Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 318 (2015) 109–127

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Adapting Hidden Markov Models for Online Learning

Tiberiu Chis[1,2] Peter G. Harrison[3]

Department of Computing
Imperial College London
London, UK

Abstract

In modern computer systems, the intermittent behaviour of infrequent, additional loads affects performance. Often, representative traces of storage disks or remote servers can be scarce and obtaining real data is sometimes expensive. Therefore, stochastic models, through simulation and profiling, provide cheaper, effective solutions, where input model parameters are obtained. A typical example is the Markov-modulated Poisson process (MMPP), which can have its time index discretised to form a hidden Markov model (HMM). These models have been successful in capturing bursty behaviour and cyclic patterns of I/O operations and Internet traffic, using underlying properties of the discrete (or continuous) Markov chain. However, learning on such models can be cumbersome in terms of complexity through re-training on data sets. Thus, we provide an online learning HMM (OnlineHMM), which is composed of two existing variations of HMMs: first, a sliding HMM using a moving average technique to update its parameters "on-the-fly" and, secondly, a multi-input HMM capable of training on multiple discrete traces simultaneously. The OnlineHMM reduces data processing times significantly and thence synthetic workloads become computationally more cost effective. We measure the accuracy of reproducing representative traces through comparisons of moments and autocorrelation on original data points and HMM-generated synthetic traces. We present, analytically, the training steps saved through the OnlineHMM's adapted Baum-Welch algorithm and obtain, through simulation, mean waiting times of a queueing model. Finally, we conclude our work and offer model extensions for the future.

Keywords: HMM, online learning, adapted Baum-Welch, autocorrelation, MMPP.

1 Introduction

Over the last decade, the performance and reliability of networks and storage systems have been key issues for international enterprises with a global online presence. On a large scale, businesses increasingly face the technical challenges that network performance may impose on their critical IP applications, remote desktops or video conferencing such as a lag in waiting times, availability and congestion with varying real-time Internet conditions in each country. Additionally, cost of storing data is

[1] Thank you to Nigel Thomas and the UKPEW Committee

[2] Email: tiberiu.chis07@imperial.ac.uk

[3] Email: p.harrison@imperial.ac.uk

http://dx.doi.org/10.1016/j.entcs.2015.10.022

increasing as the ratio of users to servers is growing, leading to competition amongst top cloud providers (i.e. Amazon EC2). Users demand applications to be available on tablets and smartphones and expect reliable performance of these mobile devices. Therefore, devices, servers and networks must all meet the required quality of service (QoS) standards determined by realistic service level agreements (SLAs) from respective clients and vendors. However, maintaining a consistently smooth network performance, by continuously upgrading infrastructure or increasing bandwidth, is expensive to match a large user base with heavy demand; a similar problem faces cloud providers w.r.t. storage.

In an attempt to solve these challenges, engineers and researchers rely on a combination of methods. Geographically, large-scale systems are spread apart to minimise communication (and therefore minimise delays) between servers located near end users and data centers. With different locations experiencing variable connectivity, a challenge is to maintain consistent service for all users, irrespective of location. At packet level, methods such as *traffic classification* helps to regulate packet transmission and bandwidth [13]. However, simply classifying IP traffic is not enough (i.e. without also modelling system load, waiting time, packet type, arrival burstiness, etc.) because non-deterministic events dictate traffic behaviour and should be modelled to improve storage systems and network performance and reliability. Workload models allow for experimenting with new storage system designs, where production systems provide key characteristics of their applications [3]. Consequently, it is desirable to extract representative workload parameters from storage traces.

Simple models, such as Poisson processes, no longer provide realistic tools for modelling Internet traffic and storage access, as they fail to account for long-range dependence (LRD) or burstiness of packets and I/O commands. Therefore, to improve this, researchers are turning their attention to more complex models, such as Markov-modulated Poission processes (MMPPs) and hidden Markov models (HMMs). In fact, the MMPP can be viewed as a discretely indexed HMM by observing intervals between events as a sequence of random variables [17]. The HMM is a bivariate Markov chain of states and transitions composed of a hidden chain (with unknown states) and an observable chain. Such models have been more successful in accounting for LRD, self-similarity, burstiness in jobs and switching modes, where we turn the reader's attention to [15,16] for reference. Similarly, Harrison et al use HMMs to obtain input to performance models of Flash memory [3]. Despite the success of these models in classifying properties of Internet traffic and storage access, there exist inefficiencies in static learning and (unnecessary) repetitive training of data.

The need to learn data in an online manner (i.e. "on-the-fly") is particularly useful for live systems where latency has a significant impact for users. Some models in the literature have adapted learning algorithms to avoid re-training on data sets, where model parameters are updated with new data [6,11]. The incremental HMM [11], in particular, provides parsimony, and portability through its adapted expectation maximisation algorithm, which trains strictly on new data points. This

work has proven that computation time to produce a parameterised model can be significantly reduced, whilst maintaining accuracy of the model. Similar variations of HMMs use a sliding window of data points based on a moving average (i.e. old points are discarded as new points arrive) and train on discrete data in an online manner, referred to as SlidHMM [9]. Both incremental and sliding approaches progressively update the model's parameters rather than periodically re-calculating them, which is appealing in terms of run-time performance. Using such models, one can reproduce meaningful and representative workload traces for simulating live systems, on which quantitative measures are made. However, this adaptive training is executed per trace and a useful upgrade would be to obtain a model capable of training on multiple traces simultaneously.

The capability for training models on multiple traces is growing in importance with the analysis of online interactions on social media and across shared systems and resources. For example, variations of HMMs have identified trends in group behaviour in Twitter data by adapting k-means clustering and the Baum-Welch algorithm to train on multiple users simultaneously [4]. This multi-input HMM (MultiHMM) was an improvement (as we scaled up) on the computationally-expensive coupled HMM [19], which used a Markov chain to represent one user and the coupling of chains was the social interaction. Despite the computations saved by the MultiHMM, an improvement through incremental training of parameters would be advantageous for online learning.

