Department of Computing
Imperial College London

Performance Modelling with Adaptive Hidden Markov Models and Discriminatory Processor Sharing Queues

Tiberiu Chis

Submitted in part fulfilment of the requirements for the degree of Doctor of Philosophy in the Department of Computing at Imperial College London

July 25, 2016

Declaration of originality

I declare that this thesis was written by myself and the work presented here is my own, unless otherwise stated.

Copyright declaration

Abstract

In modern computer systems, workload varies at different times and locations. It is important to model the performance of such systems via workload models that are both representative and efficient. For example, model-generated workloads represent realistic system behaviour, especially during peak times, when it is crucial to predict and address performance bottlenecks. In this thesis, we model performance, namely throughput and delay, using adaptive models and discrete queues.

Hidden Markov models (HMMs) parsimoniously capture the correlation and burstiness of workloads with spatiotemporal characteristics. By adapting the batch training of standard HMMs to incremental learning, online HMMs act as benchmarks on workloads obtained from live systems (i.e. storage systems and financial markets) and reduce time complexity of the Baum-Welch algorithm. Similarly, by extending HMM capabilities to train on multiple traces simultaneously it follows that workloads of different types are modelled in parallel by a multi-input HMM. Typically, the HMM-generated traces verify the throughput and burstiness of the real data. Applications of adaptive HMMs include predicting user behaviour in social networks and performance-energy measurements in smartphone applications.

Equally important is measuring system delay through response times. For example, workloads such as Internet traffic arriving at routers are affected by queueing delays. To meet quality of service needs, queueing delays must be minimised and, hence, it is important to model and predict such queueing delays in an efficient and cost-effective manner. Therefore, we propose a class of discrete, processor-sharing queues for approximating queueing delay as response time distributions, which represent service level agreements at specific spatiotemporal levels. We adapt discrete queues to model job arrivals with distributions given by a Markov-modulated Poisson process (MMPP) and served under discriminatory processor-sharing scheduling. Further, we propose a dynamic strategy of service allocation to minimise delays in UDP traffic flows whilst maximising a utility function.

Acknowledgements

I would sincerely like to thank Professor Peter Harrison for allowing me to be his PhD student. The research achieved in this thesis would not exist without his support, wisdom and contributions. Professor Harrison has guided me through difficult concepts in queueing theory and has offered valuable experience in the world of research and beyond! I will always cherish my time at Imperial College, from the early days as an indecisive undergraduate joining in 2007 to more than eight years later as a PhD candidate submitting his thesis. Throughout my struggles and (many) lessons learnt, these years in academia were truly golden and I thank Professor Harrison for giving me the chance to be a part of it.

I am also grateful to Professor William Knottenbelt for being my second supervisor during the PhD. He provided expert advice, has led by example and often encouraged me during the conferences and workshops we attended together. I would like to thank the many members of my research group, past and present, that have bettered my experience and taught me many important skills. I am proud to call the following colleagues my friends, in order of acquaintance: Anton, Chris, Tony, Jeremy, Giuliano, Gareth, Uli, Richard, Nigel, Iryna, Zhan, Weikun, Rasha, Juan, April, Silvia, Nicolai, David, Ilya, Victoria, Daniel, Pooyan, Andrea, Salvatore, Tanya and Ruth. An honourable mention goes to the reliable and helpful Amani, the friendly support staff including Ann, Barbara, Bridget, Teresa, Sarah, Mark, and the Computer Support Group including Geoff, Lloyd, and Duncan. Further, I would like to thank my DramSoc friends including Tosin, Kristen, Owain, Megan, Simon and, more recently, Fahdi from ICTV. These talented individuals allowed me to act alongside a great bunch of people in theatrical plays and short films, which I will always treasure with fond memories.

Last but not least, thank you to my amazing family, Liliana, Diana, Mircea and my grandparents, for their love, inspiration and putting up with me for the last 27 years! Without you, I would not be here. Thank you to my wonderful girlfriend, Florence, for her love and support and to her awesome family, Luke, Leonie and Ella. Another thank you goes to my legendary friends from Imperial and beyond, which include Levitt, Mikey, James, Chris Marsh, Alexandra, Dino, Jay, Anna, Hemal, Eemaan, John Paul, Gajan, Richard, Karin, Mitesh, Tom, Nic, Vera, Ravi, Frank, Christina, Jamie, Lindsay, Paul, Sinead, Ziggy, Mark, Shamini, Rebecca, Jack, Mira, Patrick and all the usual suspects. You have made these last few years a blast!

Contents

List of Figures

List of Tables

Notation

N = number of HMM states or MMPP phases.
K = number of clusters or job classes.
π = initial state distribution for an HMM.
A = state transition matrix for an HMM.
B = observation matrix for an HMM.
O_t = observation at time t.
$\alpha_t(i)$ = forward term of forward-backward algorithm.
$\beta_t(i)$ = backward term of forward-backward algorithm.
$\xi_t(i, j)$ = transition at time t between state i and j used in Baum-Welch algorithm.
$\gamma_t(i)$ = transition at time t from state i used in Baum-Welch algorithm.
T = response time of a system.
L = mean number of jobs in the system.
λ = mean arrival rate.
λ_j = mean arrival rate for a class j job.
λ_{ij} = mean arrival rate for a class j job from MMPP state i.
μ_j = mean service rate for a class j job.
ρ_j = utilisation for a class j job.
w_{ij} = superposition weights for MMPP state i for a class j job.
p_i = invariant probability for MMPP remaining in state i.
q_{ij} = MMPP rate from state i to state j.
α_j = priority weights for a class j job under DPS.
G^{*x} = the Laplace transform of G with respect to x.
$S_j(x)$ = elapsed response time for class j jobs after tagged job attains service x.
$N_j(x)$ = number of class j jobs in the system after tagged job attains service x.
$T(\cdot)$ = transform for joint distribution of response time and number of jobs.
$\mathbb{E}[T^k]$ = the k^{th} response time moment.
$\mathbb{E}[T_j^k]$ = the k^{th} response time moment for a class j job.
$\mathbb{E}[T_{ij}^k]$ = the k^{th} response time moment for a class j job from MMPP state i.
$P(T = x)$ = response time probability density function.
$P(T < x)$ = response time probability distribution function.

Chapter 1

Introduction

Chapter Description

Section 1.1 introduces motivating challenges of *performance* in modern computer systems and justifies *workload models* and *queueing models* as cost-effective solutions. Objectives in section 1.2 argue that adaptive workload models (1.2.1) and queueing models (1.2.2) represent spatiotemporal behaviour, save resources and respect service level agreements. Section 1.3 outlines the thesis structure and we present related publications in section 1.4.

1.1 Motivation

The *performance* of modern computer and communication systems is a key issue for international enterprises with a global online presence [21, 27]. Institutions and businesses increasingly face technical challenges that system performance imposes on critical IP applications, remote desktops or video conferencing [23]. Such challenges include lag in waiting times, availability and congestion with real-time, spatiotemporal-varying system conditions and user behaviour. Individual users demand availability, security and consistent performance of applications on large platforms and also on tablets and smartphones. For example, smartphone users wait, on average, approximately nine seconds for a web page to load [144] before opting for more reliable performance from competitors. Whether it is downloading files on smartphones using Wi-Fi or streaming web content on a virtual cloud environment, the delay principle still applies. To meet quality of service (QoS) goals (i.e. reducing queueing delays), application developers and content providers aim for short *response times* (or latency, i.e. the time between a job arriving and leaving the system) to minimise performance bottlenecks [7]. Typically, stochastic models are a simplistic representation of delays, given certain assumptions, and simulation offers a means of verifying analytical results. However, simply aiming to minimise mean response time (i.e. the average time users wait for one or more tasks to complete) is usually not acceptable nowadays because users tend to be equally frustrated with a highly variable service. Thus, users demand response time that is *predictable* [31] and, hence, it is desirable to obtain response time moments and distributions of traffic data in different parts of the system and at different times. Queueing models offer tractable solutions for approximating response time moments and also abstracting complex processes of networks

and storage systems. Further, queueing models represent queueing delays by approximating response time distributions that are measured against QoS goals set by realistic service level agreements (SLAs) from respective clients and vendors. For example, an SLA could demand that 95% of the time, a type of request will complete within 0.05 seconds at a specific part of the system. However, maintaining consistent performance for all parts of the system, by continuously upgrading infrastructure or increasing bandwidth, is expensive to match a large user-base with heavy demand [6]; a similar problem faces cloud service providers (CSPs) with respect to storage and service delays in data centres. In cloud environments, oversubscribing shared resources such as disks may be costly with a risk of failing strict SLA requirements [24]. Hence, a target is to obtain a class of queueing models to act as benchmarks for predicting and improving spatiotemporal performance at peak times.

Modern server systems with multiple applications and traffic from many servers need to maintain adequate performance for all applications. Despite server virtualisation reducing the number of server systems handling multiple applications, traffic to shared storage is largely varied due to operations and integration between virtualised systems [66]. This leads to a mixture of traffic flows at different time periods from many servers, and thus it is advantageous to characterise such *workloads*. In general, characterising storage workloads is useful for performance evaluation including measurement, simulation and (if possible) analytical modelling. For measurement and simulation, I/O traces can be collected from production systems. However, such traces are often very large (i.e. many gigabytes, which prolong download times), difficult to obtain and might be outdated [72]. Hence, an alternative is building a class of parsimonious workload models with few parameters, which are used to generate multiple representative traces with key characteristics. Such models are comparatively cost-effective in terms of resources than the aforementioned production systems used to collect large I/O traces. Indeed, obtaining key workload parameters (e.g. job arrivals or throughput) from traces is highly desirable, where workload models may be used in further experiments with new storage system designs (i.e. hybrid Flash and disk) [64]. Further, characterising I/O workloads offers possible answers for the demand exerted on data centres and allows construction of accurate performance models.

Workloads should typically represent time-varying correlated traffic streams that might lead to possible resource bottlenecks in different parts (or layers, as with Flash) of the system. Hence, realistic workload models should reproduce such behaviour in an economic and efficient manner, thereby incorporating important features of traffic modelling [81] that include: to be accurate, to capture spatiotemporal correlation, and to be fast at fitting. Many workload benchmarks [66, 107, 109, 110] have been constructed for profiling and system modelling in recent decades. Such benchmarks often train parameters on traffic data in batches and, hence, static learning is inadequate for obtaining characteristics of live systems due to costly computational requirements in the re-training process. Hence, we seek improvements in workload models to obtain incremental benchmarks that reduce computational complexity by training parameters online and, thus, offer runtime analysis of live systems. Additionally, it is desirable to build workload models capable of

treating customers with different classes. Multi-class models train on multiple workloads simultaneously with reduced resources and measure correlation.

Minimising *energy consumption* whilst maintaining acceptable performance is a crucial tradeoff for large storage systems and is incorporated into environmentally-friendly SLAs. Particularly, companies such as AISO.net use green technology resources via solar-powered hosting [115]. More generally, CSPs aim for environmentally-friendly cost minimisation for data centres with sustainable goals (i.e. reduce carbon emission footprint) and exploit the locational, time-varying fluctuation of electricity prices. Whilst saving operational costs in the long-term, there might be additional short-term queueing delays when re-distributing user requests to cheaper data centres, which may reduce QoS standards. Hence, queueing models with performance-energy measurements are useful in analysing energy cost of large-scale systems with conflicting performance goals [132]. With mobile technology growing rapidly, cloud services such as software-as-a-service (SaaS) and platform-as-a-service (PaaS) are expected to run efficiently on devices with full functionality and integrate with other applications. As these services scale with the number of active devices, smartphone and tablet users must adhere to recommended power consumption guidelines to save energy costs and protect battery life of device [164]. Hence, incorporating power consumption into performance models is a key target for planning efficient use of data transfers given user charging patterns. On a larger scale, distributed storage systems must address both performance guidelines to meet SLAs and reduce energy consumption for long-term sustainable goals.

1.2 Objectives

To address the aforementioned challenges and adapt existing workload models, we propose a combination of solutions, as justified in this section. The solutions translate into two main objectives of this thesis, which we list as follows:

- To build adaptive workload models that extract key characteristics from system traces with comparatively reduced resources, can train efficiently on live production systems with multiple streams and act as representative workload benchmarks.
- To build analytical queueing models that represent important characteristics of system performance, abstract real-world complex processes and approximate queueing delay through response time moments and distributions.

First, we introduce the features of the adaptive workload models. Secondly, we summarise the queueing models, which approximate important performance measurements from diverse systems. In each subsequent section, we clarify which of the aforementioned challenges have been addressed with the solution.

1.2.1 Adaptive workload models

In modern, large-scale, computer and communication systems, workload arises from multiple, time-varying, correlated traffic streams affecting different parts of the system.

Therefore, it is important to categorise and model workload in a portable and efficient way for at least four purposes:

1. To generate similar traces for system simulation.

2. To train on live systems using fewer resources.

3. To model multiple workloads simultaneously.

4. To provide input parameters for analytical performance models.

By addressing all four purposes, our aim is to build portable benchmarks to characterise workloads in diverse computer systems at different times. Building such benchmarks in terms of time-series models such as moving averages is often easy and offers fast training times, but ignores higher moments, omits seasonal trends, and smooths spikes in time-series [87]. Further, autoregressive moving average (ARMA) models [92], which are also popular in forecasting [90, 91], suffer from similar drawbacks including assuming stationarity of time-series and normality of the residuals [88]. On the other hand, the Markov-modulated Poisson process (MMPP) and the hidden Markov model (HMM) offer parsimony, accurate capture of correlation and burstiness in multi-application workloads. Such workloads might be pre-scheduled to optimise system utilisation, access time and availability. Simple stochastic models, such as Poisson processes, cannot provide realistic tools for modelling time-series of complex systems including Internet traffic or storage access, failing to account for long-range dependency (LRD) or burstiness. Therefore, to improve this, researchers are turning their attention to more complex models, such as MMPPs and HMMs [66, 86, 174]. In fact, the MMPP may also be viewed as a discretely-indexed non-stationary HMM by observing intervals between events as a sequence of dependent random variables [100]. The HMM has been successful in accounting for correlation, self-similarity and burstiness of jobs through its mode-switching capabilities [69, 83]. Applying similar models, such as the Kalman filter and the extended Kalman filter [89], are useful when the state space of the time-series is continuous and (both latent and observed) variables have Gaussian distributions. A benefit of the HMM is that it does not assume Gaussian distributions for the dynamics of the workload.

The first purpose of modelling and categorising workload is to generate traces for simulating real-world systems. The Markov chain that forms part of the HMM represents workload dynamics via modes of its generator matrix. This switching between modes offers advantages over existing models such as Box-Jenkins [49, 54] and spectral analysis [61, 101] and is represented by transitions between HMM states (i.e. evolving as a Markov chain), which is typically captured at various timescales in the time-series. Hence, HMMs are portable, comparatively cheap and parsimonious models that generate multiple representative traces with key workload characteristics to simulate realistic system behaviour, thus addressing the first purpose.

We address the second purpose by adapting HMMs to train on data in an incremental

fashion. It takes considerable system resources such as time, training-data and computing power to produce a reliably parametrised model. Hence, an incremental approach is appealing in terms of its run-time performance because a model's parameters are progressively updated rather than periodically re-calculated [66]. Further, the need to learn workload data in an online manner (i.e. "on-the-fly") is particularly useful for live systems where latency has a significant impact for users (i.e. social media and high frequency trading) [30]. The question is, therefore, are such incremental HMMs accurate? We seek an answer from several directions: validation through statistical moments (i.e. mean, standard deviation, skewness, etc.) of HMM-generated data compared to original data; obtain mean absolute percentage error of incremental HMMs and compare with standard HMMs; generate graphs of original data compared with forecasts produced by the incremental model. Further, such prediction using incremental learning of HMMs can be combined with energy measurements to analyse important performance-energy tradeoffs that are relevant in, for example, smartphone applications where data usage influences battery consumption [56].

The third purpose is addressed by constructing a multi-input HMM to train on different traces, where clustering is used as initial configuration. Typically, HMM training (via the Baum-Welch algorithm) is executed per trace and a useful upgrade would be to obtain a model capable of training on multiple traces simultaneously, thus modelling multiple workloads. Further, the capability for training models on multiple workloads is growing in importance with the analysis of online interactions on social media and across systems with shared resources. For example, variations of HMMs have identified trends in group activity on Twitter, such as coupled HMMs [82], which represents each user as a Markov chain and the coupling of chains as the social interaction. However, scaling to large data sets adds computationally-expensive coupling and, hence, this model is unrealistic in real-world applications of online social interactions. A solution is to build a scalable, adaptive multi-input HMM, which trains on multiple users in a resourceful manner without significantly reducing accuracy of synthetic traces produced [5]. Further, we model burstiness and correlation between users in social networks and recognise trends in groups of users.

The fourth purpose concerns input parameters obtained from workload models to be used in analytical performance models. We build such analytical models using queueing theory [7] and discuss the advantages offered towards obtaining realistic performance measurements (i.e. response time moments) to address SLA requirements, with important energy tradeoffs. Incorporating adaptive workload models with queueing models provides real-world benefits to storage systems, social media, and financial strategies, to name but a few. We dedicate the subsequent section to justifying analytical queueing models for approximating performance measures in real-world systems.

1.2.2 Analytical queueing models

In many modern systems, response time is a key measurement for approximating delay in different queueing scenarios and, hence, is representative of system performance [21].

Queueing delay typically arises when packets are made to wait at routers before being redirected to respective destinations. Traffic in a packet-switching network is *bursty* and delay can be averaged in two ways; as averages across all bursts or as averages within bursts. However, both such measurements ignore variability of packet burstiness. Therefore, it is important to consider the second moment (and higher) of delays at peak times, when throughput is highest and has a direct effect on utilisation. Response time distributions are approximated from moments and may be compared to SLA delay requirements to ensure performance goals of the system [145]. As part of QoS aims, there exist queue management techniques that ensure the high-priority, delay-sensitive traffic has minimal mean delay curve [146]. This is crucial for providing QoS to network traffic as the queueing delay is minimised for important jobs such as voice data. Further, measuring spatiotemporal delay (i.e. in different parts of the system at different times) benefits planning of resource allocation for large-scale storage systems, mobile technology, wireless sensor networks, etc.

Queueing models provide a tractable solution for modelling queueing delay and can approximate response times under different system conditions. When modelling queueing delays, it is imperative to avoid averaging delay for only one class of job and, therefore, priorities should be implemented in queues to facilitate multiple job classes (i.e. typically two to five levels of priority is common). Hence, discrete queueing models approximate higher moments of response time for multiple classes of jobs, whilst providing an analytical alternative to computationally-expensive simulation. Further, we utilise underlying continuous time Markov chain (CTMC) properties of queueing models in order to abstract dynamic system processes and obtain representative performance measures. Fluid queues, which were first used by Pat Moran [36], also rely on a CTMC as a generator and can approximate discrete queues, but allow continuous arrivals and typically utilise the moments of busy periods. Further, fluid queues have modelled the performance of network switches and routers [116]. For Internet applications including packets arriving at a router, it is reasonable to use discrete queues to model the packet arrivals and service requirements (i.e. discrete packet sizes). For example, in ideal scheduling algorithms such as generalised processor-sharing (GPS), the traffic modelled is assumed to be fluid, which does not match the actual (discrete) packet sizes.

Parametrising a discrete queueing model with key rates obtained from system measurements depends on the distribution of the processes under investigation. For example, using Kendall notation [11], the M/M/1 queue has Poisson arrivals, exponential service times and one server. Hence, in a simplified scenario of packets waiting at a router, we parametrise the M/M/1 queue as follows: the mean arrival rate (λ) is given by the average throughput of the packets and the mean service rate (μ) is given by transmission rate divided by the average packet size. An important assumption we make in this scenario is steady state on arrival (i.e. the utilisation is less than one). Once the queueing model is parametrised, analytical approximations of response time (i.e. representing queueing delay) and queue length distribution are calculated. Further, more complex arrival distributions (i.e. MMPP-induced arrivals) can be represented in queueing models

with a variety of scheduling disciplines such as first-come first-served (FCFS) and the equally-weighted processor-sharing (PS). Additionally, queues that schedule jobs with discriminatory processor-sharing (DPS) represent heterogeneous time-sharing systems and are popular for modelling buffers in routers and sizing links in transport networks [34]. We discuss the improvements and extensions of existing M/M/1-PS queueing models [151, 155] offering more realistic scenarios to represent Internet traffic at routers and workloads from storage systems.

We have briefly described how queueing models allow us to approximate queueing delay, achieve fairness and address QoS requirements in an efficient and mathematically tractable manner. We summarise the novel contributions for queueing models achieved in this thesis:

- Build a class of M/M/1-PS queues that abstract complex system processes and obtain explicit equations for higher response time moments for a single job class.
- Construct an automated moment-generating algorithm to obtain higher response time moments in multi-class, M/M/1-DPS queues for scenarios of different loads.
- Adapt a weighted superposition technique to obtain higher response time moments in MMPP/M/1-DPS queues from a numerical algorithm, which calculates moments iteratively. Form a dynamic allocation strategy using MMPP/M/1-DPS queues for modelling traffic flows at routers whilst minimising delay.
- Use the mean response time to approximate average delay in smartphone applications, which is combined with power consumption in an objective cost function to address a performance-energy tradeoff.

In the research reported in this thesis, we compare analytical results with numerical results to validate our queueing models. In one respect, analytical response time moments produced by a class of discrete queues should be compared to response time moments simulated in a real-world environment (or obtained from simulating models with representative parameters). On the other hand, we obtain numerical results on real data and also parametrise our queueing models with realistic inputs such as mean arrival and service rates from system traces of real traffic. Note that the analytical queueing models relate only to single queues that approximate response time moments and do not obtain closed-form network solutions. In the next section, we outline the thesis structure by chapters and summarise the novel contributions made to research. We support our contributions through peer-reviewed publications.

1.3 Thesis outline and contributions

In this section, we outline the structure of the thesis by including the main chapters and highlight the contributions, which correspond to publications in section 1.4.

Chapter 2: Background

The background covers a range of relevant topics in machine learning and queueing theory including: an introduction to model selection for unsupervised learning; advantages and disadvantages of clustering and justification of using the k-means algorithm; definition of HMMs and relevant HMM applications in storage systems [66], modelling arrivals [4], social media [82] and finance [74]; definition of MAPs and MMPPs; an introduction to fundamental queueing theory including M/M/1 queues and scheduling disciplines; existing work on response times for processor-sharing disciplines; abstractions of queueing models to measure performance in modern computer systems. To support the models used in this thesis, we describe wider applications in smartphone applications, battery models, cloud computing and data centres.

Chapter 3: Adaptive HMMs

This chapter provides detailed algorithms for modifying standard HMM processes to build parsimonious workload models that are adaptive with the following distinct benefits: through incremental learning of discrete traces, where HMM parameters are updated "on-the-fly," we obtain the **IncHMM** [1, 2] with reduced Baum-Welch convergence rate; improving the IncHMM with a fixed sliding window that discards obsolete data points as new data points are trained on the model, we form the **SlidHMM** [3]; multi-dimensional learning on groups of traces via a doubly-clustering k-means algorithm builds the **MultiHMM** [5], which reduces computational complexity of training a standard HMM sequentially per trace for multiple iterations; combining features of the SlidHMM and MultiHMM allows efficient online learning of multiple (simultaneous) traces in the **OnlineHMM** [6]. For each adaptive model, we provide specific modifications of HMM-based algorithms, present results on varied data sets from diverse applications and discuss potential improvements.

Chapter 4: Queueing models

We begin with an **M/M/1-EPS** queue, for which we obtain explicit response time moments under egalitarian processor-sharing (EPS) via an iterative algorithm [7]. To include multiple job classes, we extend this moment-generating algorithm for discriminatory PS (DPS) disciplines using the constructs of an **M/M/1-DPS** queueing system [8]. Hence, we approximate workload in smartphone activity and servers in data centres, obtaining explicit response time moments as important performance measures. Improvements to the M/M/1-DPS model include using an MMPP to distribute the multi-class queueing arrivals and obtain corresponding response time distributions to address SLAs. Hence, we build an **MMPP/M/1-DPS** queue [9]. We obtain higher response time moments and distributions using a weighted superposition of M/M/1-DPS queues under the assumption of slow switching phases of the MMPP. We parametrise the MMPP/M/1-DPS queue with traces of CAIDA IP traffic. Possible applications of this model include verifying spatiotemporal SLA requirements for Internet traffic.

Chapter 5: Applications

In this chapter, we analyse applications using our adaptive models to give the reader some ideas of relevant areas that our work may contribute to. First, we investigate the performance-energy tradeoff in **smartphone applications**, which consists of an HMM-based power consumption model and a tradeoff involving mean delay and energy consumption [10]. Smartphone data is collected from over 700 users with 100 different handsets from the OpenBattery application [124], which we use to draw conclusions about usage patterns and battery consumption of different users. Secondly, a dynamic allocation strategy uses a UDP flow-level model to minimise delay whilst maximising a utility function. Specifically, this application involves applying the MMPP/M/1-DPS queueing model to build the UDP flow-level model (aka **FlowDPS**) and compare the performance of the FlowDPS model to a simplified, static deficit weighted round robin (DWRR) scheduling algorithm.

1.4 Related publications

During my PhD (April 2012 - November 2015), I wrote the following peer-reviewed publications:

[1] Tiberiu Chis, Peter G. Harrison: **Incremental HMM with an improved Baum-Welch algorithm**, In *Proceedings of the 2nd Imperial College Computing Student Workshop (ICCSW)*, London, UK, 2012.

This workshop paper introduces two techniques for adapting the backward formula of the Baum-Welch algorithm to achieve incremental learning of HMMs. This incremental HMM (IncHMM) uses k-means clustering to pre-process discrete data and, once trained, outputs synthetic traces that we compare statistically via mean and standard deviation to original, unclustered traces. Despite achieving encouraging results, these preliminary IncHMM experiments are valid for one only data set.

[2] Tiberiu Chis, Peter G. Harrison: **iSWoM: The incremental Storage Workload Model using Hidden Markov Models**, In *Proceedings of the 20th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA)*, Ghent, Belgium, 2013.

This conference paper improved the techniques introduced in [1] by building an incremental storage workload model using the IncHMM. Results on two different data sets compare mean and standard deviation of original and IncHMM-generated storage traces, including 95% confidence intervals after many simulation runs. Despite reducing the computational requirement of the Baum-Welch algorithm, this work seeks an improvement in maintaining the size of training sets and providing an objective error measurement of the IncHMM versus the standard HMM.

[3] Tiberiu Chis: **Sliding Hidden Markov Model for Evaluating Discrete Data**, In *Proceedings of the 10th European Workshop on Performance Engineering (EPEW)*, Venice, Italy, 2013.

This workshop paper improves online training of IncHMM by incorporating a sliding window, hence forming a sliding HMM (SlidHMM). This new feature reduces the size of the observation set by discarding old data points as new ones arrive, which minimises convergence rates for the Baum-Welch algorithm. Metrics for validating results are similar to [2] and we introduce new metrics measuring the training steps saved with SlidHMM over the standard HMM.

[4] Tiberiu Chis, Peter G. Harrison: **Analysing and Predicting Patient Arrival Times**, In *Proceedings of the 28th International Symposium on Computer and Information Sciences (ISCIS)*, Paris, France, 2013.

This symposium paper applies HMMs to traces of hospital patient arrivals, providing similar validation measures as introduced in [2]. Additional features of the paper include: autocorrelation results on original and HMM-generated traces; decoding HMM-generated traces using the Viterbi algorithm; and methods for obtaining the optimal number of hidden states via merging.

[5] Tiberiu Chis, Peter G. Harrison: **Modeling Multi-User Behaviour in Social Networks**, In *Proceedings of the 22nd IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Paris, France, 2014.

This conference paper introduces a multi-input HMM (MultiHMM), capable of training on multiple traces simultaneously and reducing training time compared to a standard HMM. We collected Twitter user activity and pre-processed traces using a doubly-clustering k-means algorithm. We validated the MultiHMM against standard HMM-generated traces using moments, autocorrelation, mean absolute percentage error (MAPE) and Baum-Welch algorithm convergence rates. The MultiHMM provides promising applications for recognising group trends on social media, planning resource allocation for social networks and building user profiles for security.

[6] Tiberiu Chis, Peter G. Harrison: **Adapting Hidden Markov Models for Online Learning**, In *Elsevier Electronic Notes in Theoretical Computer Science (ENTCS)*, **318**, pp. 109-127, 2015.

This journal paper introduces the hybrid OnlineHMM, which combines methodologies of the IncHMM and the MultiHMM and, thus, trains on multiple traces simultaneously in an incremental fashion. A significant advantage of the OnlineHMM over existing HMMs is the reduced convergence rate of the adapted Baum-Welch algorithm trained on discrete time-series. We validate the OnlineHMM via centralised moments calculated on model-generated time-series and on empirical data. Further, we obtain accurate autocorrelation results from OnlineHMM-generated traces. A possible extension is using the switching rates from a converged OnlineHMM as

inputs into an MMPP, which acts as a bursty job arrival process in, for example, multi-tiered storage systems.

[7] Tiberiu Chis, Peter G. Harrison: **Moment-Generating Algorithm for Response Time in Processor Sharing Queueing Systems**, In *Proceedings of the 12th European Workshop on Performance Engineering (EPEW)*, Madrid, Spain, 2015.

This workshop paper introduces a new moment-generating algorithm for calculating response time explicitly in processor-sharing (PS) queues, which is extended for multiple job classes. The algorithm is presented in full, as implemented in Mathematica, and iteratively calculates the explicit first four moments of response time for PS queues with one job class. Additionally, we plot the first two moments of real-world traces exhibiting two distinct job classes. Our algorithm methodology is compared with existing work and extensions are offered.

[8] Tiberiu Chis, Peter G. Harrison: **Higher response time moments for M/M/1 discriminatory processing-sharing queues**, In *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, Berlin, Germany, 2015.

This short paper presents an automated algorithm to yield higher response time moments in M/M/1-DPS queues with multiple job classes. We obtain up to four moments of response time with two distinct job classes. The results in this paper reveal that analytical approximations match simulated moments well under low and high utilisation. Applications of this numerical algorithm include modelling multi-class Internet traffic, where delay addresses QoS constraints, and spatiotemporal resource allocation in networks.

[9] Tiberiu Chis, Peter G. Harrison: **Modeling Packet Delay and Traffic Flow with MMPP/M/1 Discriminatory Processing Sharing Queues**, *Submitted for publication*, 2015.

This paper provides a methodology for approximating response time moments and distributions for MMPP/M/1-DPS queues. First, an automated moment-generating algorithm is applied to yield higher moments of multiple job classes in M/M/1-DPS queues. Then, we adapted a weighted superposition technique to M/M/1-DPS queues for estimating the response time in MMPP/M/1-DPS queues, under the approximating assumption of slow switching phases of the MMPP; the analytical approximations match simulation results fairly well, but failed in some cases for high loads. Further, we utilise our response time approximations to represent packet delay in routers and as part of a flow-level allocation strategy. In doing so, we prove that DPS scheduling algorithms, coupled with MMPP bursty properties, are efficient at service differentiation at flow level (under simulated scenarios).

[10] Tiberiu Chis, Peter G. Harrison: **Modeling Performance and Energy in Smartphone Applications**, *Submitted for publication*, 2015.

This paper investigates the performance-energy tradeoff of smartphone applications by following two strategies: first, a power consumption model forecasts battery

draw using the OnlineHMM; secondly, a tradeoff structured as an objective cost function incorporates mean delay and energy consumption. For both strategies, logged smartphone activity for over 700 users is collected using an open source smartphone data-collection application. Hence, we build two hypotheses from our strategies: first, burstiness and mode-switching is present in data transfers and battery behaviour; secondly, power consumption increases as mean delay decreases.

Chapter 2

Background

Chapter Description

First, this chapter introduces different models and applications of unsupervised learning, namely model selection (2.2), clustering (2.2.2), method of moments (2.2.3) and hidden Markov models (2.2.4). Secondly, section 2.3 describes queueing models focusing on the PS discipline (2.3.3), response times (2.3.5) and Markov-modulated Poisson processes in queues (2.3.6). Lastly, we explain some performance-energy applications in section 2.4.

2.1 Introduction

This chapter introduces background information in three parts and explains fundamental theoretical concepts that are developed in subsequent chapters. In the first part, we provide an overview of unsupervised learning techniques including model selection, three types of clustering, and method of moments. Further, we define hidden Markov models (HMMs), prove the fundamental HMM algorithms and highlight the benefits and limitations of existing HMM applications. In the second part of the background, we introduce key principles of queueing theory, processor-sharing (PS) queues and existing work of response times under PS with potential extensions. Finally, we explore existing performance-energy models used in mobile technology and data centres. This chapter allows the reader to understand the basic principles of existing models, the relevant applications and limitations, and how improving such models benefits performance and saves resources and energy.

2.2 Unsupervised learning models

It was said by statistician George Box that "all models are wrong, but some are useful" [54]. This famous quote fuelled the *all models are wrong* aphorism in statistics, which attaches more importance to how useful models are over their correctness. Indeed, obtaining models for analysing, training and forecasting time-series provides many real-world benefits for social media, finance and storage systems, to name but a few. For example, a stock prediction model uses historical daily stock prices to adopt an effective investment strategy [74], which aims for diversification across asset classes and uses histori-

cal volatility to dynamically select low-risk assets. Often, time-series are "unlabelled," meaning they exhibit a hidden structure from an unknown underlying distribution. Unsupervised learning models provide solutions for modelling such unlabelled time-series and explain features of data in an efficient manner.

Many unsupervised learning models exist in the literature including clustering, method of moments and expectation maximisation (EM) [15]. Specifically, the EM algorithm iteratively finds maximum likelihood estimates of model parameters [17]. A special case of EM is the Baum-Welch algorithm [13], which is used in HMMs to find the local maximum likelihood. By considering parameter learning in HMMs via the Baum-Welch algorithm, we associate the application of EM with HMMs. To select the best unsupervised learning model given a data set, an objective criterion is typically required for model evaluation. As noted, we consider usefulness and applications as part of the model selection process. As this chapter will justify, we select HMMs as our choice of adaptive models because of their parsimony, efficiency and applicability for real-world scenarios.

2.2.1 Model selection

An important aim in training data for analysis and forecasting is selecting the optimal model from a set of candidate models [22]. It is known that with unsupervised learning techniques, possible solutions cannot be evaluated with a reward signal, as is the case with supervised learning. Therefore, to infer our own "function" from unlabelled data, we investigate the criteria for an efficient unsupervised learning model: first, we investigate variations of the mean absolute percentage error (MAPE); secondly, we focus on the well-known Akaike Information Criterion (AIC) [16].

MAPE

A validation of forecasting models is the error between the model-generated time-series and the original data. A widely used metric is the mean absolute percentage error (MAPE), which measures accuracy of the forecast value against the actual value. The MAPE formula, expressed as a percentage for a time-series of n values, is given by:

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{a_t - f_t}{a_t}\right| \tag{2.1}$$

where a_t is the actual value and f_t is the forecast value.

An averaged MAPE measure calculated on multiple time-series is sometimes distorted if particular time-series produce inflated values (i.e. significantly larger than one), which jeopardises reliable comparisons [29]. To solve such an issue, we use the symmetric MAPE (sMAPE) measure defined by:

$$\text{sMAPE} = \frac{2}{n}\sum_{t=1}^{n}\frac{|f_t - a_t|}{|a_t| + |f_t|} \tag{2.2}$$

where a_t is the actual value and f_t is the forecast value.

AIC

A well-known test for model selection is the Akaike information criterion (AIC) [16], which measures model efficiency in terms of the number of independent parameters (p) and the maximised likelihood function (L) in an effort to combat overfitting. AIC, seen as a trade-off between goodness-of-fit and model complexity, is defined as:

$$\text{AIC} = 2p - 2log(L) \tag{2.3}$$

Critics of AIC have highlighted that AIC favours models with fewer parameters and less accuracy over more accurate, complex models [18]. AIC selects models that approximate reality, suggesting that reality should always have lower dimensionality, which is not always the case. Further, AIC assumes that its models possess a likelihood function and lacks general error measures that encompass all models such as sMAPE and classification precision. Therefore, the aim is to improve weaknesses of AIC by employing an objective cost function (ideally incorporating measures such as sMAPE) as a model selector. Further, we select models that are easy to implement, train efficiently and have useful features for characterising and classifying data.

2.2.2 Clustering

Clustering is defined as the process of separating data items into homogeneous groups with similar features [20]. This is an unsupervised learning problem for providing a class structure to an apparently randomly distributed data set, where we are unaware of class labels or the number of classes in the data. A *cluster* is essentially a group, where all clustered data points share common attributes. Thus, we assign each point in the collection to a cluster, where data points from one cluster differ from the points belonging to another. Moreover, "hard" clustering dictates that data points either strictly belong to a cluster or do not, whereas "soft" clustering uses probabilities for associating a particular data point with a specific cluster. Generally, we seek clusters with high *intra-cluster* similarity and low *inter-cluster* similarity. A typical similarity criteria is geometrical distance such that each cluster contains points near one another, which is an example of *distance-based* clustering. Further, similarity criteria has other measurements excluding distance (i.e. *conceptual* clustering) and is another important feature of cluster analysis.

Clustering is used to solve a range of small and large-scale problems [83]. Some examples of clustering applications include: classifying flora and fauna in *ecology* via representative features across different communities in heterogeneous environments; assigning genotypes (i.e. the genetic makeup of an organism) in *bioinformatics*; developing future *marketing* programs via distinct groups using customer databases; detecting earthquake "danger zones" in *seismology* by clustering epicentres; identifying areas of frequent *criminal* incidents temporally.

The wide range of applications offered by clustering is supported by its simplicity and parsimony. Hence, by minimising the number of parameters, clustering is advantageous for systematically grouping data sets of variable size in a straightforward manner. Other advantages include revealing significant trends in data sets and creating profiles of individual data points. However, some types of clustering have increased computational complexity with very large data sets and specifying the number of initial clusters as an input, specifically for the k-means clustering algorithm, is often a challenge.

The three main classifications of clustering algorithms explored in subsequent sections are *hierarchical*, *density-based*, and *centroid-based* clustering. For each classification type, we elaborate on methodologies, benefits and possible drawbacks.

Hierarchical clustering

Hierarchical clustering forms new clusters from the union of two existing clusters with the smallest common distance. The first type of hierarchical clustering algorithm is *agglomerative*, where clusters are merged iteratively in a "bottom-up" fashion. The algorithm terminates when all objects form one cluster. A famous example of agglomerative clustering is Agglomerative Nesting (AGNES) introduced by Kaufmann and Rousseeuw [19]. AGNES merges nodes with common features until all nodes form one cluster and is typically implemented in statistical analysis packages such as S+. A disadvantage of agglomerative clustering is the $O(T^2)$ computational complexity for T data points, which does not scale well with large data sets. Another drawback is that steps executed in AGNES cannot be reversed. The second type is *divisive* hierarchical clustering, which splits clusters iteratively in a "top-down" process, forming successively smaller clusters until each cluster is one object. Divisive Analysis (DIANA) [19] is a typical example, which acts as the inverse of AGNES and is also implemented in S+.

Integrating hierarchical clustering with distance-based approaches, one obtains a data clustering method known as Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [33]. BIRCH uses a clustering feature tree and incrementally improves the quality of sub-clusters for multi-dimensional metric data points whilst addressing memory and time constraints. By observing that data space is typically not uniformly occupied and priorities vary among data points, BIRCH saves computation complexity and reduces I/O costs from training on a subset of data points and existing clusters.

Density-based clustering

Density-based clustering algorithms form clusters in bounded areas of high density, where lower density parts act as boundary points or noise. Therefore, important features of density-based clustering include the discovery of arbitrary-shaped clusters and the ability to handle noise. One of the most popular types of density-based clustering is Density-Based Spatial Clustering of Applications with Noise (DBSCAN), which was proposed by Ester *et al.* [35] and uses the idea of density-reachability, where a data point is within a boundary distance or neighbourhood of another point. In this scenario, a cluster is a max-

imal set of density-connected points (i.e. data points x and y are density-connected if they share a common point p such that (x, p) and (y, p) are density-reachable). DBSCAN has a run-time complexity of $O(Tlog(T))$ for T data points if it has an efficient index structure; otherwise, the complexity is $O(T^2)$.

DBSCAN has the advantage of handling noise, does not require an initial number of clusters as an input parameter and identifies arbitrarily-shaped clusters (i.e. clusters surrounded by other clusters). On the other hand, DBSCAN relies on a $getNeighbours(T)$ function, which contains the collection of points T and the distance measure that defines the ϵ error value. With multi-dimensional data, this distance metric becomes useless and it is difficult to assign an accurate value to ϵ.

Centroid-based clustering

Centroid-based clustering involves grouping data using a central point, which is usually the mean value of all data points in the cluster. The simplest centroid-based clustering is the *k-means algorithm* developed by MacQueen [44]. The k-means algorithm requires that each data point be assigned to one of k clusters, where each cluster has a centroid calculated as the mean of its data points. The k-means algorithm is summarised in four steps:

1. Choose a value for k and then calculate k initial centroids among the data points.

2. Match each data point with the nearest cluster centroid using Euclidean distance.

3. Once all the data points are assigned, re-calculate positions of the k centroids.

4. Repeat steps 2 to 3 until centroids converge (i.e. fix position) then terminate.

Note that, initially, it is important to strategically choose initial centroids as far as possible from each other. The k-means algorithm has a complexity of $O(TKI)$ where T is the number of data points, K is the number of clusters and I is the number of iterations. By increasing I, one ensures the initial randomly selected centroids have less effect on the outcome of the algorithm. Advantages of k-means include easy implementation and acceptable convergence rates for large data sets. K-means is often used as a preprocessing step for other algorithms (i.e. EM) and, thus, provides an initial configuration. For these reasons, we choose k-means as the clustering algorithm with which to partition and group our discrete data sets. Some disadvantages of k-means include: choosing a suitable value of k for the data set; k-means does not handle noisy data and is sensitive to outliers; convergence to a local minimum may not achieve the desired result even after many iterations. Preliminary testing for a range of values for k (i.e. the initial number of clusters) typically addresses the first disadvantage. Despite other disadvantages, the k-means algorithm is more suitable for modelling the data sets used in this thesis as it has the lowest training complexity compared to the other types of clustering.

2.2.3 Method of moments

The method of moments (MoM) was originally introduced by Pearson in 1894 [53] (the reader may find a modern implementation in [173]) and is described in a few steps. Let X be a random variable that takes real values, and the distribution of X has a set of k unknown parameters $\theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$. Generating a sequence of n independent random variables $X_1, X_2, \ldots, X_n$, where each X_i has the same distribution as X, we seek parameter estimators by matching sample moments with distribution moments. Let $\mu_i(\theta) = E[X^i]$, $i \in \{1, 2, \ldots, k\}$, be the i^{th} moment of X and let $M_i(X) = n^{-1} \sum_{j=1}^{n} X_j^i$, which is the i^{th} sample moment. Assuming $M_i(X)$ is an unbiased estimator of $\mu_i(\theta)$, we aim to match estimators $Y_1, Y_2, \ldots, Y_k$ with unknown parameters $\theta_1, \theta_2, \ldots, \theta_k$, which results in k equations for k unknowns:

$$\mu_i(Y_1, Y_2, \ldots, Y_k) = M_i(X_1, X_2, \ldots, X_n)$$

where $i \in \{1, 2, \ldots, k\}$.

Advantages of MoM include: the comparative ease of obtaining consistent estimators by hand, whereas equations of likelihood rely heavily on computer calculation; finding maximum likelihood estimates by repeated approximations; in cases when relevant probability distributions are unknown, method of moments is preferred to maximum likelihood when estimating specific parameters (e.g. of a utility function).

MoM presents the following disadvantages: biased estimators are often obtained; small samples may produce estimators outside the parameter space; not all relevant sample information is considered by estimators. We find that the maximum likelihood method improves MoM as it offers more consistently unbiased estimators with higher probabilities of matching original variables. Further, maximum likelihood estimates are always within the parameter space. This is another reason to select HMMs as our choice of models.

