Aligning Traces for Performance Evaluation

Todd Mytkowicz[*]
University of Colorado at Boulder
Todd.Mytkowicz@colorado.edu

Amer Diwan
University of Colorado at Boulder
diwan@cs.colorado.edu

Matthias Hauswirth
University of Lugano, Switzerland
Matthias.Hauswirth@unisi.ch

Peter F. Sweeney
IBM Thomas J. Watson Research Center
pfs@us.ibm.com

ABSTRACT

For many performance analysis problems, the ability to reason across traces is invaluable. However, due to non-determinism in the OS and virtual machines, even two identical runs of an application yield slightly different traces. For example, it is unlikely that two identical runs of an application will suffer context switches at exactly the same points. These sorts of variations across traces make it difficult to reason across traces. This paper describes and evaluates an algorithm, Dynamic Time Warping (DTW), that can be used to align traces, thus enabling us to reason across traces. While DTW comes from prior work our use of DTW is novel. Also we describe and evaluate an enhancement to DTW that significantly improves the quality of its alignments.

Our results show that for applications whose performance varies significantly over time, DTW does a great job at aligning the traces. For applications whose performance stays largely constant for significant periods of time, the original DTW does not perform well; however, our enhanced DTW performs much better.

1. INTRODUCTION

Modern systems are composed from many interacting subsystems and exhibit complex time varying behavior. For example, Java applications make use of garbage collectors, compilers, and extensive libraries and the application performance (e.g., instructions-per-cycle) is not constant throughout the run [4]. Because performance changes over time, it is not enough to use aggregate metrics (e.g., execution time) for reasoning about performance; instead we need to use traces that contain application performance over time.

[*]This work is supported by NSF CSE-0509521, NSF Career CCR-0133457, DARPA contract NBCH30390004, and a gift from Intel. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsors.

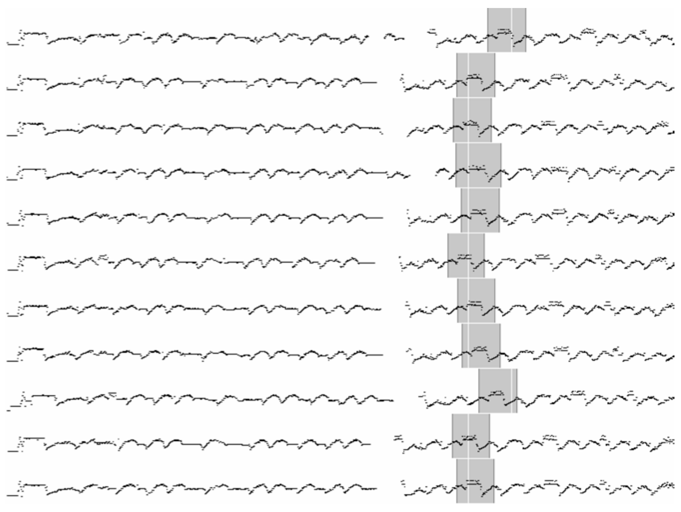

Figure 1: IPC from 10 identical runs.

Moreover, since modern systems contain many interacting subsystems, we need to collect traces that contain performance metrics from all the subsystems. Thus, we may need to collect hundreds of metrics to understand a system's performance. Collecting all of these metrics in a single run can be infeasible because i) it significantly perturbs program behavior, ii) not all of the metrics can be collected at the same time due to system constraints (e.g. hardware performance monitors have only a few registers to count events, but can count many more events), and iii) performance analysis is an interactive process where traces may be collected over time.

However, reasoning across traces is non-trivial: even two runs of a deterministic program using the same input and conditions yield slightly different behavior due to non-determinism in the OS and Java virtual machine. For example, Figure 1 shows the IPC (instructions-per-cycle) over time from the start of ten "identical" runs of an application. The runs are identical in that they all use the same inputs and run on the same lightly-loaded machine. We see that even though the general shape of the curves are the same, they are not perfectly aligned.

This paper explores a novel use of a technique, Dynamic Time Warping (DTW), from the speech recognition literature [2]. In prior work DTW has been used to determine the similarity between two time series; we use it for aligning traces (Section 2). We discuss the intuition for when DTW performs well and when it does not and use that intuition to improve DTW.

