Sliding Hidden Markov Model for Evaluating Discrete Data

Tiberiu Chis, Peter G. Harrison
Department of Computing, Imperial College London
South Kensington Campus, London, UK
{tc207, pgh}@doc.ic.ac.uk

***Abstract*—The possibility of handling infrequent, higher density, additional loads, used mainly for on-line characterization of workloads, is considered. This is achieved through a sliding version of a hidden Markov model (HMM). Essentially, a sliding HMM keeps track of processes that change with time (i.e. changing the observation set at different stages of analysis) and the constant size of the observation set helps reduce the space and time complexity of the Baum-Welch algorithm, which now need only deal with the new observations. Practically, an approximate Baum-Welch algorithm, which is incremental and partly based on the simple moving average (SMA) technique, is obtained, where new data points are added to an input trace without re-calculating model parameters, whilst simultaneously discarding any outdated observation points. The success of this technique could cut processing times significantly, making HMMs more efficient and thence synthetic workloads computationally more cost effective. The performance of our sliding HMM is validated in terms of the means and standard deviations of observations (e.g. numbers of operations of certain types) taken from the original and synthetic traces, as well as their autocorrelations, and by comparisons between model parameters.**

I. Introduction

The hidden Markov model (HMM) has been relatively popular in workload characterization [3], [4] in recent years. Its parsimony, portability and efficient training, through its expectation maximization algorithm, has made it useful for reproducing meaningful and representative workload traces for simulating live systems. Research has also complimented these applications through an incremental storage model [10], [13], on which quantitative measures were made. This work has proven that the computation time to produce a reliably parameterized model can be significantly reduced, whilst maintaining accuracy of the model. Indeed, the incremental approach, by which a model's parameters are progressively updated rather than periodically re-calculated, has been appealing in terms of run-time performance.

A. Background

To achieve an incremental model, one can adapt the standard HMM algorithms used to train the model. These statistical algorithms under investigation are essentially those solving the three fundamental problems associated with HMMs: firstly, obtain $P(O;\lambda)$, or the probability of the observed sequence O given the model λ; secondly, maximize $P(O;\lambda)$ by adjusting the model parameters λ for a given observation sequence O; thirdly, determine the most likely hidden state sequence for an observed sequence. These three problems are solved by three respective algorithms: using the Forward-Backward algorithm [1], the Baum-Welch algorithm[1] [2] and the Viterbi algorithm [11]. The solutions to the Forward-Backward and Baum-Welch algorithms are presented in the following sections.

B. Forward-Backward algorithm

The Forward-Backward algorithm, aims to find $P(O;\lambda)$, which is the probability of the given sequence of observations $O = (O_1, O_2, \ldots, O_T)$ given the model $\lambda = (A, B, \pi)$, where there are T observations, A is the state transition matrix, B is the observation matrix and π is the initial state distribution. This is equivalent to determining the likelihood of the observed sequence O occuring. We use the same format presented in [16], which is based partly on Rabiner's solution [14], [15]. Initially, the focus is on the α-pass, which is the "forward" part of the Forward-Backward algorithm. Then, we shift our attention to the corresponding β-pass, aka. the "backward" part of the algorithm.

To begin with, we define $\alpha_t(i)$ as the probability of obtaining the observation sequence up to time t together with the state q_i at time t, given our model λ. Using N as the number of states and T as the number of observations, the mathematical notation is

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, s_t = q_i; \lambda) \tag{1}$$

where $i = 1, 2, \ldots, N$, $t = 1, 2, \ldots, T$, and s_t is the state at time t.