To address some of the aformentioned issues, we propose an online, multi-input HMM capable of training on multiple traces in an incremental fashion, without decreasing in accuracy and computational efficiency. We adapt the well-known k-means clustering and Baum-Welch algorithms to support multiple traces simultaneously and obtain a discrete-time, online HMM (OnlineHMM). Applications of this model include traffic classification and workload benchmarking for live systems. To validate the effectiveness of our methods, we quantitatively compare moments and autocorrelation from the raw (i.e. unclustered), standard HMM and OnlineHMM-generated traces. Another efficiency measure for our OnlineHMM (compared to the standard and HMM variations) is the number of steps required for the model parameters to converge in the adapted Baum-Welch algorithm component. Potential online applications of our model include: characterizing workload patterns of burstiness to predict peaks of high activity; building user profiles on-the-fly in social networks based on online interactions; managing resource allocation for shared applications with intermittent patterns of activity.

The paper is organised as follows: In section 2, we outline methods of traffic classification, define MAPs and MMPPs, introduce HMM algorithms, summmarise existing variations of HMMs and explain how to merge various model features to form a superior online model; section 3 sets up the OnlineHMM, with relevant preliminaries, model training and simulation; in section 4, we present results to validate the efficiency of the OnlineHMM against other variations of models and obtain mean waiting times for a related queueing model representing an abstracted storage device; section 5 is reserved for the conclusion and future work.

2 Background

In this section, we describe existing models for traffic classification, including MAPs and MMPPs, and explain their relation to HMMs. We explore the suitability of these models for related applications and offer simple extensions into queueing theory to analyse other system features. Then, we provide an introduction to HMM training algorithms and define iterative formulas used to converge HMM parameters. The OnlineHMM is composed of two main processes: the first is a sliding window to learn data using a moving average technique; the second is an adapted k-means clustering technique requiring traces to be doubly clustered before passed as input into the online Baum-Welch algorithm. We outline both of these processes in this section, explaining how the novel online learning method works when these processes are merged and discuss the resulting advantages.

2.1 MAPs and MMPPs

The analysis and classification of Internet traffic has been vital in the understanding, planning and designing of robust, large-scale workload models. There exist different types of traffic classification. Classification by port number is fast and supported by numerous network devices, but is only useful to the applications that use fixed port numbers. In comparison, deep packet inspection (DPI), which inspects the packet payload, is slow, requires more resources and using this method rarely overcomes encryption. Another method is statistical classification and treats traffic data as a multi-dimensional time-series, which provides some nice benefits over the aforementioned methods: a faster technique than port number classification; by analyzing packet sizes and packet inter-arrival times, one gets a deeper understanding of system behaviour at various times; cycles and burstiness in packet arrivals can be detected using this method. Similar applications to Internet traffic include time series recording storage access times and sequences of I/O commands at disks. When using such techniques to build workload models, the designing and planning process can range in effort and cost. At one end, there are large-scale cloud service models, such as infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS) and software-as-a-service (SaaS), which work on data collected at distributed server clusters. At the other end, stochastic models, such as Markovian arrival processes (MAPs), batch MAPs [18] and phase-type distributions provide more analytical tools. Hence, there exist a range of models to analyse time series, where one of the simplest is the Poisson process (PP) that models inter-arrival times. The PP is a continuous-time stochastic point-process, in which inter-arrival times (between events) are independent and exponentially distributed. This process can be discretised, via partitioning time-stamped data into "bins", and turned into a portable, discrete-time stochastic process or time series. Further, PPs are generalised by MAPs, which evolve according to transitions in a continuous-time Markov chain (CTMC) [10]. We formally define a MAP in Definition 2.1:

Definition 2.1 Let $Q = D_0 + D_1$ be an infinitesimal generator for a CTMC, where D_0 and D_1 are square matrices (with non-negative off-diagonal elements of D_0 and

elements of D_1). Then, a MAP(D_0, D_1) is a point process where an event occurs in the CTMC by a transition associated with D_1. A MAP(D_0, D_1) of order two (i.e. MAP(2)) has parameters:

$$D_0 = \begin{pmatrix} -(\sigma_1) & \alpha_{12} \\ \alpha_{21} & -(\sigma_2) \end{pmatrix} \text{ and } D_1 = \begin{pmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{pmatrix}$$

where $\sigma_1 = \alpha_{12} + \sum_j \lambda_{1j}$; $\sigma_2 = \alpha_{21} + \sum_j \lambda_{2j}$; $\alpha_{ij} \geq 0$ are hidden transitions made from states i to state j; $\lambda_{ij} \geq 0$ are observable transitions of D_1, for $i, j = 1, 2$.

A subset of MAPs that considers job types on arrival are Markov-modulated Poisson processes (MMPPs). In fact, a MAP becomes an MMPP when D_1 (from definition 2.1) is a diagonal matrix. Advantages of MMPPs include closed-form expressions for correlation between inter-arrival times and modelling multiple job types. For example, in the case of the latter, an MMPP(2) (i.e. an MMPP with two states) acts as a job arrival process that switches between periods of frequent arrivals (state one), with arrival rate λ_1, and few arrivals (state two), with arrival rate (λ_2), where $\lambda_1 > \lambda_2$ [8]. Therefore, the infinitesimal generator of MMPP(2) (i.e. $D = D_0 + D_1$) is defined as follows:

$$D = \begin{pmatrix} -(\alpha_{12} + \lambda_1) & \alpha_{12} \\ \alpha_{21} & -(\alpha_{21} + \lambda_2) \end{pmatrix} + \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

where α_{ij} are hidden transitions made from state i to state j, for $i, j = 1, 2$.

By using MMPPs, one can infer the correlation between job arrivals, where underlying properties fit the behaviour of various storage and Internet workload time series. For example, taking a quasi-birth-death (QBD) process, figure 1 shows an MMPP/M/1 queue with MMPP(2) arrivals (λ_1 or λ_2), exponential service times (μ), one server and transitions between states (α_{12} or α_{21}). Note, multiple rates of service times can be specified (i.e. μ_1 and μ_2) in queueing models, such as hyper-exponential servers with probabilities ($\sum_{i=1}^{2} p_i = 1$) for the rates. The main benefit of integrating MMPPs into a queueing system is to create an abstraction of complex server behaviour and simulating measurements that are otherwise difficult to obtain from real data.