Our results show that DTW alignment performs well when the application generating the traces exhibits significant variations in performance over time. By "well" we mean that DTW's alignment is usually within one position of a perfect alignment. However, when the application's performance is relatively constant over time, DTW performs much worse. Our enhanced DTW performs well for all of our experiments, significantly outperforming the original DTW when the application performance is largely constant.

Section 2 describes DTW and Section 3 discusses why it can perform well. Section 4 presents our methodology for evaluating DTW. Section 5 presents experimental results on both the original and extended DTW and Section 6 discusses the implications of our results. Finally, Section 7 discusses related work and Section 8 concludes.

2. WHAT IS DTW

We use a technique from the speech recognition literature, dynamic time warping (DTW) [2], to align the traces. Given two sequences, X and Y, of lengths $|X|$ and $|Y|$,

$$X = (x_1, x_2, \ldots, x_i, \ldots, x_{|X|})$$
$$Y = (y_1, y_2, \ldots, y_j, \ldots, y_{|Y|})$$

dynamic time warping constructs a warp path W (w_1, w_2, $\cdots$, $w_{|W|}$) such that each w_k is a pair (x_i, y_j). The warp path satisfies the following constraints:

1. For every element x_i of X, there is at least one element in W that is $(x_i, *)$, and for every element y_j of Y, there is at least one element in W that is $(*, y_j)$, i.e., no element of X or Y is omitted
2. $w_1 = (x_1, y_1)$ and $w_{|W|} = (x_{|X|}, y_{|Y|})$, i.e., the end points of the two sequences are aligned;
3. $w_k \lhd w_{k+1}$, where $w_k = (x_i, y_j)$ and $w_{k+1} = (x_m, y_n)$ if ($i = m$ or $i + 1 = m$) and ($j = n$ or $j + 1 = n$), but not $i = m$ and $j = n$; i.e., the warp path respects the order of both sequences and each w_k consumes at least one element from one of the sequences.

DTW uses a dynamic programming algorithm to construct the warp path. The algorithm minimizes the *DTWError*:

$$\text{DTWError} = \sum_{k=1}^{|W|} |x_i - y_j| \text{ where } w_k = (x_i, y_j) \qquad (1)$$

Figure 2 shows an example of a warp path. The elements of the X sequence are the columns of the matrix and the elements of the Y sequence are the rows. The shaded squares represent the warp path. More specifically, if cell (x_i, y_j) is shaded it means that there is a k for which $w_k = (x_i, y_j)$.

Figure 3 shows the alignment implied by the warp path in Figure 2. If (x_i, y_j) is on the warp path, then x_i is aligned to y_j. The lines that go between the two sequences in Figure 3 represent the alignment. If the two sequences are identical (and thus do not need any alignment), all these lines would be vertical. Note that there are situations where a single element of X aligns with multiple elements of Y and also where a single element of Y aligns with multiple elements of X. In Figure 2, such many-to-one situations appear as vertical or horizontal sequence of shaded squares. For example in Figure 3, the four circled X elements are mapped to one circled Y element and this is represented in Figure 2 as a horizontal sequence of four shaded squares.

While DTW itself is not our invention, our use of it is novel. Prior work uses only DTWError to determine the similarity between two sequences. We do not use the DTWError but instead the warp path to align traces.

3. INTUITION BEHIND DTW

DTW is a relatively simple algorithm and our use of it is not the one for which it was designed. Yet, we have found it to be surprisingly difficult to manually construct scenarios where DTW yields a poor alignment. This section intuitively explores the strengths and weaknesses of DTW; Section 5 explores these strengths and weaknesses experimentally. These strengths and weaknesses are direct or indirect consequences of the three constraints on the warp path (Section 2).

