iSWoM: The incremental Storage Workload Model based on Hidden Markov Models

Tiberiu Chis and Peter G. Harrison

Department of Computing, Imperial College London,
Huxley Building, 180 Queens Gate, London SW7 2RH, UK
{tc207,pgh}@doc.ic.ac.uk

Abstract. We propose a storage workload model able to process discrete time series incrementally, continually updating its parameters with the availability of new data. More specifically, a Hidden Markov Model (HMM) with an adaptive Baum-Welch algorithm is trained on two raw traces: a NetApp network trace consisting of timestamped I/O commands and a Microsoft trace also with timestamped entries containing reads and writes. Each of these traces is analyzed statistically and HMM parameters are inferred, from which a fluid input model with rates modulated by a Markov chain is derived. We generate new data traces using this Markovian fluid, workload model. To validate our parsimonious model, we compare statistics of the raw and generated traces and use the Viterbi algorithm to produce representative sequences of the hidden states. The incremental model is measured against both the standard model (parameterized on the whole dataset) and the raw data trace.

1 Introduction

In modern, large-scale storage environments, workload arises from multiple, time-varying, correlated traffic streams that may create different resource bottlenecks in the system at different times. It is therefore important to categorize and model workload in a portable and efficient way for at least three purposes:

- to produce workload traces for live systems, on which quantitative measurements can then be made;
- to generate similar, representative traces for system simulation;
- to provide input parameters for analytical performance models.

We focus on the first purpose, with the emphasis on live systems. It takes considerable time, training-data and computing power to produce a reliably parameterized model and our incremental approach, by which a model's parameters are progressively updated rather than periodically re-calculated, is appealing in terms of its run-time performance. The question is, therefore, is it accurate?

Primarily, we use a Hidden Markov Model (HMM), which is a bivariate Markov chain pairing one observable state sequence with a hidden state sequence (first developed by Baum and Petrie [1]). Each hidden state produces an observation based on a probability given in a so-called observation matrix. The

transition probability from one hidden state to another is based on the state transition matrix of the Markov chain. This encodes information on the evolution of a time series and uses its parameter estimation algorithm to help infer the behaviour of the given series. As HMMs can efficiently represent workload dynamics, acting as parsimonious models that obtain trace characteristics, their applications include storage workload modelling [7], speech recognition [6, 10] and genome sequence prediction [4, 5]. HMMs act as portable benchmarks to explain and predict the complex behaviour of multiapplication workloads and therefore optimize storage and access times. One aim of this paper is to create such a benchmark, which can be applicable to a number of discrete processes, by adapting slightly the HMM mechanisms that represent these processes.

We utilize statistical algorithms to investigate the three fundamental problems associated with HMMs – here, as they pertain to workload traces. The first problem is determining $P(O;\lambda)$, which is the probability of the observed sequence O given some model λ; the second is to maximize $P(O;\lambda)$ by adjusting the model parameters λ for a given observation sequence O; and the third problem is finding the most likely hidden state sequence associated with a given observed sequence. The three fundamental problems are solved by using the Forward-Backward algorithm, the Baum-Welch algorithm[1] [2] and the Viterbi algorithm [3], respectively. Initially, this paper focuses on creating an adaptive Baum-Welch algorithm by using a forward recurrence formula to approximate certain parameters in the Forward-Backward algorithm. The Viterbi algorithm is briefly used in the results section to further validate the model.

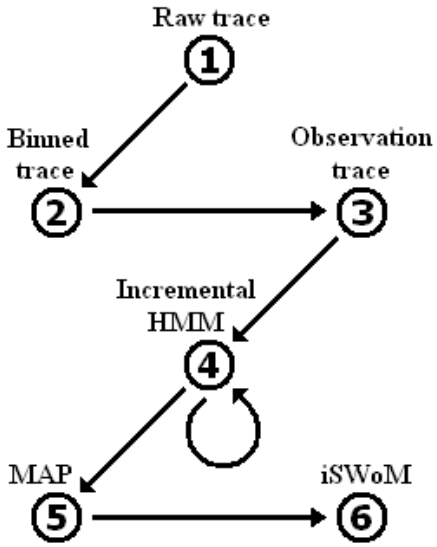

Fig. 1. A map showing our steps needed to obtain the iSWoM.