2.2.4 Hidden Markov Models

The hidden Markov model (HMM) is one of the most widely used statistical tools for modelling discrete time-series. The HMM was first developed by Baum and Petrie [12] to encode information on time-series evolution with a parameter estimation algorithm inferring the behaviour of the given series. As HMMs efficiently represent workload dynamics, acting as parsimonious models that obtain trace characteristics, their applications include storage workload modelling [66], speech recognition [65, 76] and genome sequence prediction [68]. We list a number of advantages that HMMs possess over existing models and techniques:

1. HMMs are parsimonious with only three parameters, which converge efficiently using the EM algorithm.

2. The mode-switching of HMMs captures burstiness, unlike in Poisson processes, and hidden states represent cyclic behaviour of time-series (i.e. financial markets [26]).

3. Unlike HMMs, auto-regressive moving average (ARMA) and linear regression models assume normality of residuals and omit nonlinear trends in data [87, 88].

4. HMMs offer good approximation capacities similar to artificial neural networks (ANNs), but ANNs are often poor at generalisation due to overfitting [78].

5. HMMs offer portability and can efficiently build a class of workloads used for system simulation or as inputs into analytical models.

Further, HMMs act as benchmarks to explain and predict the complex behaviour of multi-application workloads and provide representation for inputs of large, complex systems. In this thesis, we build adaptive HMMs for discrete processes by adapting the corresponding underlying HMM mechanisms for online learning of systems.

To train and test HMMs as benchmarks, one must first utilise the statistical algorithms to solve three fundamental problems associated with HMMs related to workload traces: the first problem is determining $P(O \mid \lambda)$, which is the *a priori* probability of the observed trace or sequence O given some model λ; the second is to maximise $P(O \mid \lambda)$ by adjusting model parameters of λ for a given observation sequence O; the third problem is finding the most likely hidden state sequence associated with a given observed sequence. The three fundamental problems are solved respectively by applying the forward-backward algorithm [12], the Baum-Welch algorithm[1] [13] and the Viterbi algorithm [14]. We remind the reader that the Baum-Welch algorithm is a special case of the EM algorithm, which finds the local maximum likelihood. Before elaborating further on the three algorithms, we first provide formal definitions of Markov chains and HMMs.

The HMM is defined as a bivariate Markov chain that encodes information about time-series evolutions. Essentially, there is a hidden Markov chain $\{C_t\}_{t\in\mathbb{N}}$, which has unobservable states, and a discrete time stochastic process $\{O_t\}_{t\in\mathbb{N}}$, which is observable. Combining these processes, we obtain the bivariate Markov chain $\{(C_t, O_t)\}_{t\in\mathbb{N}}$. Note that C_t governs the distribution of O_t via emission probabilities, but we reserve statistical inference for O_t, as C_t is not observed. We formally define a Markov chain as follows:

Definition 1 *Let $\{C_t\}_{t\in\mathbb{N}}$ be a stochastic process with state space $S = \{1, \ldots, r\}$. Then $\{C_t\}_{t\in\mathbb{N}}$ is a **Markov chain** if*

$$P(C_{t+1} = c_{t+1} \mid C_t = c_t, C_{t-1} = c_{t-1}, \ldots, C_1 = c_1) = P(C_{t+1} = c_{t+1} \mid C_t = c_t)$$

where $c_1, c_2, \ldots, c_{t+1} \in S$.

Given definition 1 , the HMM is formally defined as:

Definition 2 *Suppose we have a Markov chain $\{C_t\}_{t\in\mathbb{N}}$ with state space $S = \{1, \ldots, r\}$, initial distribution $\pi = (\pi_i)_{i\in S}$, transition matrix $A = (a_{ij})_{i,j\in S}$, observation matrix $B = (b_{ik})_{i\in S, k\in J}$, and a stochastic process $\{O_t\}_{t\in\mathbb{N}}$ with values in $J = \{1, \ldots, m\}$. Then, the bivariate stochastic process $\{(C_t, O_t)\}_{t\in\mathbb{N}}$ is a **hidden Markov model** if it is a Markov chain with transition probabilities*

[1]The Baum-Welch algorithm includes the forward-backward algorithm iteratively.

$$P(C_t = c_t, O_t = o_t \mid C_{t-1} = c_{t-1}, O_{t-1} = o_{t-1}) = P(C_t = c_t, O_t = o_t \mid C_{t-1} = c_{t-1}) = a_{c_{t-1}c_t} b_{c_t o_t}$$

where $c_{t-1}, c_t \in S, o_{t-1}, o_t \in J$.

A useful property that is defined conditionally on $\{C_t\}_{t\in\mathbb{N}}$ is that $\{O_t\}_{t\in\mathbb{N}}$ are independent states, which gives the following:

$$P(O_1 = o_1, \ldots, O_n = o_n \mid C_1 = c_1, \ldots, C_n = c_n) = \prod_{i=1}^{n} P(O_i = o_i \mid C_i = c_i) \tag{2.4}$$

Results similar to equation (2.4) are useful properties for HMMs and are reserved for the appendix (see A.1). Such properties provide useful mathematical tools for solving the three fundamental problems relates to HMMs. Before defining the three fundamental problems, it is necessary to define joint probability and likelihood functions. The probability of joint processes $(C_1, O_1), (C_2, O_2), \ldots, (C_n, O_n)$ allows calculation of HMM-related probabilities, where the joint probability function is given by:

$$\begin{aligned} J_\pi(c_1, o_1, \ldots, c_n, o_n) &= P(C_1 = c_1, O_1 = o_1, \ldots, C_n = c_n, O_n = o_n) \\ &= P(C_1 = c_1, O_1 = o_1) P(C_2 = c_2, O_2 = o_2 \mid C_1 = c_1) \ldots \\ &\qquad \ldots P(C_n = c_n, O_n = o_n \mid C_{n-1} = c_{n-1}) \\ &= \pi_{c_1} b_{c_1,o_1} \prod_{i=2}^{n} a_{c_{i-1},c_i} b_{c_i,o_i} \end{aligned} \tag{2.5}$$

where the initial distribution of the chain is $\pi_{c_1} = P(C_1 = c_1)$.

This is the joint probability function for observed and unobserved random variables. It follows that the likelihood function L_π of observations $o_1, o_2, \ldots, o_n$ is given by:

$$\begin{aligned} L_\pi(o_1, o_2, \ldots, o_n) &= P(O_1 = o_1, O_2 = o_2, \ldots, O_n = o_n) \\ &= \sum_{c_1,\ldots,c_n} J_\pi(c_1, o_1, \ldots, c_n, o_n) \\ &= \sum_{c_1,\ldots,c_n} \pi_{c_1} b_{c_1,o_1} \prod_{i=2}^{n} a_{c_{i-1},c_i} b_{c_i,o_i} \end{aligned} \tag{2.6}$$

Note that hereinafter the likelihood function L_π is written as $P(O \mid \lambda)$, where O is the observation set and λ is the model. In the next sections, we solve the three problems associated with HMMs, namely the forward-backward, Baum-Welch and Viterbi algorithms.

Forward-backward algorithm

The first problem solved for HMMs is described as follows: Suppose we have a sequence of T observations $O = (O_1, O_2, \ldots, O_T)$ and the model $\lambda = (\pi, A, B)$, where π is the initial distribution, A is the hidden state transition matrix and B is the observation matrix. The

goal is to find $P(O \mid \lambda)$ [12], which is the probability of the observed sequence O given the model λ. Essentially, we determine the likelihood of O. By methods in [71], we label the "forward" part of the forward-backward algorithm as the α-pass. We define $\alpha_t(i)$ as the probability of the observation sequence up to time t and of state q_i at time t, given our model λ. Hence, we write term $\alpha_t(i)$ as follows:

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, s_t = q_i \mid \lambda) \tag{2.7}$$

where $i = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T$.

We change the notation of b_{ik} (as seen in definition 2 previously) to $b_i(k)$ for given i and k values. Then, the inductive solution of $\alpha_t(i)$ is given by:

1. Initiating forward probabilities, for $i = 1, 2, \ldots, N$, we have

$$\alpha_1(i) = \pi_i b_i(O_1) \tag{2.8}$$

2. then, for $i = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T-1$, we have

$$\alpha_{t+1}(i) = b_i(O_{t+1}) \sum_{j=1}^{N} \alpha_t(j) a_{ji} \tag{2.9}$$

   where $\alpha_t(j)a_{ji}$ is the probability of the joint event that $O_1, O_2, \ldots O_t$ are observed and there is a transition from state q_j at time t to state q_i at time $t+1$.

3. it follows that

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_{T-1}(i) \tag{2.10}$$

   where we obtain $\alpha_{T-1}(i) = P(O_1, O_2, \ldots, O_T, s_{T-1} = q_i \mid \lambda)$ using equation (2.7).

The "backward" part of the forward-backward algorithm is known as the β-pass. We define the backward variable $\beta_t(i)$ as the probability of the observation sequence from time $t+1$ to the end of the sequence, given state q_i at time t and the model λ. The mathematical notation is given by:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots O_T \mid s_t = q_i, \lambda) \tag{2.11}$$

The inductive solution of $\beta_t(i)$ is written as:

1. For $i = 1, 2, \ldots, N$, we have

$$\beta_T(i) = 1 \tag{2.12}$$

2. then, for $i = 1, 2, \ldots, N$ and $t = T-1, T-2, \ldots, 1$, we have

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \tag{2.13}$$

   where any state q_j can generate the observation O_{t+1}.

Baum-Welch algorithm

The second problem solved for HMMs is described as: given the model $\lambda = (\pi, A, B)$ and the observation sequence $O = (O_1, O_2, \ldots, O_T)$, maximise $P(O \mid \lambda)$ by adjusting parameters π, A, B. This problem is solved using the iterative Baum-Welch algorithm [13]. First, we define the probability of a path being in state q_i at time t and making a transition to state q_j at time $t+1$, given O and λ, as follows:

$$\xi_t(i, j) = P(s_t = q_i, s_{t+1} = q_j \mid O, \lambda) \tag{2.14}$$

Next, we calculate $\xi_t(i, j)$ using a product of three terms: first, observations $O_1, O_2, \ldots O_t$ ending in state q_i at time t are represented by the term $\alpha_t(i)$; 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 incorporated in the term β_{t+1}. Multiplying these terms together and dividing by a normalising term (i.e. $P(O \mid \lambda)$), we rewrite $\xi_t(i, j)$ as:

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O \mid \lambda)} \tag{2.15}$$

Summing the $\xi_t(i, j)$ term over $j = 1, \ldots, N$, we obtain $P(s_t = q_i \mid O, \lambda)$, which is the probability of being in state q_i at time t, given O and λ. Therefore, this probability is written as follows:

$$\gamma_t(i) = P(s_t = q_i \mid O, \lambda) = \sum_{j=1}^{N} \xi_t(i, j) \tag{2.16}$$

Summing $\gamma_t(i)$ over time t up to T, we obtain $\sum_{t=1}^{T} \gamma_t(i)$, which is the number of times we expect to visit state q_i. Summing $\gamma_t(i)$ up to time $T-1$, we obtain the expected number of outgoing transitions from q_i, which is $\sum_{t=1}^{T-1} \gamma_t(i)$. Similarly, summing $\xi_t(i, j)$ over t up to T gives us $\sum_{t=1}^{T-1} \xi_t(i, j)$, which is the expected number of times states q_i and q_j are visited consecutively. However, terminating at $T-1$ returns the sum $\sum_{t=1}^{T-1} \xi_t(i, j)$, which is the expected number of transitions made from state q_i to state q_j. Using these terms, we define re-estimation formulas for the HMM parameters (π', A', B'):

1. Initially at $t = 1$, we have

$$\pi'_i = \gamma_1(i) \tag{2.17}$$

   where $i = 1, 2, \ldots, N$.

2. then, we update A' as

$$a'_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.18}$$

   where this formula is the expected number of transitions from q_i to q_j divided by the expected number of transitions emitted from q_i.

3. for B', it follows that

$$b'_j(k) = \frac{\sum_{t=1, O_t=k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.19}$$

where this is the expected number of times state q_j is visited whilst observing k, divided by the number of times q_j is visited.

Thus, we iteratively re-estimate our model $\lambda' = (\pi', A', B')$, where $\pi' = \{\pi'_i\}$, $A' = \{a'_{ij}\}$ and $B' = \{b'_j(k)\}$. At each iteration, check whether $P(O \mid \lambda') > P(O \mid \lambda)$ to obtain a model that maximises the likelihood of producing the given observation sequence O. Once the termination condition is met (e.g. minimal error threshold is reached), the HMM parameters (π, A, B) converge and generate synthetic observation traces.

Viterbi algorithm

The third HMM problem is described as follows: given $O = (O_1, O_2, \ldots, O_T)$ and $\lambda = (\pi, A, B)$, find an optimal state sequence for the underlying Markov chain, thus decoding the hidden part of the HMM. This problem is solved using the Viterbi algorithm [14]. Essentially, we require the best state sequence (i.e. $S = (S_0, S_1, \ldots, S_{T-1})$) such that:

$$S^* = \underset{S}{\operatorname{argmax}}\, P(S \mid O, \lambda) = \underset{S}{\operatorname{argmax}}\, P(S, O \mid \lambda) \tag{2.20}$$

where $P(O \mid \lambda)$ is independent of S.

The Viterbi algorithm calculates this optimal state sequence S^*. At each time step t, the Viterbi algorithm allows S^* to retain all optimal paths finishing at the N states. At the next time step $t + 1$, the N optimal paths are updated and S^* continues to grow in this manner. Let $S_t^*(i)$ be the optimal path ending in state i for the observations $O_1, O_2, \ldots, O_t$. Then, we define $\delta_t(i) = P(O_1, O_2, \ldots, O_t, S_t^*(i) \mid \lambda)$, which is the probability of generating observations $O_1, O_2, \ldots, O_t$ from path $S_t^*(i)$, given the model. Finally, we use an array $\psi_t(i)$ to keep track of each t and i that have maximised the last $\delta_t(i)$. The steps of the Viterbi algorithm are described as follows:

1. For $i = 1, 2, \ldots N$, initialise the variables

$$\delta_1(i) = \pi_i b_i(O_1) \tag{2.21}$$

$$\psi_1(i) = 0 \tag{2.22}$$

2. recurse for $j = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T$ on the variables

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] b_j(O_t) \tag{2.23}$$

$$\psi_t(j) = \underset{S}{\operatorname{argmax}} [\delta_{t-1}(i) a_{ij}] \tag{2.24}$$

3. terminate with

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)] \tag{2.25}$$

$$S_T = \underset{S}{\operatorname{argmax}} [\delta_T(i)] \tag{2.26}$$

4. for $t = T - 1, T - 2, \ldots, 1$, backtrack through the state sequence

$$S_t^* = \psi_{t+1}(i^*) \tag{2.27}$$

In the next section, we explain the notion of underflow, which occurs when calculating small terms for long time-series. Normalisation is offered as a possible solution to underflow occuring in the Baum-Welch and Viterbi algorithms.

2.2.5 Normalisation for underflow

The solutions to the three fundamental problems associated with HMMs are typically implemented using any double point precision language (i.e. C) and converge for discrete observation sequences. However, as we increase the number of observed points in the discrete trace, the Baum-Welch algorithm succumbs to *underflow*. This results in decreasing the size of probabilistic terms and sufficient iterations in the algorithm produces infinitesimally small values, which might returns errors if dividing by such values. Using results from [83], we solve the underflow problem for both Baum-Welch and Viterbi algorithms.

With the Baum-Welch algorithm, we normalise the α and the β sets using similar normalising procedures as [94]. First, we define normalising terms $\hat{\alpha}_t(i)$ such that $\sum_{i=1}^{N} \hat{\alpha}_t(i) = 1$ using the following equations:

$$\begin{aligned} \hat{\alpha}_t(i) &= \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} = \frac{P(O_1, O_2, \ldots, O_t, s_t = q_i \mid \lambda)}{P(O_1, O_2, \ldots, O_t \mid \lambda)} = \frac{P(O_1, O_2, \ldots, O_t, s_t = q_i, \lambda)}{P(O_1, O_2, \ldots, O_t, \lambda)} \\ &= P(s_t = q_i \mid O_1, O_2, \ldots, O_t, \lambda) \end{aligned} \tag{2.28}$$

Hence, the complete solution of $\hat{\alpha}_t(i)$ is given by:

1. For $i = 1, 2, \ldots, N$, initialise the forward probabilities

$$\hat{\alpha}_1(i) = \frac{\pi_i b_i(O_1)}{\sum_{j=1}^{N} \pi_j b_j(O_1)} \tag{2.29}$$

2. then, for $j = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T - 1$ we have

$$\hat{\alpha}_{t+1}(i) = \frac{[\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{ji}] b_i(O_{t+1})}{\sum_{k=1}^{N} [\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{jk}] b_k(O_{t+1})} \tag{2.30}$$

For the $\hat{\beta}_t(i)$ variables, we apply the α normalising terms, but note that the β values do not sum to 1 for any time t. Thus, the solution of $\hat{\beta}_t(i)$ is given by:

1. Initially, for $i = 1, 2, \ldots, N$, we have

$$\hat{\beta}_T(i) = \beta_T(i) = 1 \tag{2.31}$$

2. then, for $i = 1, 2, \ldots, N$ and $t = T-1, T-2, \ldots, 1$, it follows that

$$\hat{\beta}_t(i) = \frac{\sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{k=1}^{N} [\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{jk}] b_k(O_{t+1})} \tag{2.32}$$

Note that $\hat{\alpha}_t(i)\hat{\beta}_t(i)$ is expressed as:

$$\hat{\alpha}_t(i)\hat{\beta}_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_T(i)} = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} \tag{2.33}$$

Therefore, the $\gamma_t(i)$s are calculated, for $t = 1, 2, \ldots, T$, as follows:

$$\gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i, j) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} = \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{j=1}^{N} \hat{\alpha}_t(j)\hat{\beta}_t(j)} \tag{2.34}$$

Using the definition of $\gamma_t(i)$ terms, the $\xi_t(i, j)$ term is given by:

$$\begin{aligned} \xi_t(i, j) &= \frac{\hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{[\sum_{k=1}^{N} [\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{jk}] b_k(O_{t+1})][\sum_{j=1}^{N} \hat{\alpha}_t(j)\hat{\beta}_t(j)]} \\ &= \frac{\gamma_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{[\sum_{k=1}^{N} [\sum_{j=1}^{N} \hat{\alpha}_t(j) a_{jk}] b_k(O_{t+1})] \hat{\beta}_t(i)} \end{aligned} \tag{2.35}$$

The re-estimation formulas of the HMM parameters (π, A, B) now use the normalised $\gamma_t(i)$ and $\xi_t(i, j)$ variables.

With the Viterbi algorithm, underflow is avoided via logarithm summations. Summing logarithms of the α and the β values is achievable, but computing γ values requires manipulating a sum of $\alpha_t(i)$, which is not in the logarithm domain. Therefore, taking logarithms of terms, the enhanced Viterbi algorithm is given by:

1. Initialise the following variables, for $i = 1, 2, \ldots, N$:

$$\delta_1(i) = log(\pi_i b_i(O_1)) \tag{2.36}$$

$$\psi_1(i) = 0 \tag{2.37}$$

2. for $j = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T$, recurse on the variables

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) + log(a_{ij})] + log(b_j(O_t)) \tag{2.38}$$

$$\psi_t(j) = \operatorname*{argmax}_{S} [\delta_{t-1}(i) + log(a_{ij})] \tag{2.39}$$

3. terminate

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)] \tag{2.40}$$

$$S_T = \underset{S}{\operatorname{argmax}} [\delta_T(i)] \tag{2.41}$$

4. for $t = T - 1, T - 2, \ldots, 1$, backtrack

$$S_t^* = \psi_{t+1}(i^*) \tag{2.42}$$

2.2.6 HMM applications

Given the definition of the solutions to the three fundamental problems associated with HMMs, we have seen how to parsimoniously obtain converged parameters given discrete data. This section is dedicated for the variations and applications of HMMs in domains ranging from storage systems to finance. Further, we discuss the limitations of existing HMM applications and aim to address such drawbacks in this thesis.

HMM for Flash memory workload

Harrison *et al.* present a workload analysis technique that creates HMMs for representing workloads that generate synthetic traces [66]. Further, rates of Markov-modulated fluid processes are estimated from the HMM and used as input to an analytical performance model of Flash memory. Harrison *et al.* apply HMMs on trace-level operation sequences (i.e. reads and writes) using partitioning and clustering of traces, after which the HMM parameters are trained until convergence via the Baum-Welch algorithm. Further, HMM-generated synthetic traces are produced and validated with mean, standard deviation and autocorrelation against original, unclustered traces. Within the context of storage workload characterisation and its importance for performance evaluation, Harrison extracts key workload parameters from traces, utilising the parsimonious and burstiness capabilities of the HMM tailored for multi-application workloads. Additionally, HMMs are particularly effective when time-series dynamics are influenced by switching between modes, where each mode represents underlying characteristics.

Possible improvements of the work by Harrison *et al.* include: considering other levels of the storage stack for traces; handling infrequent, higher density, additional loads; online characterisation of workloads via an incremental HMM, where its parameters are updated "on-the-fly" as more workload data becomes available. Indeed, without this incremental model, continually generating new "static" HMMs at run-time requires further computational costs. We extend this work through the IncHMM [1, 2], which is a discrete-state, incremental HMM with reduced computational complexity of the Baum-Welch algorithm.

HMM with k-means clustering

Inspired by the aforementioned work of Harrison *et al.*, a similar HMM was constructed by Chis and applied to storage workloads and hospital patient arrivals [83]. We summarise Chis' HMM as follows: first, unlabelled data from a time-series is partitioned into

"bins" by counting the number of read and write commands per second; secondly, the k-means algorithm groups this sequence of bins into k distinct clusters ($k \in \mathbb{Z}$, $k>1$) such that the distance from the cluster centroid (i.e. mean of the cluster) to the surrounding points is minimised; thirdly, assign each bin in the time-series a value between one and k and use this discrete trace as an input into the Baum-Welch algorithm; fourthly, train the HMM via the Baum-Welch algorithm and obtain converged parameters once a minimal error boundary is reached; finally, the HMM uses its parameters to generate a likely sequence of observations (i.e. cluster centroids) and obtain representative synthetic traces.

Similarly to work by Harrison *et al.*, the HMM by Chis is statically re-trained on historical data and expected centroid values are generated accordingly. The power of this model lies in the HMM switching modes (and burstiness) and the simple categorisation of k-means as a useful preprocessing step for the EM algorithm. Additionally, Chis extends this HMM to obtain a basic incremental model (with a sliding training window for new data points) by creating a simple forward-recurrence formula to estimate the new β variables (corresponding to new data points), which originally used a backward formula requiring a one-step lookahead. Despite improving work by Stenger *et al.* [117], which sets all new β variables to one, the prediction error of Chis' model increases with each step of incremental training. Therefore, an extension of the model by Chis is obtaining a more reliable (and mathematically rigorous) incremental backward formula to enable online EM learning. Extending Harrison's work [66], the IncHMM [2] in this thesis provides such incremental capabilities and the SlidHMM [3] goes further to improve upon the IncHMM by adding the sliding window for parameter training on a dynamic data set.

Incremental HMMs

We summarise existing work on incremental learning of HMMs. A survey of techniques [47] for incremental learning of HMM parameters offers a comparison of past work in terms of block-wise (i.e. training symbols are blocks of sub-sequences) and symbol-wise (or sequential, i.e. where training symbols are observed one at a time) algorithms for measuring convergence properties and time and memory complexity. Further, the survey is also divided into objective functions, such as minimum model divergence (MMD), minimum prediction error (MPE) and maximum likelihood estimation (MLE), of which the EM algorithm offers monotonic convergence and is numerically more robust against poor initialisation. Additionally, the EM algorithm is easier to apply then gradient-based techniques since derivatives and line-search are not required, which supports its popularity as an MLE incremental technique. For example, for large observation sequences (i.e. over one million data points), incremental work on the EM algorithm by Florez-Larrahondo *et al.* [103] uses symbol-wise learning algorithms and has constant memory complexity compared to block-wise algorithms used by Mizuno et al. [48], which grow linearly as the training window size is increased. Other works employing incremental learning of HMM parameters include an online EM algorithm by Cappe [42] and adaptive ROC-based ensembles of HMMs applied to anomaly detection [40]. Mongillo *et al.* [59] calculate log of likelihood errors between batch and online Baum-Welch algorithms and extend the observation training set to continuous data.

We compare our work in this thesis with the foremost incremental EM achievements (or the algorithm with the most efficient time and memory complexity in the survey [47]), which is work by Florez-Larrahondo *et al.* [103]. Hence, we obtain novel techniques for online training of symbol-wise algorithms on a number of target applications with reduced resources, faster EM convergence and efficient validation techniques. Such work is summarised in the IncHMM [1, 2] and SlidHMM [3]. Additionally, our incremental algorithms offer the following advantages over existing work: incorporating both block-wise and symbol-wise learning algorithms on discrete-time data; incremental training has constant memory complexity as training window size increases, which matches the complexity offered by Florez-Larrahondo *et al.* [103]; new applications including spatiotemporal modelling of I/O commands in storage systems, forecasting Twitter user activity and predicting stock price movements.

Other HMM variations

Recently, the literature has seen more research into parallelisation of HMM training algorithms (i.e. Baum-Welch), mainly due to increasing applications of HMMs to multi-user modelling and pattern classification. One technique involves GPUs to parallelise the Baum-Welch algorithm using CUDA implementations [79, 80]. To avoid costly computation arising from thread synchronisation that enables parallel HMMs to train simultaneously, alternate research has relied on HMM hybrids by modifying the structure of its fundamental algorithms. For example, a factorial HMM [58], where observed outputs are combined, requires separate hidden state sequences to train the observation chain. Each sequence of hidden states is independent from the next and there is no interaction between chains via state probabilities. Other hybrids include coupled HMMs, where individual HMMs have inter-connected links between hidden states and emit an observation given a probability. The coupling of hidden state sequences, as applied to groups of Twitter users [82], is seen as user interaction and results in many possible combinations for model setup. However, with increased interaction (i.e. coupling of HMMs), Baum-Welch training becomes more computationally expensive, which is a potential problem when scaling with more users.

Layered HMMs are used for multiple sensory channels [70] with a collection of hidden state sequences corresponding to one observation sequence, which iteratively updates based on its likelihood. This model is less likely to suffer from overfitting, but will be computationally more expensive on Baum-Welch training compared to the traditional HMM. In [62], two parallel and independent HMMs are combined and information about sign language is merged for each model using a token passing algorithm. Despite the reduced computation requirement needed for the two models, scaling to higher numbers of models (i.e. to train on larger vocabularies) would not be as efficient.

To address issues with the aforementioned models, we propose a robust multi-user HMM (MultiHMM) [5] to reduce cost of training on multiple traces simultaneously, whilst maintaining accuracy of model-generated traces compared to unclustered traces. The

MultiHMM uses two layers of clustering and an adapted Baum-Welch algorithm, where multiple traces can be processed simultaneously by a single HMM.

Financial HMMs

HMMs are powerful tools for forecasting stock market behaviour [74, 75]. This is no surprise since HMMs are known for their temporal pattern recognition, parsimonious nature and burstiness. For example, HMMs are trained on historical financial time-series to predict future market state over a forecast horizon (i.e. hours or days) as either *trending* or *normal*. The HMM switching modes are ideal for capturing temporal cyclic behaviour for indices and individual stocks. Further, annualised historical volatility applied to HMM forecasts evaluates the risk of financial instruments as *low* or *high*. However, disadvantages exist for HMMs: first, pre-processing financial data into discrete time-series for input into the Baum-Welch algorithm often involves binning or clustering, which offers further partitioning challenges; secondly, using HMM parameters for multi-step forecasts requires an indication of initial direction (i.e. linear prediction or moving averages) of time-series.

Despite such limitations, HMMs offer advantages over other existing models. One example is the auto-regressive moving average (ARMA), which acts as a forecasting model in numerous fields [90, 91, 92] and is widely included in the financial literature [96]. Further, hypothesis testing for stationary ARMA processes with generalised autoregressive conditional heteroskedasticity (GARCH) errors [95] is commonly used in forecasting, which use estimating-function methodologies. However, a disadvantage of ARMA models is the assumption of stationarity of time-series, where in reality financial traces are often non-stationary and require adjustment for seasonality. Additionally, ARMA accepts normality of residuals and often omits analysis of nonlinear trends in data [87, 88]. HMMs, however, cater for nonlinear trends and represent cyclic market behaviour in hidden states. Research in [66] argues that HMMs are needed to parametrise fluid models, which then parametrise MMPPs and, more generally, MAPs. To address such applications and, more importantly, understand the models discussed in this thesis, we devote a section of the background to MAPs and MMPPs.

2.2.7 MAPs and MMPPs

Stochastic models, such as Markovian arrival processes (MAPs), batch MAPs [102] and phase-type distributions provide analytical tools for measuring time-series whilst capturing correlation and burstiness in an efficient manner. There is a range of models to analyse properties of 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, by partitioning timestamped 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) [86]. We formally define a MAP as follows:

Definition 3 *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, 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 is the Markov-modulated Poisson process (MMPP). In fact, a MAP becomes an MMPP when D_1 (from definition 3) 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 mean arrival rate λ_1, and few arrivals (state two), with mean arrival rate λ_2, where $\lambda_1 > \lambda_2$ [85]. 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 $i, j = 1, 2$.

When fitting MMPPs and (more generally) MAPs, we seek efficient methods. In subsequent sections, we compare methods of fitting MAPs and discuss the advantages and limitations for each method.

Fitting MAPs

It is important to fit MAPs for discrete data traces in a fast and accurate manner. A simple method includes obtaining mean inter-arrival times directly from data traces and parametrising an MMPP (or MAP) with the mean Poisson arrival rates. Another method is obtaining the maximum likelihood estimate (MLE), which is often used by variations of MMPPs such as hidden Markov models (HMMs) [6]. However, MLE techniques are typically computationally expensive and depend on the length of trace being fitted and a quadratic in the number of states. To solve such issues, there exist methods of fitting empirical data using moment-matching techniques, such as the KPC-Toolbox [50]. The KPC-toolbox is not only fast at fitting empirical data into a Markov model, but it automatically calculates the optimal number of states, which improves existing MLE methods. KPC-toolbox is a library of MATLAB functions for fitting an empirical data set into Markov models such as MAPs and MMPPs, where only the input trace (e.g. packet inter-arrival times) is required. There are two fitting techniques employed by the KPC-toolbox: first, the Kronecker product composition (KPC) method reduces the moment and

temporal dependence fitting problem to assigning the characteristics of smaller MMPPs with two states; secondly, the automatic attributing of order of MMPPs used in the fitting. There is an option for manually specifying the order of MMPPs before the KPC fitting commences, but this results in a trade-off between faster training or more accurate fitting. We summarise the procedure for fitting MAPs and MMPPs using KPC-toolbox in the appendix (A.4).

Other methods of fitting MAPs and MMPPs include an algorithm proposed by Heyman and Lucatoni [34] to accurately estimate parameters of superimposed MMPPs from simulated data while reducing the number of new states to ensure model tractability. In this thesis, we extend [34] by obtaining higher response time moments from MMPP queueing models, as explained in subsequent sections, whereas Heyman and Lucatoni only produce first-order performance measures (i.e. mean queue length and mean delay).

2.3 Queueing models

In this section, we introduce key queueing concepts and justify the importance of queueing models. The queueing models used in this research are single-server queues (SSQs) of varying complexity, but we do not experiment with queueing network models (QNMs), which we leave for future work. Further, this section summarises different scheduling disciplines, describes processor-sharing applications and compares several types of queueing systems. Additionally, we define response time and explain existing work for obtaining response time moments under different scheduling disciplines.

2.3.1 Scheduling

Queueing models abstract the dynamic processes governing modern, complex systems and obtain representative performance measurements, such as response time, with minimal computational cost [27]. Fundamentally, scheduling is an integral part of queueing models for obtaining such measurements. The most well-known is the first-come, first-served (FCFS) discipline, which serves jobs in order of arrival and the job that waits the longest is served first. The best example of the FCFS discipline is in the first-in, first-out (FIFO) queue when organising a data buffer. Other scheduling disciplines include last-come, first-served (LCFS), which selects the most recent job and serves it first. The most fundamental example of a data structure which implements LCFS is a stack. In terms of system utilisation (ρ), LCFS suffers from greater variability than FCFS as $\rho \to 1$ [21].

Organising servers under processor-sharing (PS) disciplines, such as egalitarian PS (EPS), allows for current jobs to be served at equal rates. Under EPS, if there are n jobs in the system with service rate 1, each job will be served at $1/n$ times the speed of the processor, which means there is no queueing and all jobs start immediately. One useful property of EPS is its *fairness*, where the expected response time of a job is directly proportional to its size [165]. Often, the equal distribution of EPS omits important real-world applications with higher priority jobs and asymmetrical distributions, and therefore, we consider

variants of PS.

Discriminatory PS (DPS) [156], where each class j job in the system receives its own percentage of the server, extends EPS by supporting multiple job classes. In DPS, K job types are served by a vector of weights ($\alpha_j > 0, j = 1, \ldots, K$) and, assuming there are n_i class i jobs ($i = 1, \ldots, K$) in the system, each class j job is served at rate:

$$r_j(n_1, \ldots, n_K) = \frac{\alpha_j}{\sum_{i=1}^{K} \alpha_i n_i}, \; j = 1, \ldots, K \tag{2.43}$$

This suggests that the share of a job class increases with the number of jobs, which prevents classes with smaller weights from starving. Note, the system is EPS if $\alpha_i = \alpha_j$, for $i, j = 1, \ldots, K$. By varying DPS weights, the choice of instantaneous service weights of different job classes enables differentiated quality of service among specific type of jobs. For example, ADSL subscribers are offered different payment rates in return for corresponding shares of available bandwidth. Existing work in the literature proves, via experiments, that the expected unconditional response time of EPS systems is reduced by 33% with DPS [162].

Round robin (RR) scheduling offers equal time slices for each job, assigned in circular order and without priorities. The EPS algorithm is seen as an idealisation of RR scheduling in time-shared computer systems [168]. The following section incorporates distributions in queues to represent mean job arrival rates and mean job service times for respective scheduling disciplines. Thus, we form queueing systems that act as benchmarks to abstract complex systems and model performance in a cost-effective manner.

2.3.2 Type of queueing systems

The most fundamental queueing system is the M/M/1-FCFS queue, which is written in standard Kendall notation [11]. This discrete queue has Poisson mean arrival rate λ and exponential mean service rate μ for one server with FCFS scheduling. Similarly, the M/M/1 queue under PS scheduling is written as M/M/1-PS. Generalising, the G/G/m queue has general (G) arrival and service time distributions for m parallel servers. Note that arrivals and service times may have specific distributions such as hyperexponential, phase-type, MMPP-induced, etc. In this thesis, we model M/M/1-DPS and MMPP/M/1-DPS queues motivated by obtaining higher moments of delay in routers and networks [9].

It is beneficial to utilise underlying continuous-time Markov chain (CTMC) properties of discrete queues for summarising transition rates involving arrivals and service times. Discrete queues offer specific advantages over fluid queues for modelling delay through response times in routers and networks. For example, fluid queues rely on a busy period, which is the length of time from when a job first arrives in an empty system until the moment the system is empty again, and this has been used for understanding delays in routers [116]. However, an assumption is that the system must be busy to behave as a fluid queue, which affects packets with small delays. In Internet traffic scenarios, discrete

queues approximate packet sizes that are variable without being infinitesimal (i.e. as is the case in fluid queues) and can model delay at single network points such as routers. Hence, we elaborate how response times obtained from discrete queues can approximate delays.

Response time is a key performance measurement because it approximates system delay for a number of applications including large-scale storage systems, routers in networks, mobile technology and wireless sensor networks (WSNs) [21]. In networks, discrete queues approximate delay at output buffers in routers and incorporate minimum packet delay through service rates using packet sizes [28]. Typically, response time distributions can be approximated using discrete PS queues, given queueing theoretic assumptions. However, it is difficult to obtain product-form solutions for response time in PS systems due to the dependency of future arrivals in the queue. The following section discusses the important applications of PS scheduling.

2.3.3 PS applications

Within queueing systems, the PS discipline has been of considerable interest for several decades. PS has been applied to modelling the performance of bandwidth-sharing protocols in packet-switched networks [134, 143, 169]. Another PS application is approximating the fair-queueing service disciplines used in communication network routers [170], where delays and congestion control are key measures. Further, PS is useful for heavy-tailed distributions [150] and bulk arrivals [152]. Stochastic analysis of PS systems dealing with power management and energy consumption has also been of interest. More specifically, a queueing model with EPS scheduling was employed when setting bounds on performance of dynamic speed scaling [148]. When predicting queueing delays, for example, the PS discipline is more complex to model than FCFS because the remaining response times in PS depend on future (i.e. uncertain) arrivals and service requirements. Nonetheless, the simplicity of PS, coupled with fairness properties, has made it easily applicable to a variety of high-speed, computer systems that are typically abstracted by queues.

We investigate why PS is an adequate discipline for modelling servers seen in smartphones and networks. Such servers are difficult to replicate precisely in a numerically tractable way; we assume a PS scheduling discipline for a number of reasons:

- PS is popular for web server design [130] and evaluating flow-level performance of end-to-end flow control mechanisms like TCP [169].
- Under PS, there is no queueing *per se* and arriving jobs start immediately to access server resources.
- The implicit *fairness* means expected job response time is directly proportional to its size.

- PS is effective for heavy-tailed service times, which may arise, for example, as short jobs are allowed to overtake long jobs. It also facilitates tractable asymptotic analysis of heavy-tailed distributions [150].

In the subsequent section, we define one of the most important performance metrics: *response time*. Further, we explore methods of approximating delay using response time obtained from efficient PS queueing models.

2.3.4 Response times

We refer to response time (sometimes called sojourn time or waiting time) as the length of time that a job spends in the system before departing from it. In queueing terms, response time T is the sum of the queueing time and the service time (i.e. duration of customer service). The average response time is computed using Little's law, irrespectively of scheduling discipline. Let λ be the mean arrival rate, μ be the mean service rate, $\rho = \lambda/\mu < 1$ be the steady system utilisation, $L = \rho/(1-\rho)$ be the mean number of jobs in the system and $\mathbb{E}[T]$ be the mean unconditional response time. Then, it follows that $L = \lambda\mathbb{E}[T]$ and re-arranging for $\mathbb{E}[T]$ gives us:

$$\mathbb{E}[T] = \frac{L}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{\lambda/\mu}{\lambda(1-\lambda/\mu)} = \frac{1}{\mu(1-\lambda/\mu)} = \frac{1}{\mu(1-\rho)} \tag{2.44}$$

When jobs require x units of service time, the mean conditional response time is given by $\mathbb{E}[T(x)] = x/(1-\rho)$. Therefore, $\mathbb{E}[T(x)]$ is linear in x, meaning that jobs with twice the size have double the response time, on average. Note that this fairness property only applies to means. Terms only affected by the mean of the service distribution exhibit the *insensitivity* property [161]. As $\rho \to 1$, the unconditional mean response time $\mathrm{E}[T]$ tends to $1/(1-\rho)$ and is independent of the variability of service time distribution.

In the FCFS case, the response time probability density function is well known [128] as:

$$f(t) = (\mu - \lambda)e^{-(\mu-\lambda)t} \tag{2.45}$$

where $t > 0$ and $f(0) = 0$.

Calculating higher moments of response time under PS scheduling requires an advanced understanding of layered branching of incoming jobs into the system [141]. In the ensuing section, we present existing work (including limitations and improvements) for approximating response times in PS queues.

2.3.5 Response time in PS queues

There are numerous works on approximating response time under PS scheduling, but few which adopt analytical queueing theory using M/M/1-PS queues. Some of the earliest significant work on M/M/1-PS queues is by Coffman *et al.* in 1970 [122], which analysed mean and variance of waiting time for PS systems and compared the results to the FCFS

discipline. In 1980, Fayolle *et al.* [114] summarised results of Kleinrock and Mitrani for DPS scheduling and also obtained average response time (both conditionally and unconditionally on job request sizes) in M/M/1-DPS queueing systems. Further, Laplace transforms provided average waiting time for multiple class types and obtained asymptotic behaviour of service demand, but no results on higher response time moments were given.

The abstraction of PS scheduling as a layered branching of incoming jobs into the system was first used implicitly by Yashkov in 1987 [141] and has led to a derivation of conditional response time moments a decade later [142], where $\mathbb{E}[T(x)^k]$ is the k^{th} moment. We present the definition of these response time moments for the reader (omitting the proof) as follows:

$$\begin{aligned}
\mathbb{E}[T(x)^k] &= -\sum_{i=1}^{k} \binom{k}{i} (-1)^i \mathbb{E}[T(x)^{k-i}] \alpha_i(x) \\
\alpha_k(x) &= \frac{k}{(1-\rho)^k} \int_{t=0}^{x} (x-t)^{k-1} F^{(k-1)*}(t) dt \\
F^{0*}(x) &= 1 \\
F^{n*}(x) &= \int_0^x F^{(n-1)*}(x-u) dF(u) \\
F(x) &= \frac{1}{\beta_1} \int_0^x (1 - B(u)) du
\end{aligned} \tag{2.46}$$

where $B(x)$ is the general service time distribution with mean length $\beta_1 < \infty$.

It is possible to approximate response time moments using equation (2.46), but calculating moments in such a nested manner is computationally expensive. Further, Yashkov does not obtain analytical higher moments in terms of utilisation rate (i.e. system load).

In 2003, Masuyama *et al.* obtained the complementary response time distribution (or survival function) using a recursive function [151]. Specifically, assuming an M/M/1-PS queue with mean arrival rate λ, mean service rate μ and job size x, the complement of response time $\bar{T}(x) = 1 - T(x)$ is defined recursively as:

$$\bar{T}(x) = \sum_{n=0}^{\infty} (1-\rho)\rho^n \sum_{k=0}^{\infty} \frac{(\lambda+\mu)^k x^k}{k!} e^{-(\lambda+\mu)x} h_{n,k} \tag{2.47}$$

where, for $n = 0, 1, \ldots,$ and $k = 0, 1, \ldots,$ we have

$$\begin{aligned}
h_{n,0} &= 1 \\
h_{n,k+1} &= \frac{n}{n+1} \frac{\mu}{\lambda+\mu} h_{n-1,k} + \frac{\lambda}{\lambda+\mu} h_{n+1,k}
\end{aligned} \tag{2.48}$$

where $h_{-1,k} = 0$.

The advantage of Masuyama's equation is omitting calculations of moments, but the computationally intensive recursive function is more costly than Yashkov's iterative solution. Storing previous terms in a buffer would speed up calculations, but evaluating multiple infinite sums is still a computational disadvantage of Masuyama's method.

In 2004, Kim and Kim offered a joint transform to obtain response time moments for K job classes in M/M/1-DPS queues [155]. Further, Kim and Kim assume a M/M/1-DPS queueing system, where $\rho_i = \lambda_i/\mu_i$, for all jobs $i = 1, \ldots, K$, and $\rho = \sum_{i=1}^{K} \rho_i < 1$ holds. Then, let N_i be the number of jobs in the system at steady state and let $Q(z_1, \ldots, z_K) = \mathbb{E}[z_1^{N_1} \ldots z_K^{N_K}]$ be the joint probability generating function in the system at steady state. Kim and Kim tag a job i with required service time greater than x. When the tagged job i attains service x, let $S_i(x)$ and $N_{ij}(x)$ denote the elapsed response time of job i and the number of class j jobs in the system, $j = 1, \ldots, K$, respectively. Then, a joint transform derives a relation on the joint distribution of $S_i(x)$ and $N_{ij}(x)$ as follows:

$$T_{ix}(s; z_1, \ldots, z_K) = \mathbb{E}[e^{-sS_i(x)} z_1^{N_{i1}(x)} \ldots z_K^{N_{iK}(x)}] \tag{2.49}$$

which is defined for $| z_i | \leq 1$, $i = 1, \ldots, K$, and $s \geq 0$.

Kim and Kim derive an expression for the joint transform $T_{ix}(s; z_1, \ldots, z_K)$ (see [155] for proof), which is governed by the following partial differential equation (PDE):

$$\begin{aligned} \frac{\partial}{\partial x} T_{ix}(s; z_1, \ldots, z_K) = &- \sum_{j=1}^{K} \frac{\alpha_j}{\alpha_i} \left\{ \left(s + \sum_{k=1}^{K} \lambda_k (1 - z_k) \right) z_j - \mu_j (1 - z_j) \right\} \frac{\partial}{\partial z_j} T_{ix}(s; z_1, \ldots, z_K) \\ &- \left(s + \sum_{j=1}^{K} \lambda_j (1 - z_j) \right) T_{ix}(s; z_1, \ldots, z_K) \end{aligned} \tag{2.50}$$

where $i = 1, \ldots, K$.