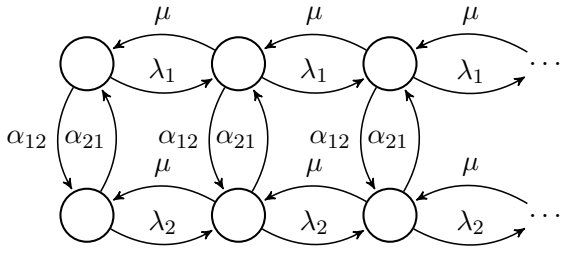

Fig. 1. QBD process for MMPP/M/1

From figure 1, one can see the parallels between MMPPs and HMMs, namely the switching modes through internal transitions of states. To be able to classify arriv-

ing jobs is useful for workload benchmarking and scheduling of MMPP-distributed tasks, but is only one part of the wider goal. By modelling servers as part of queueing systems, one can understand and (ideally) predict waiting times, variable load, system bottlenecks and resource allocation of modern servers in cloud and file storage applications. In section 4.4, we incorporate an MMPP into a first-come first-served (FCFS) queue and obtain mean waiting times for varying load. Fluid input models, such as MMPP, can have discretised time variants, such as the HMM, which is simpler to model and offers similar powerful traffic analysis. In fact, an advantage of HMMs is the parsimonious nature which allows representation of time-varying, correlated traffic streams in workload models. In the next section, the fundamental HMM algorithms are introduced.

2.2 HMM algorithms

To build the OnlineHMM and use as input to a workload model, we first adapt the standard HMM algorithms used to train the model. The statistical algorithms under investigation solve three fundamental problems associated with HMMs: first, obtain $P(O;\lambda)$, or the probability of the observed sequence O given the model λ; secondly, maximise $P(O;\lambda)$ by adjusting the model parameters of λ for a given observation sequence O; thirdly, determine the most likely hidden state sequence S for an observed sequence O. These problems are solved by three respective algorithms: the Forward-Backward algorithm (FBA) [1], the Baum-Welch algorithm [4] (BWA) [2] and the Viterbi algorithm (VA) [7]. The FBA and BWA train the HMM parameters until convergence and we describe these algorithms in more detail in the next section, including model parameters and recursion equations. In fact, the BWA re-estimation formulas, defined by equation (4), only work on a single trace of fixed size and a useful upgrade for the BWA is to handle multiple stream analysis for characterizing discrete data in an online fashion. Further, we adapt the FBA and BWA to model multi-input processes using clustering techniques and add an efficient, updating technique using a sliding window, which forms the OnlineHMM.

2.3 Forward-Backward algorithm

The Forward-Backward algorithm (FBA) solves the following problem: Given an observation sequence $O = (O_1, O_2, \ldots, O_T)$ (i.e. T is the size of the observation set) and the model $\lambda = (A, B, \pi)$, calculate $P(O;\lambda)$ (i.e. the probability of O, given the model), and obtain the likelihood of O. The parameters of λ are: the state transition matrix (A), containing probabilities for moving from one state to another; the observation matrix (B), with probabilities for each state emitting an observation; the initial hidden state distribution (π). Based on the solution in [12], we present the "forward" part of the algorithm (i.e. the α-pass), followed by the "backward" part (β-pass). We define the forward variable $\alpha_t(i)$ as: the probability of O up to time t ($1 \leq t \leq T$) and of state q_i at time t, given our model λ. So, $\alpha_t(i) = P(O_1, O_2, \ldots, O_t, s_t = q_i;\lambda)$, where $i = 1, 2, \ldots, N$ (i.e. N is the number of

[4] this algorithm uses the FBA iteratively.

hidden states in the HMM), $t = 1, 2, \ldots, T$ and s_t is the state at time t. The solution of $\alpha_t(i)$ (for $i = 1, 2, \ldots, N$) is initially $\alpha_1(i) = \pi_i b_i(O_1)$ and defined recursively (for $t = 2, 3, \ldots, T$) as follows:

$$\alpha_t(i) = [\sum_{j=1}^{N} \alpha_{t-1}(j) a_{ji}] b_i(O_t) \tag{1}$$

where $\alpha_{t-1}(j)a_{ji}$ is the probability of the joint event that $O_1, O_2, \ldots O_{t-1}$ are observed (given by $\alpha_{t-1}(j)$) and there is a transition from state q_j at time $t-1$ to state q_i at time t (given by a_{ji}); $b_i(O_t)$ is the probability that O_t is observed from state q_i. Similarly, we can define the backward variable $\beta_t(i)$ as the probability of the observation sequence from time $t+1$ to the end, given state q_i at time t and the model λ. Then, $\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots O_T; s_t = q_i, \lambda)$. The solution for $\beta_t(i)$ (for $i = 1, 2, \ldots, N$) is initially given by $\beta_T(i) = 1$ and defined recursively (for $t = T-1, T-2, \ldots, 1$) as follows:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \tag{2}$$

where the observation O_{t+1} is generated from any state q_j. For further analysis of α and β terms, the reader may refer to section 2.3 of [15]. With the α and β values computed, the BWA iterative equations can be defined.

2.4 Baum-Welch algorithm

Given the model $\lambda = (A, B, \pi)$, the Baum-Welch algorithm (BWA) trains the HMM on a fixed set of observations $O = (O_1, O_2, \ldots, O_T)$. By adjusting its parameters A, B, π, the BWA maximises $P(O; \lambda)$. As explained in section 2.3.2 of [15], the parameters of the BWA are updated iteratively (for $i, j = 1, 2, \ldots, N$, and $t = 1, 2, \ldots, T-1$) as follows:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O; \lambda)}; \ \gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{3}$$

where $\xi_t(i,j)$ is the probability of a path reaching state q_i at time t and transitioning to state q_j at time $t+1$. Summing over a set of observations, we can obtain values for the expected number of transitions from or to any arbitrary state. Thus, it is straightforward to update our HMM parameters:

$$a'_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}; \ b'_j(k) = \frac{\sum_{t=1, O_t=k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}; \ \pi'_i = \gamma_1(i) \tag{4}$$

We can now re-estimate our model parameters iteratively using $\lambda' = (A', B', \pi')$, where $A' = \{a'_{ij}\}$, $B' = \{b'_j(k)\}$ and $\pi' = \{\pi'_i\}$, as defined in equation (4). Note that with large observation sets, underflow of small terms is an issue as values tend to zero and so normalising these terms is recommended [14,15].