3.1 Weaknesses of DTW

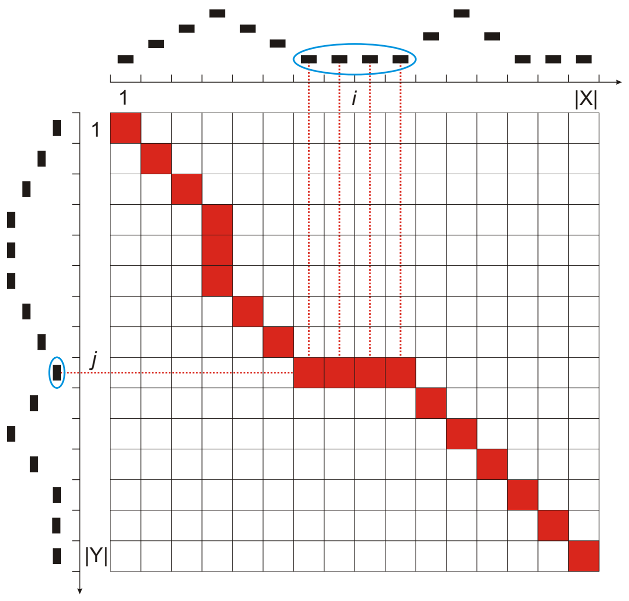

Figure 2: Dynamic Time Warping Warp Path

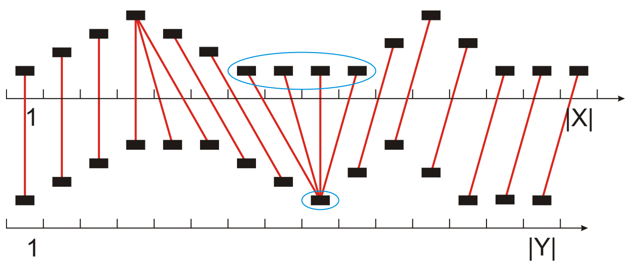

Figure 3: Warping Between Sequences X and Y

- *The warp path respects the order of both sequences.* The negative consequence of this is that DTW cannot align sequences where one sequence is reordered with respect to the other. For example, consider two runs of a Java application on a modern JVM that optimizes methods when they become hot (determined using sampling). Because sampling is non-deterministic, it may happen that in one run method A is optimized before method B and in the other method B is optimized before method A. Even worse, it may happen that the first run performs a garbage collection before compiling method A and the second run performs the corresponding garbage collection after compiling method A.

  It may seem that a simple solution to this is to enable DTW to reorder the sequences. This change, however, may degrade the other (positive) properties of DTW, as discussed below.

- *DTW allows many-to-one and one-to-many alignments.* DTW allows such alignments to support situations when a segment of one sequence is "slower" than the corresponding segment of another sequence. For example, during periods of OS activity an application may run slower with respect to another run which may encounter OS activity at different points. Even though these mappings are useful, they may also weaken DTW. Consider the task of aligning two sequences that are nearly flat during a segment that is n wide. Because both the segments are the same width, the best alignment may be a one-to-one alignment between corresponding points of the two segments. However, as an extreme case DTW may align the first point of the first segment with the first n-1 points of the second segment and the nth point of the second segment with points 2 to n of the first segment. This scenario happen when the two segments are flat and contain noise. Because of nondeterminism in the underlying system, noise is introduced into a trace, generating nearly flat segments instead of flat segments. DTW latches onto the noisy points for alignment; however, because the noise is nondeterministic DTW may find an extreme alignment.

3.2 Strength of DTW

The three constraints on DTW's warp path (presented in Section 2) collaborate to prevent DTW from performing too poorly. The reason for this is that these constraints allow DTW to make a bad alignment choice only if

- there are later points in the two sequences that enable DTW to recover from the error; this recovery must happen because the end points are aligned and DTW is not allowed to skip any points or construct an artificial recovery point by reordering events.

- the above-mentioned recovery has to be cheap otherwise DTW will reject that warp path in favor of some other cheaper path.

For example consider aligning the sequences $X = (1, 2, 3, 4)$ and $Y = (1, 3, 3, 4)$ where the second point in the two sequences is noisy; i.e., $x_2 = 2$ and $y_2 = 3$. One possible warp path that DTW may compute is (x_1,y_1), (x_2,y_1), (x_3,y_2), (x_3,y_3), (x_4,y_4); i.e., both the first and second point of the first sequence align with the first point of the second sequence. While this is undesirable, DTW can get away with it because DTW can recover relatively cheaply by doing a 1 to many mapping later on (i.e., (x_3,y_2) and (x_3,y_3)).

On the other hand, consider aligning the sequences $X = (2, 3, 6, 8)$ and $Y = (2, 4, 6, 8)$ where again the second point in the sequences is noisy; i.e. $x_2 = 3$ and $y_2 = 4$. In this case, DTW resists aligning using (x_2,y_1) (i.e., aligning the second point of the first sequence with the first point of the second sequence), because the recovery is relatively expensive: to recover from this DTW will have to align the second and third point of the second sequence with the third point of the second sequence which adds 2 to the DTWError.