An unconditional joint transform for the tagged class i job is given by $T_i(s; z_1, \ldots, z_K) = \mathbb{E}[e^{-sS_i} z_1^{N_{i1}} \ldots z_K^{N_{iK}}]$, where S_i and N_{ij} are the response time of the tagged job class i until its departure and the number of jobs of class j left behind at departure (tagged job excluded), respectively. Thus, the joint transform $T_i(s; z_1, \ldots, z_K)$ satisfies the following PDE:

$$\begin{aligned} &-\mu_i Q(z_1, \ldots, z_K) + \mu_i T_i(s; z_1, \ldots, z_K) \\ &\quad = - \sum_{j=1}^{K} \frac{\alpha_j}{\alpha_i} \left\{ \left(s + \sum_{k=1}^{K} \lambda_k (1 - z_k) \right) z_j - \mu_j (1 - z_j) \right\} \frac{\partial}{\partial z_j} T_i(s; z_1, \ldots, z_K) \\ &\qquad - \left(s + \sum_{j=1}^{K} \lambda_j (1 - z_j) \right) T_i(s; z_1, \ldots, z_K) \end{aligned} \tag{2.51}$$

where $i = 1, \ldots, K$.

Note the basic relation between the aforementioned joint transforms is defined as:

$$T_i(s; z_1, \ldots, z_K) = \int_0^\infty \mu_i e^{-\mu_i x} T_{ix}(s; z_1, \ldots, z_K) dx. \tag{2.52}$$

By successive differentiation of equation (2.50) with respect to s and z_j, $j = 1, \ldots, K$, conditional moments of response time are obtained. Similarly, unconditional moments of response time are derived from differentiating equation (4.18). In both cases, the method of Kim and Kim solves $(K+1)(K+2)/2$ linearly independent equations to obtain unknown moments M_i^{jk}, $0 \leq j \leq k \leq K$, for each i, $i = 1, \ldots, K$, which are defined as:

$$M_i^{jk} = \frac{\partial}{\partial z_j \partial z_k} T_i(s; z_1, \ldots, z_K) \bigg|_{s=0, z_1 = \cdots = z_K = 1} \tag{2.53}$$

We find such moments for the $K = 1$ case in this thesis, thereby validating our M/M/1-PS response time moments against the joint transform method of Kim and Kim. Further, we provide an automated algorithm (see A.1 in a later chapter) for calculating higher response time moments numerically for the $K = 2$ case.

With applications of modern systems exhibiting more complex arrival distributions than the simple Poisson arrival distribution, the M/M/1 queue is fast becoming a limited queueing model [39]. Hence, we turn our attention to MMPPs and the useful state-switching behaviour to model burstiness and capture significant correlation in job arrivals for varied real-world scenarios. The next section describes queueing models with arrivals determined by MMPP switching rates.

2.3.6 Queueing with MMPPs

We have discussed the advantages of modelling job arrivals with MMPPs over Poisson processes, namely for capturing correlation and burstiness in workloads. Combining queueing models with MMPPs as arrival processes increases the modelling capabilities of more complex system processes and adds to the possible applications. Indeed, incorporating an exponentially distributed server into an MMPP queueing model provides the capability of approximating important performance metrics such as delay through response time. Figure 2.1 shows the quasi-birth-death (QBD) process of 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, the exponentially distributed server is an oversimplification that can be improved, but there exists sufficient research suggesting that media file sizes follow exponential distributions [63]. Therefore, the approximation of exponential job sizes in discrete queues simplifies the modelling of TCP/IP flows and other protocol streams in Internet applications. However, multiple rates of service times can be specified (i.e. μ_1 and μ_2) in other distributions, such as hyper-exponential (or, more generally, phase-type) with corresponding probabilities ($p_1 + p_2 = 1$) for the rates.

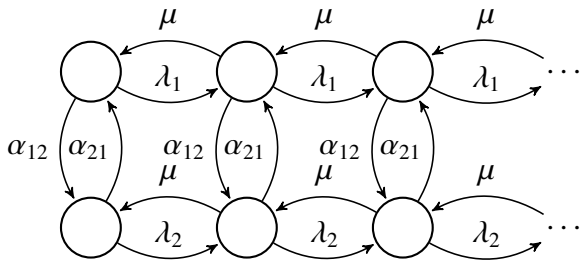

Figure 2.1: QBD process for MMPP/M/1

Queueing models allow us to abstract the dynamic processes governing modern, complex computer systems and obtain representative performance measures (i.e. response times) with minimal computational cost. By modelling multi-class jobs as part of queueing systems, one can (ideally) predict response time moments, avoid system bottlenecks and cater for resource allocation in a range of modern computer systems with spatiotemporal-specific applications.

2.3.7 Response time in MMPP/M/1 queues

Approximating response times with M/M/1-PS queues [21, 114] is sometimes viewed as insufficient for spatiotemporal Internet traffic behaviour given the weak Poisson arrival assumption. In the context of the Internet, MMPPs handle correlated streams of arrivals and, thus, can account for burstiness properties using state-switching modes. Existing work in the literature have applied MMPPs to model arrivals in FCFS queues. For example, Ciciani *et al.* [39] approximate cumulative distribution functions (CDFs) of response time for MMPP/M/1-FCFS queues through weighted superposition of separate M/M/1-FCFS queues. Incoming traffic was modelled by an MMPP with n states $(S_1, \ldots, S_n)$ and M_i/M/1-FCFS represented an M/M/1-FCFS queue[2] whose arrival rate is λ_i, as observed in state S_i. A key assumption is that if the MMPP is in state S_i long enough without transitioning to another state, the response time approaches closely the same steady state as seen in the M_i/M/1-FCFS queue. The same is assumed for the queue length. Hence, Ciciani studied two approximations for MMPP/M/1-FCFS queues based on weighted superposition of the n steady state M_i/M/1-FCFS queues: unbiased approximation and lower bound approximation.

Unbiased approximation

The mean response time ($\mathbb{E}[T]$) is averaged over the number of incoming requests, which are not distributed equally over time. For example, the rate of requests λ_i during state S_i is different from the rate λ_j in state S_j, where $i \neq j$. Nonetheless, $\mathbb{E}[T]$ for an MMPP/M/1-FCFS queue is given by a weighted sum of $\mathbb{E}[T_i]$ for M_i/M/1-FCFS queues. The weights (w_i) are scaled to account for different arrival rates for each state:

$$w_i = \frac{p_i \lambda_i}{\sum_{i=j}^{n} p_j \lambda_j} \tag{2.54}$$

[2]where the mean service rate μ is constant for all states.

Thus, $\mathbb{E}[T]$ for an MMPP/M/1-FCFS queue is given by:

$$\mathbb{E}[T] = \sum_{i=1}^{n} w_i \mathbb{E}[T_i] \tag{2.55}$$

This approximation is extended to the CDF of response time by employing the same technique of weighted superposition of the corresponding CDFs in the separate M_i/M/1-FCFS queues. However, unbiased approximation does not allow guarantees of overestimating mean response time due to errors between analytical approximations and exact values during transient periods (we refer the reader to [39] for specifics). This issue is addressed with the lower bound approximation.

Lower bound approximation

As with unbiased approximation, determining the error between the analytical approximation and the actual value is not exact. A lower bound process is constructed on the queue length by matching the behaviour of MMPP/M/1-FCFS queues during steady state periods, whilst overestimating the queue length during transient periods. The lower bound approximation is similar to the unbiased approximation in cases when the system jumps from a state of low utilisation to a state of high utilisation. However, unlike the unbiased approximation, the lower bound approximation will always jump from a state of higher utilisation to a state of lower utilisation only at the very end of the transient period – hence the overestimation. Further, Ciciani *et al.* modify the invariant probabilities p_i of the modulating Markov chain to reflect the proportions of average times spent in different states of the MMPP [39].

Initially, the lower bound approximation defines q_{ij} as the transition rate between states S_i and S_j of the MMPP. The mean arrival rates λ_i and the mean service rate μ are defined as before. Hence, the step-by-step process for the lower bound approximation is:

1. Evaluate the invariant probabilities for each state S_i in the MMPP, and denote these probabilities p_i.

2. Evaluate the transient periods for all transitions observed by using the formulas presented in [43]. Let t_{ij} be the transient period from state S_i to S_j.

3. Calculate: $p_i' = p_i + \sum_{j:\lambda_i>\lambda_j} p_i q_{ij} t_{ij} - \sum_{j:\lambda_i<\lambda_j} p_i q_{ij} t_{ij}$, where $\sum_{i=1}^{n} p_i' = 1$. Hence, this formula adds to each p_i the probability of a transient period from S_i to S_j having lower λ_j and subtracts the probability of transitioning from S_j to S_i with higher λ_j.

4. Generate a lower bound using a weighted superposition of the output processes of the MMPP states S_i and the probabilities p_i'. Further, PDFs and CDFs are obtained by similar derivations.

Ciciani *et al.* validate their approximations using data from a GRID server [43]. Initially, an analysis of normalised duration of transient periods is performed for increasing server

utilisation with a ratio metric reflecting the duration of transient periods divided by the duration of steady state periods. The results revealed that for medium to low utilisation, the ratio is less than 1.5%, but for heavy utilisation, the ratio increases to around 22%, which exposes the analytical approximations to high errors under high load scenarios. CDFs of response time are plotted under heavy and medium loads, where the simulated distribution agrees well with the analytical approximations.

In this thesis, we extend the response time approximations proposed by Ciciani *et al.* to DPS scheduling, and, thus, are able to handle multiple classes of incoming jobs. This forms an MMPP/M/1-DPS queue, which extends the EPS queue (since EPS models only one job type) and, hence, supports more real-world applications. In fact, few works have obtained higher moments of response time for DPS queues with MMPP-distributed arrivals. In terms of superposition techniques, Heyman and Lucatoni [34] reduce the number of states in superpositioned MMPPs. The MMPP/M/1-DPS queueing model used in this thesis employs a weighted superposition technique using M/M/1-DPS queues in slow transient periods of individual rates of the MMPP. With useful modelling tools such as the aforementioned queueing models, it is important to apply these efficient tools to a variety of real-world scenarios involving performance and energy measurements.

2.4 Performance-energy applications

Models used in this thesis provide a range of real-world applications for popular computer systems. This section introduces a number of applications and existing performance-energy models that have similar goals to ours. First, the focus is mobile technology including studies on measuring smartphone performance, strategies using cellular radio and battery models. Secondly, existing work on modelling performance and energy in data centres concludes this chapter. The purpose of this section is twofold: to allow the reader to gain insight into applications from which we extract important data to validate our adaptive workload models and queueing models; and to understand the limitations of existing models and, hence, highlight improvements and usefulness from our models.

2.4.1 Measuring smartphones

A recent study by Shye *et al.* on the Android G1 logged workload characteristics (i.e. phone calls, Wi-Fi sessions, etc.) for 25 users [73]. As a result, patterns in phone use, power consumption (via simple regression estimation) and Wi-Fi session durations helped characterise end-users. Further, Shye *et al.* derived a user activity model from clustering real user activity traces. Despite advantages of identifying the most power-consuming hardware components and recording changes in CPU utilisation, there were also disadvantages: CPU utilisation was logged only when the screen was on, thus ignoring background apps and syncing; no workload characterisation was attempted during Wi-Fi sessions; the "miscellaneous" state is too transient to represent specific user behaviour; studying more handsets is ideal. Work in this thesis models performance for over 100 smartphone handsets and profiles users in terms of data and power consumption in real-world contexts.

More recently, benchmarking has spawned a race for releasing the ultimate handset. For example, EEMBC [164] helps designers select optimal processors and analyse performance and energy characteristics. AndEBench-Pro evaluates Android platform performance and provides hardware tests on CPU, GPU and storage subsystems using component-specific algorithms [166]. Amongst the top handsets in 2015 (i.e. with highest AndEBench-Pro score) were Meizu-MX4 (20,683 pts) and Samsung Galaxy S5 Duos LTE (20,400 pts) [166]. Whilst measuring CPU speed and power capabilities is useful in ideal situations, most users rarely cooperate with indicated guidelines for performance optimisation. Indeed, human error leads to numerous inefficiencies such as allowing automatic syncing of apps, turning off power-saving modes, allowing frequent data downloads during intermittent Wi-Fi sessions and consistently overcharging the device for prolonged periods of time. Unfortunately, such benchmarks [164, 166] cannot measure or change user behaviour and its unpredictability. This thesis aims to forecast battery life given user behaviour through a power-consumption model and uses a performance-energy trade-off to theoretically evaluate smartphone applications.

2.4.2 Data transmission and cellular radio modes

With emerging technology companies having aimed to sell increasingly more smartphones – Xiaomi and Huawei each aimed to sell 100 million handsets in 2015 [177, 178] – wireless communication via mobile devices will only intensify. Therefore, it is important to understand the effect of asynchronous data transmission on power consumption and cost, especially from a user's point of view. Smartphones have four radios for transmitting data: cellular, Wi-Fi, bluetooth and near field communication (NFC). Cellular radio transmission occurs over large geographical areas using cellular networks and base stations. Wi-Fi is essentially a wireless local area network (WLAN) as it connects to access points from (typically) limited distances, a geographical disadvantage over cellular radio. Bluetooth is a wireless technology standard where data is exchanged between devices over short distances, typical of personal area networks (PANs). NFC allows minimal distance between mobile devices for transmitting data (i.e. less than 10 cm) and is useful for contactless payment and social networking in crowded areas.

Techniques exist for reducing the activity of the aforementioned radios to prolong battery life. For example, airplane mode disables all radios and is ideal for saving battery life when travelling or when smartphones are idle. Users might prefer to disable only unused radios and enable one that is required for the relevant activity. Cellular radio is known to drain the battery when there is no access to a cellular radio tower [182] and often is powered on for up to thirty seconds, irrespective of data transfer size [56]. Such heavy power consumption requires further investigation of cellular radio modes. Cellular radio is controlled by a state machine that balances low latency and longer battery life. When no data is transferred via radio, it enters a low-power state to save battery life. When sending data over radio, it switches to full-power mode and the application initiates data transfers. The radio waits in full-power mode for data to be transferred; if no data is detected for

five to ten seconds, the radio switches to an intermediate low-power state. Consequently, if there is no data to be transferred after a minute or so, the radio switches to stand-by mode. Switching in between modes, or state transitions, is power-consuming and drains the battery, so must be minimised.

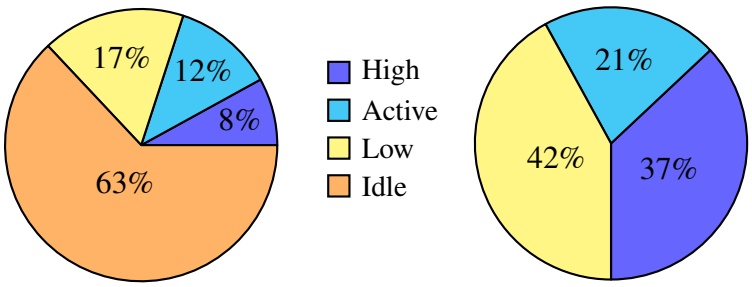

Figure 2.2: Defragmented network traffic of rare-big (left) and often-little (right) models [56].

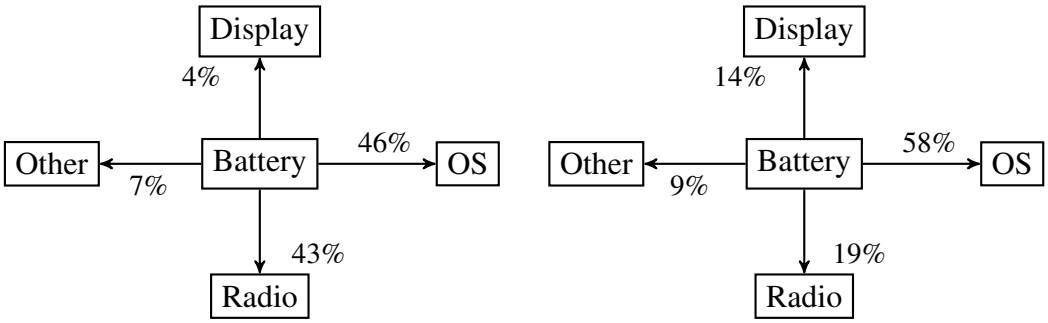

Figure 2.3: Android battery usage with high (left) and low (right) cellular radio modes [10].

There are two key transfer models for cellular radio: the "rare-big model" downloads data as infrequently as possible (with large chunks of data downloaded per session) and minimises the number of transfers whilst maximising bandwidth use; alternatively, the "often-little model" transfers small amounts of data frequently, which is costly in terms of battery discharge since radio is on nearly constantly. Figure 2.2 summarises the differences of the two techniques, where the rare-big model reduces radio usage. Figure 2.3 distinguishes between full-power and low-power modes of cellular radio for battery usage in typical Android smartphones, where monthly data was obtained from profiling. Potential user interaction exists for controlling data transfers and by switching models of data transfer, which prolongs battery life [182].

User-friendly methods of optimising data transfers exist, which can be used in conjunction with the rare-big transfer model [56]. For example, users can identify cyclic data transfers and use apps to pre-fetch this group of intermittent transfers. By generating graphs for battery usage, users create profiles and analyse patterns of updates or transfers. For example, short cyclic periods of transfers identified on the graph can be batched together, especially non-time-critical transfers. This pre-fetch of data reduces latency and

minimises full-power radio connections [32] (e.g. in six seconds, 3G pre-fetches enough information for up to 5MB of data [57]) and, thus, prolongs battery life. Such techniques assist performance prediction for mobile devices through scheduling and planning of resources. For example, a potential real-world application may schedule to use HTTP live streaming because data is transferred in bursts rather than continuously (a problem that keeps the radio consistently highly active). Updates sent to users recommending charging their device when cyclic downloads occur (i.e. app syncs and updates) are beneficial for long-term battery resource allocation. Other simple strategies include eliminating client-side polling and not relying on a refresh button [56].

2.4.3 Battery guidelines

Multinational companies including Samsung and Apple invest millions into new battery features and capabilities [183], with a focus on charging and discharging patterns and the corresponding effects on battery life. Simultaneously, smartphone researchers and scientists have debunked several battery myths originating from early 2000s. One such myth is based on the "memory effect": energy capacity of a battery is reduced after repeated recharging in a partially discharged state. This myth suggests that batteries should be fully discharged (i.e. from 100-0%) to keep their maximum capacity. In reality, this works for Ni-Mh batteries, but does not hold for modern Li-ion batteries as there is no memory effect present. Another assumption, related to heat, is that when smartphone batteries overheat this "damages" internal battery components. In fact, heat is likely to reduce battery capacity under heavy utilisation, especially for laptops where the surrounding electronics heat the battery. Similarly, maintaining a fully charged battery at elevated temperatures results in loss of capacity and shortens battery life [137].

Such "guidelines" for battery use still confuse many of today's smartphone users. Modern handsets provide self-containing energy-saving states, most of which are automatic, but may be personalised by users with an added interest. A common technique to prolong battery life is switching to a low-power mode to conserve resources [147]. For example, Android smartphones have three states: awake with screen on, awake with screen off and "deep-sleep." Therefore, when users are not actively using their smartphones, this sleep mode consumes little battery. Otherwise, when background apps are running, the use of *wakelocks* keep the phone partially awake whilst performing those processes. Some of these wakelocks can be disabled to maintain deep-sleep for longer, which acts as a power-saving mode. Other tips to save power on smartphones include: using the built-in battery usage screen to see the worst-offending apps, adjusting the backlight to be less bright, disabling GPS when not in use, and preventing apps from syncing constantly (i.e. built-in email apps). These tricks are short-term solutions and depend on user profile and activity type. For example, an avid web surfer will rarely turn radios off compared to a music listener using only local apps. Figure 2.2 and previous examples highlight the importance of considering cellular radio techniques in managing power consumption. With the help of profiling and cellular radio strategies, we address important aspects of smartphone power consumption in this thesis. Additionally, we employ a predictive model to forecast

power consumption for different handsets and measure accuracy amongst other predictive models.

2.4.4 Existing battery models

Battery behaviour is approximated via analytical models and as simulations of electrochemical processes. One example is the Battery Design Studio [126], which is a commercial electrochemical simulator; another is the Kinetic Battery Model (KiBaM) [127], which is an analytical model using kinetic abstraction. The KiBaM was especially developed to model large lead-acid storage batteries, which have a flat discharge rate, unlike Li-ion batteries found in smartphones. The KiBaM two-well model, which describes the rate capacity effect well, is adapted to different battery types, including nickel-metal hydride (NiMH) [121]. Rohner *et al.* model battery life [120], focusing on two key battery discharge behaviours: rate capacity effect, which states that battery capacity decreases as discharge rate increases, and charge recovery, which assumes that an intermittent discharge is more efficient than a continuous one. For example, the charge recovery behaviour of Rohner's simulated battery model was assumed to exhibit simple periodic loads, producing low (4mA) and high (25mA) discharge rates. However, the mode-switching of HMMs would better capture complex bursty behaviour of intermittent loads, typical of data transfer and updates in mobile apps. We model power consumption based on data activity and charging status using an online HMM.

From a queueing perspective, Jones *et al.* have attempted to prolong battery life by imposing a power-saving mode in smartphones when the charge in a rechargeable battery falls below a threshold, which can significantly prolong battery life [147]. The threshold battery level is chosen such that the power requirement of battery usage is optimised. The performance-energy models used in this thesis employs a similar idea of a power-saving mode, but relies more on minimising response times and lowering power consumption through an objective cost function. Similarly, Prabhu *et al.* also employed analytical queueing models, using batteries as servers [123], where the service system is exhaustive or non-exhaustive, thus allowing for intentional vacations during busy periods. As a result, these vacations exploit battery charge recovery increasing battery life at the expense of increased packet delay performance. A packet-delay-constraint algorithm (with server vacations) was proposed to improve the number of packets served, but did not improve mean system delay. We improve the work in [123] by extending the delay metric to higher moments and obtain a full delay distribution. Further, the unrealistic "vacations" proposed by Prabhu *et al.* may be replaced by user-defined (i.e. user-in-the-loop) switching-off of cellular radio, thus saving data and prolonging battery life.

Modelling battery life has changed with the development of smaller, wireless mobile devices. Abstracting physical processes of batteries as an equivalent circuit model (ECM) is useful for system engineers and researchers. For example, a lithium battery was modelled with thermal effects for system-level analysis as an ECM [119]. In this ECM, each component was modelled through a function with parameters that included state of charge,

temperature, current and voltage. The authors created a battery cell model using Simscape [135], which used physical modelling methods to build electrical and thermal networks. Lookup tables were used for parameter estimation, such as the voltage at specific temperatures, using Simulink [136]. For our modelling of battery life in this thesis, we use a simplified ECM setup to simulate the physical processes of Li-ion batteries, such as discharge rate.

2.4.5 Modelling data centres

We review existing techniques for modelling data centres with respect to performance and energy measurements. In fact, these terms are interrelated as a trade-off of overall costs, which must be minimised to support sustainable, energy-efficient data centres.

To recap aforementioned information, cloud service providers (CSPs) must fulfil the consumer requirements through SLAs. Typically, the SLA specifies price and quality of service (QoS). The QoS metric includes delay constraints and fairness among users. Further, predicting workloads in advance allows meeting the QoS standards and avoids overloading. SLA penalties are incurred if queueing delays of requests or other network delays occur. Key energy measurements for CSPs include electricity consumption costs, carbon emission taxes and cooling cost. Modelling power efficiency in data centres is common via power usage efficiency (PUE) and achieves an overall energy consumption measure.

The best models are often those which consider both performance and energy measurements. For example, a request routing framework named FORTE (Flow Optimisation-based framework for requesting Routing and Traffic Engineering) provides a trade-off between electricity cost, access latency and carbon footprint [132]. FORTE exploits the spatial and temporal variation of electricity price and of carbon footprint tax of grid stations by using three algorithms: the first algorithm controls the amount of user traffic directed towards each data centre; the second targets data replication given user request pattern; the third is an online algorithm which assigns user requests to data centres. Given such algorithms, FORTE reduces cost by considering the cheapest data centre with minimum latency and differentiates between request types (i.e. throughput-intensive or latency-intensive) to inform assignment decisions. Limitations of FORTE include no fairness amongst consumers, no workload prediction and no service provisioning. An improved model, known as Green-Fair, uses an algorithm to ensure fairness amongst consumers and offers constraints in latency and service capacity [133]. Green-Fair utilises green renewable energy resources to minimise queueing delays via scheduling jobs to data centres near target users. Additionally, fairness constraints apply to electricity and latency costs, which cater to strict SLA demands.

Other significant performance-energy models for data centres consider SLAs to provide energy efficient load migration and resource allocation. For example, Ghoreyshi *et al.* offer a heuristic model called VR-HM (Virtual machine Resizing Heuristic Migration), which provides online migration of VMs for handling the high failure rate in cloud en-

vironments [25]. Further, VR-HM consists of: a *global* manager to distribute jobs from users to data centres based on QoS and energy metrics; a *local* manager in each data centre to provision VMs for handling jobs, monitor fault tolerance and energy consumption of VMs, and delegate resizing and migrations of VMs. Essentially, VR-HM aims to select the most appropriate processor through considering deadline and energy consumption constraints. Efficient use of resource allocation includes intelligent scheduling combined with dynamic voltage and frequency scaling (DVFS) to keep the CPU operating at minimum voltage level, frequency and power consumption [52]. This approach reduces electricity consumption by not requiring VMs to have knowledge about underlying physical infrastructure, which is assumed in previous works.

Chapter 3

Adaptive Workload Models

Chapter Description

The adaptive workload models include: an incremental HMM (IncHMM) applied to storage workloads (3.2); a sliding HMM (SlidHMM) that updates the training set by discarding old observations (3.3); a multi-dimensional HMM (MultiHMM) that reduces computational complexity of the Baum-Welch algorithm by clustering traces (3.4); an online HMM (OnlineHMM) is formed by merging features of SlidHMM and MultiHMM (3.5).

3.1 Introduction

In this chapter, we build adaptive workload models using clustering and HMMs capable of incremental and multi-dimensional EM learning on discrete time-series. As justified in the background section, HMMs are parsimonious, can capture mode-switching and are widely applicable, thus offering advantages over Poisson processes, ARMA models and linear regression models. Further, HMMs offer portability and can efficiently build a class of workloads used for system simulation or as inputs into analytical models. The standard HMM can be improved through incremental learning, which essentially updates HMM parameters "on-the-fly" as new training data is available. This reduces the heavy computational burden of static training of standard HMMs, whilst achieving accurate trace reproduction. The incremental HMM (IncHMM) introduced in this chapter updates parameters iteratively through one-step lookahead modifications of the backward algorithm (i.e. one half of the forward-backward algorithm). The incremental training used in the IncHMM is improved by applying a sliding window to dynamically update the training set, thus forming the sliding HMM (SlidHMM). Multi-dimensional learning allows HMMs to train on multiple discrete traces simultaneously without losing accuracy of model-generated data compared to original data. The multi-dimensional HMM (MultiHMM) uses this adaptation through weights applied on clustered individual streams. Ideally, it is beneficial to combine incremental and multi-dimensional HMM features to achieve dynamic workload benchmarks capable of real-time analysis on groups of discrete workloads or time-series. Hence, we build a multi-input, online HMM (OnlineHMM) as the ideal adaptive workload model.

The data sets chosen to evaluate the adaptive HMMs should reflect the applicability and uses of each model. For example, the IncHMM, the SlidHMM, and the OnlineHMM evaluate I/O operations arriving incrementally at NetApp and Microsoft file servers. These data sets complement the online learning feature of these three models. However, the MultiHMM primarily trains on groups of traces simultaneously and, hence, evaluates Twitter data sets representing social interactions of groups of users. The MultiHMM does not support incremental learning of data and, thus, its data sets differ from those of the other three models. Nonetheless, for consistency, we provide results for MultiHMM using NetApp and Microsoft data traces. We summarise the data sets and their corresponding models in Table 3.1.

Table 3.1: The four adaptive HMMs with their corresponding data sets.

	Results on data sets	
Model	NetApp/Microsoft	Twitter
IncHMM	Section 3.2.4	-
SlidHMM	Section 3.3.6	-
MultiHMM	Section 3.5.3	Section 3.4.4
OnlineHMM	Section 3.5.3	-

The usefulness of a model should be represented by the various applications it caters for, which defines the range of data sets used to train and test the specific model. Thus, the reader should not be surprised to observe new data sets or validation strategies for each model being introduced in this chapter. We evaluate the adaptive models on different data sets via statistical methods, as follows: we compare higher moments obtained from raw traces (i.e. original and unclustered observations) with moments from synthetic traces for all adaptive HMMs after many simulation runs; the Viterbi algorithm is used as validation for the SlidHMM; correlation, burstiness and symmetric mean absolute percentage error (sMAPE) are evaluated for the MultiHMM; we calculate autocorrelation for raw and synthetic traces in the OnlineHMM; convergence rates for all adaptive HMMs highlights the most efficient model. Comparisons among adaptive HMMs in terms of Baum-Welch training complexity further support our results, which we summarise in the next section.

Training complexity

The space and time complexity for batch learning of a standard HMM using the Baum-Welch algorithm is given by $O(N^2T)$, where T is the trace length and N is the number of hidden states. Table 3.2 presents analytical results for an HMM and its variations (i.e. IncHMM, SlidHMM, MultiHMM, and OnlineHMM) measuring the following: first, the convergence complexity of the Baum-Welch algorithm when trained on H distinct traces; secondly, the convergence complexity with K incremental updates in the data set (i.e. K slides with one new data point added per slide); thirdly, the convergence complexity with K slides on H traces. Examining the four models in Table 3.2, it seems that the OnlineHMM is the least affected by scaling of H and K in terms of space and time

complexity of the Baum-Welch algorithm. For example, the IncHMM must train H separate times to learn from all H traces and the MultiHMM will learn incrementally only through re-training on the accumulated observation set with each slide. In practical scenarios found in real-world systems, the values of K and H increasing to tens of thousands would add a few seconds to the training times of standard HMMs. Further, values of K and H of over a million would add approximately a minute to the extra training times required for standard HMMs, based on online experiments. Hence, for large storage systems and networks, the extra time needed to train standard HMMs over adaptive HMMs (i.e. OnlineHMMs) would add a significant delay for detecting potential bottlenecks.

Table 3.2: Convergence for variations of the Baum-Welch algorithm.

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

3.2 Incremental HMM

We explain incremental learning in the context of storage workload modelling, where we create an incremental model for characterising I/O commands at file servers. Such models are desirable in industry for the potential of runtime analysis and planning, where the main challenges include the dependency of parameters on all preceding data. In terms of HMM dynamics, this problem requires an approximation on the new backward variables of the Baum-Welch algorithm to avoid recomputing the terms for the accumulated observation set. In fact, the difficulty in achieving an accurate approximation for the backward-recurrence formula explains why little work exists in this domain.

The IncHMM forms one part of a larger incremental storage workload model (iSWoM), which consists of various procedures and sub-models. The components of the IncHMM are given in Figure 3.1. First, we gather a raw trace consisting of I/O operations, which we transform into a binned trace using partitioning, and then into an observation trace (i.e. with observations numbered between 1 and k) using k-means clustering. Such methods are introduced to the reader in the background chapter. Secondly, our IncHMM trains on the observation trace using the adapted Baum-Welch algorithm and provides estimates for model parameters until this iterative process terminates when new data is no longer available. Thirdly, on completion, the IncHMM defines (a special case of) a discrete Markov arrival process (dMAP). The dMAP and a random distribution, which probabilistically selects input values from a specified cluster, form the iSWoM. For notational purposes, the terms IncHMM, dMAP and iSWoM are used interchangeably henceforth.

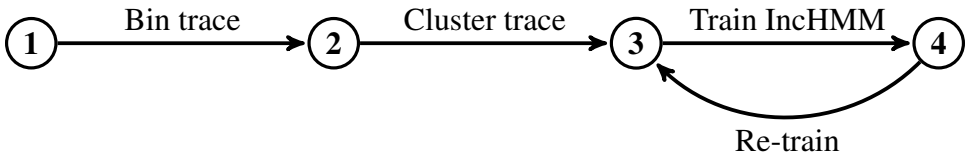

Figure 3.1: The components of the IncHMM with converged parameters at (4).

3.2.1 Motivation

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 categorise and model workload in a portable and efficient way for obtaining workload traces for live systems, on which quantitative measurements can then be made. It takes considerable time, training-data and computing power to produce a reliably parametrised 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. Therefore, the question worth investigating is: how accurate is this approach?

One of the strengths of the IncHMM is its capability of processing incoming data incrementally and updating its parameters dynamically with new data available. The IncHMM then generates unlimited discrete traces, corresponding to and representative of the respective observation traces used as input into the model. These generated traces are validated for accuracy using statistical comparisons (mean, standard deviation, etc.) against unclustered traces. The Viterbi algorithm processes both the raw and IncHMM-generated traces and produces hidden state sequences for each, where accuracy is obtained by comparing corresponding hidden state patterns.

3.2.2 Adaptive Baum-Welch algorithm

The creation of an adaptive Baum-Welch algorithm is the foundation of the IncHMM and is perceived as an updated Baum-Welch algorithm with a new forward-backward algorithm. More specifically, the backward part of the forward-backward algorithm is updated with a forward-recurrence formula such that the probabilities for the new observation set are easily calculated. We describe modifications to the forward-backward and Baum-Welch algorithms along with a mathematical approximation for the backward variables. The approximations are followed by the definition of the IncHMM parameters.

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 forms the basis of the IncHMM, which is capable of efficiently supporting real-time workload data, so demonstrating that infrequent, higher density, additional loads are handled for online characterisation of workloads as in [66]. Initially, the IncHMM is a standard HMM training on a set of observations, but

it then updates its parameters A, B, π according to 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 initially update the α and β terms of the forward-backward algorithm. First, we train a standard HMM on a trace of T observations and subsequently add M new observations. 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, obtaining $\beta_{T+1}(i)$ is more difficult because of its dependence 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, preferably a forward-recurrence formula similar to the α formula. The β approximations presented in the next sections are adapted from [1], which was a preliminary attempt on a single data trace. As explained in the following sections, the new ξ and γ variables (and also the entries a'_{ij} and $b'_j(k)$) are calculated easily once the α and β sets are complete.

First beta approximation

The first 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 $\delta(t, i)$, with respect to 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 (2.11), which calculates the β values with a one-step lookahead. All β terms ranging 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 such that $i \neq j$. We write the β approximation as:

$$\beta_t(i) \approx \beta_t(j) \tag{3.1}$$

Let us transform the β backward-recurrence formula into a forward-recurrence version. We set $N = 2$ to represent an IncHMM with two hidden states. It follows that:

$$\begin{pmatrix}\beta_t(1)\\ \beta_t(2)\end{pmatrix} = \begin{pmatrix}\sum_{j=1}^{2} a_{1j}b_j(O_{t+1})\beta_{t+1}(j)\\ \sum_{j=1}^{2} a_{2j}b_j(O_{t+1})\beta_{t+1}(j)\end{pmatrix} \tag{3.2}$$

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

$$\beta_t(1) = a_{11}b_1(O_{t+1})\beta_{t+1}(1) + a_{12}b_2(O_{t+1})\beta_{t+1}(2) \tag{3.3}$$

Assuming that $t + 1$ is sufficiently small and using (3.1) we deduce that $\beta_{t+1}(1) \approx \beta_{t+1}(2)$, giving us:

$$\beta_t(1) = \beta_{t+1}(1)(a_{11}b_1(O_{t+1}) + a_{12}b_2(O_{t+1})) \tag{3.4}$$

Re-arranging $\beta_{t+1}(1)$ as the subject results in:

$$\beta_{t+1}(1) = \frac{\beta_t(1)}{a_{11}b_1(O_{t+1}) + a_{12}b_2(O_{t+1})} \tag{3.5}$$

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

$$\beta_{t+1}(i) \approx \frac{\beta_t(i)}{\sum_{j=1}^{N} a_{ij}b_j(O_{t+1})} \tag{3.6}$$

Second beta approximation

Let us assume an IncHMM with N hidden states. By definition of β, we re-write the system of linear equations as a matrix multiplication equation:

$$\begin{pmatrix} \beta_t(1) \\ \vdots \\ \beta_t(N) \end{pmatrix} = \begin{pmatrix} a_{11}b_1(O_{t+1}) & \cdots & a_{1N}b_N(O_{t+1}) \\ \vdots & \ddots & \vdots \\ a_{N1}b_1(O_{t+1}) & \cdots & a_{NN}b_N(O_{t+1}) \end{pmatrix} \begin{pmatrix} \beta_{t+1}(1) \\ \vdots \\ \beta_{t+1}(N) \end{pmatrix} \tag{3.7}$$

Pre-multiplying by the inverse of the $N \times N$ matrix gives us:

$$\begin{pmatrix} a_{11}b_1(O_{t+1}) & \cdots & a_{1N}b_N(O_{t+1}) \\ \vdots & \ddots & \vdots \\ a_{N1}b_1(O_{t+1}) & \cdots & a_{NN}b_N(O_{t+1}) \end{pmatrix}^{-1} \begin{pmatrix} \beta_t(1) \\ \vdots \\ \beta_t(N) \end{pmatrix} = I_N \begin{pmatrix} \beta_{t+1}(1) \\ \vdots \\ \beta_{t+1}(N) \end{pmatrix} \tag{3.8}$$

where I_N is the $N \times N$ identity matrix.

Hence, terms $\beta_{t+1}(i)$, for $i = 1, \ldots, N$ are obtained from the LHS of equation (3.8), given that the $N \times N$ matrix is invertible[1]. Since we run experiments with a two-state IncHMM, we simplify equation (3.7) for the $N = 2$ case. It follows that:

$$\begin{pmatrix} \beta_t(1) \\ \beta_t(2) \end{pmatrix} = \begin{pmatrix} a_{11}b_1(O_{t+1}) & a_{12}b_2(O_{t+1}) \\ a_{21}b_1(O_{t+1}) & a_{22}b_2(O_{t+1}) \end{pmatrix} \begin{pmatrix} \beta_{t+1}(1) \\ \beta_{t+1}(2) \end{pmatrix} \tag{3.9}$$

Pre-multiplying by the 2×2 inverse matrix gives us:

$$\begin{pmatrix} a_{11}b_1(O_{t+1}) & a_{12}b_2(O_{t+1}) \\ a_{21}b_1(O_{t+1}) & a_{22}b_2(O_{t+1}) \end{pmatrix}^{-1} \begin{pmatrix} \beta_t(1) \\ \beta_t(2) \end{pmatrix} = I_2 \begin{pmatrix} \beta_{t+1}(1) \\ \beta_{t+1}(2) \end{pmatrix} \tag{3.10}$$

For the $N = 2$ case, we include the matrix inverse explicitly in the equation:

$$\begin{pmatrix} \beta_{t+1}(1) \\ \beta_{t+1}(2) \end{pmatrix} = \frac{1}{b_1(O_{t+1})b_2(O_{t+1})(a_{11}a_{22} - a_{21}a_{12})} \begin{pmatrix} a_{22}b_2(O_{t+1}) & -a_{12}b_2(O_{t+1}) \\ -a_{21}b_1(O_{t+1}) & a_{11}b_1(O_{t+1}) \end{pmatrix} \begin{pmatrix} \beta_t(1) \\ \beta_t(2) \end{pmatrix} \tag{3.11}$$

where $b_1(O_{t+1}) \neq 0$, $b_2(O_{t+1}) \neq 0$ and $a_{11}a_{22} \neq a_{21}a_{12}$.

In cases when $b_i(O_{t+1}) = 0$ for a state i, the 2×2 matrix is singular and has no inverse.

[1] For $N = 2$, we discuss cases when this matrix is singular.

Note that the third case (i.e. $a_{11}a_{22} \neq a_{21}a_{12}$) holds due to model re-parametrisation, stochastic matrix properties and initialisation of state transition probabilities. Adopting a simple β approximation for the $N = 2$ case, it follows that:

$$\begin{pmatrix}\beta_{t+1}(1)\\ \beta_{t+1}(2)\end{pmatrix} = \begin{cases}\begin{pmatrix}1.0\\ \frac{\beta_t(2)}{a_{22}b_2(O_{t+1})}\end{pmatrix}, & \text{if } b_1(O_{t+1}) = 0\\ \\ \begin{pmatrix}\frac{\beta_t(1)}{a_{11}b_1(O_{t+1})}\\ 1.0\end{pmatrix}, & \text{if } b_2(O_{t+1}) = 0\end{cases} \tag{3.12}$$

Given the two β approximations, equation (3.6) is more stable and extends to higher number of states. Inverting the matrix in equation 3.8 is more computationally complex as N increases. Further, any zero probability given by $b_i(O_{t+1})$, for $i = 1, \ldots, N$, produces a non-invertible matrix, which results in a further approximation using linear transformations (as seen in equation (3.12) for the $N = 2$ case) that are less effective with larger values of N. Hence, we use the β approximation given in equation (3.6) for validating the IncHMM on our data sets.

Approximating Baum-Welch parameters

With two β approximations defined in equations (3.6) and (3.8), we run the forward-backward algorithm and execute the α-pass, followed by the β-pass. Following the calculations of both α and β sets on the new observations $\{O_{T+1}, O_{T+2}, \ldots, O_{T+M}\}$, the ξ and γ values are defined for $T + 1 \leq t \leq T + M - 1$ as follows:

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(i)}{P(O \mid \lambda)} \tag{3.13}$$

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

$$\gamma_t = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} \tag{3.14}$$

We define the IncHMM (based on methods in [83]) using the updated forward-backward algorithm in the adaptive Baum-Welch algorithm. For each new observation, we define modified re-estimation formulas for IncHMM parameters $(\hat{\pi}, \hat{A}, \hat{B})$ as follows:

$$\hat{\pi}_i = \gamma_1(i) \tag{3.15}$$

$$\begin{aligned}\hat{a}_{ij}^{T+1} &= \frac{\sum_{t=1}^{T}\xi_t(i, j) \; + \; \xi_{T+1}(i, j)}{\sum_{t=1}^{T}\gamma_t(i) \; + \; \gamma_{T+1}(i)}\\ &= \frac{\sum_{t=1}^{T}\gamma_t(i)}{\sum_{t=1}^{T+1}\gamma_t(i)}\frac{\sum_{t=1}^{T}\xi_t(i, j)}{\sum_{t=1}^{T}\gamma_t(i)} \; + \; \frac{\xi_{T+1}(i, j)}{\sum_{t=1}^{T+1}\gamma_t(i)}\\ &= \frac{\sum_{t=1}^{T}\gamma_t(i)}{\sum_{t=1}^{T+1}\gamma_t(i)}\hat{a}_{ij}^{T} \; + \; \frac{\xi_{T+1}(i, j)}{\sum_{t=1}^{T+1}\gamma_t(i)}\end{aligned} \tag{3.16}$$

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 previous sums and in the $\hat{a}_{ij}^T$ entry).

$$\begin{aligned} \hat{b}_j(k)^{T+1} &= \frac{\sum_{t=1,O_t=k}^{T} \gamma_t(j) + \left(\gamma_{T+1}(j)|_{O_{T+1}=k}\right)}{\sum_{t=1}^{T} \gamma_t(j) + \gamma_{T+1}(j)} \\ &= \frac{\sum_{t=1}^{T} \gamma_t(i)}{\sum_{t=1}^{T+1} \gamma_t(i)} \hat{b}_j(k)^T + \frac{\gamma_{T+1}(j)|_{O_{T+1}=k}}{\sum_{t=1}^{T+1} \gamma_t(i)} \end{aligned} \tag{3.17}$$

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

We provide a summary of training the IncHMM in algorithm 1. With the training complete, subsequent sections elaborate the simulation of the IncHMM on real-world traces, which includes initial parametrisation, generating synthetic traces and obtaining relevant results.