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

Most of the paper is concerned with the development of an incremental Storage Workload Model (iSWoM), which consists of various procedures and submodels. The journey to form the iSWoM is summarised in Fig. 1. Firstly, we gather two distinct raw traces, which are transformed into binned traces using partitioning, and then into observation traces using K-means clustering. Secondly, our incremental HMM (i.e. a HMM that uses an adaptive Baum-Welch algorithm) trains on each observation trace to provide estimates for the model parameters. Note in Fig. 1 the circular arrow pointing to the incremental HMM, signifying the iterative process of training until no new data is made available. On completion, our incremental HMM defines (a special case of) a discrete Markov Arrival Process (MAP). The MAP and a random distribution (that chooses probabilistically particular input values from a specified cluster) form the iSWoM, which is capable of processing incoming data incrementally and updates its parameters on-the-fly as new trace data is available. The iSWoM can generate unlimited discrete traces, corresponding to and representative of the respective observation traces that have been inputted into the model. These generated traces can be validated using statistical comparisons (mean, standard deviation, confidence intervals) against raw traces. The Viterbi algorithm processes both the raw and iSWoM-generated traces and produces hidden state sequences for each, where accuracy is obtained by comparing correspondingly similar state patterns. Further applications can characterize workloads according to their HMM parameters and also be used to model quantitatively *a priori* similar workload environments – for example as inputs to simulation or analytical models. This paper aims for a storage workload model, formed by HMMs, capable of incremental learning of discrete data. Investigating other storage workload system benchmarks [19], we compare the iSWoM with our peers' models. Xhang et al [12] conducted measurements on three benchmarks (namely TPC-W [16], TPC-C [14] and RUBiS [13]) with the aim of understanding behaviour of e-commerce storage systems. Their findings consisted of workloads dominated by transactions (i.e. writes) requiring more storage than workloads dominated by browsing (i.e. reads). Similarities can be found between the read-dominated NetApp trace and the e-commerce writing workload, in terms of increased work which they demand. Regardless, the iSWoM provides on-line data characterization for its workloads (reads or writes), which Xhang et al have not attempted to fit on their model. Another workload characterization was presented by Kurmas at al [15], more specifically on an open mail trace using an FC-60 disk array as the storage system. The authors formed a cumulative distribution function (CDF) of read latency using a workload generator which reads values from a list. The I/O requests used in their workload included read or write commands and were replicated in a synthetic trace, much like the iSWoM-generated traces, but no incremental learning was attempted. We conclude the paper with evaluating our results in the context of other work and discuss improvements and continued research for the iSWoM. In the next section, we present the adaptive Baum-Welch algorithm as the key to the formation of the iSWoM.

2 Adaptive Baum-Welch algorithm

The creation of an adaptive Baum-Welch algorithm is the foundation of the iSWoM and can be perceived as an updated Baum-Welch algorithm with a new Forward-Backward algorithm. More specifically, the backward part of the Forward-Backward algorithm will be updated with a forward-recurrence formula such that the probabilities for the new observation set can be easily calculated. We begin this section by defining both the Forward-Backward and Baum-Welch algorithms, and then describe the modifications to each one, along with a mathematical approximation for the backward variables. In the definitions, we bear in mind the need for normalization to avoid underflow with large data sets. However, to simplify notation, the standard algorithmic terms are used in this section. The approximation is followed by the definition of the incremental HMM.

2.1 Forward-Backward algorithm

The Forward-Backward algorithm, aims to find $P(O;\lambda)$, the probability of the given sequence of observations $O = (O_1, O_2, \ldots, O_T)$ given the model $\lambda = (A, B, \pi)$, where T is the number of 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 Rabiner's solution [9] to aid our calculations, focusing initially 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 that $O_1, O_2, \ldots O_t$ are observed and we move from state q_j at time t to state q_i at $t+1$.
3. It follows that,
$$P(O;\lambda) = \textstyle\sum_{i=1}^{N} \alpha_T(i)$$

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.