Proceeding inductively, we write the solution for $\alpha_t(i)$ as follows:

1) For $i = 1, 2, \ldots, N$,
$$\alpha_1(i) = \pi_i b_i(O_1).$$
2) For $i = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T-1$,
$$\alpha_{t+1}(i) = [\textstyle\sum_{j=1}^{N} \alpha_t(j) a_{ji}] b_i(O_{t+1})$$
where $\alpha_t(j)a_{ji}$ is the probability of the joint event observing $O_1, O_2, \ldots O_t$ and moving from state q_j at time t to state q_i at time $t+1$.
3) It follows that,

[1] this algorithm uses the Forward-Backward algorithm iteratively.

$$P(O;\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$
$$\text{where } \alpha_T(i) = P(O_1, O_2, \ldots, O_T, s_T = q_i; \lambda)$$

The backward variable, $\beta_t(i)$, is defined as the probability of obtaining the observation sequence from time $t+1$ to T, given state q_i at time t and the model λ. So we have,

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T; s_t = q_i, \lambda) \tag{2}$$

and the solution of $\beta_t(i)$ is given by

1) For $i = 1, 2, \ldots, N$,
$$\beta_T(i) = 1$$
2) For $i = 1, 2, \ldots, N$ and $t = T-1, T-2, \ldots, 1$,
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$
where we note that O_{t+1} can be observed from any state q_j.

C. Baum-Welch algorithm

The Baum-Welch algorithm attempts to maximise $P(O;\lambda)$ by adjusting the parameters A, B, π given the model $\lambda = (A, B, \pi)$ and the observation sequence $O = (O_1, O_2, \ldots, O_T)$. This is done as an iterative process. We first define the probability of making a transition from state q_i at time t to state q_j at time $t+1$, given O and λ, as

$$\xi_t(i,j) = P(s_t = q_i, s_{t+1} = q_j; O, \lambda) \tag{3}$$

Computing $\xi_t(i,j)$ can be described as a three-step process. Firstly, the observations $O_1, O_2, \ldots, O_t$ finishing in state q_i at time t will be covered by $\alpha_t(i)$. Secondly, the transition from q_i to q_j, where O_{t+1} was observed at time $t+1$, is represented by the term $a_{ij} b_j(O_{t+1})$. Thirdly, the remaining observations $O_{t+2}, O_{t+3} \ldots O_T$ beginning in state q_j at time $t+1$ are covered by β_{t+1}. Putting those together, and dividing by a normalizing term ($P(O;\lambda)$) we have

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O;\lambda)} \tag{4}$$

We now sum the terms in (4) over j and notice that this gives the probability of being in state q_i at time t, given the observation sequence O and model λ. This probability is defined as

$$\gamma_t(i) = P(s_t = q_i; O, \lambda) = \sum_{j=1}^{N} \xi_t(i,j)$$

Summing $\gamma_t(i)$ over time t up to T, we get the number of times we expect to visit state q_i. Similarly, summing up to $T-1$ gives the expected number of transitions made from q_i. Thus:

$$\sum_{t=1}^{T} \gamma_t(i) = \text{Expected times state } q_i \text{ is visited.}$$
$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{Expected transitions from } q_i.$$

Similarly, we sum $\xi_t(i,j)$ over t as follows:

$$\sum_{t=1}^{T} \xi_t(i,j) = \text{Expected visits of } q_i \text{ then } q_j.$$
$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{Expected transitions } q_i \text{ to } q_j.$$

Using these terms, the re-estimation formulas for our HMM parameters are:

$$\pi_i' = \gamma_1(i)$$
$$a_{ij}' = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{j=1}^{N} \sum_{t=1}^{T-1} \xi_t(i,j)}$$
$$b_j(k)' = \frac{\sum_{t=1, O_t=k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Using these re-estimation formulas, we can update our model $\lambda' = (A', B', \pi')$, where $A' = \{a_{ij}'\}$, $B' = \{b_j(k)'\}$ and $\pi' = \{\pi_i'\}$. Our model will have fixed parameters once $P(O;\lambda') > P(O;\lambda)$, which means the iterative process to find the optimal model λ' ends.

D. Incremental model

The incremental Storage Workload Model (iSWoM) [13] made use of these algorithms with modifications of the mechanisms of the Forward-Backward algorithm (creating a forward-recurrence backward approximation). Therefore, inherently, the Baum-Welch algorithm was adapted to create a model for incremental learning of discrete data. The iSWoM essentially generated workload traces, running on live systems where quantitative measurements were made. These measurements, acting as statistical validation, included means, standard deviations and confidence intervals for both raw and iSWoM-generated traces. Also, comparisons of hidden state sequences, as generated by the Viterbi algorithm, further validated the iSWoM with a standard HMM, and found similar model parameters.