Algorithm 1 Training the IncHMM parameters

Input: $\{O_1, \ldots, O_T\}$ = existing observation set; $\{O_{T+1}, \ldots, O_{T+M}\}$ = new observation set; N = number of hidden states; K = number of clusters in k-means; A = state transition matrix; B = observation matrix; π = initial hidden state distribution; α, β = forward-backward probabilities; ξ, γ = probability sums.

```
Train HMM on the observation set {O_1, ..., O_T} to obtain A, B, π from equations (2.17), (2.18), and (2.19).
Set π̂_i = π_i.
for t = T + 1 : T + M − 1 do
    Assign a cluster value to O_t between 1 and K.
    for i = 1 : N do
        Calculate α_t(i) = [Σ_{j=1}^{N} α_{t−1}(j) a_{ji}] b_i(O_{t−1}).
        Calculate β_t(i) and β_{t+1}(i) using equations (3.6) or (3.8).
        for j = 1 : K do
            Calculate ξ_t(i, j) and γ_t(i) using equations (3.13) and (3.14), respectively.
            Update â_{ij}^t and b̂_j(k)^t using equations (3.16) and (3.17), respectively.
        end for
    end for
end for
```

Output: $A = \{\hat{a}_{ij}\}$; $B = \{\hat{b}_j(k)\}$; $\pi = \{\hat{\pi}_i\}$

3.2.3 IncHMM simulation

We train the IncHMM on two traces of I/O commands: the first is a timestamped trace of reads and writes collected from NetApp file servers (aka the NetApp trace, hereinafter); the second timestamped trace consists of reads and writes collected from Microsoft servers (aka the Microsoft trace). We set k = 3 in the k-means clustering algorithm

after testing with different values of clusters and found that higher values of k returned sparse (and sometimes empty) clusters. We initialise the IncHMM with two hidden states as the most parsimonious model to reduce complexity of the Baum-Welch algorithm. In some experiments, attributing three or four hidden states to the IncHMM returned a converged state transition matrix with two near-identical rows of probability values.

To achieve the first simulation, the NetApp trace is first partitioned and clustered into pairs of the number of reads and writes per second. Then, this discrete trace is passed as input into the Baum-Welch algorithm as an initial training set of 8000 points (i.e. 8000 seconds). The IncHMM is trained on this set (as a standard HMM) until parameters A, B, and π converge. Afterwards, 2000 new observations are added incrementally and are evaluated using the new β approximation, given in equation (3.6), from the forward-backward algorithm. We chose this β approximation as it is more stable for larger number of states. Thus, an IncHMM will converge with fixed parameters and stores information on 10000 consecutive observation points. Note that adding 2000 increments of new observations is an arbitrary number (i.e. the number of new observations added with each incremental update should be less than the size of the original observation set) and is used in this simulation for presenting adequate results. The IncHMM generates its own synthetic NetApp trace using parameters (A, B, π), where observed values are chosen using random generation sampling and runs are simulated many thousands of times. In fact, the IncHMM possesses its own distribution of NetApp reads and writes defined by mean and standard deviation, where 95% intervals are performed on simulations. We compare IncHMM-generated results with mean and standard deviation statistics for raw (i.e. original, unclustered reads and writes) 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 trace. A new set of 2000 unseen Microsoft data points is added to the training set of 8000 points and, once converged, the IncHMM produces synthetic traces.

3.2.4 Results

NetApp and Microsoft traces

Tables 3.3 and 3.5 present statistics on reads per bin and Tables 3.4 and 3.6 represent writes per bin, where the bin is a one-second interval. For example, a "raw mean of 111.35 reads/bin" implies that the raw NetApp trace produces, on average, 111.35 read commands per second. The "IncHMM mean" and "IncHMM std dev" are the mean and standard deviation of the IncHMM-generated trace, respectively. The HMM-prefixed averages are calculated from a standard HMM-generated trace with no incremental activity.

Table 3.3 shows very similar results between raw and HMM-generated mean and standard deviation. The IncHMM produces a mean of 113.32 with a 95% confidence interval of 0.60 (to two decimal places) after 10000 simulation runs, but this mean is less accurate than the HMM-generated result. The standard deviation of the IncHMM-generated trace

(255.32) matches the raw trace well and slightly outperforms the HMM-generated value. Table 3.4 presents good results for the IncHMM-generated mean and standard deviation of write commands, which again slightly underperform compared to values produced by the HMM-generated trace.

Table 3.3: Reads/bin statistics on the raw, HMM and IncHMM NetApp traces.

Trace	Mean	Std Dev
Raw	**111.35**	**254.90**
HMM	**111.26 ± 0.66**	**254.38 ± 0.65**
IncHMM	**113.32 ± 0.60**	**255.32 ± 0.59**

Table 3.4: Writes/bin statistics on the raw, HMM and IncHMM NetApp traces.

Trace	Mean	Std Dev
Raw	**0.38**	**0.21**
HMM	**0.38 ± 0.0005**	**0.21 ± 0.001**
IncHMM	**0.41 ± 0.0005**	**0.24 ± 0.001**

Table 3.5: Reads/bin statistics on the raw, HMM and IncHMM Microsoft traces.

Trace	Mean	Std Dev
Raw	**74.23**	**214.64**
HMM	**74.38 ± 0.50**	**214.30 ± 0.67**
IncHMM	**73.76 ± 0.47**	**213.18 ± 0.62**

Table 3.5 summarises the statistics for the raw, HMM and IncHMM-generated Microsoft reads. The IncHMM mean and standard deviation are only slightly outperformed by the traditional HMM-generated traces after 10000 simulation runs. Nonetheless, the computation time of training the IncHMM on live traces is heavily reduced compared to training the standard HMM, as demonstrated in Table 3.2.

Table 3.6: Writes/bin statistics on the raw, HMM and IncHMM Microsoft traces.

Trace	Mean	Std Dev
Raw	**0.24**	**0.72**
HMM	**0.24 ± 0.001**	**0.72 ± 0.001**
IncHMM	**0.24 ± 0.001**	**0.72 ± 0.001**

The statistics for the Microsoft writes in Table 3.6 reveal similar results for raw, HMM and IncHMM-generated traces. Further, the mean and standard deviation results are almost identical for the HMM and IncHMM-generated traces with the IncHMM means outperforming the HMM means. The confidence intervals at the 95% level are identical for both models based on an empirical sampling size of 10000 simulations.

3.2.5 Related work

We compare the IncHMM with existing work in workload system benchmarks and incremental EM learning. First, we explore existing storage workload benchmarks from the literature. Zhang *et al.* conduct measurements on three benchmarks (namely TPC-W [110], TPC-C [107] and RUBiS [106]) with the aim of understanding behaviour of e-commerce storage systems [104]. These findings consist of workloads dominated by transactions (i.e. writes) requiring more storage than workloads dominated by browsing (i.e. reads). Similarities exist between the I/O traces used in this chapter and the e-commerce workload, in terms of increased demand of work. Regardless, the IncHMM provides online data characterisation for its workloads (reads or writes), which Zhang *et al.* lack. A different workload characterisation is presented by Kurmas *et al.*, where open mail traces are collected at an FC-60 disk array [109]. The authors formed a cumulative distribution function (CDF) of read latency using a workload generator that 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 IncHMM-generated traces, but no incremental learning was attempted.

Notable related work on incremental EM learning includes the adaptive symbol-wise algorithm of Florez-Larrahondo *et al.* [103], which used a backward formula in its learning that was not recursive in terms of previous β values. The IncHMM backward formulas presented in this thesis, however, store all information on the complete β set and, hence, are less prone to knowledge corruption, which is a key issue surveyed by Khreich *et al.* [47]. Additionally, the IncHMM formulas are improvements on the formula used by Stenger *et al.* [117], where all β variables were equal to one. In fact, our forward-recurrence β formula in the IncHMM is statistically validated by two different data traces, namely NetApp and Microsoft data, and produces acceptable simulated results after incremental training. Time and memory complexity offered by Florez-Larrahondo's incremental model (the best among its competitors surveyed in [47]) is maintained by the IncHMM in its incremental training. Further, our IncHMM provides a range of target applications including incremental workload benchmarks. Moreover, an extension to the symbol-wise learning of the IncHMM is to provide block-wise learning (i.e. multiple observation points trained incrementally as a sequence).

3.2.6 Conclusion and future work

HMMs, combined with the supporting clustering analysis and appropriate choice of bins, are able to provide a concise, parsimonious and portable synthetic workload. This has already been established in [66], but the deficiency of such models is their heavy computing resource requirement, which essentially precludes them from any form of online analysis. The incremental model we have developed (i.e. IncHMM) has a vastly reduced computing requirement, thus making it ideal for modelling workload data in real-time. In fact, with the availability of decoding new data, the IncHMM avoids re-training on "old data" like the traditional HMM. Additionally, compared to the resource-costly HMM, the IncHMM 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 IncHMM β approximations are successful after statistical comparisons between raw and IncHMM-generated traces.

There exist possible extensions that can be made to the IncHMM regarding the training on observation sets. For example, when new data points are submitted for training, the IncHMM increases the size of its observation set, which has accumulated outdated points from previous incremental updates. It would be beneficial to discard the outdated points whilst simultaneously training on new incoming data points, which creates a dynamic observation set that is constantly updated during model training in an incremental fashion. Further extensions to the IncHMM arise from other possible validation techniques of the model including adding skewness in the comparisons between real and synthetic traces. Validating the IncHMM against standard HMMs using hidden state sequences is a possible extension, where model-generated (i.e. synthetic) traces produce such sequences via the Viterbi algorithm. Another extension might be approximating a CDF for the IncHMM workload distribution. Given any processed trace, a CDF offers probabilities, such as observing a specific number of reads or writes within a set time, and is useful for resource planning. This will highlight the IncHMM as a more transparent probabilistic model. An extension to the symbol-wise learning of the IncHMM is to provide block-wise learning such that multiple observation traces are trained incrementally simultaneously.

3.3 Sliding HMM

In previous sections, we modelled the IncHMM to form an incremental workload model [1, 2], on which quantitative measures were made. This work proved that computation time for a reliable model can be significantly reduced, whilst maintaining model accuracy. However, improvements exist for IncHMM, which we address by forming a sliding version of the HMM (SlidHMM).

3.3.1 Motivation

The IncHMM adapted the Baum-Welch algorithm through a forward-recurrence backward approximation. However, the training of new data points results in the accumulation of an increasingly large observation set. As a result, previously trained observation points become outdated after many updates and should not necessarily be included in statistical measurements of traces at present time. Thus, we seek a more efficient online training method for discrete time analysis to improve the IncHMM. Therefore, the new addition to the IncHMM is a fixed sliding window to effectively analyse discrete data traces (appropriately discarding the outdated observations) whilst updating its model parameters.

The sliding HMM (SlidHMM) has a number of benefits over the standard HMM: first, it is capable of handling infrequent, higher density, additional loads mainly for online characterisation of workloads; secondly, it reduces the space and time complexity of the

Baum-Welch algorithm. These benefits are also matched by the IncHMM, but where the SlidHMM maintains a fixed window of observations for training, the IncHMM has an observation set that grows continuously over time. This will make the SlidHMM computationally more efficient than the IncHMM for training on large data sets and, hence, reduces the complexity of the adaptive Baum-Welch algorithm. The SlidHMM allows for effectively comparing different sections of the observation set using its sliding window, a technique which neither the IncHMM nor the standard HMM can achieve. We employ the simple moving average technique on the SlidHMM, enabling the updating of terms whilst maintaining a fixed size training window.

3.3.2 Simple moving average

A *moving 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 [98]. For a simple moving average (SMA) [99], we select a fixed subset size (n) and shift along, subtracting old points from the summation whilst simultaneously adding new points. Considering a simple example, we have data points $\{x_1, x_2, \ldots, x_n\}$ with an average of:

$$ave = \frac{x_1 + x_2 + \cdots + x_n}{n} \tag{3.18}$$

Then, from (3.18) we create a SMA by adding one more data point (x_{n+1}):

$$sma = \frac{x_1 + x_2 + \cdots + x_n + x_{n+1} - x_1}{n} = ave + \frac{x_{n+1}}{n} - \frac{x_1}{n} \tag{3.19}$$

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" observations. We replace generic data points x_t by 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.

3.3.3 Sliding Baum-Welch algorithm

To perform the slide on an observation set, the adapted Baum-Welch algorithm trains on new data, whilst storing information on the original data set, in similar fashion to IncHMM. However, the training set discards outdated observations whilst adding new points before passed as input to the Baum-Welch algorithm. With the active training window, we approximate corresponding β values using either of the incremental backward formulas and obtain α, ξ and γ sets using the standard formulas. Hence, the SlidHMM re-estimation formulas for $\hat{\pi}, \hat{A}$ and $\hat{B}$, for $i = 1, \ldots, N$, are defined as follows:

$$\hat{\pi}_i = \gamma_1(i) \tag{3.20}$$

$$\hat{a}_{ij}^{T+1} = \frac{\sum_{t=1}^{T+1} \xi_t(i,j) \; + \; \xi_{T+1}(i,j) \; - \; \xi_1(i,j)}{\sum_{t=2}^{T} \gamma_t(i) \; + \; \gamma_{T+1}(i)} = \frac{\sum_{t=2}^{T+1} \xi_t(i,j)}{\sum_{t=2}^{T+1} \gamma_t(i)} \tag{3.21}$$

$$\hat{b}_j(k)^{T+1} = \frac{\sum_{t=1,O_t=k}^{T} \gamma_t(j) \; + \; \left(\gamma_{T+1}(j)|_{O_{T+1}=k}\right) \; - \; \left(\gamma_1(j)|_{O_1=k}\right)}{\sum_{t=2}^{T} \gamma_t(j) \; + \; \gamma_{T+1}(j)} = \frac{\sum_{t=2,O_t=k}^{T+1} \gamma_t(j)}{\sum_{t=2}^{T+1} \gamma_t(j)} \tag{3.22}$$

The training methodology for the SlidHMM is identical to that in algorithm 1 with the only difference being that equations (3.21) and (3.22) are used to update $\hat{a}_{ij}^{t}$ and $\hat{b}_j(k)^t$, respectively.

3.3.4 SlidHMM convergence rates

Since the SlidHMM (and the IncHMM) requires only a partial computation of the forward and backward variables (i.e. the new observations), parameters converge faster than training with the traditional Baum-Welch algorithm. If we train a model on T new observations k successive times, the number of steps (S_1) required to train the SlidHMM is kT. In the same scenario, a standard HMM will need $T + 2T + \cdots + kT$ steps (S_2). Hence, the difference between the number of steps S_2 and S_1 is given by:

$$S_2 - S_1 = [T + 2T + \cdots + kT] - [kT] = T \sum_{i=1}^{k-1} i = \frac{T(k-1)k}{2} \tag{3.23}$$

Hence, we expect to save $T(k-1)k/2$ training steps with the SlidHMM, which can have profound effects on time and memory complexity as the terms k and T scale high for continuous training. Knowledge of reduced computational complexity allows us to move on to simulating the SlidHMM to validate the accuracy of model-generated traces against original traces.

3.3.5 SlidHMM simulation

To simulate the SlidHMM on different sets of time-series, we use the NetApp and Microsoft traces, which we introduced for the IncHMM. As we did for the IncHMM, we set k = 3 in the k-means clustering algorithm and initialise the SlidHMM with two hidden states. The main reasons for this are that higher values of k produced clusters with very few data points and three hidden states of the SlidHMM generated a converged state transition matrix with two near-identical rows. Each discrete trace has a constant length of 8000 observations, with the dynamic training window updating this observation set by appending 2000 new points and simultaneously discarding 2000 old points. At each "slide," the adapted Baum-Welch algorithm iterates until the SlidHMM parameters converge. When no new observations are added to the dynamic sliding window, the SlidHMM generates its own synthetic NetApp and Microsoft traces using the A, B, and π parameters. In a similar fashion to the IncHMM simulations, the SlidHMM results are simulated thousands of times with 95% confidence intervals. We compare the

SlidHMM-generated results with mean, standard deviation and skewness for the raw and the HMM-generated traces.

3.3.6 Results

To validate the SlidHMM, we use the NetApp and Microsoft data sets that were introduced for the IncHMM. Further, it is important to train our models on different parts of the data sets, which the sliding window of the SlidHMM provides. As mentioned previously, the SlidHMM introduces new dynamics to incremental training through updating the observation set continuously to create a dynamic observation set.

Mean and standard deviation comparisons

We present results for the NetApp and Microsoft traces in Tables 3.7 - 3.10, where a bin represents a one-second interval. Table 3.7 indicates similar results between raw and HMM-generated means and standard deviation. Out of all four traces in this section, Table 3.8 reveals the best results for the SlidHMM-generated traces, which slightly underperform skewness values produced by the HMM-generated traces.

Table 3.7: Reads/bin statistics on the raw, HMM and SlidHMM-generated NetApp traces.

Trace	Mean	Std Dev	Skew
Raw	**111.35**	**254.90**	**2.28**
HMM	**112.07 ± 0.63**	**255.23 ± 0.61**	**2.28 ± 0.01**
SlidHMM	**109.15 ± 0.55**	**255.46 ± 0.58**	**2.31 ± 0.01**

Table 3.8: Writes/bin statistics on the raw, HMM and SlidHMM-generated NetApp traces.

Trace	Mean	Std Dev	Skew
Raw	**0.38**	**0.19**	**3.57**
HMM	**0.38 ± 0.001**	**0.19 ± 0.001**	**3.59 ± 0.01**
SlidHMM	**0.38 ± 0.001**	**0.19 ± 0.001**	**3.68 ± 0.01**

Table 3.9: Reads/bin statistics on the raw, HMM and SlidHMM-generated Microsoft traces.

Trace	Mean	Std Dev	Skew
Raw	**50.61**	**180.60**	**8.04**
HMM	**50.54 ± 0.35**	**180.13 ± 0.59**	**8.07 ± 0.03**
SlidHMM	**49.30 ± 0.35**	**178.07 ± 0.59**	**8.20 ± 0.03**

Table 3.10: Writes/bin statistics on the raw, HMM and SlidHMM-generated Microsoft traces.

Trace	Mean	Std Dev	Skew
Raw	**0.55**	**0.11**	**1.95**
HMM	**0.55 ± 0.001**	**0.11 ± 0.001**	**1.96 ± 0.01**
SlidHMM	**0.54 ± 0.001**	**0.11 ± 0.001**	**2.13 ± 0.01**

Viterbi hidden state sequence

The Viterbi algorithm returns a sequence of hidden states, corresponding to a sequence of observations. Essentially, the Viterbi algorithm uses the parameters A, B and π to calculate a hidden state sequence and evaluates, through this state sequence, the similarity of parameters for two different HMMs. We compare the hidden state sequence based on the original observation trace (i.e. the sequence of observations used as input into a standard HMM), which was calculated by the Viterbi algorithm using HMM parameters, with the hidden state sequence based on the SlidHMM-generated trace. This process is applied to the NetApp trace for both models, in turn. Unlike the simulation of 10000 runs that used the Baum-Welch algorithm, the Viterbi uses only one observation trace per hidden state sequence. The results of the Viterbi algorithm using HMM parameters to decode the NetApp trace and its SlidHMM equivalent are given as follows:

Table 3.11: Viterbi state sequence ratio for the NetApp trace for HMM and SlidHMM.

Trace	State 1	State 2
HMM	**7938**	**2062**
SlidHMM	**8182**	**1818**

The sequences seen in Table 3.11 match very well, with an approximate 4:1 ratio supported by both HMM and SlidHMM traces. Thus, it proves that both the HMM and the SlidHMM produce a set of parameters that are in agreement in terms of hidden state sequences. The Viterbi algorithm validates the SlidHMM accuracy given agreeable hidden state sequences and shows the SlidHMM can obtain similar model parameters to the traditional HMM whilst reducing the computational complexity of incremental training.

3.3.7 Conclusion and future work

The sliding version of the HMM developed in this work has a vastly reduced computing requirement making it ideal for modelling workload data in real-time. In fact, with the availability of new data, the SlidHMM [3] 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 IncHMM [2], the SlidHMM will handle fast-growing observation sets more efficiently, as it trains on different parts of the data. Where the IncHMM increases its observation set after every incremental training session, the SlidHMM discards outdated data points using its sliding window.

There are a few extensions which follow from the SlidHMM. First, the sensitive autocorrelation function would be a useful validation method for our model, mainly because it allows comparisons of time-series comparison through lagged versions of the original data trace. Hidden trends can be exposed in both raw and SlidHMM-generated autocorrelated data. Secondly, we seek a multi-dimensional model to train on many discrete time-series simultaneously. This block-wise learning (i.e. training on groups of traces with one model) would be a significant addition to the symbol-wise learning (i.e. sequential training with one observation at a time given one model) of the SlidHMM. Further, coupling multi-trace analysis with incremental training offers a useful application for modelling a range of live computer systems.

3.4 Multi-dimensional HMM

In social networks, there is a need to analyse the behaviour of groups of online users for a number of reasons including network resource allocation and marketing user preferences. We address this issue with a multi-dimensional HMM (MultiHMM) to act as a workload classifier for multiple users. The MultiHMM is an adaptation of the original HMM, using clustering methods and training on multiple traces simultaneously via an adapted Baum-Welch algorithm. The goals of the MultiHMM are to classify multiple online user streams with minimal processing needs, represent burstiness and correlation among groups of users and to improve security measures in social networks. Experiments are carried out using multiple traces from Twitter data, where original traces are analysed and compared with the MultiHMM-generated traces. The metrics involved in validating our model include means, standard deviation, skewness, correlation, burstiness, symmetric mean absolute percentage error (sMAPE) and convergence rates.

3.4.1 Motivation

In the last decade, leading social media companies, such as Twitter and Facebook, have grown to manage enormous online user-populations, which has led to an explosion of stored multimedia content. For example, in Twitter's short history, the average number of tweets per day has grown from 5000 in 2007 to 500 million in 2013 [163]. Thus, the individual and collective behaviour of the ever-increasing user-base is a popular research topic and, consequently, properties of online social interactions are modelled extensively. For example, researchers have used Twitter data to examine links within tweets and determine characteristic patterns that help define misinformed events [45]. Such work has evolved the analysis of multiple crisis events (i.e. the Boston marathon bombings in 2013) and aims to prevent bad information or "rumours" spreading via Twitter and similar social media sites. To accurately represent such user activity, one of the simplest models is the Poisson process, which 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 timestamped data into "bins," and turned into

a portable, discrete-time stochastic process or time-series. From such processes, parsimonious models such as the HMM can be constructed, which provide mode-switching characteristics for efficiently classifying Internet traffic data. Further, HMMs have analysed traffic burstiness from Internet packet-level sources [69] and variations of HMMs have identified trends in user behaviour in social networks [82]. More specifically, [82] introduces a new class of coupled HMMs to describe temporal patterns of user activity which incorporate user's neighbours in the social network. Each HMM corresponds to one user and the coupling of models represents user interaction. These coupled HMMs provide better explanatory and predictive power compared with existing models such as those based on renewal processes or uncoupled HMMs. However, with increased interaction, the coupling of HMMs became more complex and the training more computationally expensive. Therefore, an improvement suggested in [82] involves developing more efficient models for the social analysis of groups of users. Another example includes HMMs used for individual email communication [67], where classifying characteristics of different groups was impossible for one model.

Not surprisingly, then, a common problem that arises from such work as [67, 82] is the lack of efficient training on multiple traces, derived from communicating users, to represent and classify multi-user behaviour. We propose a model that overcomes this issue, using a hybrid HMM to classify multi-user temporal activity, without decreasing in accuracy and computational efficiency. Essentially, this is a multi-user HMM (MultiHMM), which uses k-means clustering and a multi-input Baum-Welch algorithm to obtain the required parameters to form a discrete Markov multi-arrival process (dMMAP). The terms MultiHMM and dMMAP are used interchangeably hereinafter. We describe the MultiHMM algorithm in the following section.

3.4.2 MultiHMM algorithm

The novel contribution, namely a hybrid HMM for multi-user training, consists of a k-means clustering algorithm for user traces and a weighted multi-trace Baum-Welch algorithm (MultiBWA), which trains on multiple discrete traces simultaneously and maintains accuracy with respect to moments of trace comparisons. The metrics for validating our MultiHMM are trace means, standard deviation, skewness, correlation, burstiness, sMAPE and convergence rate. Clustering is used in pre-processing of input traces for training the Baum-Welch algorithm. One method involves k-means clustering to group the data points from h traces into k distinct clusters [83]. Each trace has data points belonging to one of c categories, which produces many possible combinations for the h traces. When h is large, this leads to very large values for k (i.e. h^c). Therefore, we propose our own clustering method, using k-means, which reduces h traces to k traces: we choose a value for c by inspection, and set $k = c$. If $k < h$, reduce h to k by grouping together data points from the same cluster; if $h = k$, then points are no longer grouped and the clustering terminates. Once the traces are grouped in this manner, we apply the standard k-means algorithm. This extra clustering reduces the computational burden for the multi-input Baum-Welch algorithm to train on the doubly-clustered traces.

Algorithm 2 Training using the MultiHMM

Input: K = number of clusters; Trace(i) = i^{th} trace; Cluster(k) = k^{th} cluster; Group$_t(k)$ = group of data points in k^{th} cluster at time t; $O_t(i)$ = data point at time t from Group i; ω_k = k^{th} weight; H = number of traces; T = length of trace; N = number of hidden states; $\{O_1, \ldots, O_T\}$ = observation set; A = state transition matrix; B = observation matrix; π = initial hidden state distribution; α, β = forward-backward probabilities; ξ, γ = probability sums.

```
for i = 1 : H do
    while Cluster points not fixed do
        Perform k-means clustering on Trace(i)
    end while
end for
for t = 1 : T do
    for j = 1 : K do
        for h = 1 : H do
            if O_t(h) ∈ Cluster(j) then
                Group_t(j) ← O_t(h)
            end if
        end for
    end for
end for
while MultiBWA parameters not converged do
    for t = 1 : T do
        for i = 1 : N do
            for j = 1 : K do
                Calculate α̂_{t+1}(i) = ω_j b_i(Group_{t+1}(j)) Σ_{m=1}^{N} α_t(m) a_{mi}
                Calculate β̂_t(i) = ω_j Σ_{m=1}^{N} a_{im} b_m(Group_{t+1}(j)) β_{t+1}(m)
            end for
        end for
    end for
    for i = 1 : N do
        Calculate π̂_i = γ_1(i)
        for j = 1 : K do
            Calculate ξ_t(i, j) and γ_t(i) using equations (2.15) and (2.16) and terms α̂_t(i) and β̂_t(i).
            Calculate â_ij = (Σ_{t=1}^{T-1} ξ_t(i,j)) / (Σ_{t=1}^{T-1} γ_t(i,j)) and b̂_j(k) = (Σ_{t=1,O_t=k}^{T} γ_t(j)) / (Σ_{t=1}^{T} γ_t(j))
        end for
    end for
end while
```

Output: $A = \{\hat{a}_{ij}\}$; $B = \{\hat{b}_j(k)\}$; $\pi = \{\hat{\pi}_i\}$

The full pseudo-code for the MultiBWA is provided in algorithm 2. The algorithm initialises its weights (ω_k) with equal probabilities for each group k, but a possible extension

would be to prioritise the weights, according to the respective user streams. These priorities can define variations of the MultiHMM, depending on the groups of users involved in training the model. After training on multiple traces simultaneously, the MultiHMM now contains multi-user information, and traces generated synthetically by the model can be compared to individual user profiles. Methods of validation includes sMAPE between original user tweeting activity and MultiHMM-generated tweets. In the next section, we explain the simulation of the MultiHMM for various groups of Twitter users along with a collection of corresponding results.

3.4.3 MultiHMM simulation

We simulate a two-state MultiHMM for different groups of Twitter users: the first simulation analyses only three Twitter users; the second simulation analyses three groups of Twitter users, with each group representing a common topic (i.e. hashtag). Timestamped "tweet" information was captured from each user and we refer to this time-series of tweets as a user's "Twitter trace." Each Twitter trace was partitioned into one hour intervals (i.e. to form a binned trace) by counting the number of tweets present in each interval or "bin." This binned trace was then filtered through a k-means clustering algorithm, where we set the number of clusters k = 5 and, thus, obtained five clusters for assigning integer values for our discrete time-series (i.e. to form an observation trace). Each data point in the observation trace is an integer between one and five (inclusive). The Twitter traces all have length 3000 (i.e. users are observed for 3000 hours) and are passed as input (in various groups) to the MultiHMM, where iterative training using the MultiBWA results in model-parameter convergence (i.e. parameters A, B, π converge).

The MultiHMM, using its fixed parameters, can generate synthetic traces on all types of user groups involved in the training and, specifically, it can output the cluster centroid representing the clustered group of a specific observation (obtained from the observation matrix B during generation of synthetic traces). Therefore, we obtain synthetic Twitter traces using MultiHMM parameters (A, B, π) and random generation sampling. Further, we simulate our results 10000 times to obtain 95% confidence intervals. In fact, the MultiHMM uses its own distribution of user Twitter data defined by mean and standard deviation. We compare our MultiHMM-generated results with mean, standard deviation and skewness for raw and (standard) HMM-generated traces; the HMM-generated trace is a result of a traditional HMM trained on an observation trace of length 3000. Note that a trace length of 3000 hours will monitor the social network for weeks and choosing shorter monitoring periods is also an option.

3.4.4 Results

Trace moments

We calculated statistics (per hour) on discrete traces of user tweets (original and synthetic) in two formats: first, each trace corresponds to one user where results are averaged per user and presented in Tables 3.12 (user 1), 3.13 (user 2) and 3.14 (user 3); secondly, each

trace represents the group activity of users with a common hashtag, which we summarise by group in Table 3.15 (group 1), Table 3.16 (group 2) and Table 3.17 (group 3). The MultiHMM-generated results match the raw results well in most cases. Note, the same MultiHMM is used to generate each trace, whereas one standard HMM produces only one trace. This is an obvious advantage of the MultiHMM over the standard HMM.

Table 3.12: Twitter User 1 Traces: Raw, HMM and MultiHMM.

Trace	Mean	Std Dev	Skewness
Raw	**1.0**	**1.54**	**1.54**
HMM	**1.0 ± 0.007**	**1.53 ± 0.004**	**1.56 ± 0.011**
MultiHMM	**0.99 ± 0.002**	**1.54 ± 0.002**	**1.56 ± 0.004**

Table 3.13: Twitter User 2 Traces: Raw, HMM and MultiHMM.

Trace	Mean	Std Dev	Skewness
Raw	**0.68**	**0.82**	**1.47**
HMM	**0.67 ± 0.003**	**0.81 ± 0.002**	**1.48 ± 0.002**
MultiHMM	**0.67 ± 0.001**	**0.78 ± 0.001**	**1.56 ± 0.001**

Table 3.14: Twitter User 3 Traces: Raw, HMM and MultiHMM.

Trace	Mean	Std Dev	Skewness
Raw	**1.0**	**1.6**	**1.62**
HMM	**1.0 ± 0.007**	**1.6 ± 0.004**	**1.64 ± 0.011**
MultiHMM	**0.97 ± 0.002**	**1.61 ± 0.002**	**1.65 ± 0.004**

Table 3.15: Twitter Group 1 Traces: Raw, HMM and MultiHMM.

Trace	Mean	Std Dev	Skewness
Raw	**56.75**	**28.53**	**1.47**
HMM	**56.53 ± 0.062**	**27.37 ± 0.034**	**1.47 ± 0.003**
MultiHMM	**57.38 ± 0.108**	**25.86 ± 0.147**	**1.68 ± 0.018**

Table 3.16: Twitter Group 2 Traces: Raw, HMM and MultiHMM.

Trace	Mean	Std Dev	Skewness
Raw	**29.57**	**20.92**	**2.5**
HMM	**29.4 ± 0.050**	**19.96 ± 0.043**	**2.51 ± 0.010**
MultiHMM	**29.31 ± 0.094**	**21.0 ± 0.188**	**2.92 ± 0.030**

Table 3.17: Twitter Group 3 Traces: Raw, HMM and MultiHMM.

Trace	Mean	Std Dev	Skewness
Raw	**41.0**	**17.81**	**0.68**
HMM	**41.26 ± 0.056**	**17.0 ± 0.017**	**0.63 ± 0.004**
MultiHMM	**40.9 ± 0.116**	**19.02 ± 0.083**	**0.75 ± 0.012**

Correlation

Correlation between users and groups of users can be used to find social "intruders" and provides a useful initial measurement for security in social networks. We define Pearson's correlation coefficient [55], as applied to a sample, as follows:

$$c = \frac{\sum_{t=1}^{N}(x_t - \bar{x})(y_t - \bar{y})}{\sqrt{\sum_{t=1}^{N}(x_t - \bar{x})^2}\sqrt{\sum_{t=1}^{N}(y_t - \bar{y})^2}} \tag{3.24}$$

where $\bar{x}$ and $\bar{y}$ are the means of observations $x_1, x_2, \ldots, x_N$ and $y_1, y_2, \ldots, y_N$, respectively. Average correlation coefficients are generated (after 10000 runs) by pairing HMM and MultiHMM-generated traces with raw traces. Table 3.18 presents statistics for individual users.

Table 3.18: Average correlation coefficients for HMM and MultiHMM-generated traces.

Trace	User 1	User 2	User 3
HMM	**0.4555**	**0.9356**	**0.9785**
MultiHMM	**0.5865**	**0.9931**	**0.9992**

Analysing pairwise correlation between users is beneficial in terms of finding trends in their *online relationships*. This could be interpreted as how often a pair of users tweet each other, whether they are online at similar times, etc. Information on pairwise user correlation is summarised in Tables 3.19 and 3.20.

Table 3.19: Pairwise correlation coefficients for four Twitter users using MultiHMM.

User	1	2	3	4
1	**1.0**	**0.23**	**-0.31**	**0.42**
2	**-**	**1.0**	**0.32**	**0.46**
3	**-**	**-**	**1.0**	**0.23**
4	**-**	**-**	**-**	**1.0**

Table 3.20: Pairwise correlation coefficients for five Twitter users using MultiHMM.

User	1	2	3	4	5
1	**1.0**	**0.20**	**0.17**	**-0.01**	**0.27**
2	**-**	**1.0**	**0.21**	**0.12**	**0.15**
3	**-**	**-**	**1.0**	**0.21**	**0.07**
4	**-**	**-**	**-**	**1.0**	**-0.01**
5	**-**	**-**	**-**	**-**	**1.0**

Burstiness

We measure trace burstiness for the HMM and the MultiHMM in relation to Twitter user activity. Figure 3.2 presents tweeting activity for an arbitrary user obtained from clustered (i.e. a trace of tweets per hour grouped into clusters), HMM, and MultiHMM-generated traces. In this example, only two clusters are used to partition the tweeting activity: one

cluster with a centroid of 1.07 tweets per hour and another cluster has a centroid of 4.89 tweets per hour. Hence, if the user had a five-hour tweeting pattern of $0, 5, 1, 0, 4$, for example, then this translates to a cluster sequence of $1, 2, 1, 1, 2$. With more clusters assigned to the user's tweeting trace, a clearer representation of burstiness would also lead to higher complexity of the k-means algorithm. Given Figure 3.2, it seems the HMM-generated trace has large periods of sparse user activity (i.e. very few tweets), unlike the clustered and MultiHMM traces, which have similar, more frequent tweeting patterns.

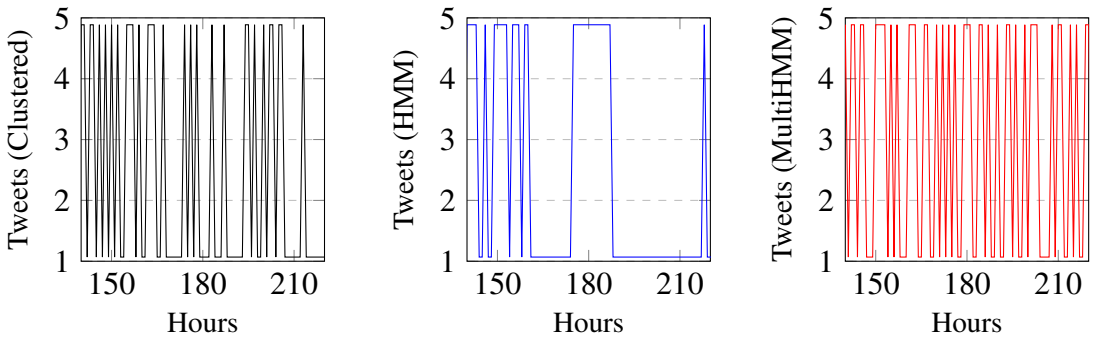

Figure 3.2: Burstiness for clustered, HMM and MultiHMM-generated data.

sMAPE

Simulations of the Baum-Welch algorithm and MultiBWA were executed 10000 times. Then, sMAPE values were obtained by comparing raw and HMM-generated traces and also comparing raw traces with MultiHMM traces. We summarise average sMAPE values in Table 3.21 for groups of users. The results reveal that the HMM is less consistent than the MultiHMM as it produces more varied errors on average after 10000 simulations.

Table 3.21: sMAPE values for Twitter groups on HMM and MultiHMM-generated traces

Trace	Group 1	Group 2	Group 3
HMM	**0.637 ± 2e-04**	**0.720 ± 2e-04**	**0.906 ± 2e-04**
MultiHMM	**0.715 ± 2e-04**	**0.741 ± 2e-04**	**0.789 ± 2e-04**

Convergence of Baum-Welch algorithm

The order of convergence of the Baum-Welch algorithm is given by $O(T^2N)$, where T is the trace length and N is the number of hidden states. We perform a simulation to analyse the computational efficiency of the Baum-Welch algorithm (trained on one trace) compared to that of the MultiBWA (multiple traces trained at once). Figure 3.3 plots the number of iterative Baum-Welch algorithm steps against the logarithm of the error (i.e. the maximum error between the state transition matrix entries at each step) and is evidence that training the MultiHMM on many traces simultaneously is more efficient than training the standard Baum-Welch algorithm per trace. Generally, the number of steps required to train the MultiHMM on h traces simultaneously (through the MultiBWA) has order $O(T^2N)$, compared to $O(T^2hN)$ for the standard HMM.

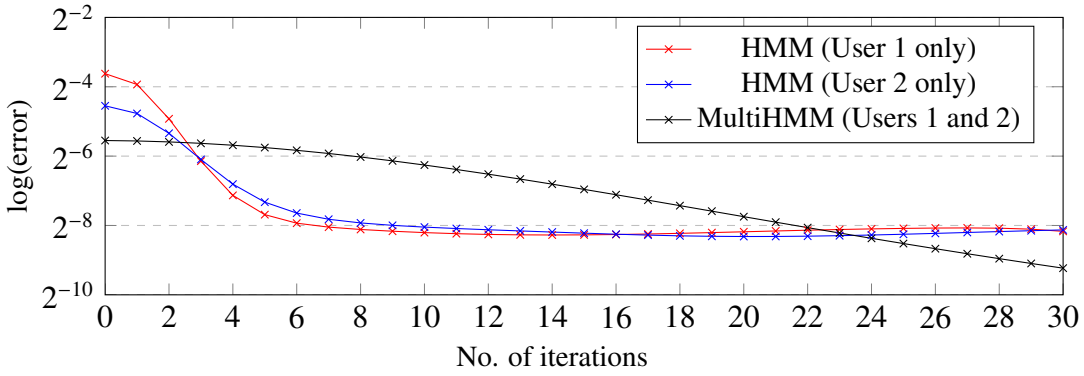

Figure 3.3: log(error) vs number of BWA and MultiBWA iterations for Twitter traces.

3.4.5 MultiHMM advantages

Based on the MultiHMM results presented in this chapter, there is statistical evidence that the accuracy of the MultiHMM matches that of the standard HMM, in terms of synthetic generation of user traces. Further, the MultiHMM reduces the computational complexity of the standard HMM whilst training on multiple traces simultaneously. We list some advantages of the MultiHMM over the standard HMM and other stochastic models:

1. Very few parameters are needed for the MultiHMM setup (i.e. A, B, π), compared to the heavy parametrisation seen in [67].

2. Periodic behaviour for the MultiHMM is not fixed to a time period, as was the case in [67], where the inter-session rate of the Poisson process was set to one week.

3. Other models (i.e. priority queueing models) fail to account for cycles and sessions of high activity (i.e. burstiness), which the MultiHMM faithfully replicates for large groups of users.

4. The MultiHMM provides inference into user behaviour through hidden states, and more so than trivial labelling of states as "passive" or "active" [67, 82].

5. The MultiHMM characterises group features of users and can help identify social "intruders" rather than prioritising a high volume of users over model features (as was done in [67]).

6. The MultiHMM saves steps in training and convergence of its MultiBWA, whereas standard HMMs are poorly-suited in situations where multiple processes interact.

7. The CoupledHMM of [82] is computationally expensive, relying on increased coupling between Markov chains to represent social influence amongst users. Our MultiHMM extends [82], acting as a basic "social influence"-driven model for groups of users.

3.4.6 Related work

We compare the MultiHMM with existing models in social media applications. In previous background sections, we described coupled HMMs with inter-connected hidden state sequences resulting in resource-costly training through many combinations of state interactions [82]. Similarly, [62] combined two parallel and independent HMMs to merge information about sign language using a token passing algorithm, but found it computationally expensive to train on very large vocabularies. The MultiHMM reduces the computational complexity of the Baum-Welch algorithm through grouping traces using two layers of clustering and, thus, is capable of training on individual traces simultaneously.

In social networks, studies show that multi-user queries result in multiple operations and are expensive in terms of performance [51]. To model these multi-user interactions, selective replicated partitioning was implemented through a temporal activity hypergraph model [51], where vertices represented users and nets corresponded to multi-user queries. The hypergraph model performed simultaneous partitioning and replication to reduce query span while respecting load balance and I/O load constraints under replication. Indeed, it was shown that the hypergraph model is the best choice (among other graph-based approaches) for predicting future query patterns and significantly improving latency and throughput. However, replicating whole sections of Twitter networks using hypergraphs method proved costly, in terms of storage and computational complexity, for increased number of users and their interactions. The MultiHMM attempts to solve some of these issues, acting as a parsimonious model (i.e. efficient training with few input parameters) capable of multi-user training with interactions on common topics.

3.4.7 Conclusion and future work

In user classification, *individual parameter estimates* fluctuate less over time than they do across individuals. Therefore, individual attributes are quite persistent and can be good candidates for characterising users as demonstrated in our multi-user classification model (i.e. MultiHMM). It has already been established that HMMs, combined with the supporting clustering analysis and appropriate choice of bins, are able to provide a concise, parsimonious and portable synthetic workload [66]. However, the lack of multi-user analysis or block-wise learning of standard HMMs results in a heavy computing resource requirement for multi-input training with such models. The MultiHMM has a vastly reduced computing requirement for parallel training making it ideal for modelling workload data with multiple streams, whilst at the same time providing excellent accuracy compared with the resource-costly traditional HMM and against the original training traces themselves. Validation of the MultiHMM, in terms of means, standard deviation, and skewness, has proved a straightforward method for verifying average "bin" probabilities of workloads (i.e. tweets per hour), in simple terms. Additionally, burstiness in a network of users has been replicated by the MultiHMM for extended periods of time. The MultiHMM improves current Markovian temporal models, with some useful potential applications for social media: *spam detection* by recognising "fake" users; modelling *group behaviour* of users on trending topics; efficient *online resource allocation* by exploiting

the highs and lows of user burstiness at peak times; user classification for *security* (e.g. normal, criminal, robot, etc.).

Extensions to our MultiHMM include using hierarchical clustering to improve the cluster allocation in different user traces. This will increase convergence times of clustering, but might give a better choice for the number of clusters and improve the accuracy of model-generated trace distributions. Also, we plan to adapt the MultiHMM training algorithm to use varying weights for each trace to represent priorities in streams of users. Another extension is to look specifically at retweets for very popular tweets (i.e. millions of users retweeting celebrities). This could predict "super busy times" in the network and thus help in resource allocation. In terms of training the model in real-time scenarios on live social networks, we seek to merge the incremental EM learning from the SlidHMM with the multi-input capabilities of the MultiHMM. In doing so, we seek to build an online benchmark to classify and predict multiple streams in a portable and parsimonious fashion.