2.2 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.

2.3 Incremental HMM

The re-estimation of the model $\lambda' = (A', B', \pi')$ only works on a fixed set of observations. The aim of the adaptive Baum-Welch algorithm is to continually read in new trace data and update its parameters on-the-fly. This will form an incremental HMM (IncHMM) capable of efficiently supporting real time workload data, so demonstrating that infrequent, higher density, additional loads can be handled, mainly for on-line characterization of workloads as in [7]. The IncHMM initially starts as a standard HMM reading a set of observations and updating its current parameters A, B, π according to new incoming data. Therefore, after the standard HMM has finished training on its observation set, we calculate the revised α, β, ξ and γ variables based on the new set of observations.

To achieve an efficient IncHMM, we first update the terms of the Forward-Backward algorithm, which are the α and β values. Firstly, a HMM is trained on a trace of T observations and then M new observations are added. To update our model incrementally, we notice that the next α value is given by $\alpha_{T+1}(i) = [\sum_{j=1}^{N} \alpha_T(j) a_{ji}] b_i(O_T)$. The knowledge of the terms $\alpha_T(j)$, a_{ji} and $b_i(O_T)$ allows the new α variables to be computed easily using the forward recurrence formula. However, to find $\beta_{T+1}(i)$ is more difficult because it is dependent on a one step lookahead $\beta_{T+1}(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{T+2}) \beta_{T+2}(j)$ and unfortunately we do not have $\beta_{T+2}(j)$. Therefore an approximation for the β variables is needed, preferrably a forward recurrence formula similar to the α formula. This β approximation will be adapted from [17], which was a preliminary attempt

on a single data trace. The new ξ and γ variables (and thus the entries a'_{ij} and $b_j(k)'$) are calculated easily once the α and β sets are complete.

The β approximation will assume that, at time t and for state i, we have that $\beta_t(i) = \delta(t, i)$ is a continuous function with parameters t and i. For any state i, the function $\delta(t, i)$, w.r.t. t, is increasing from 0 to 1. Equivalently, $\delta(t, i)$ tends to 0 as $t \to 0$. The logic of this assumption comes from the backward recurrence formula in (2), which calculates the β values with a one-step lookahead. All β terms from $t = T - 1$ to $t = 1$ are less than 1, and with every step that t decreases, $\beta_t(i)$ gets closer to 0 through the computations of the backward formula. Therefore, for a sufficiently large observation set, we obtain the approximate equality $\delta(t, i) \approx 0 \approx \delta(t, j)$, where i and j are different states. We write the β approximation as:

$$\beta_t(i) \approx \beta_t(j) \tag{5}$$

Let us now transform the β backward-recurrence formula into a forward-recurrence version, with simplified notation $b_j = b_j(O_{t+1})$. Since there are two states in our model, let $N = 2$. It then follows that

$$\begin{pmatrix} \beta_t(1) \\ \beta_t(2) \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{2} a_{1j} b_j(O_{t+1}) \beta_{t+1}(j) \\ \sum_{j=1}^{2} a_{2j} b_j(O_{t+1}) \beta_{t+1}(j) \end{pmatrix}$$

Taking $\beta_t(1)$ and expanding the summation on the RHS, we obtain:

$$\beta_t(1) = a_{11} b_1(O_{t+1}) \beta_{t+1}(1) + a_{12} b_2(O_{t+1}) \beta_{t+1}(2)$$

Assuming that $t + 1$ is sufficiently small and using (5) we can deduce that $\beta_{t+1}(1) \approx \beta_{t+1}(2)$, giving us:

$$\beta_t(1) = \beta_{t+1}(1)(a_{11} b_1(O_{t+1}) + a_{12} b_2(O_{t+1}))$$

Making $\beta_{t+1}(1)$ the subject results in:

$$\beta_{t+1}(1) = \frac{\beta_t(1)}{a_{11} b_1(O_{t+1}) + a_{12} b_2(O_{t+1})} \tag{6}$$