Using the incremental approach of the iSWoM, by which a model's parameters are progressively updated rather than periodically re-calculated, was accurate and also appealing in terms of its run-time performance. However, the training of new, incoming data points resulted in an increased storage requirement for our model, which became a burden with time. Thus, we seek a more efficient on-line characterization method for discrete time analysis. The aim is to modify the incremental model to allow for a fixed sliding window to analyse discrete data traces and automatically update model parameters.

II. Sliding HMM

The sliding HMM (SlidHMM) has a number of benefits over its standard HMM counterpart: firstly, handling infrequent, higher density, additional loads mainly for on-line characterization of workloads; secondly, to measure time-variant processes efficiently through updating the observation set at different stages of analysis; thirdly, to reduce the space and time complexity of the Baum-Welch algorithm. These benefits are also matched by the iSWoM, but where the SlidHMM maintains a fixed window of observations for training, the iSWoM has an observation set that grows continuously over time. This will make the SlidHMM computationally more efficient than the iSWoM for training on large data sets. The SlidHMM will allow for effectively comparing different sections of the obervation set using its sliding window, a technique which the iSWoM nor the standard HMM can achieve. We employ the

simple moving average technique on the SlidHMM, enabling the updating of terms whilst maintaing a fixed size window of analysis.

A. Moving average

A *moving average* [8] or *running average* is a statistical technique where a set of data points is split into subsets and averages are calculated on each of these subsets. Moving averages have seen many applications in industry, such as trend following analysis in finance [6]. For a simple moving average (SMA) [7], we select a fixed subset size (n) and shift along, subtracting old points from the summation as we add new points to it. For example, if we begin with the data points $\{x_1, x_2, \ldots, x_n\}$, then we can work out an average of these points:

$$ave = \frac{x_1 + x_2 + \cdots + x_n}{n} \tag{5}$$

Then from (5) we can create a SMA when we add one more data point (x_{n+1}):

$$\begin{aligned} sma &= \tfrac{x_t + x_{t+1} + \cdots + x_{t+n} + x_{t+n+1} - x_t}{n} \\ &= \tfrac{x_t + x_{t+1} + \cdots + x_{t+n}}{n} + \tfrac{x_{t+n+1}}{n} - \tfrac{x_t}{n} \\ &= ave + \tfrac{x_{t+n+1}}{n} - \tfrac{x_t}{n} \end{aligned}$$

The idea of SMA is applied to HMMs for observation sets with discrete data. New data points are added to the input trace without any unnecessary re-calculations of model parameters, whilst simultaneously discarding any "outdated" observation points. We replace the data points $x_t(n)$ by our model recurrence terms such as αs, βs, etc. This process is explained in the following section, where we present a simple algorithm for executing the slide on discrete data.

B. Sliding Baum-Welch algorithm

To perform the slide on an observation set, the Baum-Welch algorithm must train on new data, whilst storing information the original data set. Therefore, a new technique is required to store existing α and β values (terms worked out from the current observation set) and efficiently calculating the new α and β values for the new set of observations. For example, if we are given the observation set $\{O_{T+1}, O_{T+2}, \ldots, O_{2T}\}$ having an existing HMM defined on the observations $\{O_1, O_2, \ldots, O_T\}$, then the αs for the new set of observations will be updated as follows:

For $T \leq t \leq 2T$, we have

$$\alpha_{t+1}(i) = b_i(O_{t+1}) \sum_{j=1}^{N} \alpha_t(j) a_{ji}$$

However, we cannot compute the new β values incrementally for the new observation set $\{O_{T+1}, O_{T+2}, \ldots, O_{2T}\}$ without working out all the β values for the aggregate observation set $\{O_1, O_2, \ldots, O_{2T}\}$. Unlike the α values, the β values use a backward recursion where which set $\beta_{2T}(i)$ to 1 (as O_{2T} is our latest observation) and calculate the β values accordingly using our backward recurrence formula.