4. METHODOLOGY

To evaluate DTW's performance, we instrumented three applications (db, raytrace, and jess, all from the SPECjvm98 benchmark suite [6]) to insert application-level events into the trace. For example, every time db deletes a record we insert a delete-record event in the trace. The application-level events that we picked are all deterministic. More specifically, the nth delete-record in one trace should align exactly with the nth delete-record in another trace that is generated using the same inputs and parameters. We call these markers "milestones".

We evaluate DTW by aligning two traces using instructions-per-cycle and then evaluating the extent to which the alignment lines up the corresponding milestones in the two traces. For each element of the warp path w_k=(x_i, y_j) such that x_i contains a milestone we compute a score which is a number greater or equal to zero. A score of zero indicates that the DTW alignment is correct; i.e., the milestone in x_i also occurs in y_j. A score, n, where $n \geq 1$ means that the milestone in x_i does not occur in y_j but occurs n positions away in the second trace (forward or backward). The bottom line is that a smaller score indicates better alignment.

Our actual evaluation is a little more complicated than that described above. In order to keep the sizes of our traces manageable and to reduce the perturbation on application performance due to trace collection overhead, we do not generate a trace record on every event (e.g., instruction executed). Instead we generate a trace record roughly every 10 milliseconds; this trace record aggregates all the events that occurred during the time interval. Thus, each trace record represents an interval rather than a single point in time. For example, a trace record will give the number of instructions completed during the interval, the number of cycles elapsed during the interval, and the number of milestones encountered during the interval.

Because an interval may encounter more than one milestone and we do not know precisely when within the interval the milestone occurs, we cannot determine if DTW aligns the nth milestone in one trace exactly with the nth milestone in the other trace. However, we can determine if the DTW-aligned intervals in the two traces have milestones in common; if they do we consider that a perfect alignment.

For example, let's again consider an element of the warp path w_k=(x_i, y_j) and assume that x_i contains milestones 2 and 3 while y_j contains milestones 3 and 4. In this case, we will consider it a perfect alignment because the aligned intervals x_i and y_j have a milestone (3) in common. If instead y_j contained only milestone 4, we would not consider it a

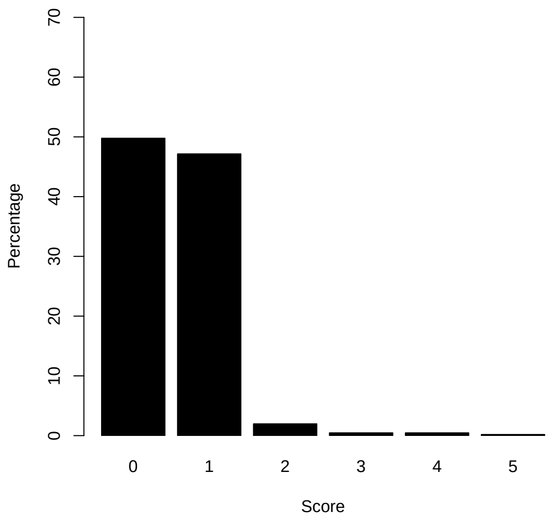

Figure 4: Alignment evaluation for db using SortSwaps.

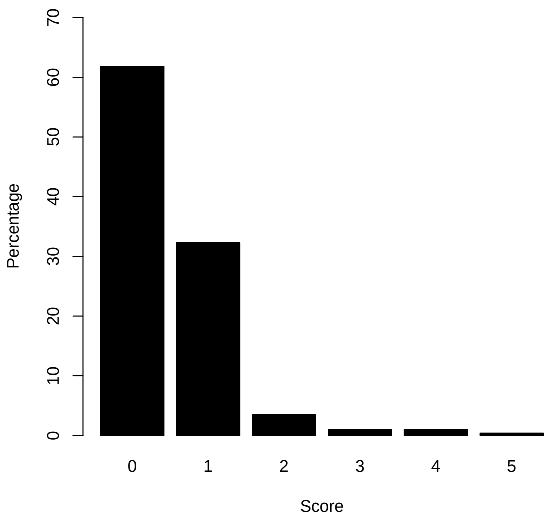

Figure 5: Alignment evaluation for jess using ActivationsFired.

perfect alignment. This example illustrates that the granulation of our trace introduces a granulation in our evaluation also.