Generalising for state i yields our forward-recurrence β approximation:

$$\beta_{t+1}(i) \approx \frac{\beta_t(i)}{\sum_{j=1}^{N} a_{ij} b_j(O_{t+1})} \tag{7}$$

With the β approximation defined in the new backward formula (7), we run the Forward-Backward algorithm and execute the α-pass, followed by the β-pass. Following the calculation of both α and β value sets on the new observations $\{O_{T+1}, O_{T+2}, \ldots, O_{T+M}\}$, the ξ and γ values are defined as follows:

For $T + 1 \leq t \leq T + M - 1$,

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

and for $T+1 \leq t \leq T+M$,

$$\gamma_t = \frac{\alpha_t(i)\beta_t(i)}{P(O;\lambda)}$$

We can now define the IncHMM (based on methods seen in [11]) using the updated Forward-Backward algorithm in the adaptive Baum-Welch algorithm. For each new observation, we define the modified re-estimation formulas for our IncHMM parameters $(\hat{\pi}, \hat{A}, \hat{B})$ as follows:

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

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

where we only compute the new $\xi_{T+1}(i,j)$ and $\gamma_{T+1}(i)$ for each new observation (note the $\xi_t(i,j)$ values for $1 \leq t \leq T$ are already stored in the $\hat{a}_{ij}^{T}$ entry).

$$\begin{aligned}
\hat{b}_j(k)^{T+1} &= \frac{\sum_{t=1,O_t=k}^{T} \gamma_t(j) + \sum_{t=T+1,O_t=k}^{T+1} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j) + \gamma_{T+1}(j)} \\
&= \frac{\sum_{t=1}^{T} \gamma_t(i)}{\sum_{t=1}^{T+1} \gamma_t(i)} \hat{b}_j(k)^{T} + \frac{\sum_{t=T+1,O_t=k}^{T+1} \xi_{T+1}(i,j)}{\sum_{t=1}^{T+1} \gamma_t(i)}
\end{aligned}$$

where we only update $\gamma_{T+1}(j)$ (such that $O_{T+1} = k$) after storing all previous γ values in the $\hat{b}_j(k)^{T}$ entries.

This concludes the definition of the IncHMM. To achieve our iSWoM, we first transform the raw traces into observation traces and use these more concise forms in the IncHMM to form a discrete Markov Arrival Process.

3 Transformation of traces

The transformation of the raw traces into binned traces and then into observation traces, acting as input for HMM training, is summarised briefly as follows:

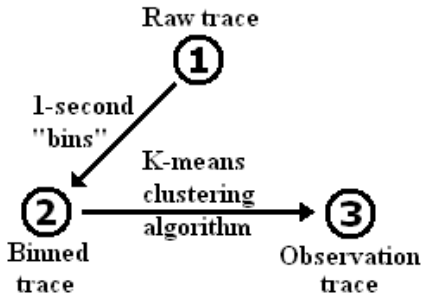

Fig. 2. The transformation of traces by various processes.

In this section, we discuss, in the order presented in Fig. 2, the processes which evolve the discrete time series data. These include binning and clustering raw traces to transform data into a more manageable form for HMM processing.

3.1 Raw Traces

The first step of the process is analysing the raw NetApp trace, which contains hundreds of thousands of entries collected from NetApp storage servers. A CIFS (Common Internet File System) network trace (of about 750 GB) was gathered from file servers at the NetApp headquarters, where the servers were accessed mainly by Windows desktops and laptops using various applications. In this paper, the aformentioned trace will be known simply as the NetApp trace.