3.5 Online HMM

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 MMPP, which can have its time index discretised to form an HMM. These models have been successful in capturing bursty behaviour and cyclic patterns of I/O operations and Internet traffic by 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 and we have addressed this with the IncHMM and the SlidHMM. Ideally, adding multi-input features of MultiHMM to such incremental models would be beneficial for modelling correlated and time-varying workload in an online fashion. 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" (i.e. SlidHMM) and, secondly, a multi-input HMM capable of training on multiple discrete traces simultaneously (i.e. MultiHMM). The OnlineHMM reduces data processing times significantly and, thence, synthetic workloads are made more computationally cost effective. We measure the accuracy of generating representative traces through comparisons of moments on original data points and HMM-generated synthetic traces. Further, we compare autocorrelation functions (ACFs) between standard HMMs and the OnlineHMM in order to validate the dynamics of the generated workload traces in terms of inter-bin correlation. We present, analytically, training steps saved from the adapted Baum-Welch algorithm used by the OnlineHMM compared to other HMM variations. Finally, we conclude our work and offer model extensions for the future.

3.5.1 Motivation

In storage systems, it is important to model workloads exhibiting spatio-temporal characteristics with multiple job classes. As discussed earlier, generated workload traces can save resources and time, where burstiness and correlation are common features of such traces. Indeed, non-deterministic events dictate traffic behaviour and should be modelled to improve storage system performance and allow for experimenting with new storage system designs. As we have seen with the IncHMM and the SlidHMM, it is beneficial to build online models for training on live system traces to reduce computational complexity and save resources. Additionally, modelling multiple job classes, as with the MultiHMM, provides correlation and burstiness for groups of workloads simultaneously, which also reduces training time. A useful extension is to combine the incremental and the multi-dimensional training of the aforementioned models to save further resources.

To address such issues, we propose an online, multi-input HMM capable of training multiple traces in an incremental fashion, without decreasing accuracy and computational efficiency. We adapt the well-known k-means clustering and Baum-Welch algorithms to support multiple traces simultaneously and obtain the discrete-time OnlineHMM. Applications of this model include multi-job 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. Potential online applications of our model include: characterising 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.

3.5.2 OnlineHMM simulation

The OnlineHMM has two hidden states and the adapted Baum-Welch algorithm is trained on observations until parameter convergence (i.e. A, B, π become fixed). Traces used for simulating the OnlineHMM include the NetApp and Microsoft traces. We choose eleven clusters for initialising the MultiHMM-adapted k-means clustering, after optimising the distance-based iterations during trial runs. The online Baum-Welch algorithm slides along the data set, which is updated incrementally, and trains on multiple traces simultaneously. This learning is a combination of the adapted Baum-Welch algorithms from the SlidHMM and the 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, the OnlineHMM, having first trained on a block of T observations, trains on T new observations, but retains information on a set of $2T$ data points. On the other hand, a standard HMM trains on T data points and then re-trains on $2T$ points when a new block of T observations are available, etc. Hence, as the observation set grows increasingly large, the standard HMM re-trains on this accumulated data set in a batch fashion. We summarise the training of the OnlineHMM in algorithm 3.

Algorithm 3 Training the OnlineHMM

Input: K = number of clusters; Trace(i) = i^{th} trace; Cluster(k) = k^{th} cluster; Group$_t(k)$ = group of data points in k^{th} cluster at time t; $O_t(i)$ = data point at time t from Group i; ω_k = k^{th} weight; $\{O_1, \ldots, O_T\}$ = existing observation set; $\{O_{T+1}, \ldots, O_{T+M}\}$ = new observation set; N = number of hidden states; A = state transition matrix; B = observation matrix; π = initial hidden state distribution; $\alpha, \beta, \xi, \gamma$ = probabilities.

Train on $\{O_1, \ldots, O_T\}$ using the MultiHMM as seen in algorithm 2 to obtain A, B, π.

for $t = T + 1 : T + M$ **do**
  for $i = 1 : H$ **do**
    while Cluster points not fixed **do**
      Perform k-means clustering on Trace(i).
    end while
  end for
  for $h = 1 : H$ **do**
    Assign a cluster value to $O_t(h)$.
    for $j = 1 : K$ **do**
      if $O_t(h) \in$ Cluster(j) **then**
        Group$_t(j) \leftarrow O_t(h)$.
      end if
    end for
  end for
end for
for $i = 1 : N$ **do**
  Calculate $\hat{\pi}_i = \gamma_1(i)$.
end for
for $t = T + 1 : T + M - 1$ **do**
  while Online BWA parameters not converged **do**
    for $i = 1 : N$ **do**
      for $j = 1 : K$ **do**
        Calculate $\hat{\alpha}_t(i) = \omega_j b_i(\text{Group}_{t-1}(j)) \sum_{m=1}^{N} \alpha_{t-1}(m) a_{mi}$.
        Calculate $\hat{\beta}_t(i) = \omega_j \beta_{t-1}(i) / \sum_{m=1}^{N} a_{im} b_m(\text{Group}_t(j))$.
        Calculate $\hat{\beta}_{t+1}(i) = \omega_j \beta_t(i) / \sum_{m=1}^{N} a_{im} b_m(\text{Group}_{t+1}(j))$.
      end for
    end for
    for $i = 1 : N$ **do**
      for $j = 1 : K$ **do**
        Calculate $\xi_t(i, j)$ and $\gamma_t(i)$ using equations (3.13) and (3.14), respectively.
        Update $\hat{a}_{ij}^t$ and $\hat{b}_j(k)^t$ using equations (3.21) and (3.22), respectively.
      end for
    end for
  end while
end for

Output: $A = \{\hat{a}_{ij}\}$; $B = \{\hat{b}_j(k)\}$; $\pi = \{\hat{\pi}_i\}$

For the OnlineHMM simulation, we initialise A, B, and π with equiprobable distributions and the length of traces (T) ranges from approximately 600 to 60000 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 with respect to computation complexity of training, which we summarise in the subsequent section. We obtain mean, standard deviation, skewness and autocorrelation on original and synthetic reads and writes, which we present in the results section.

3.5.3 Results

Trace moments

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 10000 times and corresponding statistics are evaluated per bin with 95% confidence intervals. For example, Table 3.26 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, the "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. the MultiHMM was used). A variety of traces are presented in Tables 3.22 - 3.29. As expected, the HMM provides better estimates of mean, standard deviation and skewness in most of the results. The clustering techniques and backward approximation equation used in the MultiHMM and the OnlineHMM may be responsible for the difference in accuracy of these estimates. Nonetheless, the OnlineHMM meets minimum benchmarks for accuracy, sometimes outperforming the standard HMM (see Tables 3.24 and 3.26).

Table 3.22: Twelfth Netapp read after no slides: raw, HMM and MultiHMM.

Trace	Mean	Std Dev	Skewness
Raw	**62.52**	**152.92**	**3.03**
HMM	**62.34 ± 0.001**	**151.92 ± 0.008**	**3.04 ± 0.001**
MultiHMM	**62.61 ± 0.12**	**136.30 ± 0.21**	**3.08 ± 0.005**

Table 3.23: Seventieth Netapp write after no slides: raw, HMM and MultiHMM.

Trace	Mean	Std Dev	Skewness
Raw	**0.50**	**5.54**	**19.12**
HMM	**0.50 ± 0.004**	**4.75 ± 0.05**	**15.59 ± 0.1**
MultiHMM	**0.52 ± 0.006**	**5.20 ± 0.06**	**15.81 ± 0.13**

Table 3.24: 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**

Table 3.25: Fourth Netapp write after four slides: raw, HMM and OnlineHMM.

Trace	Mean	Std Dev	Skewness
Raw	**0.43**	**0.12**	**-0.37**
HMM	**0.43 ± 0.001**	**0.12 ± 0.002**	**-0.41 ± 0.004**
OnlineHMM	**0.41 ± 0.001**	**0.21 ± 0.001**	**-0.27 ± 0.009**

Table 3.26: Fourth Microsoft read after no slides: raw, HMM and MultiHMM.

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**
MultiHMM	**49.09 ± 0.14**	**165.66 ± 0.31**	**4.29 ± 0.008**

Table 3.27: Sixth Microsoft write after no slides: raw, HMM and MultiHMM.

Trace	Mean	Std Dev	Skewness
Raw	**0.66**	**0.16**	**1.02**
HMM	**0.67 ± 0.002**	**0.16 ± 0.002**	**0.96 ± 0.012**
MultiHMM	**0.62 ± 0.001**	**0.22 ± 0.001**	**1.10 ± 0.009**

Table 3.28: 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 3.29: Twentieth Microsoft write after three slides: raw, HMM and OnlineHMM.

Trace	Mean	Std Dev	Skewness
Raw	**0.45**	**1.61**	**4.28**
HMM	**0.45 ± 0.003**	**1.65 ± 0.006**	**4.43 ± 0.02**
OnlineHMM	**0.45 ± 0.002**	**1.79 ± 0.005**	**4.51 ± 0.01**

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 lag k autocorrelation function (ACF) for observations $y_1, y_2, \ldots, y_N$ (with mean $\bar{y}$) is defined as follows:

$$p_k = \frac{\sum_{t=1}^{N-k}(y_t - \bar{y})(y_{t+k} - \bar{y})}{\sum_{t=1}^{N}(y_t - \bar{y})^2} \tag{3.25}$$

We investigate the ACFs for both NetApp and Microsoft data and compare the autocorrelation obtained from the raw (i.e. unclustered) data points with the corresponding synthetic traces as generated by the HMM and the OnlineHMM. Figure 3.4 shows varied behaviour in ACF for raw, HMM and OnlineHMM-generated NetApp reads. Self-similarity of raw reads is best captured by the OnlineHMM from lag 32 to 62 and from lag 70 to 80. For the first 30 lags, both the HMM and the OnlineHMM ACFs reveal less autocorrelation than the raw ACF, but only the OnlineHMM produced any negative values of ACF from lag 72 to 104; this differs from the ACF vaues produced by the HMM, which display spikes with values of 0.1 during the same lag period. Figure 3.5 shows that the behaviour of raw writes is matched by the OnlineHMM more accurately than the HMM, with no autocorrelation from HMM writes. The lags in which the peaks of raw ACF match the OnlineHMM ACF include 12, 36, 49, 60, and 67. Indeed, this matching of ACFs gives the OnlineHMM power as a bursty classifier, with applications in I/O management and resource allocation for disks and file servers. In Figure 3.6, there is a similar ACF pattern evident in both HMM and OnlineHMM-generated Microsoft reads, which offer slightly less ACF compared to raw reads from lag 0 to 60. Seemingly, the OnlineHMM ACF stays above zero for most lags, but the HMM reads are repeatedly moving into negative ACF values. Further, the OnlineHMM ACF matches the spikes of the raw Microsoft reads at lag 77 and again at lag 88. Figure 3.7 reveals that the HMM-generated Microsoft writes match ACF values for the first 30 lags more accurately than the OnlineHMM-generated writes. However, after lag 30, the only model to vaguely represent any significant ACF value of original writes is the OnlineHMM, acknowledging the positive peak at lag 68, whereas HMM-generated writes have ACF values only below zero.

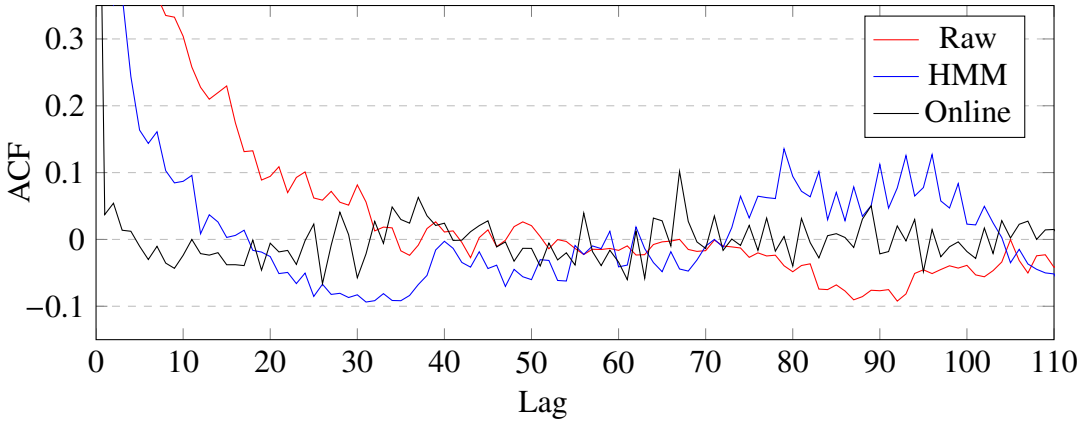

Figure 3.4: Autocorrelation for Netapp reads.

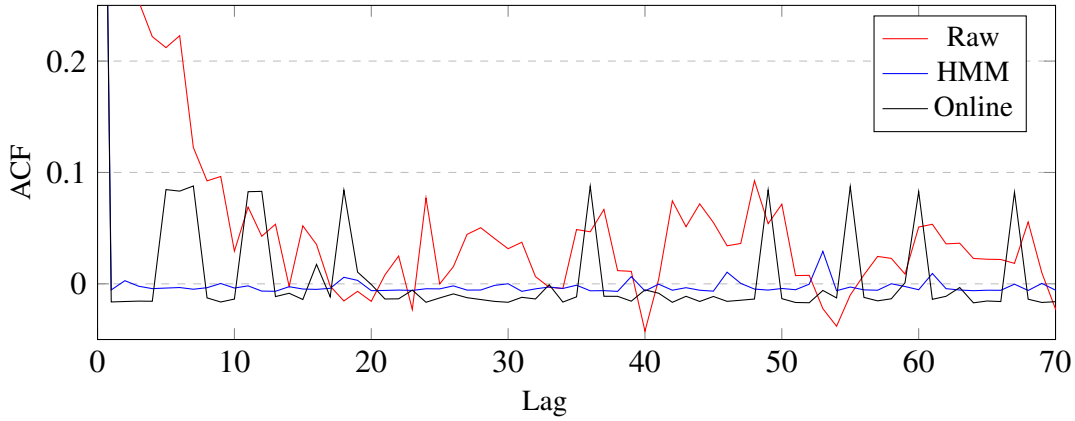

Figure 3.5: Autocorrelation for Netapp writes.

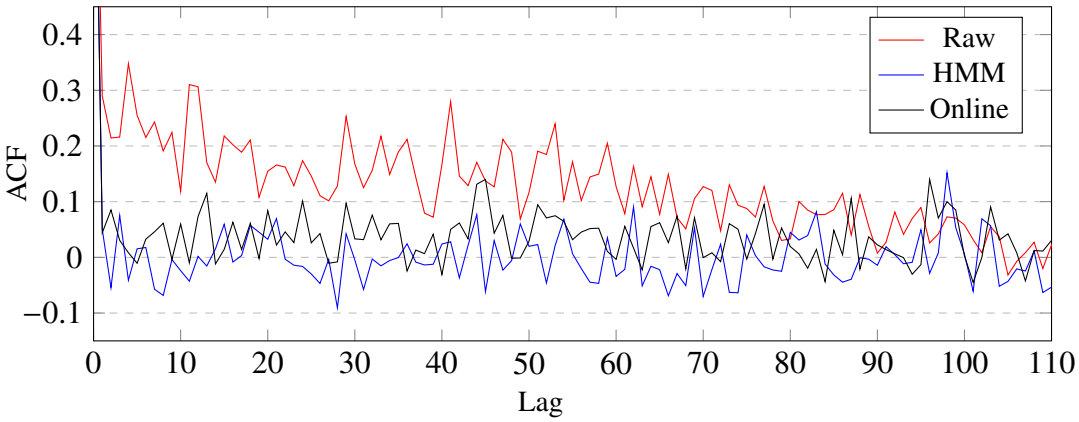

Figure 3.6: Autocorrelation for Microsoft reads.

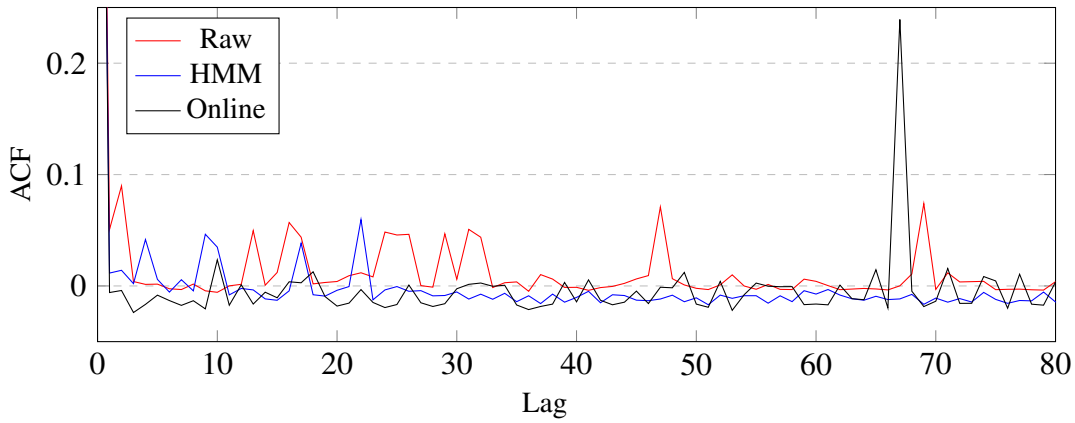

Figure 3.7: Autocorrelation for Microsoft writes.

3.5.4 Conclusion and future work

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 online HMM (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, as summarised in Table 3.2. Further, the OnlineHMM is ideal for modelling multi-class, workload data in real-time, such as collecting synthetic traces to build a profile of I/O commands at disks or of 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 and 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 after K slides) and train on a variety of traces to obtain a wider synthetic representation base. Nonetheless, from an analytical point-of-view, the OnlineHMM outperforms the standard HMM and other variations of HMMs with its adapted Baum-Welch algorithm that is based on the SlidHMM and the MultiHMM.

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 with respect to moments. By eliminating clustering altogether, it is possible to reduce computational time during training even further. 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 (or flows) to servers. Another addition to our work is obtaining an OnlineHMM that is capable of training using a continuous state space. This would require a continuous version of the Baum-Welch algorithm, as seen in [84], and would act as an online, sliding algorithm for continuous time-series.

Chapter 4

Queueing Models

Chapter Description

The queueing models in this chapter include: the M/M/1-EPS queue for calculating response time through a moment-generating algorithm (4.2); the M/M/1-DPS queue for approximating delays via response time moments for smartphones and cloud applications (4.3); the MMPP/M/1-DPS queue using a weighted superposition technique to approximate response time moments and distributions for multi-class jobs in Internet traffic (4.4).

4.1 Introduction

In this chapter, we utilise the adaptive HMMs from the previous chapter to construct efficient queueing models that represent important system performance metrics. The aims are to use the transition rates of HMMs to input into MMPPs (i.e. discretely-indexed non-stationary HMMs), which model jobs arriving at a server with processor-sharing scheduling, and to approximate queueing delay through higher response time moments. However, we first obtain response time moments using the simpler Poisson arrival process via M/M/1 queues and then build up to the more complex MMPP/M/1 queues. Hence, we present work on three types of single server queueing models to represent queueing delays for real systems. The first model is the M/M/1/EPS queue, which has long been used to model round-robin scheduling, and from which we approximate explicit response time moments via an iterative moment-generating algorithm. The M/M/1-EPS queue approximates response time moments for one-class jobs and we compare analytical results against simulated results obtained from JSIMwiz, which is a simulation package that forms part of the Java Modelling Tools (JMT) framework [37]. The second model is the M/M/1-DPS queue, which models response time for multiple job classes, where different job classes have different time quanta (i.e. larger for high priority classes). Such queueing models are approximations for servers in smartphone applications and data centres, where response time moments are the performance measures to meet quality of service (QoS) guidelines. We obtain data to test the M/M/1-DPS queue in two ways: first, we use mean arrival rates and packet sizes of TCP/IP packets captured on a network; secondly, we parametrise the DPS queue with values obtained directly from a GRID network application [43]. The third queueing model is an MMPP/M/1-DPS queue, which uses the Markov-modulated

Poisson process (MMPP) to model correlated and mode-dependent traffic (i.e. packet arrivals at routers) and can approximate long-range dependency (LRD). We obtain higher response time moments and density functions using a weighted superposition of M/M/1-DPS queues under the assumption of slow switching phases of the MMPP. Hence, response time density functions and corresponding CDFs allow comparison with realistic service level agreement (SLA) guidelines of delay. Applications of the MMPP/M/1-DPS queue include approximating delay of packets arriving at routers and building a dynamic allocation strategy to minimise mean and variance of delay whilst maximising a utility function.

4.2 M/M/1-EPS queues

4.2.1 Motivation

We have discussed how processor-sharing (PS) queueing models provide an abstraction for modern computer systems and allow analytical response time metrics to represent system delay and, hence, performance. Minimising mean response time alone is usually not acceptable nowadays because users tend to be equally frustrated with a highly variable service. They demand response time that is *predictable* [140, 165], which makes it important to calculate moments, at least, and ideally response time distributions, which is not straightforward. Further, approximating delay distributions through response time obtained from PS queues is useful for meeting QoS standards (i.e. queueing delay at routers) and verification of SLA requirements without replicating system behaviour. Additionally, varying the level of utilisation in the parametrisation of discrete queues offers useful system scenarios and stress-testing of analytical approximations. In the past three decades, existing work has addressed response time in various ways using PS queues [21, 138, 141, 142, 155, 160]. In the present work, we introduce a novel moment-generating algorithm to calculate higher response time moments analytically from M/M/1-EPS queues. Hence, we offer the following contributions:

- Iterative computation of moments, in terms of mean service rate (μ) and utilisation (ρ) of the system, using a partial differential equation for the Laplace transform of response time density.
- Obtain explicit expressions for the first four moments of response time for one job class under egalitarian PS (EPS).
- Compare analytical response time moments with simulated moments for low to high load levels.

In the background section 2.3.5, we have described other methods of obtaining response time approximations in PS queues, notably that of Kim and Kim [155]. In subsequent sections, we introduce a novel moment-generating algorithm that can *iteratively* calculate arbitrary moments of response time, thus improving an aspect of Kim and Kim's method in this respect. We explain some important assumptions for our discrete EPS queues in the next section.

4.2.2 EPS queue assumptions

Initially, we make three assumptions for our EPS queueing systems:

- The queueing system is stable (i.e. utilisation is less than one).
- Job service times are exponentially distributed.
- Slow phases allow individual Poisson arrivals in each phase.

First, it is reasonable to assume that the total offered load at the server is less than one (i.e. $\rho < 1$), which avoids saturation of resources. Secondly, existing research proposes that media file sizes follow exponential distributions [63] and, therefore, it is beneficial to simplify our analysis by choosing applications with exponential job sizes. Thirdly, it is reasonable to assume different Poisson arrivals for different phases of applications (e.g. frequent or infrequent packet arrivals). Further, it allows approximation of response time moments from separate M_i/M/1-EPS queues for each phase i.

We use discrete M/M/1-EPS queues to approximate moments of delay (i.e. represented here as response time), given the aforementioned assumptions. Such queues abstract the complex scheduling of processors when serving multiple classes of exponentially-sized jobs. Note that increasing the number of parallel processors physically adds a multi-core functionality, but in an EPS queueing model all servers are considered collectively and with a single rate. This approximation is accurate at moderate to high loads where cores can be kept busy.

The EPS queues fix discrete arrival and service rates at any given time and, therefore, differ from fluid models. However, the discrete queues distinguishes two types of job: the aggregate class of those present under equilibrium conditions and a particular new arrival with its own specific characteristics. For example, when a job arrives at the queue, n other jobs are already present with geometric probability $(1-\rho)\rho^n$ at equilibrium. Each of these n queueing jobs has a system demand given by the specified exponential distribution, having arrived from different applications with individual arrival rates[1] $(\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_n)$, which are well approximated by a superposition of independent renewal processes with aggregate arrival rate $\lambda_{all} = \sum_{i=1}^{n} \lambda_i$, which is close to Poisson if no individual arrival stream dominates. On the other hand, the server treats the newly arrived job differently from the class of existing jobs in the system in that it may have any specified service demand, x. In this way, our EPS queues act as pseudo-multi-class queues. We proceed to the pseudo-algorithm for obtaining explicit higher moments of response time, which is explained in the next section.

4.2.3 Obtaining response time moments for EPS queues

In a PS queue with utilisation ρ, the response time T of an arriving customer that requires x units of service time is known to have a probability density function that has Laplace

[1] Assuming arrivals are independent and identically distributed.

transform:

$$W^*(s \mid x) = \frac{(1-\rho)(1-\rho r^2)e^{-[\rho\mu(1-r)+s]x}}{(1-\rho r)^2 - \rho(1-r)^2 e^{-[1/r-\rho r]\mu x}} \tag{4.1}$$

where r is the smaller root of the equation $\rho r^2 - (\rho + 1 + s/\mu)r + 1 = 0$. The result is long known, see for example [21, 122], and is derived by solving a partial differential equation (PDE) for a certain generating function $G(z, s, x)$, viz.

$$(\mu z^2 - (\rho\mu + \mu + s)z + \rho\mu)\frac{\partial G}{\partial z} - \frac{\partial G}{\partial x} = (\rho\mu + s - \mu z)G \tag{4.2}$$

which yields $W^*(s \mid x) = (1-\rho)G(\rho, s, x)$. We make the following observations:

1. The unconditional response time density for an arriving customer that has exponentially distributed service time requirement with mean $1/u$ is the product of u and the Laplace transform of $W^*(s|x)$ with respect to x, evaluated at Laplace parameter u.

2. To calculate moments, the generating function's derivatives need only be computed at $s = 0$.

3. There is no need to solve the differential equation (4.2) for the generating function G since the moments are given by its derivatives evaluated at $s = 0$ and $z = \rho$, corresponding to the geometric equilibrium queue length probability distribution.

4. The Laplace transform of derivative $\partial G/\partial x$ yields the term $uG^{*x}(z, s, u) - G(z, s, 0)$, where G^{*x} denotes the Laplace transform of G with respect to x and the initial value $G(z, s, 0)$ is known to be $1/(1-z)$.

5. At $s = 0$ and $z = \rho$, the coefficient of $\partial G/\partial x$ vanishes. Thus, by successive differentiation of the Laplace-transformed equation (4.2), we can determine the moments recursively.

In this way, we obtain the following unconditional moments for response time:

$$\mathbb{E}[T] = \frac{1}{\mu(1-\rho)} \tag{4.3}$$

$$\mathbb{E}[T^2] = \frac{4}{\mu^2(2-\rho)(1-\rho)^2} \tag{4.4}$$

$$\mathbb{E}[T^3] = \frac{12(\rho+2)}{\mu^3(2-\rho)^2(1-\rho)^3} \tag{4.5}$$

$$\mathbb{E}[T^4] = \frac{48(48 + 52\rho - 10\rho^2 - 6\rho^3 - 24\rho^4 + 9\rho^5)}{\mu^4(2-\rho)^3(1-\rho)^3(3-2\rho)(4-3\rho)} \tag{4.6}$$

In Table 4.1, we summarise response time moments with fixed $\mu = 1$ whilst increasing ρ and also obtain moments with fixed $\rho = 0.5$ whilst increasing μ. The mean service rate (μ)

is measured per second and we assume one job class. We extend the moment-generating algorithm to multiple job types in subsequent sections.

Table 4.1: Moments for $\mu = 1$ with varying ρ (left) and varying μ with fixed $\rho = 0.5$ (right).

Moment	$\rho = 0.2$	$\rho = 0.5$	$\rho = 0.8$
$\mathbb{E}[T]$	**1.25**	**2.0**	**5.0**
$\mathbb{E}[T^2]$	**3.47**	**10.67**	**83.33**
$\mathbb{E}[T^3]$	**15.91**	**106.7**	**2.9e+03**
$\mathbb{E}[T^4]$	**105.3**	**1.6e+03**	**1.1e+05**

Moment	$\mu = 0.5$	$\mu = 2.5$	$\mu = 8.5$
$\mathbb{E}[T]$	**4.0**	**0.8**	**0.235**
$\mathbb{E}[T^2]$	**42.67**	**1.71**	**0.147**
$\mathbb{E}[T^3]$	**853.3**	**6.82**	**0.174**
$\mathbb{E}[T^4]$	**2.5e+04**	**40.5**	**0.303**

This method of obtaining explicit higher moments of response time should be compared against the algorithm of Kim and Kim [155]. We dedicate the following section to this comparison, which includes derivation of the same PDE for EPS queues and advantages of our moment-generating methodology over that of Kim and Kim.

4.2.4 Kim and Kim's response time moments for EPS queues

Conditional and unconditional joint transforms of response time are given in equations (2.50) and (4.18), respectively. This allows calculation of conditional and unconditional moments of response time [155], where there are K job classes. For the $K = 1$ case, let us assume the following conditions for Kim and Kim's M/M/1-EPS queueing model:

1. The mean arrival rate is λ and the mean service rate is μ.
2. Utilisation is $\rho = \lambda/\mu < 1$.
3. $z_1 = z/\rho$, where z is the parameter from equation (4.2).

Let $Q(z_1) = \mathbb{E}[z_1^{N_1}]$ be the probability generating function in the system steady state for one job type, where N_1 is the number of jobs in the system at steady state. Note that $z_1 = z/\rho$, where z is a parameter from equation (4.2), which is the difference of deconditioning on ρ. Further, Kim and Kim tag a job with required service time greater than x. Therefore, when the tagged job attains service x, let $S_1(x)$ and $N_1(x)$ denote the elapsed response time and the number of jobs in the system, respectively. Then, Kim and Kim use a joint transform to derive a relation on the joint distribution of $S_1(x)$ and $N_1(x)$:

$$T_x(s; z_1) = \mathbb{E}[e^{-sS_1(x)} z_1^{N_1(x)}] \tag{4.7}$$

which is defined for $| z_1 | \leq 1$ and $s \geq 0$.

Kim and Kim obtain an expression for the joint transform $T_x(s; z_1)$, governed by the PDE in equation (2.50), where a proof is given in [155] and is omitted here. We evaluate the expression for this PDE, for the $K = 1$ case, such that we obtain equation (4.2) as follows:

$$\frac{\partial}{\partial x} T_x(s; z_1) = -\frac{\alpha_1}{\alpha_1}\Big((s + \lambda(1 - z_1))z_1 - \mu(1 - z_1)\Big)\frac{\partial}{\partial z_1} T_x(s; z_1) - (s + \lambda(1 - z_1))T_x(s; z_1)$$

Simplifying terms gives us

$$\frac{\partial}{\partial x}T_x(s;z_1) = -\Big(sz_1 + \lambda z_1 - \lambda z_1^2 - \mu + \mu z_1\Big)\frac{\partial}{\partial z_1}T_x(s;z_1) - (s + \lambda - \lambda z_1)T_x(s;z_1)$$

Substituting z/ρ for z_1, we have

$$\frac{\partial}{\partial x}T_x(s;z_1) = -\rho\Big(s\frac{z}{\rho} + \lambda\frac{z}{\rho} - \lambda\frac{z^2}{\rho^2} - \mu + \mu\frac{z}{\rho}\Big)\frac{\partial}{\partial z}T_x(s;z_1) - (s + \lambda - \lambda\frac{z}{\rho})T_x(s;z_1)$$

Simplifying terms further and using relation $\rho = \lambda/\mu$ gives us

$$\frac{\partial}{\partial x}T_x(s;z_1) = \Big(- sz - \lambda z + \lambda\frac{z^2}{\rho} + \rho\mu - \mu z\Big)\frac{\partial}{\partial z}T_x(s;z_1) - (s + \lambda - \lambda\frac{z}{\rho})T_x(s;z_1)$$

$$\frac{\partial}{\partial x}T_x(s;z_1) = \Big(- sz - \rho\mu z + \mu z^2 + \rho\mu - \mu z\Big)\frac{\partial}{\partial z}T_x(s;z_1) - (s + \rho\mu - \mu z)T_x(s;z_1)$$

Replacing G for $T_x(s;z_1)$ and rearranging terms gives us equation (4.2) as follows

$$\Big(\mu z^2 - (\rho\mu + \mu + s)z + \rho\mu\Big)\frac{\partial G}{\partial z} - \frac{\partial G}{\partial x} = (\rho\mu + s - \mu z)G$$

Obtaining unconditional moments of response time uses repeated differentiation of the PDE given in equation (4.18), where we use the joint transform $T(s;z_1)$ for the $K = 1$ case. To obtain the first moment of response time T (i.e. $\mathbb{E}[T]$), we use Little's law. The second moment requires derivation of $(K+1)(K+2)/2$ linearly independent equations with unknown moments $L^j, M^0, M^j, M^{00}, M^{0j}$, $j = 1,\ldots,K$, and L^{jk}, M^{jk}, $k = 1,\ldots,K$, $0 \le j \le k \le K$. For $K = 1$, the moments are defined as follows:

$$L^1 = \frac{\partial}{\partial z_1}Q(z_1)\Big|_{z_1=1},\ M^0 = \frac{\partial}{\partial s}T(s;z_1)\Big|_{s=0,z_1=1},\ M^1 = \frac{\partial}{\partial z_1}T(s;z_1)\Big|_{s=0,z_1=1},$$

$$M^{00} = \frac{\partial^2}{\partial s^2}T(s;z_1)\Big|_{s=0,z_1=1},\ M^{01} = \frac{\partial^2}{\partial s\partial z_1}T(s;z_1)\Big|_{s=0,z_1=1}, \tag{4.8}$$

$$L^{11} = \frac{\partial^2}{\partial z_1^2}Q(z_1)\Big|_{z_1=1},\ M^{11} = \frac{\partial^2}{\partial z_1^2}T(s;z_1)\Big|_{s=0,z_1=1}$$

Evaluating derivatives for these moments gives us

$$L^1 = \mathbb{E}[N_1 z_1^{(N_1-1)}]\Big|_{z_1=1},\ M^0 = \mathbb{E}[-S_1 e^{-sS_1} z_1^{N_1}]\Big|_{s=0,z_1=1},\ M^1 = \mathbb{E}[e^{-sS_1} N_1 z_1^{(N_1-1)}]\Big|_{s=0,z_1=1},$$

$$M^{00} = \mathbb{E}[S_1^2 e^{-sS_1} z_1^{N_1}]\Big|_{s=0,z_1=1},\ M^{01} = \mathbb{E}[-S_1 e^{-sS_1} N_1 z_1^{(N_1-1)}]\Big|_{s=0,z_1=1},$$

$$L^{11} = \mathbb{E}[N_1(N_1-1) z_1^{(N_1-2)}]\Big|_{z_1=1},\ M^{11} = \mathbb{E}[e^{-sS_1} N_1(N_1-1) z_1^{(N_1-2)}]\Big|_{s=0,z_1=1} \tag{4.9}$$

Substituting values for s and z_1, we have

$$L^1 = \mathbb{E}[N_1],\ M^0 = \mathbb{E}[-S_1],\ M^1 = \mathbb{E}[N_1],\ M^{00} = \mathbb{E}[S_1^2],\ M^{01} = \mathbb{E}[-S_1 N_1],$$
$$L^{11} = \mathbb{E}[N_1(N_1-1)],\ M^{11} = \mathbb{E}[N_1(N_1-1)] \tag{4.10}$$

Note that $L^1 = M^1$ and $L^{11} = M^{11}$ such that these terms are used interchangeably hereinafter. Further, it is known that $\mathbb{E}[N_1] = \rho/(1-\rho)$ and $\mathbb{E}[-S_1] = -1/\mu(1-\rho)$. In the $K = 1$ case, taking partial derivatives of equation (4.18) gives us three linearly independent equations from which we solve the moments. The first equation is obtained by taking partial derivatives twice in equation (4.18) with respect to s and evaluating at $s = 0, z_1 = 1$:

$$\mu M^{00} + 2M^{01} = -2M^0 \tag{4.11}$$

Then, we take partial derivatives of equation (4.18) with respect to s and z_1 and evaluate at $s = 0, z_1 = 1$:

$$(2\mu - \lambda)M^{01} + M^{11} = \lambda M^0 - 2M^1 \tag{4.12}$$

Again we take partial derivatives twice in equation (4.18), but this time with respect to z_1 and evaluate at $s = 0, z_1 = 1$:

$$(\mu - \lambda)M^{11} = 2\lambda M^1 \tag{4.13}$$

Solving equations (4.11), (4.12) and (4.13), we obtain the following values for the moments:

$$M^{00} = \frac{4}{\mu^2(2-\rho)(1-\rho)^2};\ M^{01} = \frac{-\lambda(3-\rho)}{\mu^2(2-\rho)(1-\rho)^2};\ M^{11} = \frac{2\rho^2}{(1-\rho)^2} \tag{4.14}$$

Therefore, we verify that values for M^{00} from equation (4.14) and $\mathbb{E}[T^2]$ from equation (4.4) are indeed the same for the $K = 1$ case. Extending analysis to the third moment (with notation M^{000} or $\mathbb{E}[T^3]$) is computationally more complex and Kim and Kim do not provide explicit values for M^{000} as we do for $\mathbb{E}[T^3]$ in equation (4.5).

4.2.5 Simulating M/M/1-EPS response time moments

We approximate response time for jobs with one class ($K = 1$). Hence, we parametrise M/M/1-EPS queues with mean service rate $\mu = 1$ per second and varying utilisation from 0.2 to 0.8. We simulate the centralised response time moments from the JMT tool for M/M/1-EPS queues on six million observations and obtain 95% confidence intervals from 10000 simulated samples. Hence, we present analytical and simulated moments in Table 4.2 under varying utilisation.

Table 4.2: M/M/1-EPS moments (sec) for $\mu = 1$.

	$\lambda = \rho = 0.2$	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**1.25**	**1.25 ± 7e-04**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**1.91**	**1.94 ± 2e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**6.80**	**7.08 ± 9e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**50.97**	**52.60 ± 0.06**
	$\lambda = \rho = 0.4$	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**1.67**	**1.68 ± 1e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**4.17**	**4.19 ± 4e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**26.62**	**25.38 ± 3e-02**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**329.24**	**348.54 ± 0.34**
	$\lambda = \rho = 0.6$	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**2.50**	**2.50 ± 1e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**11.61**	**11.31 ± 0.01**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**146.05**	**152.83 ± 0.14**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**3027.4**	**2871.8 ± 2.75**
	$\lambda = \rho = 0.8$	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**5.00**	**4.94 ± 3e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**58.33**	**56.83 ± 0.05**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**1916.7**	**2006.6 ± 1.68**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**6.7e+04**	**6.6e+04 ± 13.2**

Further, we plot the mean and variance of response time in Figure 4.1 obtained from the centralised moments in Table 4.2 under increasing utilisation. The plots display the well-known "delay curve" and is an effect of queueing delay, where the average delay a packet experiences increases as utilisation increases. The analytical moments match the results of Kim and Kim [155] for the one class (i.e. $K = 1$) case.

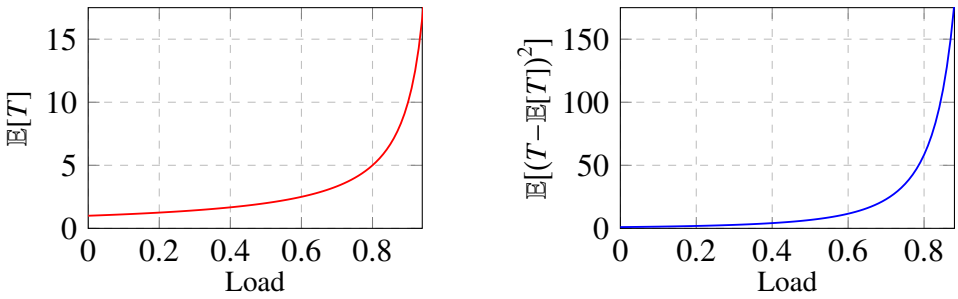

Figure 4.1: $\mathbb{E}[T]$ (left) and $\mathbb{E}[(T-\mathbb{E}[T])^2]$ (right) for increasing load.

4.2.6 Conclusion and future work

We have obtained a moment-generating algorithm for response time in M/M/1-EPS queues. Our explicit moment formulas were compared to simulated moments obtained from a JMT framework and tested under low, medium and high levels of utilisation. Comparisons with existing work by Kim and Kim [155], where a joint transform was used, reveal similarities through the reliance on the same PDE for EPS queues. However, the advantages of our methodology include iterative calculation of higher moments and explicit formula obtained for each moment.

We extend our M/M/1-EPS queues to discriminatory PS (DPS) scheduling, which supports modelling of multiple job classes as used by Kim and Kim. Further, the next queueing model experiments with more applications where priorities determine the weight of service given to each job class. Additionally, it is desirable to obtain response time densities given response time moments, which is straightforward given distribution-fitting algorithms. Such densities are useful for delay distributions used in SLA requirements for performance of many systems.

4.3 M/M/1-DPS queues

4.3.1 Motivation

We have shown some applications of EPS that have benefited from the implicit fairness properties. Often, the equal distribution of EPS omits applications with priorities and, hence, it is important to consider variants of EPS such as discriminatory PS (DPS) [156]. We have defined DPS scheduling more rigorously in the background section 2.3.1. In DPS, each job class has weights to determine the service received and the share of a job class increases with the number of jobs, which prevents classes with smaller weights from starving. By varying DPS weights, the choice of instantaneous service weights of different job classes enables differentiated quality of service among specific type of jobs. For example, ADSL subscribers are offered different payment rates in return for corresponding shares of available bandwidth. Existing work in the literature proves, via experiments, that the expected unconditional response time of PS systems is reduced by 33% with DPS [162]. With such work focusing primarily on reducing mean response times, it is important to consider higher moments to understand the effects of variance and skewness on response time distributions. Similarly, existing work for approximating response times in M/M/1-DPS queues [7, 21, 114, 155] is limited by the number of moments obtained or the number of job classes considered. Hence, we provide an automated algorithm to iteratively calculate higher response time moments (i.e. more than two) for multiple job classes in M/M/1-DPS queues. Our contributions are as follows:

- Extend the two-moment equations introduced by Kim and Kim [155] (utilised in an algorithm by Chis and Harrison [7]) via an automated algorithm.
- Provide an explicit formula for the third moment of response time in M/M/1-DPS

queues for two classes.

- Obtain arbitrary (up to any order) response time moments numerically for multi-class M/M/1-DPS queues.