2.5 Variations of HMMs

To understand the OnlineHMM, we must first analyse the processes that compose it. The first is an adaptation of the sliding HMM (SlidHMM) [9], which is based on a simple moving average and uses a sliding window technique for data measurement. The second is a multi-input HMM (MultiHMM) capable of training on multiple discrete traces simultaneously. The OnlineHMM attempts to merge both techniques and create a novel online learning workload benchmark.

2.5.1 Sliding HMM

The concept of a sliding window to update data sets on which HMM parameters are trained on-the-fly is appealing in terms of run-time performance and online workload characterization. By updating the data set with new arrivals and simultaneously discarding the oldest observations, one can measure time-variant processes parsimoniously. The sliding window effect improves the incremental learning of IncHMM [11], which accumulates an increasingly large observation set as the outdated data points are included in iterative updates of HMM parameters.

In order to implement the SlidHMM, the HMM algorithms are modified w.r.t. the FBA, thus creating a forward-recurrence backward approximation. Therefore, the BWA is adapted to the new FBA and updates HMM parameters with relative ease. First, we describe the principle behind the backward approximation, which uses the α and β values. Say the HMM is trained on a discrete trace of T observations and M new observations are added to this set. To update our model, we first use the FBA definition of $\alpha_{T+1}(i) = [\sum_{j=1}^{N} \alpha_T(j)a_{ji}]b_i(O_T)$. 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 depends on a one step lookahead (i.e. $\beta_{T+1}(i) = \sum_{j=1}^{N} a_{ij}b_j(O_{T+2})\beta_{T+2}(j)$) and, unfortunately, the value of $\beta_{T+2}(j)$ is unknown. By adapting an approximation for the β variable [6,11], one can obtain a forward recurrence formula similar to the α term. Then, the BWA variables (ξ , γ , a'_{ij} and $b'_j(k)$) are easily obtainable once the α and β sets are complete.

The β approximation assumes that $\beta_t(i) = \delta(t, i)$ is a continuous function with parameters t (time) and i (state). 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 $t \to 0$. All β terms from $t = T - 1$ to $t = 1$ are less than 1, and with every step that t decreases, $\beta_t(i)$ tends to 0 through the computations of the backward formula (2). Therefore, for large observation sets, we obtain the approximate equality $\delta(t, i) \approx 0 \approx \delta(t, j)$, where i and j are different states. The β approximation is defined as follows:

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

We transform the backward-recurrence formula into a forward-recurrence version using equation (5). First, lets set $N = 2$ and equation (2) gives us:

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

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

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

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

Generalising for state i yields our forward-recurrence β approximation [11]:

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

With the β approximation defined in (7), we execute the FBA α-pass and β-pass, in a forward-recurrence fashion. With both α and β sets evaluated on the new observations $\{O_{T+1}, O_{T+2}, \ldots, O_{T+M}\}$, the ξ and γ values are still defined by equation (3). Since observations are added sequentially, we define the modified re-estimation formulas for HMM parameters $(\hat{A}, \hat{B}, \hat{\pi})$ in incremental steps. Initially, for $i = 1, \ldots, N$, we have $\hat{\pi}'_i = \gamma_1(i)$. For $\hat{A}$, we have:

$$\begin{aligned}
\hat{a}_{ij}^{T+1} &= \frac{\sum_{t=2}^{T} \xi_t(i,j) + \xi_{T+1}(i,j)}{\sum_{j=1}^{N}\sum_{t=2}^{T} \xi_t(i,j) + \sum_{j=1}^{N} \xi_{T+1}(i,j)} \\
&= \frac{\sum_{t=2}^{T} \gamma_t(i)}{\sum_{t=2}^{T+1} \gamma_t(i)} \frac{\sum_{t=2}^{T} \xi_t(i,j)}{\sum_{t=2}^{T} \gamma_t(i)} \;+\; \frac{\xi_{T+1}(i,j)}{\sum_{t=2}^{T+1} \gamma_t(i)} \\
&= \frac{\sum_{t=2}^{T} \gamma_t(i)}{\sum_{t=2}^{T+1} \gamma_t(i)} \hat{a}_{ij}^{T} \;+\; \frac{\xi_{T+1}(i,j)}{\sum_{t=2}^{T+1} \gamma_t(i)}
\end{aligned}$$

Thus, only the new $\xi_{T+1}(i,j)$ and $\gamma_{T+1}(i)$ for O_{T+1} need to be calculated because the $\xi_t(i,j)$ values for $1 \le t \le T$ are stored in the $\hat{a}_{ij}^{T}$ entry. Notice that terms at $t = 1$ have been subtracted from the sums, thus discarding outdated values. For $\hat{B}$, we have:

$$\begin{aligned}
\hat{b}_j^{T+1}(k) &= \frac{\sum_{t=2, O_t=k}^{T} \gamma_t(j) + \sum_{t=T+1, O_t=k}^{T+1} \gamma_t(j)}{\sum_{t=2}^{T} \gamma_t(j) + \gamma_{T+1}(j)} \\
&= \frac{\sum_{t=2}^{T} \gamma_t(j)}{\sum_{t=2}^{T+1} \gamma_t(j)} \hat{b}_j^{T}(k) + \frac{\sum_{t=T+1, O_t=k}^{T+1} \gamma_t(j)}{\sum_{t=2}^{T+1} \gamma_t(j)}
\end{aligned}$$

where updating $\gamma_{T+1}(j)$ (such that $O_{T+1} = k$) is sufficient because previous γ values are included in $\hat{b}_j^{T}(k)$ entries.