There exists a solution, or rather an approximation, of these unknown β values for the new observation set $\{O_{T+1}, O_{T+2}, \ldots, O_{2T}\}$. The technique used by Stenger et al. in 2001 [12] assumes the following simple approximation:

For $1 \leq i \leq N$, we have

$$\beta_T(i) = \beta_{T+1}(i) = \beta_{T+2}(i) = \cdots = \beta_{2T}(i) = 1 \tag{6}$$

However, from the knowledge of the traditional backward recurrence formula for the β values, we can deduce that the sequence $\beta_{2T}(i), \beta_{2T-1}(i), \ldots, \beta_T(i)$ decreases in value, where $\beta_{2T-1}(i)$ is significantly less than $\beta_{2T}(i)$, etc. Eventually, this decreasing sequence of β values should tend exponentially to zero. Therefore, setting all the new β values to one as seen in (6) is not the most efficient solution.

We can attempt a more accurate approximation for the β values by assuming that only $\beta_{2T}(i) = 1$ and then use the normal β recurrence formula to update the terms: $\beta_{2T-1}(i), \beta_{2T-2}(i), \ldots \beta_{T+1}(i)$. Notice the change only for the new observations in terms of β.

So, $\beta_{2T-1}(i)$ is calculated as follows:

$$\begin{aligned} \beta_{2T-1}(i) &= \textstyle\sum_{j=1}^{N} a_{ij} b_j(O_{2T}) \beta_{2T}(j) \\ &= \textstyle\sum_{j=1}^{N} a_{ij} b_j(O_{2T}) \end{aligned}$$

And therefore $\beta_{2T-2}(i)$ is:

$$\begin{aligned} \beta_{2T-2}(i) &= \textstyle\sum_{k=1}^{N} a_{ik} b_k(O_{2T-1}) \beta_{2T-1}(k) \\ &= \textstyle\sum_{k=1}^{N} a_{ik} b_k(O_{2T-1}) [\sum_{j=1}^{N} a_{kj} b_j(O_{2T})] \end{aligned}$$

Continuing in this fashion, until we obtain a value for $\beta_{T+1}(i)$ in terms of all new β values. Essentially, this methodology utilizes part of the backward formula, but ignores any "old" β values. Once a complete approximation of both α and β sets, we calculate the ξ and γ values for the set $\{O_{T+1}, O_{T+2}, \ldots, O_{2T}\}$:

For $T + 1 \leq t \leq 2T - 1$ we have

$$\xi_t(i,j) = \tfrac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(i)}{\sum_{i=1}^{N} \alpha_t(i) \beta_t(i)}$$

and for $T + 1 \leq t \leq 2T$ we have

$$\gamma_t = \tfrac{\alpha_t(i) \beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i) \beta_t(i)}$$

However, these points are added incrementally and therefore the Baum-Welch algorithm adds one new term for each new observation (in T separate steps). Hence, for each new observation added, the modified re-estimation formulas for our incremental HMM parameters ($\hat{\pi}, \hat{A}, \hat{B}$) are as follows:

Initially at $t = 1$, we have

$$\hat{\pi}'_i = \gamma_1(i)$$

where $i = 1, \ldots, N$

For $\hat{A}$, we have

$$\hat{a}_{ij}^{T+1} = \frac{\sum_{t=1}^{T} \xi_t(i,j) + \xi_{T+1}(i,j)}{\sum_{j=1}^{N}\sum_{t=1}^{T} \xi_t(i,j) + \sum_{j=1}^{N} \xi_{T+1}(i,j)}$$
$$= \frac{\sum_{t=1}^{T} \gamma_t(i)}{\sum_{t=1}^{T+1} \gamma_t(i)} \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)} + \frac{\xi_{T+1}(i,j)}{\sum_{t=1}^{T+1} \gamma_t(i)}$$
$$= \frac{\sum_{t=1}^{T} \gamma_t(i)}{\sum_{t=1}^{T+1} \gamma_t(i)} \hat{a}_{ij}^{T} + \frac{\xi_{T+1}(i,j)}{\sum_{t=1}^{T+1} \gamma_t(i)}$$

Thus, only the new $\xi_{T+1}(i,j)$ and $\gamma_{T+1}(i)$ for O_{T+1} need to be calculated. This is because the $\xi_t(i,j)$ values for $1 \leq t \leq T$ are stored in the $\hat{a}_{ij}^{T}$ entry.