4.3.2 Moment-generating algorithm for DPS queues

We build an automated moment-generating algorithm for multiple job classes, which supports DPS scheduling for K job classes and incorporates service weights α_i for each job class $i, i = 1, \ldots, K$. For simplicity of presentation, we use equal job weights (i.e. $\alpha_i = \alpha_j, i, j = 1, \ldots, K$), but this is not a requirement of our method. Adapting a multi-class version of the PDE given in equation (4.2), we apply repeated differentiation to determine moments recursively. Assuming two job classes (i.e. $K = 2$), with mean arrival rates λ_1 and λ_2, mean service rates μ_1 and μ_2, and utilisation $\rho_1 = \lambda_1/\mu_1$ and $\rho_2 = \lambda_2/\mu_2$ such that $\rho_1 + \rho_2 < 1$, we obtain respective mean response times $\mathbb{E}[T_1]$ and $\mathbb{E}[T_2]$ as:

$$\mathbb{E}[T_1] = \frac{1}{\mu_1(1-\rho_1-\rho_2)}; \; \mathbb{E}[T_2] = \frac{1}{\mu_2(1-\rho_1-\rho_2)} \tag{4.15}$$

Further, we derive second moments of response time $\mathbb{E}[T_1^2]$ and $\mathbb{E}[T_2^2]$ as:

$$\mathbb{E}[T_1^2] = \frac{4(\mu_1(1+\rho_2)+\mu_2(1-\rho_2))}{\mu_1^2(1-\rho_1-\rho_2)^2(\mu_1(2-\rho_1)+\mu_2(2-\rho_1-2\rho_2))} \tag{4.16}$$

$$\mathbb{E}[T_2^2] = \frac{4(\mu_1(1-\rho_1)+\mu_2(1+\rho_1))}{\mu_2^2(1-\rho_1-\rho_2)^2(\mu_1(2-2\rho_1-\rho_2)+\mu_2(2-\rho_2))} \tag{4.17}$$

These expressions were obtained by solving the moment equations through repeated differentiation of equation (4.18) up to two times. The algorithm to do this, written in Wolfram's Mathematica, is shown in Figure 4.2. Obtaining the variance (i.e. $\sigma_i^2 = \mathbb{E}[T_i^2] - \mathbb{E}[T_i]^2$) of a class i job reveals the spread of the response time distribution from the mean. Further, calculating higher moments of response time is useful for predicting performance in a variety of multi-class applications where jobs have different priorities – or shares of a PS server. As with the second moment, higher moments are derived by differentiating equation (4.18) and defining the steady state generating function $Q(\cdot)$, which is straightforward to derive. This is the approach used in Figure 4.2 for just two moments, which is easy to extend to any higher moments. The difficulty that arises is the number of calculations needed, since every partial derivative up to p is required to calculate moment p – a rapidly increasing number, especially if there are many classes. A symbolic solution is surely intractable, but mathematical software could easily cope with a numerical solution when values are pre-set for the parameters of the model. Using such an automated multi-class algorithm, it is straightforward to estimate the probability distribution of response time for K job classes; good approximations can usually be found from the first four moments or so.

```
K = 2;
L1sub = Solve[ (L# == λ# / μ# + Sum[αj (λj Lj# + λ# Lj) / (αj μj + α# μ#), {j, 1, K}]) & /@
    Range[K], L# & /@ Range[K]] // Simplify;
Do[M0,i = ((-Li / λi) /. L1sub) // Simplify, {i, 1, K}];
e1 =
  ((μi + μ# α# / αi) M0,#,i - λ# Sum[M0,j,i αj / αi, {j, 1, K}] + Sum[M#,j,i αj / αi, {j, 1, K}] ==
      λ# M0,i - (1 + α# / αi) M#,i) & /@ Range[K];
e2 = ((μi + 2 μ# α# / αi) M#,#,i - 2 λ# Sum[M#,j,i αj / αi, {j, 1, K}] ==
      μi L#,# + 2 (1 + α# / αi) λ# M#,i) & /@ Range[K];
e3 = Table[(μi + μk αk / αi + μm αm / αi) Mm,k,i - λk Sum[Mm,j,i αj / αi, {j, 1, K}] -
      λm Sum[Mk,j,i αj / αi, {j, 1, K}] == μi Lm,k + λk (1 + αm / αi) Mm,i + λm (1 + αk / αi) Mk,i,
    {m, 1, K}, {k, 1, m - 1}] // Flatten;
eM3 = Join[e1, e2, e3] /. Mx_,y_,i /; x > y → My,x,i /. Lx_,y_ /; x > y → Ly,x;
eM = ((μi + μ# α# / αi) M#,i == λ# Sum[Mj,i αj / αi, {j, 1, K}] + μi Li# + λ#) & /@ Range[K];
M2sub = Solve[(eM /. L1sub) // Flatten, M#,i & /@ Range[K]] // First;
eL2 = Table[Lj,k - Sum[αi (λj Lk,i + λk Li,j + λi Lj,k) / (αj μj + αk μk + αi μi), {i, 1, K}] ==
    (αj + αk) (λj Lk + λk Lj) / (αj μj + αk μk), {k, 1, K}, {j, 1, k}];
L2sub = Solve[(eL2 /. L1sub /. Lx_,y_ /; x > y → Ly,x) // Flatten,
    Table[La,b, {b, 1, K}, {a, 1, b}] // Flatten] // First // Simplify;
M0,0,i_ := (-2 M0,i - 2 Sum[M0,j,i αj / αi, {j, 1, K}]) / μi;
M3sub = Solve[eM3 /. L2sub /. M2sub, Join[Table[M0,b,i, {b, 1, K}],
    (Table[Ma,b,i, {b, 1, K}, {a, 1, b}] // Flatten)]] // First;
M0,1
```

Figure 4.2: Mathematica code for two K-class moments.

4.3.3 Case study- M/M/1-DPS analytical response times

We obtained workload traces from two applications, which we abstract using M/M/1-DPS queueing models, each with two job classes (i.e. $K = 2$). The first application is an HTC One (M7) smartphone transmitting data via 4G cellular radio, where a timestamped trace was recorded from a transmission period of 30 minutes. We summarise this HTC trace with the following mean service rates for each job class: $\mu_1 = 0.6$ and $\mu_2 = 2.4$. The second application is an Apache CloudStack VM executing programs on an Intel Core i7-2600 CPU @ 3.40GHz host machine. The CloudStack trace was recorded with mean service rates $\mu_1 = 1.4$ and $\mu_2 = 6.1$. Using equation (4.15), we plot mean response times (i.e. $\mathbb{E}[T_i]$, for $i = 1, 2$) in Figure 4.3 with increasing system load (i.e. $\rho_1 + \rho_2$) for the HTC and CloudStack traces. Further, using equations (4.16) and (4.17), we plot variance (i.e. σ_i^2, for $i = 1, 2$) of response time with increasing values of ρ_1 and ρ_2 for the HTC and CloudStack traces in Figures 4.4 and 4.5, respectively. Note that for the variance, the total system load (i.e. $\rho_1 + \rho_2$) does not exceed one. The measurements for response times are in milliseconds (ms) and throughout the analytical approximation we assume equal α weights ($\alpha_1 = \alpha_2 = 0.5$) for two job classes. For systems with more than two job classes, it is important to measure performance via response time moments for resource planning whilst considering different system load. Indeed, the DPS moment-generating algorithm

allows such measurements for any K job classes.

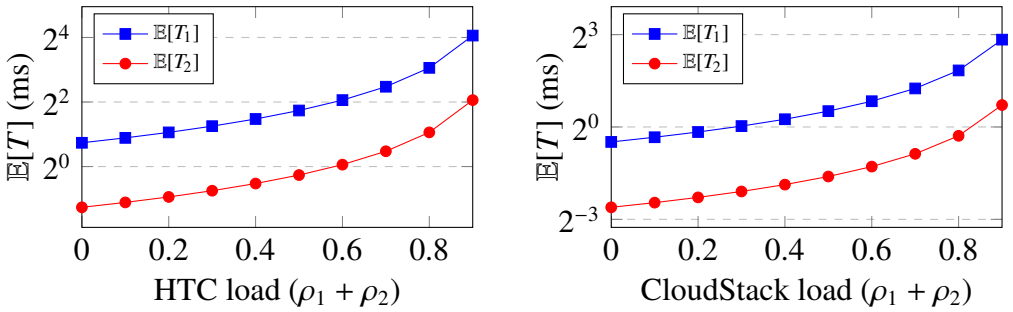

Figure 4.3: $\mathbb{E}[T_1]$ and $\mathbb{E}[T_2]$ for HTC (left) and CloudStack (right) traces under increasing load.

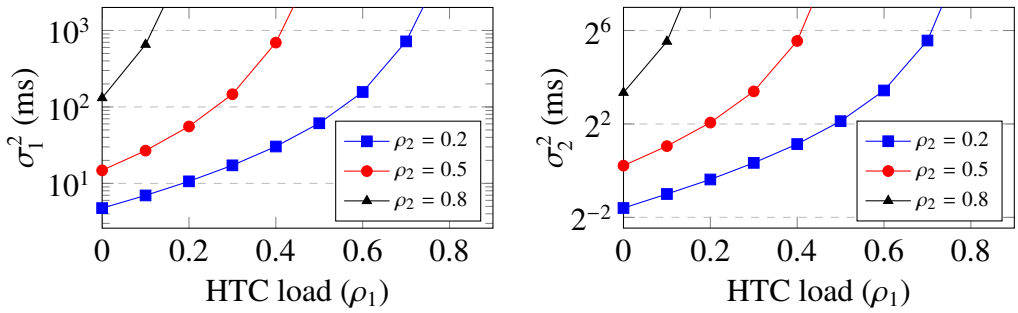

Figure 4.4: σ_1^2 (left) and σ_2^2 (right) for the HTC trace under increasing load.

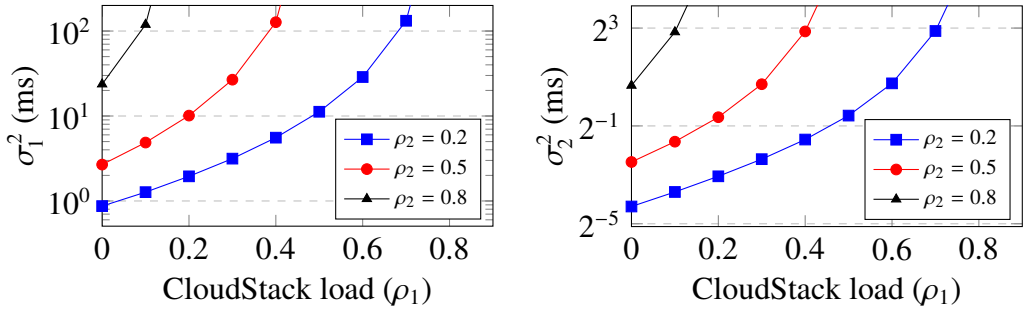

Figure 4.5: σ_1^2 (left) and σ_2^2 (right) for the CloudStack trace under increasing load.

4.3.4 Numerical algorithm for higher response time moments

We present an automated algorithm for obtaining higher response time moments from M/M/1-DPS queues. To obtain the general k^{th} response time moment in M/M/1-DPS queues, we extend the work of [7] by forming an iterative algorithm, implemented in Wolfram's Mathematica. This numerical algorithm is based on the direct approach of solving the moment-equations obtained by differentiating Kim and Kim's PDE [155] repeatedly (i.e. k times for the k^{th} moment), which is given in equation (4.18). The full details are summarized in algorithm 4 and we explain the fundamental concepts briefly. For two job classes, Kim and Kim define $T_i(s; z_1, z_2) = \mathbb{E}[e^{sS_i} z_1^{N_{i1}}, z_2^{N_{i2}}]$ as the Laplace transform of the

unconditional joint density of response time and probability generating function of class populations, for $i = 1, 2$. Note that z_0 is taken to be the Laplace-parameter s. Further, $Q(z_1, z_2) = \mathbb{E}[z_1^{N_1}, z_2^{N_2}]$ is the joint probability generating function for two job classes. The aforementioned joint transform and generating function are governed, for $i = 1, 2$, by the following PDE:

$$\begin{aligned}
&\mu_i\Big(T_i(s; z_1, z_2) - Q(z_1, z_2)\Big) = \\
&-\sum_{j=1}^{2} \frac{\alpha_j}{\alpha_i} \left\{ \Big(s + \sum_{k=1}^{2} \lambda_k(1 - z_k)\Big) z_j - \mu_j(1 - z_j) \right\} \frac{\partial}{\partial z_j} T_i(s; z_1, z_2) \\
&-\Big(s + \sum_{j=1}^{2} \lambda_j(1 - z_j)\Big) T_i(s; z_1, z_2)
\end{aligned} \tag{4.18}$$

Unconditional moments of response time are derived by differentiating equation (4.18). Initially, we substitute $s = 0$ into equation (4.18) and check that the LHS equals the RHS. The LHS evaluates to zero, as $T_i(s; z_1, z_2) = Q(z_1, z_2)$ when $s = 0$, and the RHS also evaluates to zero. The next step is to notice that the terms independent of s can be pre-calculated in advance of the terms that depend on s. Thus, differentiating w.r.t. z_1 and z_2, we calculate all possible combinations of derivatives for the k^{th} moment. For example, to obtain the fourth unconditional moment, given that $j_1 + j_2 + j_3 + j_4 = 4$, we define:

$$M_i^{j_1 j_2 j_3 j_4} = \frac{\partial^4}{\partial z_{j_1} \partial z_{j_2} \partial z_{j_3} \partial z_{j_4}} T_i(s; z_1, z_2) \bigg|_{s=0, z_1 = z_2 = 1} \tag{4.19}$$

Hence, the fourth unconditional response time moment for a class i job is simply M_i^{0000} from equation (4.19) as we set $s = z_0$ and $j_1 = j_2 = j_3 = j_4 = 0$. Using the derivatives w.r.t. z_1 and z_2 of the $(k-1)^{th}$ moment, we can form a recursive relationship with terms used to obtain the k^{th} moment. Notice how one partial derivative of $T_i(s; z_1, z_2)$ in the RHS of equation (4.18) is being differentiated w.r.t. to z_j, for $j = 1, 2$. Hence, to obtain the k^{th} moment (M_i^k) for class i, we differentiate k times to construct a simplified recursive relation (i.e. letting $T_i(\cdot) = T_i(s; z_1, z_2)$ and omitting non-important terms) as follows:

$$M_i^k = -\sum_{j=1}^{2} \frac{\alpha_j}{\alpha_i} k \frac{\partial^{k-1}}{\partial s^{k-1}} \left[\frac{\partial}{\partial z_j} T_i(\cdot) \right] - k \frac{\partial^k}{\partial s^{k-1}} T_i(\cdot) \tag{4.20}$$

Note that, by Leibniz's rule, we can swap the positions of $\partial^{k-1}/\partial s^{k-1}$ and $\partial/\partial z_j$ on the RHS of equation (4.20). Hence, we obtain a recursive relation of derivatives, whereby the current moment can be calculated directly given the previous moment. Mathematica implements this algorithm efficiently and solves the recursive equations for any k^{th} moment symbolically. The full description of the moment-generating algorithm for M/M/1-DPS queues is given in algorithm 4.

Algorithm 4 Two-class moments for M/M/1-DPS queues

Input: Equilibrium queue length distribution on arrival.
Input the following queueing parameters:
α_1, α_2 = priority weights
λ_1, λ_2 = mean arrival rates
μ_1, μ_2 = mean service rates
k = the number of moments
Calculate all possible derivatives w.r.t to z_1 and z_2, as shown in equation (4.19), and store in a queue length array.
Differentiate the PDE in equation (4.18) k times to obtain the k^{th} response time moment recursively using the pre-calculated queue length array.
Using equation (4.20), Mathematica evaluates derivatives of the joint transform only at $s = 0$ and $z_1 = z_2 = 1$.
Output: The first k unconditional response time moments.

Using this algorithm, we display the third moment $\mathbb{E}[T_1^3]$ in equation (4.21) symbolically for job class 1 (with corresponding class 2 moment $\mathbb{E}[T_2^3]$ obtained by inverting subscripts 1 and 2). Note that we set $\alpha_1 = \alpha_2 = 0.5$ for presentation purposes, but the service weights α can have any distribution given all weights sum to one:

$$\begin{aligned} d_1\mathbb{E}[T_1^3] = {} & \mu_1^2\left(\rho_1\left(\rho_2-1\right)-2\left(\rho_2^2+4\rho_2+1\right)\right)-2\mu_1\mu_2\left(\rho_1-2\rho_2+2\right)\left(\rho_2+1\right) \\ & +\mu_2^2\left(\rho_1-2\rho_2+2\right)\left(\rho_2-1\right) \end{aligned} \tag{4.21}$$

where $12d_1 = \mu_1^3\left(\rho_1+\rho_2-1\right)^3\left(\mu_1(\rho_1-2)+\mu_2(\rho_1+2\rho_2-2)\right)^2$.

Note that using algorithm 4, third (and higher) moments may be expressed in terms of their general α weights, but we omit such results for presentation purposes.

4.3.5 Case study- M/M/1-DPS analytical and simulated moments

We calculate results from two workloads: first, we collect inter-arrival times of TCP traffic from experiments on Intel Core i7-2600 CPU @ 3.40GHz machines, which form the TCP data set; secondly, we use parameters from a GRID network application [43]. Both data sets exhibit multiple job classes with different access priorities and, hence, are suitable for our M/M/1-DPS queue. Class priorities are calculated using packet type and size. We approximate analytical response time moments using the numerical algorithm in Figure A.1. To obtain simulated moments, we execute 10000 runs of one million observations using the JMT tool and provide 95% confidence intervals.

TCP data sets

Data is collected from servers running data applications with TCP packet delivery. Using inter-arrival times from a TCP data set, we parametrise our M/M/1-DPS queue as follows: $\lambda_1 = 0.2$, $\lambda_2 = 0.3$, $\mu_1 = 1$, $\mu_2 = 2$, $\alpha_1 = \alpha_2 = 0.5$. Hence, we obtain analytical

and simulated response time moments (in seconds) for class 1 and class 2 moments in Table 4.3. Analytical moments are calculated from the numerical algorithm in Figure A.1, where all moments are centralised to reveal the spread from the mean. Further, we set appropriate priority weights for each job class, which is typical of TCP packets arriving with different-sized requests. For example, one class i may have higher share of the server (i.e. larger α_i value), but have lower mean service rate (μ_i). We collect mean arrival and service rates from a second TCP data set and parametrise our queue with $\lambda_1 = 0.2$, $\lambda_2 = 0.6$, $\mu_1 = 2$, $\mu_2 = 1.2$, $\alpha_1 = 0.33$ and $\alpha_2 = 0.67$. The corresponding multi-class response time moments are summarised in Table 4.4.

Table 4.3: Response time moments (sec) from TCP data set 1.

	Class 1	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**1.54**	**1.54 ± 9e-04**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**3.25**	**3.29 ± 2e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**16.64**	**16.86 ± 0.01**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**174.62**	**177.15 ± 0.10**
	Class 2	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**0.77**	**0.78 ± 4e-04**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**0.88**	**0.89 ± 5e-04**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**2.67**	**2.69 ± 2e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**16.71**	**16.95 ± 0.01**

Table 4.4: Response time moments (sec) from TCP data set 2.

	Class 1	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**1.68**	**1.69 ± 1e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**6.30**	**6.37 ± 4e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**64.88**	**65.73 ± 0.04**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**1286.8**	**1305.5 ± 0.74**
	Class 2	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**2.0**	**2.02 ± 1e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**7.27**	**7.35 ± 4e-03**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**71.40**	**72.34 ± 0.04**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**1394.6**	**1414.8 ± 0.80**

The applicability of modelling multiple arrivals and the flexibility of setting arbitrary weights allows the automated algorithm in Figure A.1 to improve existing work; previously, either all α_i weights were equal, as with Chis and Harrison's work [7], or different weights were used to approximate response time for up to two moments only, as with Kim and Kim's work [155].

GRID network

For the second experiment, we parametrise an M/M/1-DPS queue with values obtained directly from a GRID network application [43] as follows: $\lambda_1 = 22.1$, $\lambda_2 = 7.16$, $\mu_1 = 50, \mu_2 = 20$, $\alpha_1 = 0.25$ and $\alpha_2 = 0.75$. We obtain response time moments for two job classes in M/M/1-DPS queues presented in Table 4.5.

Table 4.5: Response time moments (sec) from GRID data set.

	Class 1	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**0.15**	**0.15 ± 8e-05**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**0.06**	**0.06 ± 3e-05**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**0.06**	**0.07 ± 4e-05**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**0.14**	**0.14 ± 8e-05**
	Class 2	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**0.18**	**0.18 ± 1e-04**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**0.07**	**0.07 ± 4e-05**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**0.07**	**0.07 ± 4e-05**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**0.14**	**0.14 ± 8e-05**

4.3.6 Conclusion and future work

We proposed an automated moment-generating algorithm that calculates response times analytically using single server M/M/1-DPS queues. This iterative algorithm uses a partial differential equation for recursively evaluating terms in a Laplace transform for multiple job classes. For the results, we analytically examined workloads from a smartphone and a VM and obtained mean and variance of response time. Further, we obtained analytical and simulated results (i.e. the first four moments of response time) on two case studies, namely TCP traffic and a GRID network application, each with two priority classes. Possible applications include resource allocation in data centres and prediction of peak delays for multi-class workloads. Indeed, response times have become essential components of SLAs and, thus, support the long-term performance goals of many systems.

It is possible to extend this research by collecting more data from smartphone applications. Specifically monitoring delay, approximated as response time, on smartphone applications running on different handsets might build useful user profiles. Another extension is to use the response time moments to approximate distributions of response time to meet SLA requirements (e.g. 99% of jobs are executed within 0.01 seconds). Further, we seek to improve modelling job arrivals by replacing the Poisson distribution with more bursty distributions such as the MMPP. In this way, we utilise the switching modes of the MMPP to represent the different characteristics of job types (or traffic flows).

4.4 MMPP/M/1-DPS queues

4.4.1 Motivation

In the context of Internet traffic, handling correlated streams of arrivals motivates the use of MMPPs, which can account for burstiness properties and state-switching [39, 66]. Therefore, approximating response times with M/M/1-DPS queues [114, 21] is insufficient for spatiotemporal Internet traffic behaviour given the simplified Poisson arrival assumption. MMPPs are useful in capturing the spatiotemporal behaviour in jobs arriving in smartphone applications and multi-tiered cloud platforms, where obtaining response times from such systems is beneficial for predicting delays and for resource planning. Essentially, we aim to combine MMPPs with discrete processor-sharing queues parametrised from empirical data to model delays at routers and in networks. In the literature, there are techniques of approximating response time moments in MMPP/M/1-FCFS queues from M/M/1-FCFS queues assuming slow phases of MMPP. Hence, we adapt Ciciani's FCFS weighted superposition method [39] to DPS queues and, therefore, approximate response time moments for MMPP/M/1-DPS queues. With weighted superposition, we use the existing moment-generating algorithm for M/M/1-DPS queues from the previous section. Further, it is possible to use the switching rates from the IncHMM to input into the MMPP arrival process, which we reserve for future work.

4.4.2 Weighted superposition for DPS queues

Obtaining higher response time moments through an MMPP/M/1-DPS queueing model is novel work, to our best knowledge. To achieve this queueing model, we first adapt the weighted superposition technique of Ciciani *et al.* to DPS queues, given an important assumption. In order to apply the technique of weighted superposition of Poisson processes to form an arrival process based on MMPPs, we assume (as an approximation) the *quasi-stationarity* property. This property states that, provided the queue dynamics are sufficiently faster than the state changes of the underlying MMPP, the queue reaches equilibrium immediately in each state of the MMPP. Therefore, with the arrival process in state i, the response times of the multiple job classes of the DPS queue can be estimated using the known response time moments of the M_i/M/1-DPS queue. Further, the unconditional response time of the MMPP/M/1-DPS queue can be estimated using a weighted average of the response time in each M/M/1-DPS queue, where each weight is the equilibrium probability of the MMPP being in each of its states.

Ciciani *et al.* use a notion of quasi-stationarity for FCFS queues, but do not obtain explicit response time moments for DPS queues. Hence, we adapt their weighted superposition methodology for MMPP/M/1-DPS queues, with N phases in the MMPP arrival process:

1. For each state i, ensure $\sum_{j=1}^{N} q_{ij} = 0$. Use rates q_{ij} to calculate p_i, which is the invariant probability that the MMPP remains in state i, and is calculated [2] using linear equations.

[2] for $N = 2$, we have $p_1 = q_{21}/(q_{12} + q_{21})$ and $p_2 = q_{12}/(q_{12} + q_{21})$.

2. For each class j job, calculate the weights w_{ij} for all states $i = 1, \ldots, N$ using the following equation:

$$w_{ij} = \frac{p_i \lambda_{ij}}{\sum_{k=1}^{N} p_k \lambda_{kj}} \tag{4.22}$$

3. For a class j job, calculate the k^{th} moment of response time ($\mathbb{E}[T_{ij}^k]$) for all M_i/M/1-DPS queues, $i = 1, \ldots, N$, using the algorithm from Figure A.1.

4. Approximate the k^{th} moment of response time for a class j job in an MMPP/M/1-DPS queue as:

$$\mathbb{E}[T_j^k] = \sum_{i=1}^{N} w_{ij} \mathbb{E}[T_{ij}^k] \tag{4.23}$$

Note, the assumption of slow MMPP phases is essential in applying the weighted superposition technique. However, we plan improvements to this approximation, namely modelling an MMPP with faster switching rates, as future work.

4.4.3 Response time density

After obtaining response time moments, we approximate a response time probability density function (PDF) when it is too complex to numerically invert the Laplace transform. One such distribution-fitting approximation is the generalised lambda distribution (GLD) [154, 157, 159], which inputs the first four moments of response time as parameters. The GLD is parametrised with mean (m_1), variance (σ^2), skewness (g) and kurtosis (κ) to approximate distribution ($F(t)$) and density ($f(t)$) functions. Let $Q(u) \equiv F^{-1}(u)$ (where $0 < u < 1$) be a quantile function for a four-parameter generalisation of Tukey's lambda family of distributions [158]:

$$Q(u) = \Lambda_1 + \frac{1}{\Lambda_2}\left(\frac{u^{\Lambda_3} - 1}{\Lambda_3} - \frac{(1-u)^{\Lambda_4} - 1}{\Lambda_4}\right) \tag{4.24}$$

where the k^{th} moment is finite if $min(\Lambda_3, \Lambda_4) > -\frac{1}{k}$ [159].

The four Λ parameters are defined as follows: Λ_1 is the location parameter, Λ_2 is the scale of data and Λ_3 and Λ_4 capture the shape of the empirical distribution. Note, $\Lambda_3 = \Lambda_4$ in a symmetric distribution. Therefore, the goal is to find parameters Λ_i, for $i = 1, 2, 3, 4$, such that the moments of the theoretical GLD match the empirical moments (i.e. m, σ^2, g and κ). Hence, we define equations, inspired from Lakhany *et al.* [159], as follows:

$$g = \frac{v_3 - 3v_1v_2 + 2v_1^3}{(v_2 - v_1^2)^{3/2}} \; ; \; \kappa = \frac{v_4 - 4v_1v_3 + 6v_1^2v_2 - 3v_1^4}{(v_2 - v_1^2)^2} \tag{4.25}$$

where quantities v_k are defined, for $k = 1, 2, 3, 4$, as

$$v_k = \sum_{j=0}^{k} \frac{(-1)^j}{\Lambda_3^{k-j}\Lambda_4^j} \binom{k}{j} \beta(\Lambda_3(k-j) + 1, \Lambda_4 j + 1) \tag{4.26}$$

where $\beta(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1} dt$ for Re(x),Re(y)> 0.

After deriving Λ_3 and Λ_4, we compute Λ_1 and Λ_2 using the following formulae:

$$\Lambda_2 = \frac{(v_2 - v_1^2)^{1/2}}{\sigma} \; ; \; \Lambda_1 = m + \frac{1}{\Lambda_2}\left(\frac{1}{\Lambda_3 + 1} - \frac{1}{\Lambda_4 + 1}\right) \tag{4.27}$$

Thus, we approximate the distribution $F(t)$ by letting $t = Q(u)$ and $F(t) = u$ (where $0 < u < 1$) [153]. Similarly, the response time probability density ($f(t)$) is obtained parametrically [131] using $t = Q(u)$:

$$f(Q(u)) = \frac{\Lambda_2}{u^{\Lambda_3 - 1} + (1 - u)^{\Lambda_4 - 1}} \tag{4.28}$$

Algorithm 5 Response times from MMPP/M/1-DPS queues

Input: $\{I_t\}$ is the trace of packet inter-arrival times.
Other terms are defined as follows:
N = number of MMPP states.
D = infinitesimal $N \times N$ generator matrix of MMPP.
K = number of job classes.
λ_{ij} = mean arrival rate for a class j job from state i in MMPP.
λ_j = mean arrival rate for a class j job.
μ_j = mean service rate for a class j job.
$\rho_j = \lambda_j/\mu_j$ is the utilisation for a class j job.
w_{ij} = superposition weights for MMPP state i for a class j job.
α_j = priority weights for a class j job under DPS.
$\mathbb{E}[T_{ij}^k]$ = the k^{th} response time moment for a class j job from state i in MMPP.
$\mathbb{E}[T_j^k]$ = the k^{th} response time moment for a class j job.
The methodology begins as follows:
Fit $\{I_t\}$ by matching moments and autocorrelation in KPC-toolbox to obtain an $N \times N$ MMPP(D_0,D_1) with fixed rates.
for $j = 1 : K$ **do**
  for $i = 1 : N$ **do**
    Calculate the weights w_{ij} using equation (4.22).
    Parametrise rates λ_{ij} and μ_j for M_i/M/1-DPS queue.
    Obtain moments $\mathbb{E}[T_{ij}^k]$ from algorithm in Figure A.1.
    Calculate $\mathbb{E}[T_j^k] \mathrel{+}= w_{ij}\mathbb{E}[T_{ij}^k]$.
  end for
  From moments $\mathbb{E}[T_j^k]$, use GLD equations (4.24) - (4.28) to approximate the response time PDF $P(T_j = x)$ and calculate the corresponding CDF $P(T_j < x)$.
end for
Output: For a class j job, output the response time PDF $P(T_j = x)$ and CDF $P(T_j < x)$.

The quality of fit is ascertained only through a goodness-of-fit test because closed-form solutions do not exist for the lambda parameters. Lakhany *et al.* found that the GLD moment-matching technique produced a lower Kolmogorov-Smirnov (KS) test statistic when compared to the least squares and starship methods [159]. With response time

moments obtained from equation (4.23), it is straightforward to parametrise the GLD and obtain approximative PDFs. We summarise this process in algorithm 5. In the next section, we discuss the data sets used to test our MMPP/M/1-DPS queueing model.

4.4.4 Data sets

The data sets consist of anonymised CAIDA passive data collected from backbone OC192 links. Every month since April 2008, one-hour traces are collected on the OC192 monitors between 13:00 UTC and 14:00 UTC, with statistical information including flow and protocol characteristics. Among this information, we note that the maximum load for the OC192 link is approximately 10 billion bits/sec. For more information, the reader may explore the CAIDA anonymised Internet traces [77]. We present a summary of statistics for one recent trace from Equinix-Chicago (dirB) in Table 4.6. The collection of data analysed consists of 25 traces from Equinix-Chicago (dirB) spanning the two most recent years 2014 and 2015. In total, this is the equivalent of analysing approximately of 50 billion IPv4 packets of Internet traffic.

Table 4.6: Statistics for CAIDA Equinix-Chicago data (dirB) collected on Oct 15$^{\text{th}}$ 2015.

Duration (hh:mm:ss)	01:02:01
Total IPv4 packets	1.5 bill.
Total IPv6 packets	30.2 mill.
Mean IPv4 packet size	856 bytes
Mean IPv6 packet size	970 bytes
Mean packets/sec	414.6k/sec
Mean flows/sec	10.2k/sec
Mean utilisation	0.286

For two arbitrary data sets, with similar statistics as shown in Table 4.6, we plot the packet inter-arrival times for an initial session in Figure 4.6.

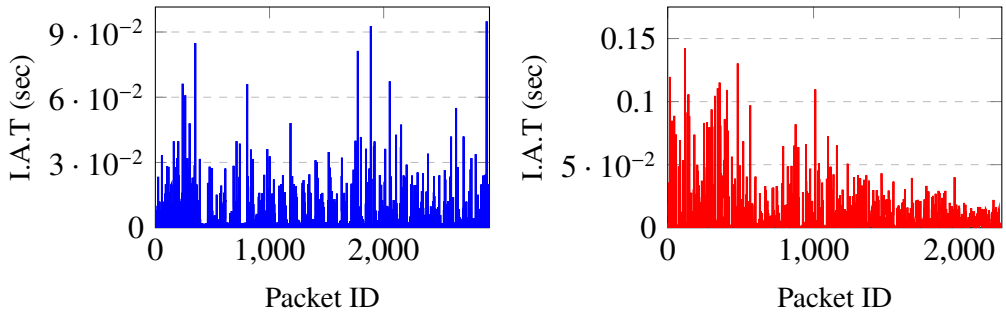

Figure 4.6: Inter-arrival times (I.A.T.) of packets from CAIDA data set 1 (left) and 2 (right).

4.4.5 Results

We approximate response time moments from MMPP/M/1-DPS queues by modelling traces of packets arriving at an anonymised router. We use KPC-toolbox to fit the traces of packet inter-arrival times and, hence, parametrise the MMPP arrival process. The steps are summarised in algorithm 5. Whilst the number of priority classes ($K = 2$) remains fixed, a number of parameters in the MMPP/M/1-DPS queueing system are varied for two CAIDA data sets in separate experiments. First, we vary the number of MMPP states (or phases) to take values two and four, therefore capturing the burstiness of packet inter-arrival times with MMPP(2) and MMPP(4) models, respectively. Secondly, the load (or utilisation) is parametrised with low ($\rho = 0.4$) and high ($\rho = 0.9$) values. The wide range of utilisation is chosen specifically to stress-test our DPS-queueing model and, hence, identify at what load level the weighted superposition technique of response time moments fails. Hohn *et al.* argue that high utilisation scenarios with significant delay are of most interest [116] and we investigate this through our high load experiments. The results are further validated by applying the two-sample Kolmogorov-Smirnov (KS) test, which compares two distributions and returns 0 (i.e. accept null hypothesis) if the sample vectors are from the same continuous distribution. Otherwise, it returns a value of 1 (i.e. reject null hypothesis).

We simulate response time moments using the Java Modelling Tools (JMT) software [37]. We input relevant workload and queueing parameters obtained from the CAIDA data sets into the simulation models. To obtain response time moments for a MMPP/M/1-DPS, the JMT software offers JSIMwiz, which is a user-friendly interface for performance analysis measurements given different scenarios involving routers and networks. We use JSIMwiz to simulate over 60 million MMPP-distributed packet arrivals at a router, which are sampled over one million times.

4.4.6 Moments for MMPP(2)/M/1-EPS queue

Before experimenting with the CAIDA data sets, we first parametrise a two-state MMPP (i.e. MMPP(2)) under EPS scheduling with values obtained directly from a GRID network application [43] as follows: $q_{12} = 0.17$, $q_{21} = 0.08$; $p_1 = 0.32$, $p_2 = 0.68$. Further, assuming only one job class ($K = 1$), the arrival rates are given by $\lambda_1 = 22.1$, $\lambda_2 = 7.16$ and the weights are $w_1 = 0.59$, $w_2 = 0.41$. Hence, analytical and simulated centralised response time moments (in seconds) are produced for MMPP(2)/M/1-EPS queues. By varying the mean service rates, we experiment with low load and high load to compare response time approximations.

Low load

For the low load scenario, we set the following parameters: $\mu = 80$, $\rho_1 = 0.28$ and $\rho_2 = 0.09$. This gives a combined load of $\rho = 0.37$. The analytical and simulated centralised moments of response time are presented in Table 4.7 with corresponding 95%

confidence intervals. The simulation results match the analytical approximations well for the first three moments and with reasonable accuracy for the fourth moment.

Table 4.7: MMPP(2)/M/1-EPS moments (sec) on low load GRID data ($\rho = 0.37$).

Moment	Analytical	Simulated
$\mathbb{E}[T]$	**0.016**	**0.016 ± 2e-06**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**3.24e-04**	**3.24e-04 ± 6e-08**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**1.65e-05**	**1.64e-05 ± 4e-09**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**1.77e-06**	**1.43e-06 ± 3e-10**

High load

For the high load scenario, we set the following service and load parameters: $\mu = 35$, $\rho_1 = 0.63$ and $\rho_2 = 0.2$. Such parameters result in a combined utilisation of $\rho = 0.83$, which we consider high load (under equilibrium) in the GRID network. In Table 4.8, we present analytical and simulated response time moments.

Table 4.8: MMPP(2)/M/1-EPS moments (sec) on high load GRID data ($\rho = 0.83$).

Moment	Analytical	Simulated
$\mathbb{E}[T]$	**0.0602**	**0.0596 ± 6e-06**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**0.0078**	**0.0074 ± 1e-06**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**0.0031**	**0.0028 ± 5e-07**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**0.002**	**0.002 ± 2e-07**

Using the analytical and simulated moments from Tables 4.7 and 4.8, we approximate probability density functions (PDFs) on the GRID data. GLD fits the distribution parameters from the given empirical moments and outputs representative PDFs for low and high load. In both cases, the GLD returns a beta prime distribution, or a beta distribution of the second kind. The beta prime distribution has Pearson distribution type VI and is one particular parametrisation of the F distribution. Hence, we present response time PDFs for the GRID data set in Figure 4.7 for low and high loads. The analytical GLD-fitted PDFs match the simulated PDFs well, with slightly higher errors for higher load (i.e. $\rho = 0.83$).

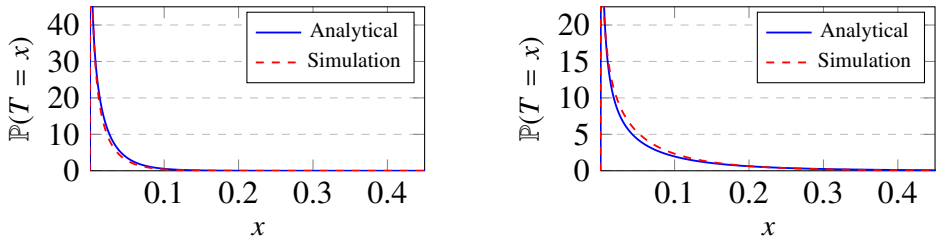

Figure 4.7: Response time PDFs via moments in Tables 4.7 (left) and 4.8 (right).

4.4.7 Moments for MMPP(2)/M/1-DPS queue

We return to DPS queues and experiment with the CAIDA data sets for two job classes ($K = 2$). After data-fitting with KPC-toolbox, an MMPP(2) is first parametrised for class 1 jobs. The MMPP(2) transition rates to other states are $q_{12}^{(1)} = 0.18$ and $q_{21}^{(1)} = 16.2$ for class 1 jobs. Hence, state probabilities are calculated as $p_1^{(1)} = 0.01$, $p_2^{(1)} = 0.99$. The MMPP(2) mean arrival rates are $\lambda_{11} = 83.3$ and $\lambda_{21} = 429.1$. Further, we obtain weights w_{i1} for class 1 jobs in state i as follows: $w_{11} = 0.001$ and $w_{21} = 0.999$. Similarly, for class 2 jobs, the MMPP(2) transition rates are $q_{12}^{(2)} = 0.44$ and $q_{21}^{(2)} = 45.7$, with corresponding state probabilities $p_1^{(2)} = 0.01$ and $p_2^{(2)} = 0.99$. The mean arrival rates for class 2 jobs are $\lambda_{12} = 1960.7$ and $\lambda_{22} = 11933.9$. Hence, we obtain weights $w_{12} = 0.002$ and $w_{22} = 0.998$. Essentially, we observe that for both job classes, the second state is heavily weighted, which influences the MMPP(2) arrival process into behaving (for the most part) as a Poisson arrival process with one mean rate. We set priority weights $\alpha_1 = 0.4$ and $\alpha_2 = 0.6$ for each class of data packet as a function of packet size and mean arrival rate. We parametrise an MMPP(2)/M/1-DPS queue and obtain analytical and simulated response time moments for both job classes under low and high loads. For both loads, we stress-test our queueing model and verify if the analytical and simulated results are from the same distribution with the KS test.

Table 4.9: MMPP(2)/M/1-DPS moments (sec) on low load ($\rho = 0.4$).

Low load ($\rho = 0.4$)		
	Class 1 jobs	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**5.92e-04**	**5.50e-04 ± 2e-06**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**4.33e-07**	**3.24e-07 ± 6e-09**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**7.36e-10**	**4.16e-10 ± 4e-13**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**2.65e-12**	**9.20e-13 ± 3e-16**
	Class 2 jobs	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**2.03e-05**	**2.10e-05 ± 4e-07**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**5.67e-10**	**5.12e-10 ± 3e-12**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**4.13e-14**	**3.19e-14 ± 3e-17**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**6.49e-18**	**3.92e-18 ± 3e-21**

Low load

We experiment with CAIDA data set 1 for low total load ($\rho = 0.4$) on our MMPP(2)/M/1-DPS queueing system. Since, we have two job classes ($K = 2$), it follows that $\rho_1 = \rho_2 = 0.2$, which gives us $\rho = \rho_1 + \rho_2 = 0.4$. Note, to achieve such specific load requirement, we modify the mean service rate for each job class. It follows that the mean service rate for job class 1 is $\mu_1 = 2562$ and for job class 2 it is $\mu_2 = 69473$. We present response time moments for classes 1 and 2 in Table 4.9. The GLD fits the distribution parameters from the given empirical moments and outputs representative PDFs for classes 1 and 2 under

low load. In both cases, the GLD returns a beta prime distribution. We performed KS tests on both job classes to test the accuracy of the analytical approximation compared to the simulated response time distribution. We plot the low load PDFs of response time in Figure 4.8. For class 1 jobs, the KS test returned 0 and we accept the null hypothesis that distributions match in this scenario. For class 2 jobs, the KS test also returned 0, which indicates that analytical and simulated distributions are in agreement.

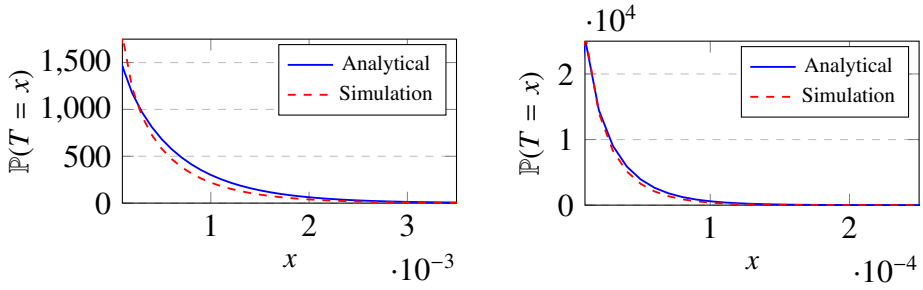

Figure 4.8: Response time PDFs via moments in Table 4.9 for class 1 (left) and 2 (right) jobs.

Table 4.10: MMPP(2)/M/1-DPS moments (sec) on high load ($\rho = 0.9$).

High load ($\rho = 0.9$)		
	Class 1 jobs	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**0.0037**	**0.0022 ± 3e-05**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**2.66e-05**	**8.28e-06 ± 2e-07**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**4.15e-07**	**1.36e-07 ± 1e-10**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**2.12e-08**	**5.26e-09 ± 5e-12**
	Class 2 jobs	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**1.09e-04**	**7.99e-05 ± 2e-06**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**2.51e-08**	**1.39e-08 ± 1e-11**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**1.69e-11**	**1.22e-11 ± 5e-15**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**2.32e-14**	**2.41e-14 ± 2e-18**

High load

We experiment with CAIDA data set 1 for high total load ($\rho = 0.9$). The individual load for the two job classes is $\rho_1 = 0.45$ and $\rho_2 = 0.45$. The mean service rates for job classes 1 and 2 are $\mu_1 = 1139$ and $\mu_2 = 30877$, respectively. The mean arrival rates (λ_{ij}), superposition weights (w_{ij}) and priority weights (α_j) for classes 1 and 2 are fixed with the same values as those used in the low load experiment previously. Hence, response time moments for classes 1 and 2 are presented in Table 4.10. For class 1 results, the analytical moments overestimate the simulated moments. The class 2 analytical moments are also overestimated for the first three moments, but match the simulated moments more closely

with excellent accuracy for the fourth moment. We plot the response time PDFs in Figure 4.9 for both job classes based on the moments presented in Table 4.10. For class 1 jobs, the KS test returns a value of 1, which means we reject the null hypothesis. Further, this signifies that the analytical approximation produced a different distribution than the JMT simulated results under high utilisation. For class 2 jobs, the KS test returns a value of 0 and, hence, we accept the null hypothesis.

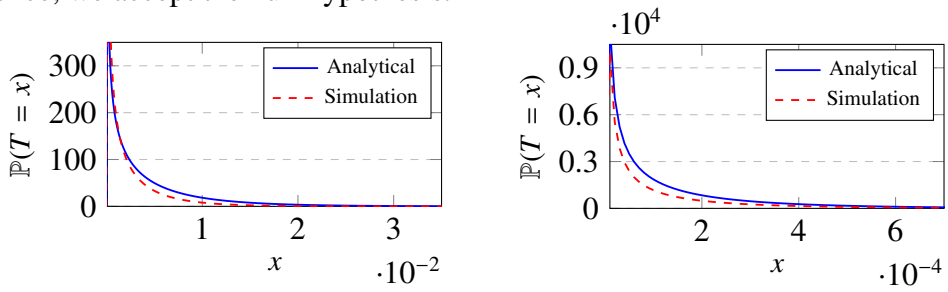

Figure 4.9: Response time PDFs via moments in Table 4.10 for class 1 (left) and 2 (right) jobs.

4.4.8 Moments for MMPP(4)/M/1-DPS queue

We obtain results for a four-state MMPP and parametrise our MMPP(4)/M/1-DPS queue with 8 different mean arrival rates. For clarity, we list these rates for each job class as follows: class 1 jobs have mean arrival rates $\lambda_{11} = 41103.6$, $\lambda_{21} = 857.5$, $\lambda_{31} = 611.5$, $\lambda_{41} = 3112.7$; class 2 jobs have mean arrivals rates $\lambda_{12} = 26.8$, $\lambda_{22} = 60.7$, $\lambda_{32} = 11.1$, $\lambda_{42} = 191.8$. We reserve calculations of the MMPP transition rates and probabilities from both job classes in appendix (A.3). Hence, we define 8 weights for the superposition approximation, namely w_{ij}, for $i = 1, \ldots, 4$ and $j = 1, 2$, using equation (4.22). Priority weights are set as $\alpha_1 = 0.7$ and $\alpha_2 = 0.3$ given the packet sizes. We compare the results of the MMPP(4) in modelling bursty traffic with the MMPP(2) via stress-testing under higher utilisation.