This NetApp trace is a part of a larger network trace (about 12 GB in size) consisting of I/O commands (single CIFS reads and writes). Each entry of the collected trace is of the form:

```
Cmd: Write
Timestamp: 3.4051253054
PID: 2520
IP: 10.58.48.58
Filename: 3G314EG822563KF312D4G4F
Size: 72
Offset: 0
```

For each of these entries, the command (Cmd), either "Read" or "Write," and its "Timestamp" (i.e. the time in seconds when the command was made) were collected and stored in separate read and write arrays using an *InputStream-Reader*. Similarly, the Microsoft trace was collected over 15-minute periods and captures the complete build process. The trace events were primarily Disk IO (block level) and File IO, where we focused on reads and writes. Other trace specifications include a Windows Server 2008 OS and two socket, quad-core processors. Analysing the primary fields of interest for our storage workload research, we used the timestamp of the event (measured in microseconds from the

start of the trace) and the type of trace event (i.e. FileIoRead and FileIoWrite). Other data, which were not considered for this paper, included size of the request in bytes and elapsed time from start of event to completion event for a specific IO (also in microseconds). The next stage of the transformation process is assigning "bins" to our traces, thus creating two distinct, binned traces.

3.2 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. Each bin contained two values: the number of read entries and the number of write entries during that time interval. Consistency is kept in both the NetApp and Microsoft binned traces through the formation of reads and writes and the same size interval. The bin sizes were 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 was 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[7] = 85 and writes[7] = 20 represents 85 reads and 20 writes in the seventh second. A vector list, holding a pair of read and write values as each data point, was formed from the arrays. Similarly to the NetApp trace, we binned the Microsoft trace in one-second intervals. After transforming the timestamped microseconds into seconds, we filtered through the raw trace and chose only the read and write events. The result was two arrays of length 9000 (i.e. 9000 seconds is 150 minutes), for reads and for writes. Both NetApp and Microsoft binned traces (acting as vectors with (read,write) tuples as data points) are inputted into a K-means clustering algorithm to obtain observation traces.

3.3 K-means Clustering

The next step of the transformation was to apply a clustering algorithm to the binned traces and further reduce them to more manageable format (i.e. the observation trace). We implemented the K-means clustering algorithm, which grouped data into K clusters. Each cluster contains a pair of values representing the centroid (i.e. the mean number of reads and mean number of writes for that group of points) and all the data points belonging to that cluster (i.e. all the reads and writes in the group). A Euclidean-distance itervative algorithm calculates 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 four clusters for the Microsoft trace. These values were not too large (giving surplus or even empty clusters) nor too little (missing out significant differences among clusters) for our data traces. The NetApp trace lists seven clusters as vectors, with each centroid written as a pair of values, reads and writes, respectively.

$$\begin{pmatrix} 964.53 & 2.18 \\ 221.87 & 0.35 \\ 1.49 & 0.69 \\ 160.92 & 0.78 \\ 637.5 & 0.37 \\ 394.08 & 0.2 \\ 77.35 & 2.95 \end{pmatrix} \tag{8}$$

As shown in (8), observation values represent low writes with increasing reads. Some read values are 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 Microsoft observation trace was given four clusters to assign its data points to (i.e. $1, 4, 2, 3, 1, 3, \dots$). Finalising with two discrete data traces, we proceed to simulating the iSWoM. Model validation includes comparing statistical averages of raw, HMM and iSWoM-generated traces, and also using Viterbi algorithm to produce hidden state.

4 iSWoM Simulation and Results

4.1 Simulating the iSWoM using NetApp and Microsoft traces

To achieve the first simulation, the NetApp observation trace is inputted into the Baum-Welch algorithm as a training set of 8000 points (i.e. 8000 seconds). The IncHMM is trained on this set until parameter convergence (i.e. A, B, π converge). Afterwards, 2000 new observations are added and are evaluated using the new β approximation from the Forward-Backward algorithm. Thus, an incremental MAP (IncMAP) with fixed parameters is formed, which stores information on 10000 consecutive observation points. Our IncMAP then generates its own synthetic NetApp trace using its parameters (A, B, π). The IncMAP reproduces the observed values using random generation sampling, and simulated 1000 times, thus forming the iSWoM. In fact, the iSWoM possesses its own distribution of NetApp reads and writes defined by mean and standard deviation, where 95% intervals are performed on both statistics. We compare iSWoM-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. The second simulation involves a Microsoft data trace as input and follows the same process as that of the NetApp simulation, except that the original training set holds 7000 points. A new set of 2000 unseen Microsoft data points is added to this training set and the process follows identically. In the next section, we present the results of our simulation.