For $\hat{B}$, we have

$$\hat{b}_j(k)^{T+1} = \frac{\sum_{t=1,O_t=k}^{T} \gamma_t(j) + \sum_{t=T+1,O_t=k}^{T+1} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j) + \gamma_{T+1}(j)}$$
$$= \frac{\sum_{t=1}^{T} \gamma_t(j)}{\sum_{t=1}^{T+1} \gamma_t(j)} \hat{b}_j(k)^{T} + \frac{\sum_{t=T+1,O_t=k}^{T+1} \gamma_t(j)}{\sum_{t=1}^{T+1} \gamma_t(j)}$$

where updating $\gamma_{T+1}(j)$ (such that $O_{T+1} = k$) is sufficient because the previously calculated γ values lie in the $b_j(k)^T$ entries.

Under these modified parameters (similar to those seen in [12], a sliding version of the Baum-Welch algorithm is created(referred to as **SlidHMM**). It requires only a partial computation of the forward and backward variables and thus can converge to fixed results much quicker than the traditional Baum-welch algorithm (which has time and space complexity $O(N^2T)$, where T is the number of observations and N is the number of states [5]). To summarise our sliding methodology for any discrete observation sets, we have four simple steps:

1) Train HMM on observations $\{O_1, O_2, \ldots, O_T\}$ until convergence.
2) Perform slide on new observations $\{O_{T+1}, O_{T+2}, \ldots, O_{T+M}\}$ using SlidHMM.
3) Calculate new α, β, ξ and γ sets for new observations.
4) Update parameters (A, B, π) for SlidHMM and continue to step 2.

III. Collecting and processing traces

In order to train our SlidHMM on discrete data, we need to process various traces such that they pass as eligible inputs into the Baum-Welch algorithm. Two different traces are used to train the model: first, the NetApp trace made up of thousands read and write commands from a server; second observes patient arrivals at a hospital over a period of several weeks. In the following sections, these raw data points are transformed into binned traces and finally into discrete observation sets.

A. Raw traces

The raw NetApp data points, which contains hundreds of thousands of entries collected from NetApp storage servers, essentially form a CIFS (Common Internet File System) network trace (of about 750 GB). These file servers, located at the NetApp headquarters, were accessed mainly by Windows desktops and laptops using various applications. We denote the aforementioned trace as the "NetApp trace" for the remainder of this paper. The trace used for analysis consisted of I/O commands (single CIFS reads and writes) and a timestamp entry (i.e. the time in seconds when the command was made). The data was transferred from a web page into read and write arrays (in a local Java class) using an *InputStreamReader*.

The data describing patient arrival times was collected from an internal PostgreSQL database (namely *aesop_artery*) from the Department of Computing at Imperial College London. Within this database, we accessed the *arrivals* table to extract the arrival times for a period of four weeks using a simple SQL query. The resulting "Hospital trace" was outputted into a csv file, which was read into our Java class and paired with the NetApp trace. With both traces collected and stored in arrays, the next stage of the transformation process is assigning "bins" to these traces.

B. Binned traces

We partitioned the entries of the raw traces into uniform bins of a pre-defined size. These bins are essentially fixed-size intervals, dividing the raw data into a discrete time series. For the NetApp trace, each bin contains two values: the number of read entries and the number of write entries (during each time interval). For the Hospital trace, each bin is an interval (i.e. an hour) where a number of patients can arrive at the hospital.

The size of the bin was decided by the timescale required for the modelling exercise. For example, if the raw trace spans a time period of several days, then we expect much larger bin sizes than if we had a raw trace spanning several hours. Also, the level of detail at which the raw trace is operating (e.g. at the Application level) is also an important factor in determining the bin size. After experimenting with the NetApp raw trace, we found the best bin size to be one second. Having tried 100 milliseconds resulted in too many empty time intervals, whilst with a larger time interval (i.e. five seconds), there were issues of missing out low-level, operation sequence characteristics such as mode transitions. Using one second bin sizes allowed us to represent each index in an array as a second. Counting the number of commands occuring each second (separately for reads and writes) resulted in filling our arrays easily. For example, reads[3] = 76 and writes[3] = 23 represented 76 reads and 23 writes occuring in the third second. A vector list, holding a pair of read and write values as each data point, was formed from the arrays.