Table 1 lists the benchmarks and the milestones for the benchmarks. The "# milestones" column gives the number of milestones in each trace; note that since the milestones are deterministic, both traces have exactly the same number of milestones. The "# intervals" and "% intervals with milestones" colums give pairs of numbers where the first number in the pair is for the first trace and the second number for the second trace. The "# intervals" columns gives the number of intervals in the two traces. The "% intervals with milestones" gives the percentage of intervals that contain a milestone. Note that to improve our confidence in our results, for each benchmark we experiment with two different metrics for milestones.

From the "# Intervals" column we see that the two traces being aligned always have a different number of intervals; in other words, the alignment is not trivial. We also see that the number of milestones and the percentage of intervals with milestones varies significantly across benchmarks and milestones. Our results, however, are consistent across milestones (Section 5): i.e., they do not change significantly when using one milestone over another for a given benchmark.

5. RESULTS

Sections 5.1 and 5.2 present the data for the original DTW algorithm. Section 5.3 presents the data for DTW with an extension of our design.

5.1 When DTW performs well

Figure 4 presents the score distribution when we use DTW to align two traces from the db benchmark and use the SortSwaps milestone to evaluate the alignment. Figure 5 presents similar data for jess when using the ActivationsFired milestone. The two traces that are aligned for each graph use the same program inputs and parameters but collect different performance metrics. Both traces contain the IPC metric which we use for the alignment. Given two traces and a warp path w, the height of a bar gives the percentage of warp path elements $((x_i, y_j))$ that are aligned with the score given on the x-axis label. For example, the 0 bar gives the percentage of warp path elements that DTW aligns correctly (with respect to the milestone) and the 1 bar gives the percentage of warp path elements on which DTW is off by one interval.

From Figures 4 and 5 we see that DTW performs well for db and jess: more than half of the warp path elements are aligned perfectly and most of the intervals are aligned so that they are off by no more than one position.

Since our evaluation compares DTW's alignment to the deterministic alignment provided by the milestones, we wanted to determine if DTW was indeed performing well or if it just appeared to perform well due to a poor choice of a milestone. Thus, we repeated our experiments with the second milestone for db and jess (Table 1). Figure 6 shows the alignment scores for db using the two milestones. The first bar in each pair of bars is the same as the bars in Figure 4 and the second bar gives the score when using the SortSwaps milestone. We see that while there are differences between the alignment scores with the two milestones, the basic trends are the same: most alignments are off by one or less. We omit a similar graph for jess since it shows similar trends.

5.2 When DTW performs poorly

Figure 7 presents the alignment's score distribution for the raytrace benchmark when using the SphereNormalsFound milestone. In contrast to DTW's performance for jess and db, DTW performs extremely poorly for raytrace: DTW's alignment is often off by more than 60 positions. To investigate this poor alignment further, we aligned a different pair of traces with DTW and computed the scores: we found

Benchmark	Milestone Event	# Milestones	# Intervals	% Intervals with milestones
db	DeleteRecord	584	(3692, 3686)	(8.0, 8.1)
	SortSwaps	21,548,358	(3692, 3686)	(82.8, 82.7)
jess	ActiviationsFired	360	(2078, 2082)	(23.1, 23.3)
	InMemoryDataComparisons	5,258,864,926	(2078, 2082)	(58.9, 59.0)
raytrace	SphereNormalsFound	106,338	(2424, 2416)	(24.8, 24.5)
	PolyTypeNormalsFound	542,182	(2424, 2416)	(71.9, 72.6)

Table 1: Benchmarks

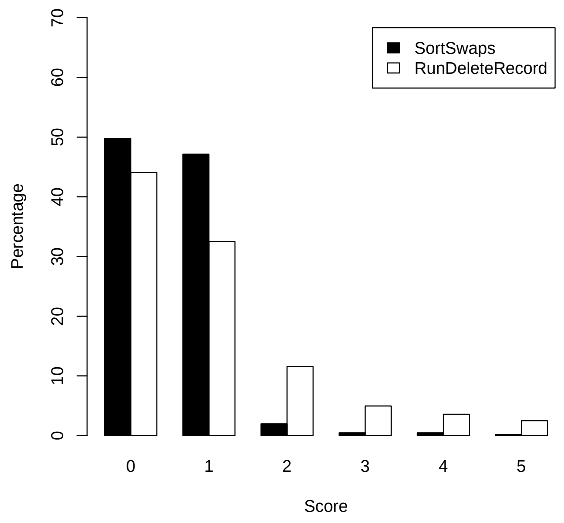

Figure 6: Alignment evaluation for db when using different milestones.