Once HMM parameters $(\hat{A}, \hat{B}, \hat{\pi})$ converge, the model is ready for generating synthetic workload traces with an underlying model distribution. The power of the sliding BWA, as part of SlidHMM, is the reduced computation of the forward and backward variables (on new data only) and thus converges model parameters

quicker than the traditional BWA. If we train a model on T new observations k successive times, the additional steps needed for training a standard HMM (S_1) compared to a SlidHMM (S_2) is the difference in steps (i.e. $S_1 - S_2$) given by:

$$\begin{aligned} S_1 - S_2 &= (T + 2T + \ldots + kT) - (T + T + \ldots + T) \\ &= (T \sum_{i=1}^{k} i) - kT \\ &= \frac{Tk(k+1)}{2} - kT \\ &= \frac{Tk(k-1)}{2} \end{aligned}$$

With the SlidHMM methodology evaluated, the other feature of the OnlineHMM is the multi-input adaptation of the BWA, which we describe in the next section.

2.5.2 Multi-input HMM

The multi-input HMM (MultiHMM) [4] comprises of a k-means clustering algorithm and a weighted BWA, which trains on multiple discrete traces simultaneously and maintains accuracy w.r.t. comparisons of trace moments. We present the doubly clustered methodology and the full MultiHMM algorithm.

A simple clustering technique, used for pre-processing input traces for BWA training, is k-means, which groups data points from H traces (i.e. H-tuples) into K distinct clusters [15] ($K \leq H$). Each data point from a chosen input trace belongs to one of C categories, which produces H^C combinations for the H-tuple during clustering. To avoid this high value of combinations, Chis et al reduced H traces to K by grouping together data points from the same cluster [4] and then assigned weights to each trace before BWA training. The pseudo-code for the MultiHMM is provided in algorithm 1, which initialises weights ω_k with equal probabilities (i.e. $\omega_k = 1/K$, where $1 \leq k \leq K$), for all K traces. An extension is to prioritise these weights, according to the strength of individual features of each trace, and define variations of the MultiHMM.

Having defined the multi-input feature, which is adapted from MultiHMM, the next aim is to compose this process with the aforementioned sliding window learning technique, adapted from SlidHMM. The next section merges these two useful properties to achieve a novel OnlineHMM learning algorithm.

3 OnlineHMM preliminaries

In this section, preliminaries of the OnlineHMM are discussed. These preliminaries include trace collection, the binning process, k-means clustering technique and parameter initialisation of the OnlineHMM.

3.1 Netapp and Microsoft traces

We collected millions of I/O commands (i.e. timestamped reads and writes) from NetApp storage servers, from a common Internet file system (CIFS) network trace (of about 500GB). These file servers, located at NetApp HQ, were accessed mainly

Algorithm 1 Training using MultiHMM [4]

Require: K Clusters $\wedge$ H Traces $\wedge$ $size$ = Trace.length $\wedge$ ω_k = weight
  for $i = 1 : H$ Raw Traces **do**
    while Cluster Points not Fixed **do**
      K-means Clustering on Trace$_i$
    end while
  end for
  for $j = 1 : K$ **do**
    $Group_j = \{\}$
    for $t = 1 : size$, $i = 1 : H$ **do**
      if Point$_i(t) \in Cluster_j$ **then**
        $Group_j(t) \leftarrow$ Point$_i(t)$
      end if
    end for
  end for
  while BWA parameters not converged **do**
    for $k = 1 : K$ Observation Traces **do**
      $\hat{\alpha} = \sum_{i=1}^{N} \omega_k \alpha_i$; $\hat{\beta} = \sum_{i=1}^{N} \omega_k \beta_i$; $\hat{\xi} = \sum_{i=1}^{N} \omega_k \xi_i$
    end for
    $\gamma_t(i) = \sum_{j=1}^{N} \hat{\xi}_t(i,j)$; $\pi'_i = \gamma_1(i)$; $a'_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\xi}_t(i,j)}{\sum_{j=1}^{N} \sum_{t=1}^{T-1} \hat{\xi}_t(i,j)}$; $b'_j(k) = \frac{\sum_{t=1, O_t=k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$
  end while

by Windows desktops and laptops using various applications. The two types of traces are therefore denoted as "Netapp reads" and "Netapp writes." Similarly, we collected Microsoft storage data from available servers and divided the sequential I/O commands into reads and writes (i.e. "Microsoft reads" and "Microsoft writes").

With the trace data collected, reads and writes were partitioned into uniform "bins" (i.e. fixed-size intervals). The bin size should represent intervals with significant variance present in data (i.e. avoid too many empty intervals) and one-second intervals were chosen. Once traces are binned, we apply a doubly clustered k-means algorithm [4], as described in section 2.5.2. We chose eleven clusters for initialisation, after optimising the distance-based clustering algorithm during trial runs. After k-means clustering, we obtain a vector of eleven centroid pairs with read and write values, which represent individual clusters. Finally, the raw trace was transformed by assigning the data point (i.e. number of reads or writes per second) to a cluster (i.e. an integer between one and eleven) to which the data point corresponds to. Therefore, this "observation trace" is passed as input for adapted BWA training, which is covered in the next section.

3.2 OnlineHMM training and simulation

The OnlineHMM has two hidden states and the adapted BWA is trained on observations until parameter convergence (i.e. A, B, π become fixed). The online BWA slides along the data set, which is updated incrementally, and trains on multiple traces simultaneously. It is a combination of adapted BWAs from SlidHMM and MultiHMM, but reduces parameter convergence time on both. With each new point added to the data set, an outdated point is discarded by the OnlineHMM whilst parameters are updated. Therefore, an OnlineHMM, having first trained on T observations, trains on M new observations (i.e. in M slides) and now retains information on $T + M$ data points. A standard HMM trains on T data points and then re-trains on $T + M$ points, etc.

For our simulation, we initialise A, B, and π with equiprobable distributions and T ranges from 500 to 100000 seconds. We perform up to 10000 consecutive slides using Netapp and Microsoft reads and writes, with groups of up to 1000 traces. Once the parameters converge, the model generates individual synthetic traces. Note, the same OnlineHMM generates many traces (one-to-many relationship), whereas a standard HMM produces only one trace. This is an obvious advantage of the OnlineHMM over the standard HMM. We obtain mean, standard deviation, skewness and autocorrelation on original and synthetic reads and writes, which we present in the next section.