The Hospital trace was binned in a similar fashion, but only contained one entry (i.e. the number of patient arriving every hour). After analysing the frequency of patient arrivals over four weeks, almost one third of cases had no activity. On the other hand, two patients arrived in the same hour about 17% of the time. In fact, on very few occasions were there more than eight patients in one hour. Thus, choosing the one hour bin sizes resulted in an ideal range of values for forming clusters around our data points. In the next section, the NetApp and Hospital traces (acting as vectors with paired and single tuple values, respectively) are inputted into a K-means clustering algorithm to obtain our observation traces.

C. K-means clustering

Apply a clustering algorithm to the binned traces further reduces them to more manageable format (i.e. the observation

trace). We implemented the K-means clustering algorithm, which essentially groups data into K clusters. Each cluster contains either a pair of values (i.e. the mean number of reads and mean number of writes) or single value (i.e. number of patient arrivals) to represent the centroid. Logically, the cluster also contains every data points belonging to that cluster. A Euclidean-distance itervative algorithm calculated the cluster centroids over and over again until they became fixed. As we inputted K manually, we chose a value of seven clusters for the NetApp trace and three clusters for the Hospital trace. These values were not too large (which gives surplus or even empty clusters) nor too small (missing out significant differences among clusters) for our data traces. As an example of the NetApp trace, the seven clusters are listed below as vectors. Note that the centroid is written as a pair of values, the first value representing reads and the second value for writes. These are essentially our seven observation values:

$$\begin{pmatrix} 964.53 & 2.18 \\ 221.87 & 0.35 \\ 1.49 & 0.69 \\ 160.92 & 0.78 \\ 637.5 & 0.37 \\ 394.08 & 0.2 \\ 77.35 & 2.95 \end{pmatrix} \tag{7}$$

As shown in (7), observation values represent low writes with increasing reads. The read values start off low and progress to medium and high values. It is expected that there are not many varying writes (i.e. medium or high writes) in this read-dominated trace. The Hospital trace was limited to five or less clusters because after clustering with $K = 6$, there were two empty clusters present (i.e. with centroids of 0.0). As we inputted the value of K manually, we decided to use a value of three clusters, as it gave closer means to the raw data when compared to HMM-generated data. The three clusters are listed here and are essentially our observation values:

$$\begin{pmatrix} 4.99 \\ 0.39 \\ 2.4 \end{pmatrix} \tag{8}$$

As we can see above in (8), observation values from top to bottom represent: frequent arrivals, very few arrivals and moderately frequent arrivals. Having performed the K-means clustering on both our NetApp and Hospital traces, we obtain observation traces ideal for input into the Baum-Welch algorithm. Essentially, the SlidHMM will train on these observation traces as slides are performed on various sections of observations.

IV. Simulation of SlidHMM

To achieve both simulations of the SlidHMM, for both NetApp and Hospital traces, each observation trace is inputted into the Baum-Welch algorithm as a training set of 8000 points. A HMM (with two hidden states) is trained on this set until parameter convergence (i.e. A, B, π become fixed). Afterwards, 2000 new observations are added to this set, evaluating the 2000 points using the sliding technique and the new β approximation from the Forward-Backward algorithm. Thus, a sliding MAP (SlidMAP) with fixed parameters is formed, which stores information on 10000 consecutive observation points. Our SlidMAP then generates its own synthetic NetApp and Hospital traces using its initial state distribution (π), state transition matrix (A) and observation matrix (B). The SlidMAP reproduces the observed values using random generation sampling, which are simulated 1000 times, summarised as means and standard deviations, where 95% intervals are performed on both statistics. We compare SlidHMM-generated results with mean and standard deviation for raw and HMM-generated traces. Note that the HMM-generated trace is a result of a traditional HMM trained on an observation trace of length 10000, with no incremental learning.