Table 4.11: MMPP(4)/M/1-DPS moments (sec) on low load ($\rho = 0.4$).

Low load ($\rho = 0.4$)		
	Class 1 jobs	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**5.34e-06**	**4.81e-06 ± 3e-08**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**3.42e-11**	**2.51e-11 ± 2e-14**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**5.08e-16**	**2.97e-16 ± 4e-19**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**1.61e-20**	**6.08e-21 ± 5e-24**
	Class 2 jobs	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**7.90e-04**	**7.78e-04 ± 2e-06**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**6.95e-07**	**6.20e-07 ± 3e-09**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**1.35e-09**	**1.03e-09 ± 6e-12**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**5.75e-12**	**2.70e-12 ± 7e-15**

Table 4.11 presents response time moments of MMPP(4)/M/1-DPS queues for both job classes experiencing low load ($\rho = 0.4$) and Table 4.12 presents results for high load ($\rho = 0.9$). From the moments in Tables 4.11 and 4.12, we plot the low load and high load density functions of response time in Figures 4.10 and 4.11, respectively. Similar to previous plots, the density functions in Figure 4.10 exhibit a beta-prime distribution for both classes. However, the KS test rejects the null hypothesis after comparing distributions from class 1 jobs. For class 2 jobs, the KS test returns 0, which accepts the null hypothesis that analytical and simulated plots belong to the same distribution.

Table 4.12: MMPP(4)/M/1-DPS moments (sec) on high load ($\rho = 0.9$).

High load ($\rho = 0.9$)		
	Class 1 jobs	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**1.67e-05**	**1.56e-05 ± 4e-07**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**4.25e-10**	**3.05e-10 ± 3e-13**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**2.81e-14**	**1.53e-14 ± 5e-17**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**3.77e-18**	**1.32e-18 ± 1e-23**
	Class 2 jobs	
Moment	Analytical	Simulated
$\mathbb{E}[T]$	**0.0022**	**0.0018 ± 3e-05**
$\mathbb{E}[(T-\mathbb{E}[T])^2]$	**6.49e-06**	**3.48e-06 ± 1e-09**
$\mathbb{E}[(T-\mathbb{E}[T])^3]$	**4.71e-08**	**1.56e-08 ± 2e-11**
$\mathbb{E}[(T-\mathbb{E}[T])^4]$	**7.17e-10**	**1.15e-10 ± 4e-13**

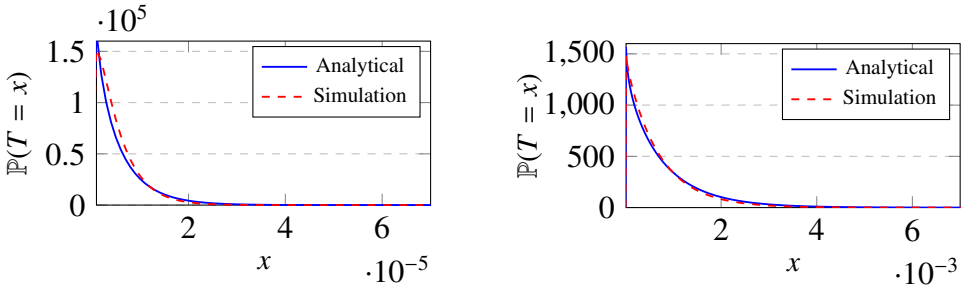

Figure 4.10: Response time PDFs under low load ($\rho = 0.4$) for class 1 (left) and 2 (right) jobs.

Examining the high load density functions in Figure 4.11, which we obtained from the moments in Table 4.12, it follows that, for class 1 jobs, the KS test accepts the null hypothesis. Unsurprisingly, the KS test rejects the null hypothesis for class 2 jobs, which indicates that analytical and simulated results are from different distributions. Hence, this stress-test, combined with the KS test, indicates that for high utilisation (i.e. ≥ 0.9), the MMPP(4)/M/1-DPS model fails to accurately match realistic packet delay distributions at router queues. The same can be said regarding moments for high load jobs from class 1 that were approximated using the MMPP(2)/M/1-DPS queueing model and presented

in Figure 4.9. Possible explanations linked to such results include: failure of the quasi-stationarity property since there is no equilibrium solution in any phase of the MMPP for high enough loads; or a poor initial choice of priority weights (i.e. $\alpha_1 = 0.7$ and $\alpha_2 = 0.3$) that are fixed for the two job classes. In practice, such weights are automatically assigned by the network manager to optimise flow in a network scenario. In the next chapter, we develop a dynamic allocation strategy using a UDP flow model and an MMPP/M/1-DPS queue to assign optimal priority weights to flows.

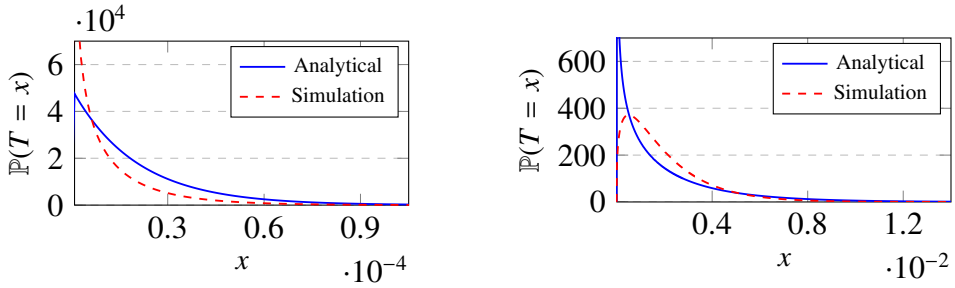

Figure 4.11: Response time PDFs under high load ($\rho = 0.9$) for class 1 (left) and 2 (right) jobs.

4.4.9 Conclusion and future work

We have provided a methodology for obtaining response time moments and PDFs using MMPP/M/1-DPS queues. First, an automated moment-generating algorithm was applied to yield higher moments for multiple job classes using M/M/1-DPS queues. Then, we applied a weighted superposition technique to estimate the response time moments using MMPP/M/1-DPS queues, under the approximating assumption of slow switching phases of the MMPP; the analytical approximations match the JMT simulation results fairly well. Applications include modelling multi-class, correlated Internet traffic, where delay metrics address SLA constraints and offer spatiotemporal resource allocation policies. For example, in the following chapter, we present a dynamic, service weight allocation strategy for flows are routers, whilst minimising the mean and variance of delay.

Possible extensions of this work include generalising job arrivals to full MAPs – ultimately, possibly, to any inter-arrival time distribution – and hence finding response time moments in MAP/M/1-DPS (or G/M/1-DPS) queues. Similarly, adding different service time distributions to our queueing system, such as hypo-exponential, hyper-exponential or phase-type, would increase the range of modelling scenarios, whilst maintaining mathematical tractability through the Markov property. Currently, such DPS queueing models assume steady state (i.e. $\lambda < \mu$) in the response time approximations and an extension is to handle overutilisation of the system when load is greater than one. Possible solutions include spectral analysis, which has been used for Markov-modulated queues [101], and matrix geometric methods [108]. Another possible extension is incorporating energy cost into our performance models, which would approximate modern SLA requirements more realistically. Indeed, battery models are popular in the literature [119, 120, 121, 123, 124, 147], but there is scarce analysis on power consumption related to performance via higher response time moments for multiple job classes. The

next chapter considers application of various adaptive models and discrete queues, which includes a performance-energy trade-off with mean delay and power consumption.

Chapter 5

Applications

Chapter Description

The applications we explore are based on adaptive models from previous chapters and include: a forecasting strategy for stocks and indices using IncHMMs (5.2); a performance-energy trade-off in smartphone applications and an HMM-based power consumption model that trains on recent data transfer activity (5.3); a dynamic allocation strategy for a UDP flow-level model to minimise delay whilst also maximising a utility function (5.4).

5.1 Introduction

In this chapter, we analyse new applications for our adaptive models. The first application uses the IncHMM used to forecast FTSE and NASDAQ stocks. We plot graphs of close-of-day prices generated from the IncHMM against the real stocks and indices. In the second application, we adapt the OnlineHMM to consider the performance-energy trade-offs of smartphone applications. Smartphone data is collected from over 700 users with 100 different handsets from the OpenBattery application [124]. Using this data to draw conclusions about usage patterns and battery consumption of different users, we investigate two strategies: use the OnlineHMM to forecast power consumption given user data activity; formulate a performance-energy trade-off using an objective cost function incorporating mean delay and power consumption. The third application involves applying the MMPP/M/1-DPS queueing model to form a dynamic allocation strategy for a UDP flow-level model (i.e. FlowDPS). This allocation strategy updates the weights of the DPS algorithm to minimise the delay mean and variance, whilst maximising a utility function. The performance of the FlowDPS model is compared to a simplified static DWRR scheduling algorithm.

5.2 Financial forecasting strategy

5.2.1 Motivation

Online forecasting time-series provides benefits for real-world applications such as intra-day algorithms and high frequency trading in finance [26]. For example, stock prediction

uses historical daily stock prices to adopt an effective investment strategy, which aims for diversification across asset classes and uses historical volatility to select low-risk assets. We have discussed in this thesis how HMMs provide temporal pattern recognition, parsimony and can be modified to act as online models for forecasting and run-time analysis. Online models are desirable in industry for potential analysis of live systems and resource planning, where main challenges include the dependency of model parameters on all preceding data. In terms of HMM dynamics, such online models require an approximation of the backward formula used in the Baum-Welch algorithm to save time computing the terms for the accumulated observation set. Hence, we use the IncHMM, along with its backward approximation, and build a live forecasting strategy for financial time-series. We summarise our contributions as follows:

- Collect and pre-process financial time-series, which are the close-of-day prices for stocks and indices from FTSE and NASDAQ stock markets.
- Using the IncHMM, build an online forecasting strategy that predicts future stock and indices movements given historical prices. Benefits include forecasting trends in stocks for risk prediction and using stock correlation for asset allocation.

5.2.2 Collecting time-series

Historical daily stock prices were collected from several financial websites (e.g. Yahoo finance). Time-series of close-of-day prices on 35 American and European stocks were updated by replacing the actual price with the price difference from the day before. Essentially, for a stock price s_t, we calculated the change in price from the day before and recorded the difference (i.e. $s_t - s_{t-1}$). Each time-series was clustered using a value of k = 2 (i.e. stock either moves up or down) in the k-means algorithm and then used as input into both the standard HMM and the IncHMM, which were each initialised with two hidden states. We reserve experiments with different values of k for the k-means clustering algorithm as future work.

It is important to note that non-stationary time-series are typically adjusted for seasonality using such models as X-12-ARIMA [38], developed by the US Census Bureau. Our real-world data sets are also seasonally adjusted through time-series decomposition in the following way: first, we checked if the seasonal component acts additively for the stock time-series; secondly, we calculated the variation of seasonality for each time-series (i.e. every three months for stocks); thirdly, we subtract this seasonal component from our original time-series. With the stock time-series now decomposed, we focus on trends and cyclic behaviour. After pre-processing the time-series, we validate the IncHMM by using the error between the model-generated time-series and the original data. Thus, via the sMAPE metric, we form a forecast strategy that involves iterative training on updated time-series using the IncHMM. We summarise this strategy in algorithm 6 for a generic forecast model M. Additionally, we provide 95% confidence intervals for $S = 10000$ simulation runs, as presented in the results section. There are many combinations of training windows and forecast horizons for the IncHMM. Experiments with heatmaps that output

average sMAPE values from forecasts revealed that a training period of 250 days was the best choice that produced the smallest errors.

Algorithm 6 Forecasting strategy

Input: $\{X\}$ = time-series; L = size of $\{X\}$; M = forecast model; T = training window; F = forecast horizon; S = simulation runs; $c = 0$; $T < t < (L - F)$.
  while $c < S$ **do**
    At time t, start with point X_t.
    while $t < L$ **do**
      Train model M on $\{X_{t-T+1},...,X_t\}$.
      Update model parameters for M.
      Forecast F new points $\{X'_{t+1},...,X'_{t+F}\}$.
      $t = t + F$.
    end while
    $c = c + 1$.
  end while
Output: $\{X'\}$ = forecast time-series.

5.2.3 FTSE and NASDAQ traces

From our experiments, we plot one-day forecasts for four stocks and two indices in Figures 5.1 and 5.2 , which reveal representative IncHMM-generated prices. For our forecasts, we chose a period from June 29th 2006 to September 1st 2006. However, the training period dates back to 2005 as it constitutes 250 days in the past. The IncHMM often forecasts sudden jumps in stock fairly accurately including the rise in BARC on day 15 to reach 600 and the small peak in FORD on day 31 just below 8. Similarly, such accurate forecasts are evident in the FTSE index on day 5 and day 31. Further, there are some directional changes in stock prices that the IncHMM predicts one day ahead of the actual event. For example, notice the sharp drop of FORD stock from 8 to 7 predicted by the IncHMM on day 55, which occurs on day 56. Whilst there exist some IncHMM-generated forecasts that are in the opposite direction to the actual stock movements, such movements will experience a mean reversion effect (i.e. eventually the stock prices will move towards an average in the market). Nonetheless, the incremental capacity of the IncHMM allows such forecast strategies to run alongside live systems without increasing computation costs or heavily reducing accuracy of model-generated results.

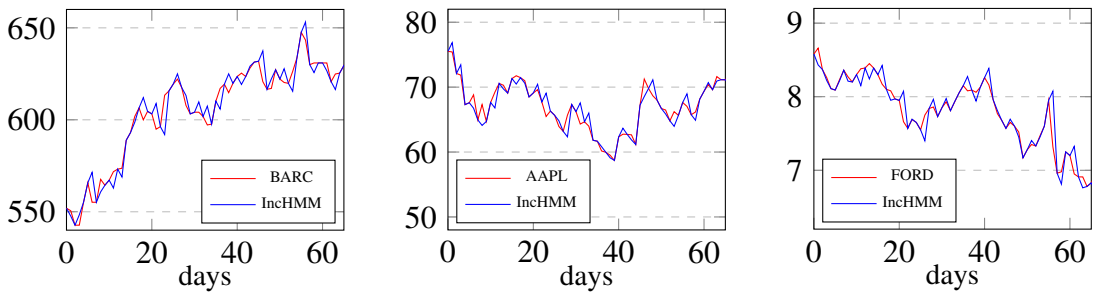

Figure 5.1: IncHMM forecasts of BARC, AAPL and FORD prices.

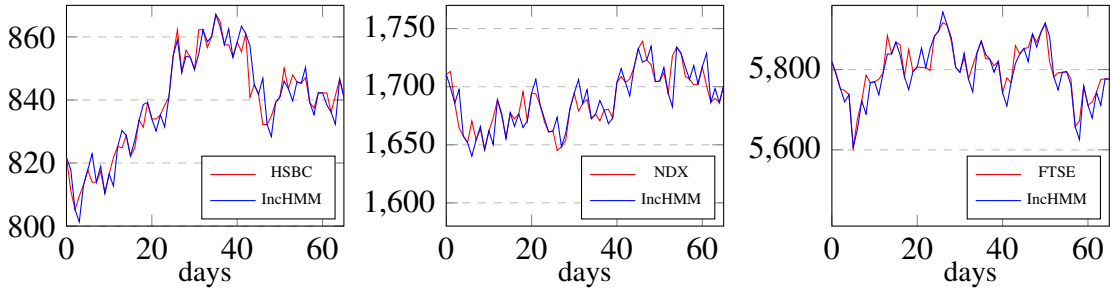

Figure 5.2: IncHMM forecasts of HSBC, NDX and FTSE prices.

5.2.4 Conclusion

In this work, we used the IncHMM to build a forecasting strategy to predict movements of stock and indices. We collected and pre-processed historical close-of-day prices for stocks and indices from FTSE and NASDAQ stock markets, then trained our IncHMM to predict future prices. The main advantage of using the IncHMM is its incremental capacity that allows forecasting to run alongside live systems without increasing computation costs. Further, the IncHMM produced pleasing results by replicating cyclic patterns and burstiness in price movements on a range of stocks.

Possible extensions include obtaining the implied volatility from our IncHMM strategy, which is the expected future variance of prices. We can currently calculate the historical volatility as the standard deviation of annualised log returns of prices, which is an estimate of the price "risk". However, option traders require an estimate of the implied volatility (or "view") to be able to price their options. Another extension is calculating the future stock correlation from the predicted prices, which is useful for investors in their asset allocation strategies to achieve more diverse portfolios.

5.3 Performance-energy modelling in smartphones

5.3.1 Motivation

In the last decade, smartphone technology has advanced faster than almost any other, with widespread applications in e-commerce [179], healthcare [176] and personal use [73, 129]. However, despite the development of handsets with faster multi-core CPUs, smaller physical components and high resolution displays, smartphone batteries are yet to improve at commensurate rates. Further, smartphone batteries are misused and applications are managed poorly, leading to elevated discharging rates and inefficient re-charging cycles. Additionally, Wi-Fi and background applications drain battery life despite providing obvious advantages to user needs. Therefore, a greater aim is to understand the performance-energy trade-off: maintaining high performance standards for smartphones without severely limiting battery life.

Some argue that performance is driving the smartphone industry. For example, we men-

tioned at the beginning of this thesis that smartphone users wait, on average, just over nine seconds for a web page to load [144]. Whether it's using smartphones to download files, stream web content or run background apps, the same delay principle applies. From a queueing perspective, delay and response time are closely related, but measuring delay is computationally expensive in real systems. We have seen the appealing properties of DPS scheduling, coupled with the abstraction properties of queueing systems to obtain response times. Real traces of smartphone activity and battery data may be used to feed a power consumption model and so characterise representative battery behaviour. Further, queueing models provide a more cost-effective alternative to replicating real smartphone servers and/or running applications. One of our strategies is to employ an adaptive workload model to forecast battery levels given current charging state and data activity.

To address the aforementioned challenges, we propose two strategies to investigate performance and energy in smartphone applications:

- Use the OnlineHMM to forecast power consumption, given recent data activity history of user and current charging state of device.

- Pose a performance-energy trade-off via an objective cost function incorporating mean delay and energy consumption.

From these strategies, we form two hypotheses: burstiness and mode-switching is observed in data activity and in power consumption rate of batteries; and power consumption increases as mean delay decreases. The rest of this section is organised as follows: we summarise the data collection process, describe each strategy in separate subsections and conclude with possible extensions of each strategy.

5.3.2 Data set

This section provides an overview of the data set used in this work and some data processing before building the two strategies. We use the original, unanonymised smartphone data from OpenBattery [124], which is a data collection application chronologically logging the following information: smartphone model, manufacturer and timestamped battery data ranging from February 2012 to September 2014. More specifically, each row of battery data stored in the OpenBattery database consists of: a timestamp (UTC), charge level (%), temperature, health status, plugged status, charging status, current capacity and voltage. Additionally, we collect timestamped packet data transferred for specific handsets from experiments to complement the battery data. An overview of data statistics is shown in Table 5.1, where there is collectively over 1.8 million aggregate hours of battery data.

Table 5.1: Statistics of battery data traces analysed.

Total smartphones	766
Total applications	4,857
Total charge cycles	> 58.2k
Total aggregate hours	> 1.8m
Median trace duration	367.5

Table 5.2 presents a sample of 12 different handset models (from a total of 766) and the corresponding manufacturer, as collected by OpenBattery. Among this sample, we also record the first data log for each model. It is important to collect data from a range of handset models to avoid overfitting the training parameters. Further, the more diverse data we obtain in testing the models, the more impact our claims have.

Table 5.2: Sample of handset models with manufacturer and date of first log.

Model	Manufacturer	Date
Blade	ZTE	2012-02-23
LS670	LGE	2012-03-18
LT15i	Sony Ericsson	2012-03-24
Huawei-M920	Huawei	2012-08-16
Novo10 Hero	MID	2013-04-24
Nebula 6.9	ZIGO	2013-08-07
Q800	XOLO	2013-10-19
Chaser	TeleEpoch	2013-12-28
Xperia Z C6603	Sony	2014-04-12
i867	Motorola	2014-07-04
HTC One M8	HTC	2014-07-23
GT-I8190N	Samsung	2014-09-12

Using this data set, we investigated the charging and discharging habits of different users. More specifically, we plotted time-series of power consumption for 100 users (i.e. a representative sample of the data set) to find any distributions or patterns among charging behaviour. For example, Figure 5.3 presents the distribution of session durations with charging and discharging for 100 users from March 2014 to July 2014; both distributions exhibit an exponential-like distribution, where about 90% of users never charge smartphones for over ten hours.

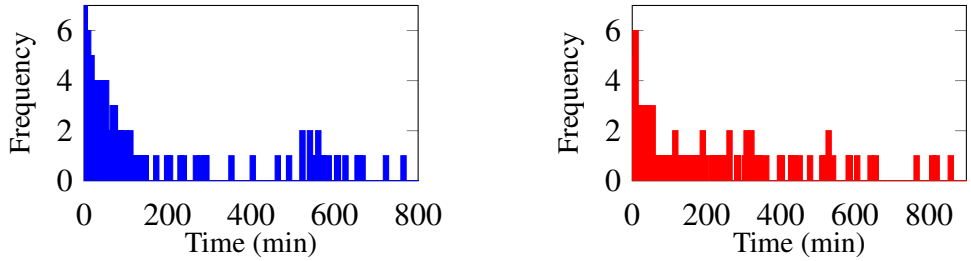

Figure 5.3: Distribution of charging (left) and discharging (right) sessions for 100 users.

Figure 5.4 reveals patterns in 3G data transfers for a HTC One user: on the left, the user is browsing websites using Chrome, which results in small packet sizes being transmitted; on the right, the user has two sessions of streaming YouTube videos resulting in two large bundles of packets (of almost 50KB) being transferred. The bursty nature of such transfers requires parsimonious models with switching modes and capabilities to generate correlated streams, which makes the HMM (and its adaptive variations) suitable for fitting on such data.

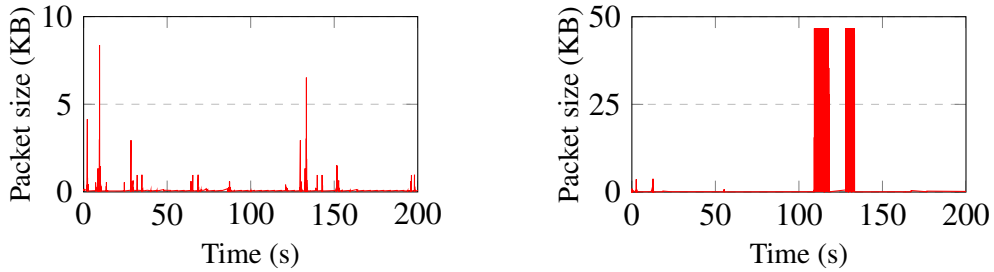

Figure 5.4: Data transfers for an HTC One user by browsing (left) and streaming (right).

Such examples of data transfers on smartphones invoke the inevitable question: how can we reduce unnecessary data transfers and prolong battery life in the process? We investigate this question through two strategies.

5.3.3 Strategy 1: Power consumption model

One of our aims is to obtain a power consumption model for smartphones. First, we consider the charge behaviour of two users given specific handsets. Figure 5.5 summarises the session durations and the corresponding increase in charge level; considering charging sessions under 100 minutes, user 329 has a positive (reasonably linear) correlation of duration vs charge, whereas longer sessions offer a more uniform distribution of charge. Thus, charging (and discharging) activity are key to understanding the diurnal power consumption of smartphones.

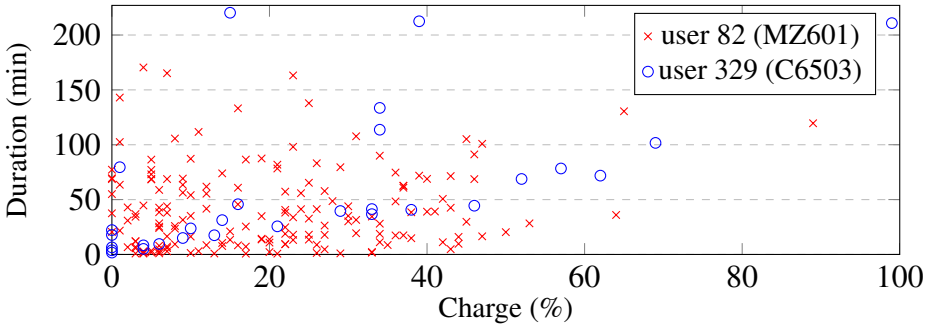

Figure 5.5: Charge increase with durations for two users.

Strategy 1 focuses on **forecasting power consumption to reveal trends in battery discharge for smartphone users**. To incorporate bursty and cyclic data transfers into the power consumption model, we employ an OnlineHMM. Notably, burstiness and mode-switching are two well-known phenomena occurring in data transmission and Internet traffic, which gives the OnlineHMM advantages over simplistic models such as the Poisson process. The hidden states of the OnlineHMM represent the sizes of data transfers by the applications of the smartphone. For example, in a two-state OnlineHMM, if the smartphone cellular radio is frequently in full-power mode (state 1), data transfers will often be larger than if the cellular radio were in low-power mode (state 2). The benefit of forecasting with HMMs is knowing the most recent hidden state, which provides a likelihood of the type of forecast likely to occur given the state. Further, the Viterbi algorithm can produce the sequence of hidden states to match the corresponding observations from the data trace. Multiple traces of timestamped data transfers are clustered using the k-means algorithm and the OnlineHMM trains on the clustered traces simultaneously in an incremental fashion (i.e. dynamically updating as new data becomes available) to converge its parameters as described in algorithm 3. Given the user's data transfer history, the charging status of the smartphone and the current hidden state of the OnlineHMM, the model outputs the most likely change in power (approximated using real rates obtained from profiling with battery manufacturers' parameters) and updates the smartphone battery level accordingly. In this way, the OnlineHMM can predict battery life for 24 hours and aid in resource planning and energy-saving strategies. Using sMAPE as validation, we executed 10000 simulations with 95% confidence intervals and compared results of the two-state OnlineHMM against two models: the first is a simple regression model similar to [73], which fits a nonlinear curve of expected power consumption; the second is a moving average of recent data consumption that averages recent power consumption to forecast future consumption.

Table 5.3: sMAPE for several smartphone users comparing three predictive battery models.

Model	HTC One	GT-I9300	Nexus 7
OnlineHMM	**0.32±2e-4**	**0.28±1e-4**	**0.35±4e-4**
Moving Avg.	**0.54±8e-9**	**0.24±4e-9**	**0.44±1e-8**
Regression	**0.40±1e-8**	**0.33±6e-9**	**0.44±7e-9**

Table 5.3 shows the OnlineHMM as the most consistent forecast model. The moving average model performs well on steady charging or discharging rates as observed for the GT-I9300 trace, but poorly on traces with intermittent behaviour (i.e. HTC One and Nexus 7 traces). The regression model is more consistent than the moving average, but produces higher errors by over 20%, on average, than our OnlineHMM predictive model. These results highlight that HMM forecast models are useful for capturing intermittent battery use for smartphones and, hence, can plan optimal charging times and durations. The OnlineHMM power consumption model we have investigated for three smartphone users proves that **burstiness and mode-switching exists in both smartphone data activity and battery behaviour**. The next strategy formulates a theoretical performance-energy trade-off of smartphone applications, involving mean delay and energy.

5.3.4 Strategy 2: Performance-energy trade-off

Strategy 2 investigates **the effect of mean delay of smartphone applications on power consumption rates**. To address this, we formulate an objective cost function, C, as part of a performance-energy trade-off. Recall that μ is the service rate, $\rho < 1$ is the utilisation and the arrival rate $\lambda = \mu\rho$ holds at equilibrium for positive terms. Let r_{all} be the total power consumption rate for general OS processes, background applications and active power cellular radio modes. Let r_{rad} be the power consumption rate for active radio only excluding other applications or processes. The rates r_{all} and r_{rad} are based on profiling devices from the OpenBattery data set and using Li-ion manufacturers' parameters [180, 181]. Such parameters are extracted from typical Li-ion discharge curves (i.e. voltage vs capacity) and include: voltage (~3.9V) and capacity (~1.1Ah) when the exponential zone ends; voltage (~3.6V) and capacity (~5.1Ah) when the nominal zone ends; capacity (~5.7Ah) at cut-off voltage (~2.5V); internal resistance is set to ~0.05Ohm. It is important to incorporate such realistic rates into power consumption given varying levels of data transfers and cellular radio modes.

The performance term of the cost function is the mean response time under EPS (for one job class): $\mu^{-1}(1-\rho)^{-1}$. The energy term is more complex: let B be the busy period, which represents the duration that the cellular radio is in active power mode continuously; let t_1 be the time needed to switch the radio to a low power mode; let I be the idle time spent with the radio in low power mode; let t_2 be the time needed to switch the radio to active power again; and define $t = t_1 + t_2$ as the total time spent switching between these states. The system operates as a sequence of $B, I, B, I, \ldots$ delay cycles and averaging over B and I yields:

$$\frac{r_{all}E[B] + r_{rad}E[I]P(I \leq t)}{E[B+I]} \tag{5.1}$$

where we use $P(I \leq t) = 1 - e^{-t\mu\rho}$, $\rho = E[B]/E[B+I]$ and $(1-\rho) = E[I]/E[B+I]$ to obtain:

$$r_{all}\rho + r_{rad}(1-\rho)(1-e^{-t\mu\rho}) \tag{5.2}$$

Let $0 < \alpha < 1$ be a scaling parameter for performance and energy terms. Having introduced all terms, we define the cost function C as follows:

$$C = \frac{\alpha}{\mu(1-\rho)} + (1-\alpha)(r_{all}\rho + r_{rad}(1-\rho)(1-e^{-t\mu\rho})) \tag{5.3}$$

Observe that when cellular radio is in low power mode, we eliminate the term $r_{rad}(1-\rho)(1-e^{-t\mu\rho})$ and, therefore, save energy in the process. In Figure 5.6, we plot μ against C for increasing values of t and fix $\rho = 0.5$, $\alpha = 0.05$, $r_{all} = 20$, $r_{rad} = 7$, where battery profiling and manufacturers' information was used as guidance. Given the three values of t, we observe in Figure 5.6 that C has a global minimum and is essentially a minimisation problem with respect to μ (i.e. service rate). Further, μ is the most important parameter since it is used to calculate mean delay and energy consumption from cellular radio. Note that changing the parameter values used in the cost function would alter the slopes of the minimisation curves for different values of t.

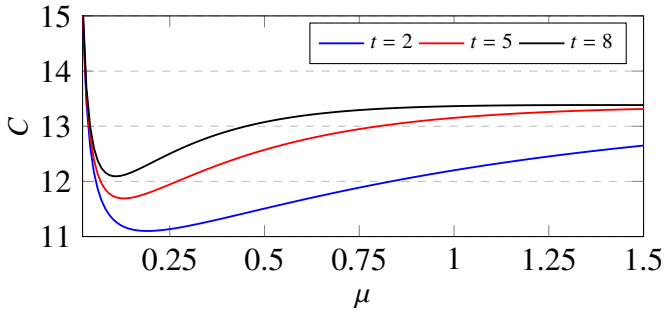

Figure 5.6: Cost function for varying μ.

Exploring both performance and energy constraints of the cost function, we hypothesise that **power consumption increases as mean delay decreases**. In other words, there is a trade-off between saving energy and speeding up performance of applications. This is supported by our cost function in equation (5.3) as increasing μ (i.e. the service rate) decreases the mean response time (i.e. the term $\mu^{-1}(1-\rho)^{-1}$), but increases the power consumption required to, for example, serve more jobs from smartphone applications. Smartphone users typically observe small changes in delay of applications only when downloading files or waiting for a webpage to load. The cost function we formulated, although theoretical, is useful when leveraging delays of non-urgent applications to reduce battery draw in bursts rather than over extended time periods.

5.3.5 Conclusion and future work

In this work, we have investigated two strategies: first, to forecast power consumption using OnlineHMMs; and secondly, to build a performance-energy trade-off structured as an objective function incorporating mean delay and power consumption. As a result, we have formulated two hypotheses. First, adaptive HMMs are useful models to represent burstiness and mode-switching for data activity and battery behaviour compared to other models such as moving averages and regression-based models. Secondly, power consumption in smartphones increases as mean delay of its applications decreases according

to our performance-energy trade-off. Important observations in data usage highlight the need to consider both performance (i.e. representing delay as response time) and energy (i.e. power consumption) for a number of applications.

Extensions include improvements to each of the strategies. We extend strategy 1 by executing more forecasts on different handsets and focusing on specific applications to understand the effect on the battery life. Strategy 2 can be extended by incorporating higher response time moments into the objective cost function C and modelling more job classes with DPS scheduling to replace the simplified EPS assumption. Further experiments are planned using the useful OpenBattery data including: test the cost function from strategy 2 on real smartphone data for specific applications; investigate the effect of two cellular radio modes (i.e. rare-big and often-little) on the data saved (i.e. from 3G or 4G transfers) and the battery life of varied smartphone users.

5.4 Traffic flow model

5.4.1 Motivation

A traffic flow (or packet flow) is a sequence of packets sent from a particular source to a particular destination, as defined by the RFC 2722 [41]. Typically, a flow consists of packets in a specific transport connection, but is not a one-to-one map to that respective connection. Alternatively, a flow can represent a media stream with a finite time interval. A flow is uniquely identified within a time period by source and destination IP addresses and ports, as well as the layer 4 protocol (i.e. UDP, TCP, etc.). In UDP, all packets with the same source address/port and destination address/port within a time interval belong to one flow. A TCP connection begins with a three-way handshake and creates two flows, which ends in a four-way handshake or a time-out. Since UDP is uni-directional, it is simpler to model its traffic flows than handling the handshaking of TCP or ICMP, for example. Further, it is important to model dynamic flows (i.e. existing flows close as new flows appear) in real-world Internet traffic scenarios and, ideally, to represent such dynamic flows in a parsimonious and cost-effective manner.

By using our DPS queueing models, we aim to model traffic flows dynamically in a resourceful way that allows comparisons (via simulation) with existing flow scheduling policies such as deficit weighted round robin (DWRR). Unlike the FCFS assumption at the system input and a fluid queue at the output used by Hohn *et al.* [116], the discrete queues in this chapter measure delay (i.e. the delay in the output buffer of the router) under DPS scheduling. This way, packets of varying sizes from different interfaces are considered in the delay approximation. Note that we model traffic flows at a single point (i.e. a router), but a greater aim is to extend our performance analysis to networks [125]. In the next section, we define a flow-level model specifically for UDP packets.

5.4.2 UDP flow-level model

We describe the parameters of the UDP flow-level model (FlowDPS) covering the main conditions given in [46]. Let $L = \{1, \ldots, \mathrm{L}\}$ be a set of links. Each link $l \in L$ has capacity $C_l > 0$. Let a_k be the per-flow rate limit (i.e. access rate is limiting). A class k is characterised by route L'_k, which is a specific subset of links. Let $x = (x_1, \ldots, x_k)$, where x_i is the number of flows of class i in progress. Let ϕ_k be the total capacity allocated to flows of class k; then the allocation satisfies the following conditions:

$$\sum_{k:l \in L'_k} \phi_k(x) \leq C_l\,,\ l = 1, \ldots, \mathrm{L} \ \text{ and } \ \phi_k(x) \leq x_k a_k\,,\ k = 1, \ldots, K \tag{5.4}$$

The allocation is Pareto efficient if there exists a saturated link l on L'_k. In other words, $\exists\, l \in L'_k$ s.t. $\phi_k(x) = x_k a_k$. We set traffic conditions for utilisation (ρ_k) as follows:

$$\sum_{k:l \in L'_k} \rho_k < C_l\,,\ l \in L \tag{5.5}$$

The metrics for the class k arrival rate (λ_k), service rate (μ_k) and system utilisation (ρ_k) in a DPS queueing model are equivalent to the flow-level model as follows:

$$\lambda_k = \mu_k \rho_k = \frac{1}{\mathbb{E}[x_k]} \rho_k \tag{5.6}$$

Given the conditions of the flow-level model, we make an important claim involving DWRR and DPS scheduling algorithms.

Claim Suppose there is a system with N classes (or flows) of packets scheduled by a DWRR algorithm and let the rate of the server be one for simplicity. Then, the rate for serving a class j job is proportional to the DPS rate.

Justification Let R be the rate of the server, R_j be the rate of serving a class j job in the DWRR scheduling algorithm, Q_j be the quantum for class j, and n_j be the number of class j jobs in the system. Setting $R = 1$, $N = K$ and $Q_j = \alpha_j n_j$, it follows that:

$$R_j = \frac{Q_j}{\sum_{i=1}^{N} Q_i} R = \frac{\alpha_j n_j}{\sum_{i=1}^{K} \alpha_i n_i} = r_j(n_1, \ldots, n_K) n_j \tag{5.7}$$

where $r_j(n_1, \ldots, n_K)$ is the DPS rate from equation (2.43). This claim also follows from the Kuhn-Tucker theorem, which states that the capacity allocated to flows under the same constraining links is shared in proportion to their weights [46]. Having proved similar properties between DWRR and our DPS flow model (FlowDPS), we turn our attention to dynamic flows.

5.4.3 Dynamic flows

In existing work, network utility is modelled with a fixed number of permanent flows [172]. In reality, flows do not last indefinitely. Each flow corresponds to the transfer of a

finite volume of data (or flow size) and ceases when the transfers complete. The evolution of the number of flows in progress clearly depends on the way new flows are generated, their sizes, and how bandwidth is shared between competing flows. In particular, the fact that an allocation is optimal in the sense of some utility function in a static scenario does not necessarily imply that this allocation is optimal in a dynamic scenario. The allocation may well lead to a steady state where overall utility is low. For example, maximising mean flow rate may lead to unstable traffic conditions and allow the number of flows to increase indefinitely. Hence, bandwidth-sharing objectives cannot reasonably be defined without taking flow-level dynamics into account. When building our dynamic allocation strategy, we change priority weights in the FlowDPS model to optimise a flow set that updates periodically. Further, in order to adhere to QoS demands, we model Internet traffic at flow level and also at packet level to obtain delay metrics. The MMPP/M/1-DPS queue allows us to obtain higher response time moments to approximate delay distributions at routers given bursty packet arrivals with multiple classes. The dynamic allocation strategy is introduced in the following section.

5.4.4 Dynamic allocation strategy

To utilise the service differentiation of multi-class jobs offered by DPS scheduling and the response times obtained from our MMPP/M/1-DPS model, we build a dynamic allocation strategy for modelling UDP flows. Using expected throughput from the most recent set of UDP flows, we measure delay given this throughput and aim to maximise a log utility function by optimally allocating DPS priority weights. We experiment with different loads to achieve the best QoS targets and minimise mean and variance of delay.

Algorithm 7 Dynamic Allocation Strategy for UDP flows

Input: $F = \{f_1, \ldots, f_K\}$ is the set of active UDP flows.
  while F not empty **do**
    for $k = 1 : K$ **do**
      Calculate expected throughput λ_k for flow f_k.
      Using the MMPP/M/1-DPS queue, calculate response time density function $P(T_k = x)$ for flow f_k using algorithm 5.
      Solve maximisation problem in equation (5.8) with given α_k weights.
      Check if mean and variance of delay meet acceptable QoS levels:
      while Mean and variance of delay are not minimised **do**
        Obtain mean and variance of response time from MMPP/M/1-DPS queue to represent delay.
        Update allocation weights α_k in equation (5.8) to re-calculate mean and variance whilst maximising U.
      end while
    end for
    Update active flow set F.
  end while
Output: Collection of mean and variance of delay for all flows in F.

The allocation provided by a system with DPS scheduling can be represented by the allocation that maximises the aggregate utility of the different classes for a given number of jobs. Hence, it is important to achieve some weighted proportional fairness from the allocation associated with a log utility function U, which is increasing and strictly concave. In a single server system with K traffic classes (or flows), we assume that there are n_j class j jobs, for $j = 1, \ldots, K$. As suggested in [46], the logical formulation of a minimisation problem is given by:

$$\max_{r_j, j=1,\ldots,K} \left\{ \sum_{j=1}^{K} \alpha_j n_j U\left(\frac{r_j}{\alpha_j n_j}\right) \right\} \text{ s.t. } \sum_{j=1}^{K} n_j r_j < 1 \tag{5.8}$$

It can be shown that the rates that solve the utility function in equation (5.8) are given by DPS rates r_j from equation (2.43) [60]. Generally, discriminatory allocations help realise service differentiation allowing users with higher rates to obtain better quality of service. Hence, DPS introduces an implicit fairness into flow allocations in networks. We summarise our dynamic allocation strategy using the FlowDPS model and MMPP/M/1-DPS queue in algorithm 7. The following section presents results for the FlowDPS model.

5.4.5 Results

We simulate our FlowDPS model as part of a dynamic allocation strategy for UDP flows using JSIMwiz and measure delay under two load scenarios. We experiment with our dynamic allocation strategy, as summarised in algorithm 7, using statistics from 20 CAIDA data sets as input into JSIMwiz simulations. We divide the data sets into two scenarios: low and high load UDP flows. From the CAIDA data sets, we select ten monthly traces with a low mean load of $\rho = 0.41$ and ten monthly traces with a very high mean load of $\rho = 0.88$. We adapt the dynamic allocation strategy in algorithm 7 for two scheduling algorithms: first, the DWRR algorithm with fixed priority weights (i.e. once the first flow begins, the α weights do not change with the simulation); secondly, the FlowDPS scheduling algorithm with dynamic priority weights that update as flows evolve in the simulation. Hence, we present mean (i.e. $\mathbb{E}[T]$) and variance (i.e. $\mathbb{E}[(T - \mathbb{E}[T])^2]$) of delay under both scheduling algorithms in Table 5.4. Based on the results in Table 5.4, we make the following observations and offer relevant recommendations given appropriate scenarios:

- For low loads, FlowDPS does not notably decrease the mean delay over DWRR, but it significantly reduces the variance of delay (as seen for class 1 jobs).
- For high loads, FlowDPS prioritises one class (i.e. class 2) and optimises weights to reduce mean and variance of delay when compared to DWRR.
- DPS allocation is suitable for service differentiation in UDP flows if it is dynamic (i.e. priority weights are updated regularly with new flows).
- Dynamic allocation reduces delay for high priority UDP packets with small sizes (i.e. voice data).

Table 5.4: Dynamic allocation results for DWRR (fixed) and FlowDPS (dynamic).

Low load scenario (ρ = 0.41)		
Model	Class 1 jobs	
	$\mathbb{E}[T]$	$\mathbb{E}[(T-\mathbb{E}[T])^2]$
DWRR	**3.0e-04 ± 2e-06**	**2.6e-07 ± 4e-09**
FlowDPS	**2.6e-04 ± 5e-06**	**1.5e-10 ± 2e-13**
	Class 2 jobs	
	$\mathbb{E}[T]$	$\mathbb{E}[(T-\mathbb{E}[T])^2]$
DWRR	**5.6e-03 ± 9e-05**	**7.2e-05 ± 3e-07**
FlowDPS	**4.2e-03 ± 1e-05**	**3.9e-05 ± 5e-07**

High load scenario (ρ = 0.88)		
Model	Class 1 jobs	
	$\mathbb{E}[T]$	$\mathbb{E}[(T-\mathbb{E}[T])^2]$
DWRR	**0.075 ± 3e-04**	**0.014 ± 8e-04**
FlowDPS	**0.076 ± 4e-04**	**0.02 ± 6e-04**
	Class 2 jobs	
	$\mathbb{E}[T]$	$\mathbb{E}[(T-\mathbb{E}[T])^2]$
DWRR	**2.5e-03 ± 3e-05**	**2.1e-05 ± 7e-07**
FlowDPS	**1.3e-03 ± 3e-05**	**2.8e-06 ± 5e-08**

5.4.6 Conclusion

Queueing management is a key requirement of QoS to reduce delays in networks. We have utilised our response time approximations to represent packet delay at routers and as part of a flow-level allocation strategy. The dynamic allocation strategy was simulated at low and high loads with mean and variance of delay collected. The results indicate that FlowDPS reduces delay mean and variance using its dynamic allocation of priority weights compared to a simplified DWRR scheduling algorithm with static weights for all flows. Further, we have proved that DPS scheduling algorithms, coupled with MMPP bursty properties, are efficient for service differentiation at flow level.