4.2 Results of iSWoM

Tables 1 and 3 present statistics on Reads/bin and Tables 2 and 4 represent Writes/bin, where the "bin" is a 1-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 "iSWoM Mean" and "iSWoM Std Dev" are the averages of the iSWoM-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:

Table 1. Reads/bin statistics on the raw, HMM and iSWoM NetApp traces

Trace	Mean	Std Dev
Raw	111.350	254.904
HMM	111.26 ± 0.66	254.38 ± 0.65
iSWoM	113.32 ± 0.60	253.18 ± 0.58

Table 1 shows very similar results between raw and HMM-generated means and standard deviations. The iSWoM 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 iSWoM-generated trace (253.18) matches the raw trace well, but is outperformed slightly by the value of the HMM trace. Table 2 presents good results for the iSWoM-generated mean and standard deviation, which again slightly underperform compared to the values produced by the HMM trace.

Table 2. Writes/bin statistics on the raw, HMM and iSWoM NetApp traces

Trace	Mean	Std Dev
Raw	0.382	0.208
HMM	0.38 ± 0.0005	0.21 ± 0.001
iSWoM	0.41 ± 0.0005	0.24 ± 0.001

Table 3. Reads/bin statistics on the raw, HMM and iSWoM Microsoft traces

Trace	Mean	Std Dev
Raw	1.153	20.6
HMM	1.14 ± 0.01	20.33 ± 0.16
iSWoM	1.15 ± 0.01	20.42 ± 0.15

Table 3 summarises the statistics for the raw, HMM and iSWoM-generated Microsoft reads. The iSWoM mean and standard deviation have outperformed the traditional HMM-generated traces.

Table 4. Writes/bin statistics on the raw, HMM and iSWoM Microsoft traces

Trace	Mean	Std Dev
Raw	0.242	0.719
HMM	0.243 ± 0.0005	0.719 ± 0.001
iSWoM	0.242 ± 0.0005	0.718 ± 0.001

The statistics for the Microsoft Writes/bin reveal very similar results for raw, HMM and iSWoM traces. Both sets of confidence intervals at 95% level are identical for the HMM and iSWoM traces, as a result of the small mean and standard deviation values and also setting the population size equal to 1000 (i.e. 1000 simulations). The next section focuses on using the Viterbi algorithm as a means of validating the iSWoM.

4.3 Viterbi hidden state sequence analysis

The Viterbi algorithm is used to return a sequence of hidden states, corresponding to a sequence of observations. Viterbi uses the parameters A, B and π to calculate its hidden state sequence. Therefore, Viterbi signifies, through its hidden state sequence, if the parameters for two different HMMs are similar. We compare the hidden state sequence based on the raw observation trace (i.e. the sequence of observations originally inputted into HMMs) with that of the iSWoM-generated synthetic observation trace. This process is applied to both NetApp and Microsoft traces, in turn. Unlike the simulation of 1000 runs which used the Baum-Welch algorithm, the Viterbi uses only one observation trace per hidden state sequence. The results of the Viterbi algorithm based on this raw NetApp trace and its iSWoM equivalent are as follows:

Table 5. Viterbi state sequence ratio on the raw and iSWoM NetApp trace

Trace	State 1	State 2
Raw	7938	2062
iSWoM	8182	1818

The sequences seen in Table 5 match very well, with an approximate 4:1 ratio supported by both raw and iSWoM traces. Thus, it proves, most importantly,

that the HMM and iSWoM each produce a set of parameters which are in agreement. The following Viterbi results were obtained for the raw Microsoft trace and iSWoM-generated trace, and show an identical ratio of hidden states:

Table 6. Viterbi state sequence ratio on the raw and iSWoM Microsoft trace

Trace	State 1	State 2
Raw	1	8999
iSWoM	1	8999

From Table 6, it is clear both raw and iSWoM traces have the majority of their observations from one state. This is due to the transition matrix heavily favouring one state, with a very small probability of exiting. The Viterbi algorithm has sustained the validity of the iSWoM given agreeable hidden state sequences and thus similar model parameters to a traditional HMM.