V. Results

A. Statistical comparison of discrete data traces

The Baum-Welch parameters (i.e. transition matrix and initial state distribution) obtained from performing the SlidHMM simulation on the NetApp data were as follows:

$$A = \begin{pmatrix} 0.97 & 0.03 \\ 0.1 & 0.9 \end{pmatrix} \tag{9}$$

$$B = \begin{pmatrix} 0.0 & 0.04 & 0.92 & 0.0 & 0.01 & 0.02 & 0.01 \\ 0.22 & 0.12 & 0.06 & 0.04 & 0.36 & 0.16 & 0.04 \end{pmatrix} \tag{10}$$

$$\pi = (0.0, 1.0) \tag{11}$$

The transition matrix has expected entries, very close to the original HMM. The emission matrix is also well approximated for this new observation set. From these new parameters, the SlidHMM generated a new NetApp observation trace. Table I presents statistics on Reads/bin and table II represents Writes/bin, where the "bin" is a one-second interval. For example, a "Raw Mean of 111.350 Reads/bin" implies that the raw NetApp trace produces, on average, 111.350 read commands per second. The "SlidHMM Mean" and "SlidHMM Std Dev" are the averages of the SlidHMM-generated trace. The "HMM"-prefixed averages are calculated from a standard HMM-generated trace with no incremental activity. The results for the NetApp trace are summarised as follows:

Trace	Mean	Std Dev
Raw	111.350	254.904
HMM	111.26 $\pm$ 0.66	254.38 $\pm$ 0.65
SlidHMM	113.32 $\pm$ 0.60	253.18 $\pm$ 0.58

TABLE I. Reads/bin statistics on the raw, HMM and SlidHMM-generated NetApp traces after 1000 simulations

Table I shows very similar results between raw and HMM-generated means and standard deviations. The SlidHMM produces a mean of 113.32 with a 95% confidence interval of 0.6, which is pleasing after 1000 simulations, but this mean is less accurate than the traditional HMM. The standard deviation of the SlidHMM-generated trace (253.18) matches the raw trace

well, but again is outperformed slightly by the value of the HMM trace.

Trace	Mean	Std Dev
Raw	0.382	0.208
HMM	0.38 $\pm$ 0.0005	0.21 $\pm$ 0.001
SlidHMM	0.392 $\pm$ 0.0005	0.23 $\pm$ 0.001

TABLE II. WRITES/BIN STATISTICS ON THE RAW, HMM AND SLIDHMM-GENERATED NETAPP TRACES AFTER 1000 SIMULATIONS

Table II shows good results for the SlidHMM-generated mean and standard deviation, which slightly underperform the values produced by the HMM trace. The Hospital observation trace is also generated by the SlidHMM, with means and standard deviations as follows:

Trace	Mean	Std Dev
Raw	1.483	1.565
HMM	1.461 $\pm$ 0.003	1.551 $\pm$ 0.001
SlidHMM	1.474 $\pm$ 0.002	1.572 $\pm$ 0.001

TABLE III. ARRIVALS/BIN STATISTICS ON THE RAW, HMM AND SLIDHMM-GENERATED HOSPITAL TRACES AFTER 1000 SIMULATIONS

Table III show bin-means that match well, and more pleasingly, the standard deviations are even closer to raw values. We can conclude, from these statistics alone, that our SlidHMM faithfully reproduces meaningful representations of patient arrival times. Continuing model validation, the next section presents autocorrelation functions which justify the synthetic, SlidHMM-generated traces.

VI. CONCLUSION

HMMs, combined with the supporting clustering analysis and appropriate choice of bins, is able to provide a concise, parsimonious and portable synthetic workload. This has already been established, in [4] for example, but the deficiency of such models is their heavy computing resource requirement, which essentially precludes them from any form of on-line analysis. The sliding model developed in the paper, SlidHMM, has a vastly reduced computing requirement making it ideal for modelling workload data in real-time. In fact, with the availability of decoding new data, the SlidHMM avoids re-training on "old data" like the traditional HMM. Additionally, compared with both the resource-costly HMM and raw traces, the SlidHMM provides excellent accuracy of training data. In comparison to the previously mentioned iSWoM [13], the SlidHMM will handle fast-growing observation sets more efficiently, as it slides and trains on different parts of the data in ant given order. In fact, the iSWoM can only compare new observations with its most recent observation set, a drawback caused by its incremental Baum-Welch learning algorithm.