4 Results

4.1 Mean, standard deviation and skewness

We calculated statistics on discrete traces of Netapp and Microsoft reads and writes using original, HMM and OnlineHMM-generated data points. Particular traces have been simulated 1000 times and corresponding statistics are evaluated per bin with 95% confidence intervals. For example, table 1 shows a mean of HMM-generated Microsoft reads of "48.84 ± 0.08," indicating that the HMM produced, on average, 48.84 reads per second, with a confidence interval of 0.08. Further, "Fourth Microsoft read after no slides" means we chose the fourth trace (out of a group of 1000) of Microsoft reads, where no new reads were added during training (i.e. no slides). A variety of traces are presented in tables 1 to 5. As expected, the HMM provides better estimates of mean and standard deviation. The clustering techniques and backward approximation equation used in the OnlineHMM may be responsible for the difference in these estimates. Nonetheless, the OnlineHMM meets minimum benchmarks for accuracy, sometimes outperforming the standard HMM (tables 1 and 5).

Table 1
Fourth Microsoft read after no slides: Raw, HMM and OnlineHMM

Trace	Mean	Std Dev	Skewness
Raw	49.09	167.49	4.32
HMM	48.84 ± 0.08	164.76 ± 0.16	4.36 ± 0.012
OnlineHMM	49.09 ± 0.14	165.66 ± 0.31	4.29 ± 0.008

Table 2
Sixth Microsoft write after no slides: Raw, HMM and OnlineHMM

Trace	Mean	Std Dev	Skewness
Raw	0.663	0.164	1.019
HMM	0.665 ± 0.002	0.162 ± 0.002	0.96 ± 0.012
OnlineHMM	0.620 ± 0.001	0.219 ± 0.001	1.102 ± 0.009

Table 3
Second Microsoft read after five slides: Raw, HMM and OnlineHMM

Trace	Mean	Std Dev	Skewness
Raw	47.87	166.34	4.36
HMM	46.78 ± 0.25	160.61 ± 0.50	4.51 ± 0.013
OnlineHMM	51.39 ± 0.12	149.21 ± 0.30	4.75 ± 0.009

Table 4
Fourth Netapp write after four slides: Raw, HMM and OnlineHMM

Trace	Mean	Std Dev	Skewness
Raw	0.431	0.124	-0.365
HMM	0.431 ± 0.001	0.122 ± 0.002	-0.409 ± 0.004
OnlineHMM	0.409 ± 0.001	0.214 ± 0.001	-0.269 ± 0.009

Table 5
First Netapp read after nine slides: Raw, HMM and OnlineHMM

Trace	Mean	Std Dev	Skewness
Raw	100.96	248.52	2.54
HMM	102.07 ± 0.60	245.35 ± 0.68	2.6 ± 0.012
OnlineHMM	102.77 ± 0.25	250.81 ± 0.33	2.51 ± 0.004

4.2 Autocorrelation

A major benefit of autocorrelation is observing trends or cycles in the self-correlated time series. As autocorrelation is normalised *autocovariance*, the two terms are, unfortunately, sometimes used interchangeably in industry. The autocorrelation function (ACF) for observations $y_1, y_2, \ldots, y_N$ (with mean $\bar{y}$) is defined as follows:

$$p_k = \frac{\sum_{t=1}^{N-k}(y_t - \bar{y})(y_{t+k} - \bar{y})}{\sum_{t=1}^{N}(y_t - \bar{y})^2} \tag{8}$$

We investigate the ACFs for both Microsoft and Netapp data and compare these raw (i.e. unclustered) data points with the corresponding synthetic traces as generated by the HMM and OnlineHMM.

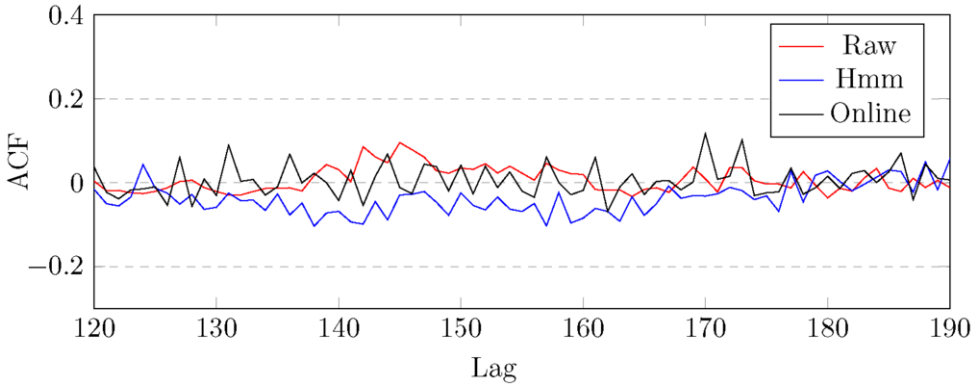

Fig. 2. Autocorrelation for Netapp reads.

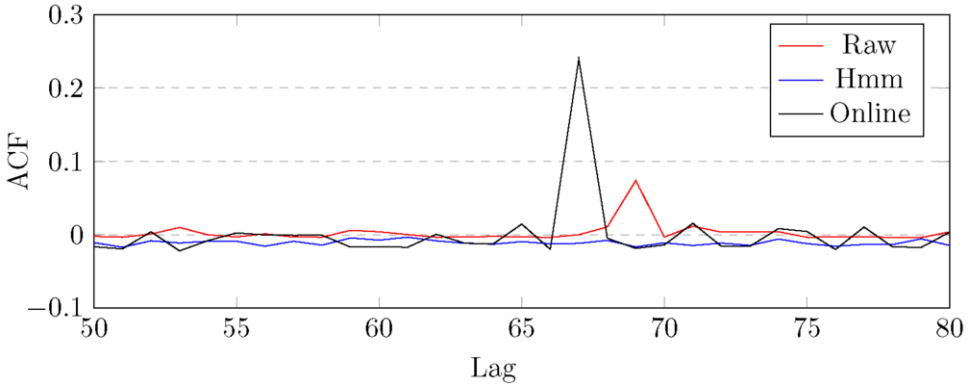

Fig. 3. Autocorrelation for Netapp writes.