5 Conclusions and Further Work

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 [7], but the deficiency of such models is their heavy computing resource requirement, which essentially precludes them from any form of on-line analysis. The incremental model we have developed, iSWoM, 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 iSWoM avoids re-training on "old data" like the traditional HMM. Additionally, compared with both the resource-costly HMM and raw traces, the iSWoM provides excellent accuracy of training data. Such mathematical descriptions of workload 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 incremental model. Nonetheless, the iSWoM β approximation has been successful after statistical comparisons between raw and iSWoM-generated traces. Other related work, for example, the incremental model from [8], used a backward formula in its learning that was not recursive in terms of previous β values. The iSWoM backward formula, however, stores all information on the complete β set, unlike the formula used in [18] where all β variables were equal to 1. In fact, our forward-recurrence β formula used in the iSWoM is validated by NetApp and Microsoft data traces and produces satisfying simulated results. Extensions to this paper arise from other possible validation techniques of the iSWoM. The sensitive autocorrelation function would indeed be a useful validation method for our model, mainly because it allows time series comparison through lagged versions of the original data trace. Hidden trends can be exposed in both raw and iSWoM-generated autocorrelated

data. Another extension might be a cumulative distribution function (CDF) for the iSWoM workload distribution. Given any processed trace, a CDF can be used to obtain specific time series probabilities, such as observing k reads or writes within a time t, etc. This will highlight the iSWoM as a more transparent probabilistic model.

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. 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)
4. Krough, A., Brown, M., Mian, S., Sjolander, K., Haussler, D.: Hidden Markov Models in Computational Biology, In: *Journal of Molecular Biology*, pp. 1501-1531, Computer and Information Sciences, University of California, Santa Cruz, USA, (1994)
5. Burge, C., Karlin, S.: Prediction of complete gene structures in human genomic DNA, In: *Journal of Molecular Biology*, pp. 78-94, Computer and Information Sciences, Stanford University, California, USA (1997)
6. Ashraf, J., Iqbal, N., Khattak, N. S., Zaidi, A. M.: Speaker Independent Urdu Speech Recognition Using HMM (2010)
7. 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)
8. 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)
9. Rabiner, L. R., Juang, B. H.: An Introduction to Hidden Markov Models, In *IEEE ASSP Magazine*, **3**, pp. 4-16 (1986)
10. Rabiner, L. R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, In *IEEE*, **77**, pp. 257-286 (1989)
11. Chis, T.: Hidden Markov Models: Applications to Flash Memory Data and Hospital Arrival Times, Department of Computing, Imperial College London (2011)
12. Zhang, X., Riska, A., Riedel, E.: Characterization of the E-commerce Storage Subsystem Workload, Proceedings of Quantitative Evaluation of SysTems (2008)
13. RUBiS Implementation: http://rubis.objectweb.org/.
14. On-Line Transaction Processing (OLTP) Benchmark: http://www.tpc.org/tpcc.
15. Kumas, Z., Keeton, K., Becker-Szendy, R.: I/O Workload Characterization, CAECW-01, before HPCA-7 (2001)
16. Transactional Web e-Commerce Benchmark: http://www.tpc.org/tpcw.
17. Chis, T., Harrison, P. G.: Incremental HMM with an improved Baum-Welch Algorithm, Proceedings of Imperial College Computing Student Workshop (2012)
18. 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)
19. Keaton, K., Veitch, A., Obal, D., Wilkes, J.: I/O characterization of commercial workloads, CAECW-00 (2000)