Such mathematical descriptions of workloads and arrivals should be measured quantitatively against independent data (i.e. traces not used in model construction) that they represent, and more extensive tests are planned for our sliding model. Nonetheless, the SlidHMM β approximation has been successful after statistical comparisons between raw and SlidHMM-generated traces (on two independent traces). Analysing current work in this field, for example, the incremental model from [5] used a backward formula in its learning that was not recursive in terms of previous β values. The SlidHMM backward formula, however, stores all information in the β set, unlike the formula used in [12] where all β variables were equal to one and where accuracy is lost over Baum-Welch iterations.

VII. FUTURE WORK

With SlidHMM, we approximated the unknown β values for the new observations by using the backward algorithm recurrence formula. Another possibility for approximating the β values using the backward formula for the β values from [13]. The result would be an incremental model (namely the forward-recurrence updating of the βs) with a "sliding window" learning technique. As this paper illustrates, SlidHMM applies to discrete time series, in fact on two different traces. The next step would deriving a sliding model for continuous time. This would require a continuous version of the Baum-Welch algorithm, as seen in [9], which would then behave as a sliding algorithm for a continuous time series. The intervals would not be observation points as in the discrete case, but rather fixed time intervals which slide along the continuous time series. Therefore, a degree of accuracy is needed in choosing the time intervals for the sliding HMM, perhaps to the nearest millisecond of the timestamp to represent each boundary value.

REFERENCES

[1] Baum, L. E., Petrie, T.: Stastical Inference for Probabilistic Functions of Finite Markov Chains, In *The Annals of Mathematical Statistics*, **37**, pp. 1554-63 (1966)

[2] Baum, L. E., Petrie, T., Soules, G., Weiss, N.: A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, In *The Annals of Mathematical Statistics*, **41**, pp. 164-171 (1970)

[3] Ashraf, J., Iqbal, N., Khattak, N. S., Zaidi, A. M.: Speaker Independent Urdu Speech Recognition Using HMM (2010)

[4] Harrison, P. G., Harrison, S. K., Patel N. M., Zertal, S.: Storage Workload Modelling by Hidden Markov Models: Application to Flash Memory, In: *Performance Evaluation*, **69**, pp. 17-40 (2012)

[5] Florez-Larrahondo, G., Bridges, S., Hansen, E. A.: Incremental Estimation of Discrete Hidden Markov Models on a New Backward Procedure, Department of Computer Science and Engineering, Mississippi State University, Mississippi, USA (2005)

[6] Burghardt, G., Duncan, R., Liu, L.: What You Should Expect From Trend Following (2004)

[7] Chou, Y.: Statistical Analysis, In *Holt International*, **17.9** (1975)

[8] Whittle, P.: Hypothesis Testing in Time Series Analysis, Almquist and Wicksell (1951)

[9] Zraiaa, M.: Hidden Markov Models: A Continuous-Time Version of the Baum-Welch Algorithm, Department of Computing, Imperial College London, London (2010)

[10] Chis, T., Harrison, P. G.: Incremental HMM with an improved Baum-Welch Algorithm, Proceedings of Imperial College Computing Student Workshop (2012)

[11] Viterbi, A. J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, In *IEEE Transactions on Information Theory*, **13**, pp. 260-269 (1967)

[12] Stenger, B., Ramesh, V., Paragois, N., Coetzee, F., Buhmann, J. M.: Topology free Hidden Markov Models: Application to background modeling, Proceedings of the International Conference on Computer Vision, pp. 297-301 (2001)

[13] Chis, T., Harrison, P. G.: iSWoM: An Incremental Storage Workload Model using Hidden Markov Models, Department of Computing, Imperial College London, *To be published* (2013)

[14] Rabiner, L. R., Juang, B. H.: An Introduction to Hidden Markov Models, In *IEEE ASSP Magazine*, **3**, pp. 4-16 (1986)

[15] Rabiner, L. R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, In *IEEE*, **77**, pp. 257-286 (1989)

[16] Chis, T.: Hidden Markov Models: Applications to Flash Memory Data and Hospital Arrival Times, Department of Computing, Imperial College London (2011)