Figure 2 shows varied behaviour in ACF for raw, HMM and OnlineHMM Netapp reads. Similarity of raw reads is best captured by the OnlineHMM for lags 140 to 160, where the ACF is above zero. Notably, around lag 170, two jumps in the raw data are enhanced by two spikes in the OnlineHMM, with little correlation from the HMM. Similarly, figure 3 shows the behaviour of raw writes matched by the OnlineHMM (lags 65 to 70), with no correlation from HMM writes. Indeed, this matching of ACFs gives OnlineHMM power as a bursty classifier, with applications in I/O management and resource allocation for disks and file servers.

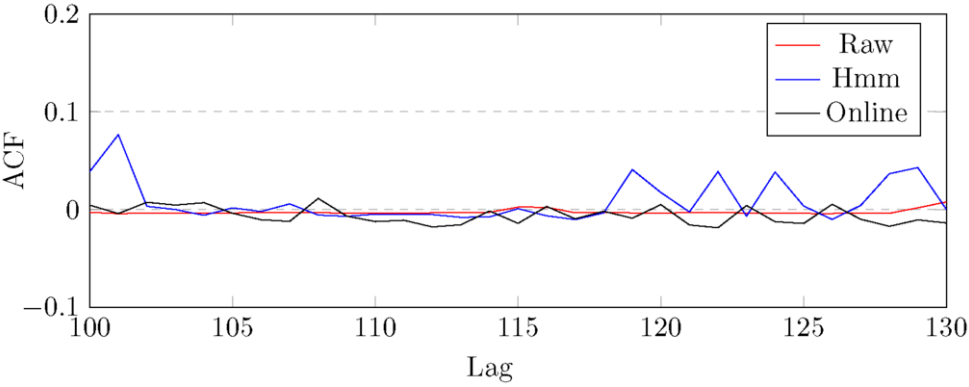

Fig. 4. Autocorrelation for Microsoft reads.

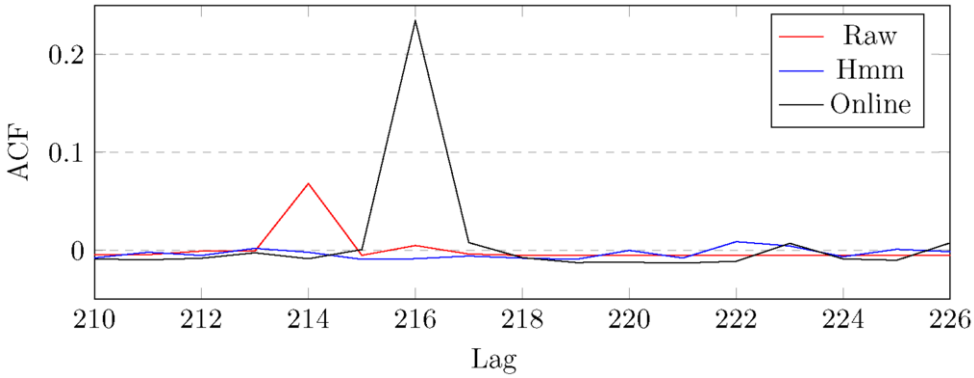

Fig. 5. Autocorrelation for Microsoft writes.

In figure 4, there is little ACF for raw, HMM and OnlineHMM reads. However, at lag 100 and during lags 118 to 130, the HMM reads exhibit jumps in ACF, which differs from a flat ACF behaviour given by both original and OnlineHMM reads. Figure 5 again demonstrates the power of the OnlineHMM, this time for Microsoft writes, in lags 213 to 217; the only model to vaguely represent any significant ACF value of original writes is the OnlineHMM. Moving on from simulated results to more analytical results, the next section presents advantages of the OnlineHMM in terms of convergence of its adapted BWA.

4.3 Convergence of BWA

The space and time complexity for batch learning of a standard HMM (using the BWA) is given by $O(N^2T)$, where T is the trace length and N is the number of hidden states. Table 6 presents analytical results for a two-state HMM and its variations (i.e. SlidHMM, MultiHMM, and OnlineHMM) measuring the following: first, the convergence of BWA when models train on H distinct traces; secondly, BWA convergence with K incremental updates in the data set (i.e. K slides with one new data point added per slide); thirdly, BWA convergence with K slides on H traces. Of all the models, the OnlineHMM is the least affected by scaling of H and K in terms of space and time complexity.

Table 6
Convergence rates for variations of BWA using HMM, SlidHMM, MultiHMM and OnlineHMM

Model	H traces	K slides	K slides on H traces
HMM	$O(HN^2T)$	$O(N^2(T+K(T+\frac{K+1}{2})))$	$O(HN^2(T+K(T+\frac{K+1}{2})))$
SlidHMM	$O(HN^2T)$	$O(N^2(T+K))$	$O(HN^2(T+K))$
MultiHMM	$O(N^2T)$	$O(N^2(T+K(T+\frac{K+1}{2})))$	$O(N^2(T+K(T+\frac{K+1}{2})))$
OnlineHMM	$O(N^2T)$	$O(N^2(T+K))$	$O(N^2(T+K))$

4.4 Mean waiting time of queueing models

Modelling job arrivals (i.e. packets, reads and writes, etc.) at routers, disks and other storage devices can be composed with queueing systems to obtain important system metrics such as mean waiting time (MWT). Queueing models, including server scheduling discplines, add an extra level of abstraction to research and are relatively efficient and cheaper compared to recording live data from real systems. Using the OnlineHMM (after training on Netapp reads and writes) to input rates into an MMPP, we model an MMPP/M/k/FCFS queue: arrivals are given by an MMPP (a version of the OnlineHMM), jobs have exponential service times and the system has 'k' parallel servers, which schedule jobs in a FCFS fashion. This queueing model acts as an abstraction for various modern systems, including routers, switches or even static RAM. Therefore, since latency is a significant factor for performance in all these systems, it is important to obtain MWT measurements. We present values of MWT (in seconds) for the MMPP/M/k/FCFS queue, using multiple servers and varying load (i.e. utilisation), in figure 6.