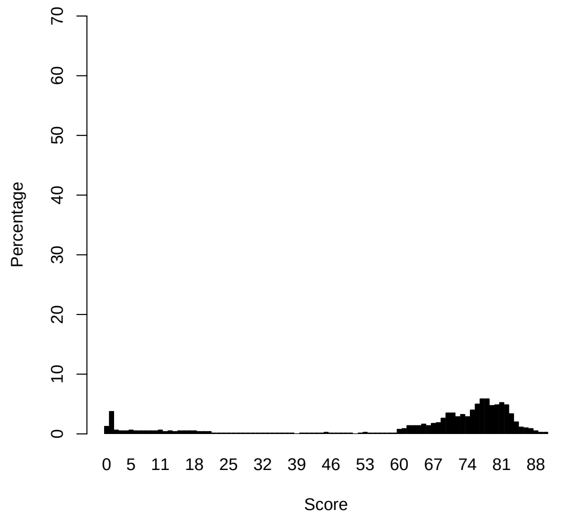

Figure 7: Alignment evaluation for raytrace using SphereNormalsFound.

that DTW performed much better (but still worse than on db and jess). In other words DTW's performance for raytrace is rather finicky: it can perform very poorly if we end up with an unfortunate pair of traces. Note that we did not see this instability with the other benchmarks (i.e., DTW's performance for db and jess was the same for both pairs of traces we tried).

As discussed in Section 3, DTW makes a poor alignment only if it will be able to recover from it subsequently (Section 3). The reason for this is that DTW always aligns the end points of the traces (Section 2). If a trace is long and the metric used for alignment

- *has sharp frequent transitions* then DTW will most likely align these transitions and end up with a good overall alignment.
- *is relatively flat or has significant relatively flat segments* then there is greater potential for DTW to perform poorly. This is because flat areas do not have any transitions onto which DTW can latch onto. Unfortunately, in practice segments are not completely flat but have slight variations. DTW latches on to these variations resulting in poor alignments.

Figure 8 graphs the IPC over time for db, jess and raytrace. The jess and db traces illustrate that there are frequent sharp transitions and thus DTW performs well and creates good alignments for these two benchmarks. In contrast, the raytrace trace illustrates relatively flat segments and thus DTW performs poorly and creates a poor alignment.

To get further insight for why DTW performs poorly for raytrace, Figure 9 shows DTW's alignment for a segment of raytrace's run. The two graphs in Figure 9 give the IPC (instructions-per-cycle) from the two traces for raytrace and the lines that go between the two graphs (i.e., from top to bottom) give DTW's alignment. We omit the alignment lines for intervals that do not contain a milestone, because these intervals do not feature in the score computation. For each interval, we draw one hollow circle each time the interval participates in an alignment and the diameter of the circle gives the score of that alignment. For example consider the first alignment: its circle is tiny so it appears as a point, indicating a score of zero. Now consider the first situation where a single point in the upper graph is aligned with many points in the lower graph. Because the point in the upper graph participates in many alignments, we draw many concentric circles for it, ranging from zero error onwards; thus we see a solid circle even though what we really have is 12 concentric circles.

Note that the IPC values (the vertical position of the points) are flat with minor variations. Thus even though a good alignment would be one-to-one mappings between

