of time. For example, a small architectural exploration of just 30 different configurations would take 90 days with the `sim-outorder` simulator from SimpleScalar. This is assuming 500 million instructions per hour, with 1 billion instructions simulated for each of the 36 SPEC CPU2000 benchmarks. Compared with this, the amount of work done in choosing simulation points is relatively tiny.

5 Conclusions and future work

The purpose of SimPoint is to cluster program traces in order to choose representative intervals (simulation points). Detailed simulations are expensive, so researchers need methods that reduce the amount of a benchmark they have to simulate in order to get accurate ideas of overall performance statistics. SimPoint does a good job of clustering benchmark data with the k-means algorithm, and choosing simulation points. These simulation points are run through new

hardware designs simulated in software, offering accurate predictions about performance without requiring the entire benchmark to be fully simulated.

We proposed using EDCM mixture models to cluster the same data as k-means, and we found that EDCM mixtures provide very accurate prediction performance. For fixed simulation budgets, a researcher will want to spend as little time as possible in simulation, translating into a desire for fewer simulation points and smaller values of k. For small values of k ($2 \leq k \leq 8$, which equates to 200 to 800 million detailed instructions simulated), we find that the EDCM mixture model outperforms k-means. We see that the average error, variance in the errors of the predictions, and the maximum prediction error using the EDCM are all lower on the SPEC CPU2000 suite of benchmarks than using k-means. For larger values of k ($k > 10$), the two methods both predict performance with around 1% error, on average.

The EDCM mixture offers more than empirical reasons why it is a favorable clustering model. First, it is able to use the entire dimension of the data. In practice, k-means requires dimension reduction before it can begin clustering, which may collapse true clusters of behavior, causing information about the data to be lost to k-means. Second, the EDCM mixture is able to handle bursty data well. Third, the EDCM mixture is a probability density function that assigns probabilities to each example, giving nice statistical interpretations to our data that k-means does not offer.

Future work includes implementing an optimized version of the EDCM mixture algorithm implemented in the SimPoint software, so that others may use it. We also want to further analyze the phenomenon of burstiness in program traces, examine the differences between the clusterings generated by SimPoint and the EDCM mixture, and leverage the probabilities assigned to each example by the EDCM mixture. The current work uses BBVs of a fixed length. It would be interesting to examine the dynamics of clustering intervals with variable lengths.

References

1. Sherwood, T., Perelman, E., Hamerly, G., Calder, B.: Automatically characterizing large scale program behavior. In: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems. (October 2002) 45–57
2. Sanghai, K., Su, T., Dy, J.G., Kaeli, D.R.: A multinomial clustering model for fast simulation of computer architecture designs. In: Knowledge Discovery and Data Mining (KDD). (2005) 808–813
3. Hamerly, G., Perelman, E., Calder, B.: Comparing multinomial and k-means clustering for SimPoint. In: Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS). (2006) 131–142
4. Madsen, R., Kauchak, D., Elkan, C.: Modeling word burstiness using the Dirichlet distribution. In: International Conference on Machine Learning. (2005) 289–296
5. Elkan, C.: Clustering documents with an exponential-family approximation of the Dirichlet compound multinomial distribution. In: International Conference on Machine Learning. (2006) 545–552

6. Wunderlich, R., Wenisch, T., Falsafi, B., Hoe, J.: SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In: Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA). (June 2003) 84–95
7. Burger, D.C., Austin, T.M.: The SimpleScalar tool set, version 2.0. Technical Report CS-TR-97-1342, U. of Wisconsin, Madison (June 1997)
8. Sherwood, T., Perelman, E., Calder, B.: Basic block distribution analysis to find periodic behavior and simulation points in applications. In: International Conference on Parallel Architectures and Compilation Techniques. (September 2001)
9. Dasgupta, S., Gupta, A.: An elementary proof of the Johnson-Lindenstrauss lemma. Technical Report TR-99-006, Berkeley, CA (1999)
10. Hamerly, G., Perelman, E., Lau, J., Calder, B., Sherwood, T.: Using machine learning to guide architecture simulation. Journal of Machine Learning Research **7** (February 2006) 343–378
11. Denning, P., Schwartz, S.C.: Properties of the working-set model. Communications of the ACM **15**(3) (March 1972) 191–198
12. Balasubramonian, R., Albonesi, D.H., Buyuktosunoglu, A., Dwarkadas, S.: Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In: 33rd International Symposium on Microarchitecture. (2000) 245–257
13. Dhodapkar, A., Smith, J.: Comparing program phase detection techniques. In: 36th International Symposium on Microarchitecture. (December 2003) 217–227
14. Dhodapkar, A., Smith, J.E.: Dynamic microarchitecture adaptation via co-designed virtual machines. In: International Solid State Circuits Conference. (February 2002) 198–199
15. Dhodapkar, A., Smith, J.E.: Managing multi-configuration hardware via dynamic working set analysis. In: 29th Annual International Symposium on Computer Architecture. (May 2002) 233–244
16. Gracia Pérez, D., Berry, H., Temam, O.: Iddca: A new clustering approach for sampling. In: MoBS: Workshop on Modeling, Benchmarking, and Simulation MoBS: Workshop on Modeling, Benchmarking, and Simulation, Madison, Wisconsin (2005)
17. Feng, Y., Hamerly, G.: PG-means: learning the number of clusters in data. In Schölkopf, B., Platt, J., Hoffman, T., eds.: Advances in Neural Information Processing Systems 19, Cambridge, MA, MIT Press (2007) 393–400
18. Minka, T.: Estimating a Dirichlet distribution. `http://research.microsoft.com/~minka/papers/dirichlet/` (2003)
19. Church, K.W., Gale, W.A.: Poisson mixtures. Natural Language Engineering (1) (1995) 163–190
20. Katz, S.M.: Distrubution of content words and phrases in text and language modelling. Natural Language Engineering (2) (1996) 15–59
21. Aggarwal, C.C., Hinneburg, A., Keim, D.A.: On the surprising behavior of distance metrics in high dimensional space. In: Proc. Eighth International Conference of Database Theory, London (2001) 420–434
22. Arthur, D., Vassilvitskii, S.: How slow is the k-means method? In: SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry, New York, NY, USA, ACM (2006) 144–153