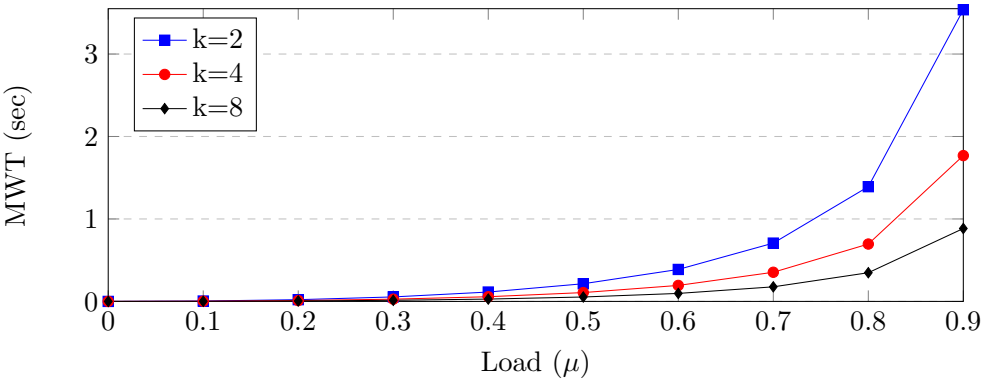

Fig. 6. Mean waiting time (MWT) for MMPP/M/k/FCFS queue.

5 Conclusion

We have analysed variations of HMMs that, combined with clustering analysis and adapted online learning algorithms, provide parsimonious and portable synthetic workloads. By proposing an OnlineHMM that learns efficiently using a sliding data training window and is capable of multi-input evaluation, we have reduced the heavy computing resource requirement of the Baum-Welch algorithm. Further, the OnlineHMM is ideal for modelling workload data in real-time, such as collecting synthetic traces to build a profile of I/O commands at disks or packet arrivals at routers. By analysing long-term behaviour of a live system, through abstracting low-level implementations using queueing theory, the OnlineHMM aims to improve scheduling of jobs, manage system resources and (crucially) bottlenecks.

Obtaining realistic synthetic traces, in terms of matching moments to original data points, has been important for validating accuracy of this research. Also, matching autocorrelation to raw data has validated the dynamics of the generated

workload traces, focusing on the inter-bin correlation. This has added benefits to the OnlineHMM, in terms of observing burstiness and self-similarity for extended periods of time. Despite our mathematical approximation of workloads, the OnlineHMM should cyclically recalibrate (i.e. train a standard HMM every M slides) and train on a variety of traces to obtain a wider synthetic representation base. Nonetheless, from an analytical point-of-view, OnlineHMM outperforms the standard HMM and other variations of HMMs (see table 6) with its adapted Baum-Welch algorithm. Using the OnlineHMM to input rates into an MMPP, which was part of an MMPP/M/k/FCFS queueing system, was useful to obtain mean waiting times for storage devices with increasing load, whilst scaling up the number of servers.

Possible extensions to the OnlineHMM include using hierarchical clustering to improve the cluster allocation during data pre-processing. Currently, a challenge with k-means is choosing an optimal value for the initial number of clusters, which affects the model performance in generating accurate traces w.r.t moments. Also, we plan to adapt the OnlineHMM training algorithm to use varying weights for each trace, which gives priorities to groups of traces and is useful for scheduling important jobs to servers. Extensions to the MMPP/M/k/FCFS queueing system include adding different rates for service times (i.e. have a hyper-exponential distribution) and change the FCFS scheduling discipline to processor sharing (PS), thus catering for a wider range of servers. Another addition to our work is obtaining a continuous OnlineHMM, capable of training on continuous time data. This would require a continuous version of the Baum-Welch algorithm, as seen in [5], and would act as an online, sliding algorithm for a continuous time series.

References

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

[2] L. E. Baum, T. Petrie, G. Soules, N. Weiss: A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, In *The Annals of Mathematical Statistics*, **41**, p. 164-71, 1970

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

[4] T. Chis, P. G. Harrison: Modeling Multi-User Behaviour in Social Networks, In *Proc. IEEE MASCOTS*, 2014

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

[6] T. Chis, P. G. Harrison: Incremental HMM with an improved Baum-Welch Algorithm, In *Proc. OASIcs ICCSW*, 2012

[7] A. J. Viterbi: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, In *IEEE Transactions on Information Theory*, **13**, p. 260-69, 1967

[8] T. Osogami: Analysis of Multi-Server Systems via Dimensionality Reduction of Markov Chains, Computer Science Department, Carnegie Mellon University, 2005

[9] T. Chis: Sliding Hidden Markov Model for Evaluating Discrete Data, In *Proc. Springer EPEW*, 2013

[10] G. Casale: Building Accurate Workload Models Using Markovian Arrival Processes, In *ACM SIGMETRICS Tutorial*, June, 2011

[11] T. Chis, P. G. Harrison: iSWoM: An Incremental Storage Workload Model using Hidden Markov Models, In *Proc. Springer ASMTA*, 2013

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

[13] H. Y. Wei, S. C. Tsao, Y. D. Lin: Assessing and Improving TCP Rate Shaping over Edge Gateways, In *IEEE Transactions*, **53**, p. 259-75, 2004

[14] C. X. Zhai: A Brief Note on the Hidden Markov Models (HMMs), Department of Computer Science, University of Illinois at Urbana-Champaign, 2003

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

[16] J. Domanska, A. Domanski, T. Czachorski: A HMM Network Traffic Model, In *Proc. ICNFI*, p. 17-20, 2012

[17] S. L. Scott, P. Smyth: The Markov Modulated Poisson Process and Markov Poisson Cascade with Applications to Web Traffic Data. In *Bayesian Statistics*, **7**, p 671-80, 2003

[18] P. Salvador, A. Pacheco, R. Valadas: Modeling IP traffic: Joint Characterization of Packet Arrivals and Packet Sizes using BMAPs, In *Computer Networks*, **44**, p. 335-52, 2004.

[19] V. Raghavan, G. Steeg, A. Galstyan, A. Tartakovsky: Coupled Hidden Markov Models For User Activity In Social Networks, In *Proc. IEEE ICME*, p. 1-6, 2013