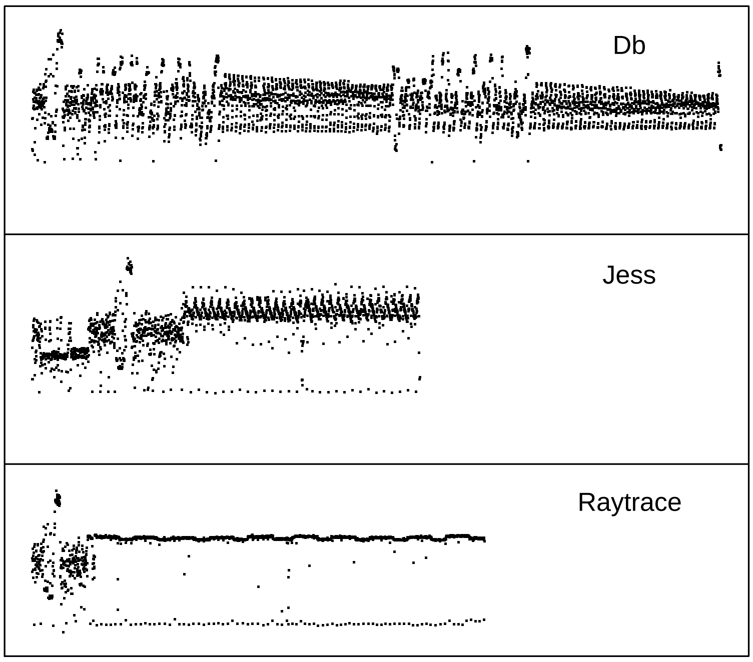

Figure 8: The IPC for all three benchmarks.

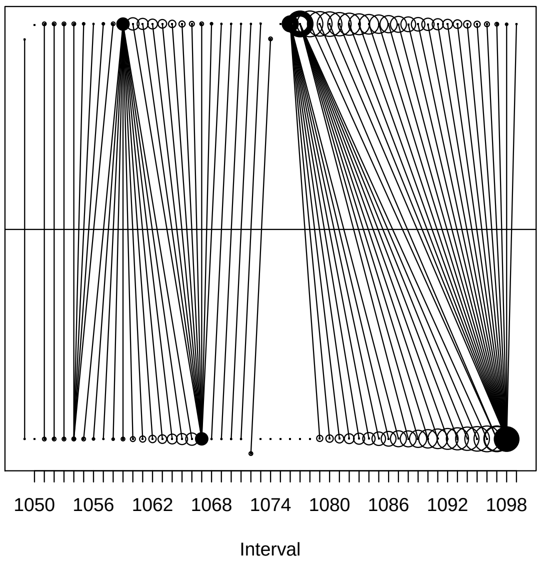

Figure 9: The alignment of a subsequence of the IPC signal for raytrace.

the points in the two subsequences (i.e., vertical lines for the alignment) we get many-to-one and one-to-many mappings. These mappings result in the poor alignment scores. This graph illustrates that the minor variations in the flat sequence cause DTW to get confused and perform poorly.

5.3 Results for Extended DTW

Based on the insights in the previous section, one possible way to improve DTW's performance is to split the trace into a number of segments using milestones as anchors and to use DTW to align each segment in the first trace with its corresponding segment in the second trace. Because DTW is now aligning shorter traces it will have less opportunity to make a bad alignment. We call this modified DTW "ADTW" for Anchored DTW.

Figure 10 illustrates what happens when we use ADTW with 32 (roughly) evenly distributed anchors. To improve legibility of the graph we truncated it at a score of 19. The black bars ("Full") give the score distribution for DTW and are the same as the first 20 bars from Figure 7. The white bars ("Split(32)") give the score distribution for ADTW with 32 anchors. ADTW performs dramatically better than DTW: 45.9% of the milestones align perfectly with ADTW compared to 1.25% with DTW. Furthermore, ADTW quickly tails off with few milestones that have a score of more than 18. On the other hand, as illustrated by Figure 7, most of the milestones have a score of greater than 60 with DTW.

We have experimented with different numbers of anchors (between 1 and 32) and have found that particularly for raytrace ADTW performance improves significantly with each doubling of anchor points. We should, however, be moderate with the anchoring: while we can use a huge number of anchors and get precise alignment of the milestones (which are necessary for the anchors), unnecessarily tracing milestone

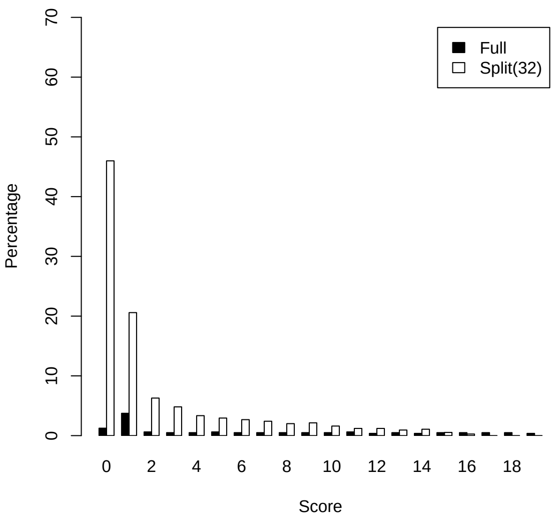

Figure 10: Alignment evaluation for raytrace.

events may perturb application behavior (e.g., the cost to write out the milestones to the trace may affect the IPC). Thus, we should strategically insert anchors only when their benefits will far outweigh their costs. One heuristic is to concentrate the anchors in relatively flat areas of behavior and omit anchors from active areas.

6. DISCUSSION

Our prior work [3] on understanding the performance of modern Java applications (which included raytrace) found that DTW performed well; more importantly we did not find any situation where DTW performed poorly. This is in contrast to the current paper where we find that DTW produces a poor alignment for raytrace.

To understand this apparent discrepancy we need to understand how we used DTW in prior work. In prior work we selected an area in one trace and used DTW to select corresponding areas in all the other traces. After we had done this we performed all analysis (e.g., correlations etc.) within a trace. Thus, as long as the trends in the aligned portions were similar enough, we did not care if DTW's alignment was perfect. In this work, we evaluate DTW in more detail and use absolute events within an application run to see how well DTW aligns two runs. Thus the conclusions of this paper differ from those in our prior work.

Whether or not DTW is good enough depends on how one intends to use the alignment. If the alignment is going to be used to enable reasoning across traces then most likely DTW is not good enough especially if the metrics being aligned are relatively flat. If the alignment is going to be used to select "matching" areas in different traces and all further reasoning will be within a trace then DTW is most likely good enough. In any case, ADTW, our extension to DTW, yields comparable if not better results than DTW and we recommend using that instead of DTW when milestones are available.

7. RELATED WORK

We are not aware of other research on aligning traces. Our approach for trace alignment is based on the dynamic time warping (DTW) technique [5], first introduced into the data mining community in 1994 [2]. The data mining community primarily uses DTW for the comparison of two time series, or two subintervals in a time series. This interpretation of the DTW distance as a similarity measure has been used for many data mining applications, such time series database queries [7], but also for the alignment of gene expression time series [1]. To the best of our knowledge we are the first to use DTW for aligning performance traces, and thus to enable the automatic performance analysis across multiple complementary traces produced by non-deterministic (real) systems.

8. CONCLUSIONS

For many performance analysis problems, the ability to reason across traces is invaluable. However, due to non-determinism in the OS and virtual machines, even two identical runs of an application yield slightly different traces. For example, it is unlikely that two identical runs of an application will suffer context switches at exactly the same points. These sorts of variations across traces make it difficult to reason across traces. This paper describes and evaluates an algorithm, Dynamic Time Warping (DTW), that can be used to align traces, thus enabling us to reason across traces. While DTW comes from prior work our use of DTW is novel.

We present a novel way to evaluate DTW's alignment of two traces by using application metrics whose position is deterministic across runs; that is, the milestones occur at the same point in the application's computation in every run and are unaffected by the non-determinism of the underlying OS or virtual machine. Using the milestones we compute a score for the alignment.

Our evaluation reveals that even though DTW is a simple algorithm it does surprisingly well. More specifically, when the metrics used for the alignment exhibit sharp frequent transitions DTW produces a good alignment. However, DTW can produce poor alignments when the metric used for the alignment has flat but noisy areas.

Based on these insights we came up Anchored DTW which improves upon DTW even for flat areas. More specifically Anchored DTW uses a handful of milestones (preferably in flat areas of the trace) as anchors which restricts it from generating poor alignments. We show that Anchored DTW dramatically outperforms DTW especially for traces where DTW performs poorly.

9. REFERENCES

[1] John Aach and George Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, Vol. 17:495–508, June 2001.

[2] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Working Notes of the Knowledge Discovery in Databases Workshop*, pages 359–370, July 1994.

[3] Matthias Hauswirth, Amer Diwan, Peter F. Sweeney, and Michael C. Mozer. Automating vertical profiling. In *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications*, pages 281–296, New York, NY, USA, 2005. ACM Press.

[4] Matthias Hauswirth, Peter F. Sweeney, Amer Diwan, and Michael Hind. Vertical profiling: understanding the behavior of object-priented applications. In *OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 251–269, New York, NY, USA, 2004. ACM Press.

[5] Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*, August 2004.

[6] Standard performance evaluation corporation. SPECjvm98 benchmarks. http://www.spec.org/jvm98.

[7] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. Efficient retrieval of similar time sequences under time warping, February 1998.