Extensions to the FlowDPS model include adding more representative service time distributions to represent packets with varying requests at routers. This can be achieved with an MMPP/PH/1-DPS queueing model, where PH represents the phase-type service time distribution, and the model should support multiple phases. Further, it is interesting to investigate heavy-tailed service time distributions and, specifically, understand the correlation of response time tails to the length of busy periods. Alternatively, it is possible to replace discrete DPS queues with fluid queues to utilise the busy period and measure delay of output buffers found in routers. It would be beneficial to compare the FlowDPS model to a wider range of scheduling algorithms, including shortest-job-first (SJF) and last-come first-served (LCFS), to investigate how delay varies under allocation weights and higher loads with such scheduling disciplines.

Chapter 6

Conclusion

Chapter Description

Section 6.1 summarises the achievements and contributions of this thesis with respect to adaptive HMMs and processor-sharing queueing models. In section 6.2, extensions include improving accuracy of model-generated results and combining adaptive HMMs with DPS queues to form new dynamic strategies to save resources. In section 6.3, we evaluate our research in terms of model components and discuss the impact of our work.

6.1 Summary of achievements

In the work presented in this thesis, we succeeded in adapting HMMs as online workload models and building discrete queues for processor-sharing systems to approximate delay measures. First, the incremental and multi-input HMMs act as efficient and portable benchmarks for measuring and modelling: throughput in live storage systems, communication patterns in social media for groups of users and power consumption in smartphones. Adaptive models such as the SlidHMM, the MultiHMM and, combining the best of both models, the OnlineHMM have reduced training times of standard (batch-trained) HMMs by clustering and incremental EM learning of the Baum-Welch algorithm. Secondly, our discrete queueing models have approximated delay through response time moments obtained from traces in Internet traffic scenarios by parametrising M/M/1-DPS and MMPP/M/1-DPS queues. Explicit moment formulas obtained from an iterative algorithm offer fast calculation of response time moments and density/distribution functions for approximating queueing delay and, more importantly, meet SLA requirements. Hence, through this research we have developed strategies to measure delay efficiently, with respect to QoS definitions, whilst saving resources in online measurements of computer systems. These strategies are supported by parsimonious models for generating representative traces reflecting spatiotemporal behaviour of system performance, which include throughput and delay.

We list our successful contributions in this thesis as follows:

- We formed adaptive models including the IncHMM (3.2), the SlidHMM (3.3), the MultiHMM (3.4) and the OnlineHMM (3.5) that reduce computational complex-

ity of training whilst providing incremental and/or multi-class learning of discrete data. Hence, classes of parsimonious workload benchmarks generate representative traces for replicating spatiotemporal-specific behaviour in a portable and efficient manner.

- We approximated important performance metrics such as delay using response time moments calculated from single M/M/1-DPS queues (4.3) under certain assumptions. An automated algorithm produced explicit formulas for higher response time moments. Further, we used MMPP/M/1-DPS queues (4.4) to approximate response time distribution using an improved iterative algorithm, which modelled bursty packet arrivals with DPS scheduling serving multiple job classes. SLA requirements can be matched against the response time distributions obtained from the MMPP/M/1-DPS queueing models for different loads.

- We formed strategies for the adaptive HMMs and queueing models including: a financial strategy for stock prediction (5.2); an incremental power consumption model and an objective function for performance-energy tradeoffs in smartphone applications (5.3); a dynamic allocation strategy at flow-level (5.4), which modelled queueing delay of Internet traffic at routers.

Given the achievements in this thesis, there is room for improvements and further applications for new research to extend our models. We discuss the extensions in detail in the subsequent section.

6.2 Future work

For future work, we seek a wider impact of our research and identify the methods of achieving this and outline the improvements required. This section presents the extensions for adaptive HMMs, discrete processor-sharing queues and improvements related to combining HMMs with queueing models. Future directions are also discussed in previous chapters, namely at the end of each section for the appropriate model. First, we discuss improvements related to the adaptive HMMs, which include:

- Building an incremental k-means algorithm to update clusters with new data points. As it stands, new data points are assigned to an existing cluster (i.e. formed in the k-means algorithm preceding incremental training) and these clustered data points are passed as input into the incremental HMM. An incremental k-means algorithm would save resources and improve classification accuracy in cases when new observations require a new cluster. Alternatively, the k-means algorithm can be repeated at periodic times to form new clusters.

- Obtaining a calibration period for the incremental HMMs in order to determine when batch training should re-occur (periodically) to stabilise model parameters. For example, after a number of incremental slides, train the model using a standard Baum-Welch algorithm. Once this batch training is complete, incremental slides

continue to save time and memory complexity through the adaptive Baum-Welch algorithm.

- For the multi-input training used by the MultiHMM and the OnlineHMM, we recommend experiments with unequal weights used in the clustering of traces. For example, after clustering all data points per trace and clustering these traces again into groups with similar characteristics, these doubly-clustered traces (used in the adapted forward-backward algorithm) can be weighted to prioritise specific groups of traces. This would essentially add priorities to multiple job classes according to the importance of respective traffic streams and, hence, build different classes of MultiHMMs and OnlineHMMs.

- Build a continuous-time OnlineHMM with continuous state spaces. For example, we could adapt a continuous version of the Baum-Welch algorithm used in [84] or use approximate methods such as the extended Kalman filter.

For the class of discrete, processor-sharing queues used in this thesis, we list possible extensions as follows:

- Extending response time analysis of DPS to heavy-tailed service time distributions and, hence, define M/G/1-DPS and MMPP/G/1-DPS queues in Kendall notation. Ideally, the G service time distributions may be approximated using hyper-exponential (i.e. two or more weighted exponential phases) by defining new population terms in the PDE in (4.2) to represent the new exponential phases. Further, continuous phase-type distributions represent light-tailed service time distributions and also approximate heavy-tailed distributions.

- Improving the performance-energy trade-off problem in (5.3.4) by incorporating higher moments of the response time into the objective cost function in (5.3). This adds more dimensions to the optimisation problem that requires new constraints, but can be handled using nonlinear global optimisation frameworks [113]. Ideally, distributions estimated from the higher moments can target SLAs in an optimised scenario.

- Single DPS queues assume steady state such that mean arrival rate is never more than the mean service rate; whilst this holds most of the time, on average, there exist cases when the router experiences overutilisation (i.e. there are more jobs than can be processed by the router) and the steady state assumption fails. Hence, in cases of overutilisation, the response time and the number of jobs present in the system can be obtained using spectral expansion methods [101].

- Approximating response times for networks to represent delay on a wider scale for more realistic systems. For example, to measure end-to-end delays for networks, an improvement is to group our DPS queueing models into clusters and measure the total response time for a one-directional traffic flow.

A major aim of our future work is to incorporate queueing models with features of adaptive workload models. In this way, we can obtain representative performance measures whilst saving training time and reducing memory complexity in different applications. To justify incorporating workload models into queueing models, we list potential applications with corresponding benefits:

- Job arrivals in discrete queues can be modelled using incremental HMMs to provide the transition probabilities for MMPPs, which saves time over batch training, and, hence, can predict throughput for live systems.

- Using explicit moment equations, response time can be calculated "on-the-fly," given incremental HMM-workload prediction, to approximate delay distributions, which ensures rapid verification of SLA targets for delay-sensitive applications.

There exist multiple paths for developing this work with such adaptive queueing models including: modelling spatiotemporal performance of multi-tiered systems; inferring new statistical properties of workloads and higher number of servers under varying loads, and replicating behaviour of servers with other scheduling disciplines (i.e. shortest job first). This thesis provides only a flavour of the possibilities of the queueing theory discipline in modern computer systems and should serve as a starting point for a well-known approach with new-found adaptations.

6.3 Evaluation

To evaluate the research reported in this thesis, we explain the choice of models including the relevant parameters and validation techniques. In doing so, we discuss the lessons learnt from our performance modelling and the impact of our research. We summarise the evaluation separately in two subsections, focusing on our adaptive workload models and queueing models.

6.3.1 Adaptive workload models

The adaptive HMMs, which behaved as workload benchmarks for different experiments, consisted of the following components: clustering algorithms to pre-process discrete data for batch HMM training and also to group traces for multi-input HMM training; backward approximation formulas to allow incremental learning using a forward-recurrence equation; parametrisation and validation of adaptive HMMs. We discuss these three components in the subsequent paragraphs.

Clustering

K-means clustering was used as a pre-processing technique to assign discrete data to an integer value representing a cluster. This was also applicable in incremental learning where data points were selected from traces "on-the-fly" and assigned a cluster from an

existing set of clusters. K-means was chosen as a clustering algorithm due to its comparatively low convergence rate for big data sets, unlike the expensive training cost of hierarchical clustering, which was quadratic in terms of trace length. Further, k-means was simple to implement for the discrete data sets in our experiments (i.e. no more than two dimensions required) and we could manipulate the algorithm for assigning weights in multi-input training scenarios (i.e. for the MultiHMM). Ideally, another classification algorithm to reduce the convergence rate of the k-means whilst pre-processing traces would be beneficial to improve performance of adaptive models.

Backward formula

The backward approximations (or β formulas) used in the adaptive Baum-Welch algorithm offered a forward-recurrence relationship in terms of the time variable, much like the α formula, and was necessary for incremental learning. Our β formulas improved simple approximations where new β terms were assumed to all have a value of one. The formula provided by Florez-Larrahondo *et al* [103] allowed for incremental learning with acceptable accuracy of HMM-generated traces and eliminated the reliance of β variables altogether. In the survey by Khreich *et al* [47], we found that retaining the values of the β variables was important to avoid knowledge corruption achieved via a complete set of the forward-backward probabilities. Hence, building an adaptive Baum-Welch algorithm with a forward-recurrence formula for the backward variable addressed this issue whilst providing incremental learning for symbol-wise (i.e. IncHMM and SlidHMM) and block-wise (i.e. MultiHMM and OnlineHMM) learning.

Parametrisation and validation

The adaptive HMMs used in our experiments were parametrised with two hidden states. After initial tests on discrete data sets, we found that three or more hidden states produced duplicate rows of probabilities in the state transition matrix. Further, by reducing the number of hidden states in the online HMMs, the training complexity of the Baum-Welch algorithm (i.e. $O(N^2T)$ with N hidden states and T observations) was minimised. Generating synthetic HMM traces allowed comparisons with original traces and validated the models through mean error measurements (i.e. sMAPE) and autocorrelation functions. We note that the error of the likelihood function is also a possible validation technique between adaptive HMMs, but we found sMAPE and autocorrelation metrics more reflective of model usefulness with respect to the data sets investigated and the correlation or burstiness replicated. A Monte Carlo simulation is another option for replicating similar results given the data sets to validate the HMM-generated traces.

Impact

The impact of the adaptive HMMs exists in the cost-effectiveness and applicability of such parsimonious models acting as online workload benchmarks for storage systems (i.e. modelling read and write operations at file servers), social networks (i.e. predicting

activity of Twitter users in groups), smartphone battery (i.e. predicting power consumption given user activity) and financial strategies (i.e. forecasting stock movements for different indices). A prerequisite of the adaptive HMMs is that the traces have discrete data, but the incremental and multi-input modifications save computational complexity of training models on live systems (compared to standard HMMs trained in a batch manner) that are typically latency-sensitive (i.e. high-frequency trading and users interactions on social media). Combining the resource-saving capabilities of the adaptive HMMs with the burstiness and mode-switching features of standard HMMs allows modelling spatiotemporal activity of computer systems to avoid bottlenecks during peak times and, hence, facilitate resource planning. Future applications of adaptive HMMs include characterising workloads according to their HMM parameters and modelling similar workload environments either as inputs to simulation or analytical models (i.e. Markov-modulated fluid models).

6.3.2 Queueing models

We evaluate the discrete, processor-sharing queueing models. Specifically, we address the methods for approximating response time moments and discuss the impact of the queueing models for different applications.

Response time approximation

For M/M/1-EPS and M/M/1-DPS queues (i.e. discrete queues with egalitarian and discriminatory processor-sharing disciplines, respectively), we approximate response time moments by differentiating a Laplace-transformed partial differential equation (PDE) and evaluating at specific values to obtain moment expressions. Continuing with successive differentiations, we determine response time moments recursively, via an automated algorithm, and obtain explicit formulas. Such response time moments essentially represent delays in storage systems and Internet traffic (i.e. queueing delays at routers). In contrast to work by Kim and Kim [155] that manipulates a similar PDE to produce an algorithm for obtaining only the first two moments, we provide explicit formulas of higher response time moments (i.e. up to four) for one and two job classes. Given more than one job class, the formulas for higher moments were lengthy with hundreds of terms, most of which were priority weights as part of DPS scheduling. We validated our analytical response time moments against simulated moments, obtained from JSIMwiz queueing simulations as part of the JMT modelling suite. We extended our response time approximations for MMPP/M/1-DPS queues, which were able to capture the burstiness of multi-class job arrivals using a weighted superposition technique of M/M/1-DPS queues. Further, we built density functions from response time moments using distribution-fitting algorithms, such as the general lambda distribution (GLD), that allow comparisons with service level agreements (SLAs). Having experimented with single queues, an aim is to build a network of DPS queues to represent delays in more complex systems.

Impact

The queueing models have multiple applications for modelling delay in different computer systems. Using MMPP/M/1-DPS allowed modelling of Internet traffic, specifically packets arriving at a router, which allowed the discrete packet arrivals to be serviced by DPS queues. Typically, fluid queueing models allow arrivals to be continuous and have modelled the performance of network switches and routers [116]. The theoretical delay moments obtained from single DPS queues should be compared with delay information based on busy periods from fluid models. The flow delay strategy (i.e. FlowDPS), which used response time moments obtained from MMPP/M/1-DPS queues, offered dynamic allocation of multi-class jobs to minimise mean and/or variance of delays. The applications of modelling queueing delay for Internet traffic such as voice data, typically found in small data packets, offer incentives to reduce jitter (i.e. minimise variance of packet delays). Such strategies are preliminary experiments for at least two purposes: first, to find similarities with existing scheduling disciplines (i.e. DWRR) that can be modelled and replicated; secondly, to obtain performance measurements in a more resourceful setting with aims to reduce delay, speed up allocation, and train on live systems.

Bibliography

[1] T. Chis, P. G. Harrison: Incremental HMM with an improved Baum-Welch Algorithm, In *Proc. ICCSW*, p. 29-34, London, UK, 2012

[2] T. Chis, P. G. Harrison: iSWoM: The incremental Storage Workload Model using Hidden Markov Models, In *Proc. ASMTA*, p. 127-141, Ghent, Belgium, 2013

[3] T. Chis: Sliding Hidden Markov Model for Evaluating Discrete Data, In *Proc. EPEW*, p. 251-262, Venice, Italy, 2013

[4] T. Chis, P. G. Harrison: Analysing and Predicting Patient Arrival Times, In *Proc. ISCIS*, p. 77-85, Paris, France, 2013

[5] T. Chis, P. G. Harrison: Modeling Multi-User Behaviour in Social Networks, In *Proc. IEEE MASCOTS*, p. 168-173, Paris, France, 2014

[6] T. Chis, P. G. Harrison: Adapting Hidden Markov Models for Online Learning, In *Elsevier Electronic Notes in Theoretical Computer Science*, **318**, pp. 109-127, 2015

[7] T. Chis, P. G. Harrison: Moment-Generating Algorithm for Response Time in Processor Sharing Queueing Systems, In *Proc. EPEW*, p. 80-95, Madrid, Spain, 2015

[8] T. Chis, P. G. Harrison: Higher response time moments for M/M/1 discriminatory processing-sharing queues, In *Proc. EAI VALUETOOLS*, Berlin, Germany, 2015

[9] T. Chis, P. G. Harrison: Modeling Packet Delay and Traffic Flow with MMPP/M/1 Discriminatory Processing Sharing Queues, *Submitted for publication*, 2015

[10] T. Chis, P. G. Harrison: Modeling Performance and Energy in Smartphone Applications, *Submitted for publication*, 2015

[11] D. G. Kendall: Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain, In *The Annals of Mathematical Statistics*, **24**(6), pp. 338-354, 1953

[12] L. E. Baum, T. Petrie: Stastical Inference for Probabilistic Functions of Finite Markov Chains, In *The Annals of Mathematical Statistics*, **37**, pp. 1554-1563, 1966

[13] 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**, pp. 164-171, 1970

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

[15] A. P. Dempster, N. M. Laird, D. B. Rubin: Maximum Likelihood from Incomplete Data via the EM Algorithm, In *Journal of the Royal Statistical Society*, **39**(1), pp. 1-38, 1977

[16] H. Akaike: A new look at the statistical model identification, In *IEEE Transactions on Automatic Control*, **19**(6), pp. 716-723, 1974

[17] R. Sundberg: Maximum likelihood theory and applications for distributions generated when observing a function of an exponential family variable, PhD Thesis, Institute for Mathematical Statistics, Stockholm University, 1971

[18] H. Bozdogan: Akaike's Information Criterion and Recent Developments in Information Complexity, In *Journal of Mathematical Psychology*, **44**, pp. 62-91, 2000

[19] L. Kauffman, P. J. Rousseeuw: Finding Groups in Data: An Introduction to Cluster Analysis, In *Wiley*, **1**, 1990

[20] A. K. Jain, M. N. Murty, P. J. Flynn: Data Clustering: A Review, In *ACM Computing Surveys*, **31**(3), pp. 264-323, 1999

[21] P. G. Harrison, N. M. Patel: Performance Modelling of Communication Networks and Computer Architectures, *Addison-Wesley*, 1993

[22] U. Gahde, S. Hartmann, J. H. Wolf: Models, Simulations, and the Reduction of Complexity, *De Gruyter*, **1**, 2013

[23] M. Hajjat, X. Sun, Y. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, M. Tawarmalani: Cloudward Bound: Planning for Beneficial Migration Enterprise Applications to the Cloud, In *Proc. ACM SIGCOMM*, p. 243-254, New Delhi, India, 2010

[24] S. A. Baset, L. Wang, C. Tang: Towards an Understanding of Oversubscription in Cloud, In *Proc. Hot-ICE*, p. 1-6, San Jose, USA, 2012

[25] S. M. Ghoreyshi: Energy-Efficient Resource Management of Cloud Data Centers under Fault Tolerance Constraints, In *Proc. IGCC*, p. 1-6, Arlington, USA, 2013

[26] C. Brownlees, R. Engle, B. Kelly: A Practical Guide to Volatility Forecasting through Calm and Storm, In *Jour. Risk*, **14**(2), p. 3-22, 2012

[27] M. N. O. Sadiku, S. M. Musa: Performance Analysis of Computer Networks, In *Routledge*, **1**, 2013

[28] R. S. Prasad, C. Dovrolis, M. Thottan: Router Buffer Sizing Revisited: The Role of the Output/Input Capacity Ratio, In *Proc. CoNEXT*, p. 1-12, New York, USA, 2007

[29] C. Tofallis: A Better Measure of Relative Prediction Accuracy for Model Selection and Model Estimation, In *Journal of Operational Research Society*, **66**, pp. 1352-1362, 2014

[30] C. C. Moallemi, M. Saglam: The Cost of Latency in High-Frequency Trading, In *Operations Research*, **61**, pp. 1070–1086, 2013

[31] D. F. Galletta, Y. Zhang: Human-Computer Interaction and Management Information Systems: Applications, In *Springer*, **1**, 2013

[32] A. Parate, M. Bohmer, D. Chu, D. Ganesan, B. M. Marlin: Practical Prediction and Prefetch for Faster Access to Applications on Mobile phones, In *Proc. UbiComp*, p. 275-284, Zurich, Switzerland, 2013

[33] T. Zhang, R. Ramakrishnan, M Livny: BIRCH: an efficient data clustering method for very large databases, In *Proc. ACM SIGMOD*, p. 103-114, Montreal, Canada, 1996

[34] D. P. Heyman, D. Lucatoni: Modeling Multiple IP Traffic Streams With Rate Limits, In *IEEE/ACM Transactions on Networking*, **11**(6), pp. 948-958, 2003

[35] M. Ester, H. Kriegel, J. Sander, X. Xu: A density-based algorithm for discovering clusters in large spatial databases with noise, In *Proc. KDD*, p. 226-231, Portland, USA, 1996

[36] P. A. P. Moran: A probability theory of dams and storage systems, In *Journal of Applied Sciences*, **5**, pp. 116-124, 1954

[37] M. Bertoli, G. Casale, G. Serazzi: JMT: performance engineering tools for system modeling, In *ACM SIGMETRICS Performance Evaluation Review*, **36**(4), pp. 10-15, 2009

[38] X-12-ARIMA, http://www.census.gov/srd/www/x12a/

[39] B. Ciciani, A. Santoro, P. Romano: Approximate Analytical Models for Networked Servers Subject to MMPP Arrival Processes, In *Proc. IEEE NCA*, p. 25-32, Rome, Italy, 2007

[40] W. Khreich, E. Granger, A. Miri, R. Sabourin: Adaptive ROC-based ensembles of HMMs applied to anomaly detection, In *Journal of Pattern Recognition*, **45**(1), pp. 208-230, 2012

[41] RFC 2722, https://tools.ietf.org/html/rfc3697

[42] O. Cappe: Online EM Algorithm for Hidden Markov Models, In *Journal of Computational and Graphical Statistics*, **20**(3), pp. 728-749, 2011

[43] H. Li, M. Muskulus, L. Wolters: Modeling Job Arrivals in a data-intensive Grid, In *Proc. JSSPP Workshop*, p. 210-231, St. Malo, France, 2006

[44] J. B. MacQueen: Some Methods for classification and Analysis of Multivariate Observations, In *Proc. BSMSP*, p. 281-297, 1967

[45] K. Starbird, J. Maddock, M. Orand, P. Achterman, R. M. Mason: Rumors, False Flags, and Digital Vigilantes: Misinformation on Twitter after the 2013 Boston Marathon Bombing, In *Proc. iConference*, p. 654-662, 2014

[46] T. Bonald, A. Proutiere: Insensitive bandwidth sharing in data networks, In *Queueing Systems*, **44**, pp. 69-100, 2003

[47] W. Khreich, E. Granger, A. Miri, R. Sabourin: A survey of techniques for incremental learning of HMM parameters, In *Elsevier Information Sciences*, **197**, pp. 105-130, 2012

[48] J. Mizuno, T. Watanabe, K. Ueki, K. Amano, E. Takimoto, A. Maruoka: On-line Estimation of Hidden Markov Model Parameters, In *Proc. Discovery Science*, p. 155-169, Kyoto, Japan, 2000

[49] G. E. P. Box, G. Jenkings: Time series analysis: forecasting and control, *Holden-Day*, 1976

[50] G. Casale, E. Z. Zhang, Smirni: KPC-Toolbox: Simple Yet Effective Trace Fitting Using Markovian Arrival Processes, In *Proc. QEST*, St. Malo, France, 2008

[51] C. Aykanat, A. Turk, O. Selvitopi, H. Ferhatosmanoglu: Temporal Workload-Aware Replicated Partitioning for Social Networks, In *IEEE Transactions on Knowledge and Data Engineering*, p. 1-14, 2014

[52] R. N. Calheiros, R. Buyya: Energy-Efficient Scheduling of Urgent Bag-of-Tasks Applications in Clouds through DVFS, In *Proc. IEEE CloudCom*, p. 342-349, Singapore, 2014

[53] K. Pearson: Contributions to the mathematical theory of evolution, In *Philosophical Transactions of the Royal Society of London, A.*, pp. 71, 1894

[54] G. E. P. Box: Robustness in the strategy of scientific model building, In *Robustness in Statistics*, Academic Press, p. 201-236, 1979

[55] K. Pearson: Notes on regression and inheritance in the case of two parents, In *Proc. Royal Society of London*, **58**, pp. 240-242, 1895

[56] DevBytes: Efficient Data Transfers - Understanding the Cell Radio, https://www.youtube.com/watch?v=cSIB2pDvH3E

[57] If You're Programming A Cell Phone Like A Server You're Doing It Wrong, http://highscalability.com/blog/2013/9/18/if-youre-programming-a-cell-phone-like-a-server-youre-doing.html

[58] Z. Ghahramani, M. Jordan: Factorial hidden Markov models, In *Machine Learning*, **29**, pp. 245-273, 2012

[59] G. Mongillo, S. Deneve: Online Learning with Hidden Markov Models, In *Neural Computation*, **20**, pp. 1706-1716, 2008

[60] E. Altman, K. Avrachenkov, U. Ayesta: A Survey on Discriminatory Processor Sharing, In *Queueing Systems*, **53**(1), pp. 53-63, 2006

[61] M. B. Priestly: Spectral analysis and time series, Academic Press, 1981

[62] C. Vogler, D. Metaxas: Parallel Hidden Markov Models for American Sign Language Recognition, In *Proc. ICCV*, p. 116-122, Kerkyra, 1999

[63] L. Guo, E. Tan, S. Chen, Z. Xiao, X. Zhang: The Stretched Exponential Distribution of Internet Media Access Patterns, In *Proc. PODC*, Toronto, Canada, 2008

[64] P. G. Harrison, N. M. Patel, S. Zertal: Response Time Distribution of Flash Memory Accesses, In *Elsevier Performance Evaluation*, **67**, pp. 248-259, 2010

[65] J. Ashraf, N. Iqbal, N. S. Khattak, A. M. Zaidi: Speaker Independent Urdu Speech Recognition Using HMM, In *Proc. IEEE INFOS*, p. 1-5, 2010

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

[67] R. D. Malmgren, J. M. Hofman, L. A. N. Amaral, D. Watts: Characterizing Individual Communication Patterns, In *Proc. KDD*, p. 226-231, Paris, 2009

[68] C. Burge, S. Karlin: Prediction of complete gene structures in human genomic DNA, In *Journal of Molecular Biology*, **268**, pp. 78-94, 1997

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

[70] N. Oliver, A. Garg, E. Horvitz: Layered Representations for Learning and Inferring Office Activity from Multiple Sensory Channels, In *Computer Vision and Image Understanding*, **96**, pp. 163-180, 2004

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

[72] A. J. Smith: Trace Driven Simulation in Research on Computer Architecture and Operating Systems, In *Proc. NDSMC*, p. 43-49, Tokyo, Japan, 1994

[73] A. Shye, B. Scholbrock, G. Memik, P. A. Dinda: Characterizing and Modeling User Activity on Smartphones, Technical Report, Northwest University, 2010

[74] M. Hassan, B. Nath: Stock Market Forecasting Using Hidden Markov Model: A New Approach, In *Proc. IEEE ISDA*, p. 192-196, Wroclaw, Poland, 2005

[75] L. de Angelis, L. J. Paas: A dynamic analysis of stock markets using a hidden Markov model, In *Journal of Applied Statistics*, **40**(8), pp. 1682-1700, 2013

[76] L. R. Rabiner: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, In *Proc. IEEE*, **77**(2), pp. 257-286, 1989

[77] The CAIDA UCSD Statistical information for the CAIDA Anonymized Internet Traces, http://www.caida.org/data/passive/passive_trace_statistics.xml

[78] G. Zhang, E. Patuwo, M. Hu: Forecasting with Artificial Neural Networks: The State of The Art, In *International Journal of Forecasting*, **14**(1), pp. 35-62, 1998

[79] C. Liu: cuHMM: a CUDA Implementation of Hidden Markov Model Training and Classification, http://liuchuan.org/pub/cuHMM.pdf

[80] S. Hymel: Massively Parallel Hidden Markov Models for Wireless Applications, MSc Thesis, Virginia Polytechnic Institute and State University, 2011

[81] M. Wang, A. Ailamaki, C. Faloutsos: Capturing the spatio-temporal behavior of real traffic data, In *Elsevier Performance Evaluation*, **49**(1/4), pp. 147-163, 2002

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

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

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

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

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

[87] A. Barbulescu, E. Bautu: A Hybrid Approach for Modeling Financial Time Series, In *International Arab Journal of Information Technology*, **9**(4), pp. 327-335, 2012

[88] A. Lo: The Adaptive Markets Hypothesis: Market Efficiency from an Evolutionary Perspective, In *Journal of Portfolio Management*, **30**(1), pp. 15-29, 2004

[89] R. Kalman: A New Approach to Linear Filtering and Prediction Problems, In *Journal of Basic Engineering*, **82**(1), pp. 35-45, 1960

[90] G. Box, G. M. Jenkins, G. C. Reinsel: Time Series Analysis: Forecasting and Control, *Prentice-Hall*, **3**, 1994

[91] P. J. Brockwell, R. A. Davis: Introduction to Time Series and Forecasting, *Springer*, **2**, 2010

[92] P. Whittle: Hypothesis Testing in Time Series Analysis, PhD Thesis, University of Uppsala, 1951

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

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

[95] M. Ghahramani, A. Thavaneswaran: Financial applications of ARMA models with GARCH errors, In *Journal of Risk Finance*, **7**(5), pp. 525-543, 2006

[96] J. H. Cochrane: Time Series for Macroeconomics and Finance, Lecture Notes, Chicago Booth School of Business, 1997

[97] S. W. M. Au-Yeung, U. Harder, E. McCoy, W. J. Knottenbelt: Predicting patient arrivals to an accident and emergency department, In *Emergency Medicine Journal*, **26**, pp. 241-244, 2009

[98] G. Burghardt, R. Duncan, L. Liu: What You Should Expect From Trend Following, In *Alternative Edge Research Note*, 2004

[99] Y. Chou: Statistical Analysis, In *Holt International*, **17**, 1975

[100] S. L. Scott, P. Smyth: The Markov Modulated Poisson Process and Markov Poisson Cascade with Applications to Web Traffic Data. In *Bayesian Statistics*, **7**, pp. 671-680, 2003

[101] I. Mitrani: Spectral Expansion Solutions for Markov-Modulated Queues, In *Network Performance Engineering*, **5233**, pp. 423-446, 2011

[102] P. Salvador, A. Pacheco, R. Valadas: Modeling IP traffic: Joint Characterization of Packet Arrivals and Packet Sizes using BMAPs, In *Computer Networks*, **44**, pp. 335-352, 2004

[103] G. Florez-Larrahondo, S. Bridges, E. A. Hansen: Incremental Estimation of Discrete Hidden Markov Models on a New Backward Procedure, In *Proc. AAAI*, pp. 758-763, Pittsburgh, USA, 2005

[104] X. Zhang, A. Riska, E. Riedel: Characterization of the E-commerce Storage Subsystem Workload, In *Proc. QEST*, **5**, p. 297-306, St. Malo, France, 2008

[105] J. Curtis: 10 top cloud computing providers for 2014, http://www.cbronline.com/news/enterprise-it/it-services/10-top-cloud-computing-providers-for-2014-4401618

[106] RUBiS Implementation, http://rubis.objectweb.org/

[107] On-Line Transaction Processing (OLTP) Benchmark, http://www.tpc.org/tpcc

[108] J. F. Perez, B. Van Houdt: Quasi-birth-and-death processes with restricted transitions and its applications, In *Elsevier Performance Evaluation*, **68**(2), pp. 126-141, 2011

[109] Z. Kumas, K. Keeton, R. Becker-Szendy: I/O Workload Characterization, In *Proc. CAECW*, 2001

[110] Transactional Web e-Commerce Benchmark, http://www.tpc.org/tpcw

[111] C. Velazco: Google gives students unlimited cloud storage, http://www.engadget.com/2014/09/30/google-drive-for-education/

[112] J. Wray: Where's The Rub: Cloud Computing's Hidden Costs, http://www.forbes.com/sites/centurylink/2014/02/27/wheres-the-rub-cloud-computings-hidden-costs/

[113] R. Misener, C. A. Floudas: ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations, In *Journal of Global Optimization*, **59**(2), pp. 503-526, 2014

[114] G. Fayolle, R. Iasnogorodski, I. Mitrani: Sharing a Processor Among Many Job Classes, In *Journal of the ACM*, **27**(3), pp. 519-532, 1980

[115] AISO.net, http://www.aiso.net/index.html

[116] N. Hohn, D. Veitch, K. Papagiannaki, C. Diot: Bridging router performance and queueing theory, In *Proc. SIGMETRICS/Performance*, p. 355-366, New York, USA, 2004

[117] B. Stenger, V. Ramesh, N. Paragois, F. Coetzee, J. Buhmann: Topology Free Hidden Markov Models: Application to Background Modeling, In *Proc. ICCV*, p. 297-301, Vancouver, Canada, 2001

[118] K. Keeton, A. Veitch, D. Obal, J. Wilkes: I/O Characterization of Commercial Workloads, In *Proc. CAECW*, p. 1-9, Toulouse, France, 2000

[119] T. Huria, M. Ceraolo, J. Gazzarri, R. Jackey: High fidelity electrical model with thermal dependence for characterization and simulation of high power lithium battery cells, In *Proc. IEEE IEVC*, p. 1-8, Greenville, USA, 2012

[120] C. Rohner, L. M. Feeney, P. Gunningberg: Evaluating Battery Models in Wireless Sensor Networks, In *Proc. LNCS WWIC*, **7889**, p. 29-42, St. Petersburg, Russia, 2013

[121] V. Rao, G. Singhal, A. Kumar, N. Navet: Battery model for embedded systems, In *Proc. IEEE VLSID*, p. 105-110, Washington DC, USA, 2005

[122] E. G. Coffman Jr., R. R. Muntz, H. Trotter: Waiting Time Distributions for Processor-Sharing Systems, In *Journal of the ACM*, **17**, pp. 123-130, 1970

[123] B. J. Prabhu, A. Chockalingam, V. Sharma: Performance Analysis of Battery Power Management Schemes in Wireless Mobile Devices, In *Proc. IEEE WCNC*, p. 825-831, Orlando, USA, 2002

[124] G. L. Jones: Open Battery, http://www.openbattery.com/

[125] O. Younes, N. Thomas: Modelling and performance analysis of multi-hop ad hoc networks, In *Elsevier Simulation Modelling Practice and Theory*, **38**, pp. 69-97, 2013

[126] Battery Design Studio, http://www.batdesign.com/batterydesign.html

[127] J. F. Manwell, J. G. McGowan: Lead acid battery storage model for hybrid energy systems, In *Elsevier Solar Energy*, **50**, pp. 399-405, 1993

[128] W. J. Stewart: Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling. *Princeton University Press*, pp. 409, 2009

[129] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, P. Bahl: Anatomizing Application Performance Differences on Smartphones, In *Proc. ACM Mobisys*, p. 165-178, San Francisco, USA, 2010

[130] M. A. Kjaer, M. Kihl, A. Robertsson: Response-Time Control of a Processor-Sharing System using Virtualised Server Environments, In *Proc. IFAC*, p. 3612-3618, Seoul, South Korea, 2008

[131] S. W. M. Au-Yeung, N. J. Dingle, W. J. Knottenbelt: Efficient Approximation of Response time Densities and Quantiles in Stochastic Models, In *Proc. ACM WOSP*, p. 151-155, San Francisco, USA, 2004

[132] P. X. Gao, A. R. Curtis, B. Wong, S. Keshav: It's not easy being green, In *Proc. ACM SIGCOMM*, p. 211-222, Helsinki, Finland, 2012

[133] R. Y. Alawnah, I. Ahmad, E. A. Alrashed: Green and Fair Workload Distribution in Geographically Distributed Data, In *Journal of Green Engineering*, **4**, pp. 69-98, 2014

[134] L. Massoulie, J. W. Roberts: Bandwidth sharing and admission control for elastic traffic, In *Telecommunication Systems*, **15**, pp. 185-201, 2000

[135] Simscape, http://www.mathworks.co.uk/products/simscape/

[136] Simulink, http://www.mathworks.co.uk/products/simulink/

[137] BU-808: How to Prolong Lithium-based Batteries, http://batteryuniversity.com/learn/article/ how_to_prolong_lithium_based_batteries

[138] T. J. Ott: The Sojourn-Time Distribution in the M/G/1 Queue with Processor Sharing, In *Journal of Applied Probability*, **21**, pp. 360-378, 1984

[139] A. Wierman: Scheduling for Today's Computer Systems: Bridging Theory and Practice, School of Computer Science, Carnegie Mellon University, 2007

[140] A. Wierman, M. Harchol-Balter: Classifying Scheduling Policies with Respect to Higher Moments of Conditional Response Time, In *Proc. ACM SIGMETRICS*, p. 229-240, Banff, Canada, 2005

[141] S. F. Yashkov: Processor-Sharing Queues: Some Progress In Analysis, In *Queueing Systems*, **2**, pp. 1-17, 1987

[142] A. P. Zwart, O. J. Boxma: Sojourn time asymptotics in the M/G/1 processor sharing queue, In *Queueing Systems*, **35**, pp. 141-166, 2000

[143] J. W. Roberts: A survey on statistical bandwidth sharing, In *Computer Networks*, **45**, pp. 319-332, 2004

[144] S. Lohr: For Impatient Web Users, an Eye Blink Is Just Too Long to Wait, http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html

[145] P. Bodik, C. Sutton, A. Fox, D. Patterson, M. Jordan: Response-Time Modeling for Resource Allocation and Energy-Informed SLAs, Technical Report, University of California, Berkeley, USA, 2010

[146] M. Curado, R. Veludo, E. Monteiro: Queue Management and QoS Routing for Traffic Differentiation, In *Proc. ICOMP*, p. 209-215, Las Vegas, USA, 2005

[147] G. L. Jones, P. G. Harrison, U. Harder, T. Field: Fluid Queue Models of Battery Life, In *Proc. IEEE MASCOTS*, p. 278-285, Singapore, 2011

[148] A. Wierman, L. L. H. Andrew, A. Tang: Power-aware speed scaling in processor sharing systems: Optimality and robustness, In *Elsevier Performance Evaluation*, **69**(12), pp. 601-622, 2012

[149] G. Casale, P. G. Harrison: AutoCAT: Automated Product-Form Solution of Stochastic Models, In *Matrix-Analytic Methods in Stochastic Models*, **27**, pp. 57-85, 2013

[150] R. N. Queija: Sojourn times in non-homogeneous QBD processes with processor sharing, In *Stochastic Models*, **17**, pp. 61-92, 2001

[151] H. Masuyama, T. Takine: Sojourn time distribution in a MAP/M/1 processor-sharing queue, In *Operations Research Letters*, **31**, pp. 406-412, 2003

[152] N. Bansal: Analysis of the M/G/1 processor-sharing queue with bulk arrivals, In *Operations Research Letters*, **31**, pp. 401-405, 2003

[153] A. Lebrecht: Queueing network models of Zoned RAID system performance, PhD Thesis, Department of Computing, Imperial College London, 2009

[154] J. Ramberg, B. Schmeiser: An approximate method for generating asymmetrics random variables, In *Communications of the ACM*, **17**, pp. 78-82, 1974

[155] J. Kim, B. Kim: Sojourn time distribution in the M/M/1 queue with discriminatory processor-sharing, In *Elsevier Performance Evaluation*, **58**, pp. 341-365, 2004

[156] L. Kleinrock: Time-shared systems: A theoretical treatment, In *Journal ACM*, **14**(2), pp. 242-261, 1967

[157] J. Ramberg, E. Dudewicz, P. Tadikamalla, E. Mykytka: A probability distribution and its uses in fitting data, In *Technometrics*, **21**, pp. 201-214, 1979

[158] M. Freimer, G. Mudholkar, G. Kollia, C. Lin: A study of the generalized Tukey Lambda family, In *Communications in Statistics*, **17**, pp. 3547-3567, 1988

[159] A. Lakhany, H. Mausser: Estimating the parameters of the General Lambda Distribution, In *ALGO Research Quarterly*, **3**, pp. 47-58, 2000

[160] A. R. Ward, W. Whitt: Predicting Response Times in Processor-Sharing Queues, In *Proc. Fields Institute Conference on Communication Networks*, 2000.

[161] F. Kelly: Stochastic Networks and Reversibility, In *Wiley*, **1**, 1979

[162] K. Avrachenkov, U. Ayesta, P. Brown, R. Nunez-Queija: Discriminatory Processor Sharing Revisited, In *Proc. IEEE Infocom*, p. 784-795, Miami, USA, 2005

[163] Twitter Usage Statistics, http://www.internetlivestats.com/twitter-statistics/# trend

[164] Embedded Microprocessor Benchmark Consortium (EEMBC), http://eembc.org/

[165] A. Wierman: Scheduling for Today's Computer Systems: Bridging Theory and Practice, PhD Thesis, School of Computer Science, Carnegie Mellon University, 2007

[166] AndEBench-Pro, http://eembc.org/andebench/index_pro.php

[167] M. Taboga: F distribution, http://www.statlect.com/F_distribution.htm

[168] S. Aalto, U. Ayesta, S. Borst, V. Misra, R. Nunez-Queija: Beyond Processor Sharing, In *ACM SIGMETRICS Performance Evaluation Review*, **34**, pp. 36-43, 2007

[169] A. A. Kherani, A. Kumar: On Processor Sharing as a Model for TCP Controlled HTTP-like Transfers, In *Proc. IEEE ICC*, p. 2256-2260, Paris, France, 2004

[170] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, N. McKeown: Processor Sharing Flows in the Internet, In *Proc. LNCS IWQoS*, **3552**, p. 271-285, Passau, Germany, 2005

[171] N. L. Johnson, S. Kotz, N. Balakrishnan: Continuous Univariate Distributions, Wiley, **2**, 1995

[172] P. Hande, S. Zhang, M. Chiang: Distributed Rate Allocation for Inelastic Flows, In *IEEE/ACM Transactions on Networking*, **15**(6), pp. 1240-1253, 2007

[173] A. Moitra, G. Valiant: Settling the Polynomial Learnability of Mixtures of Gaussians, In *Proc. IEEE FOCS*, p. 93-102, Las Vegas, USA, 2010

[174] W. Wei, B. Wang, D. Towsley: Continuous-time hidden Markov models for network performance evaluation, In *Elsevier Performance Evaluation*, **49**(1/4), pp. 129-146, 2002

[175] K. Pearson: Mathematical Contributions to the Theory of Evolution XIX. Second Supplement to a Memoir on Skew Variation, In *Philosophical Transactions of the Royal Society of London* **216**, pp. 429-457, 1916

[176] Y. Park, J. V. Chen: Acceptance and adoption of the innovative use of smartphone, In *Industrial Management and Data Systems*, **107**, pp. 1349-1365, 2007

[177] M. Huilgol: Xiaomi aims to sell 100 million smartphones in 2015, http://www.bgr.in/news/xiaomi-aims-to-sell-100-million-smartphones-in-2015/

[178] M. Moore: Huawei Looks To Shift 100 Million Smartphones in 2015, http://www.techweekeurope.co.uk/mobility/huawei-100m-smartphones-2015-160308

[179] I. Macleod: Two-fifths of online sales to be via smartphones and tablets by 2018, http://www.thedrum.com/news/2014/06/23/two-fifths-online-sales-be-smartphones-and-tablets-2018

[180] BU-204: Lithium-based batteries, http://batteryuniversity.com/learn/article/lithium_based_batteries

[181] PSIM Tutorial: How To Use Lithium-Ion Battery Model, http://powersimtech.com/wp-content/uploads/2013/04/Tutorial-How-to-use-Lithium-Ion-battery-model.pdf

[182] J. P. Cohen: Cell Radio ShutOff, https://play.google.com/store/apps/details?id=the.radioshutoff

[183] P. Sawers: Samsung drives $17M investment round in Seeo to help build better batteries for electric cars, http://venturebeat.com/2014/12/09/samsung-drives-17m-investment-round-to-help-build-better-batteries-for-electric-cars/

Appendix A

A.1 Proof of HMM properties

We prove useful properties of HMM equations introduced in the background chapter. Such mathematical properties prove the equations used as part of the Baum-Welch algorithm. Note, we use the shorthand term O_t to represent $O_t = o_t$ and, similarly, C_t replaces $C_t = c_t$.

Property 1 For all integers t and l such that $1 \leq t \leq l \leq T$, we have:

$$P(O_l, \ldots, O_T \mid C_t, \ldots, C_T) = P(O_l, \ldots, O_T \mid C_l, \ldots, C_T) \tag{A.1}$$

Proof From the definition of conditional probability, we re-write the LHS as:

$$\begin{aligned}
&P(O_l, \ldots, O_T \mid C_t, \ldots, C_T) \\
&\quad = \frac{1}{P(C_t, \ldots, C_T)} \sum_{c_1, \ldots, c_{t-1}} P(O_l, \ldots, O_T \mid C_1, \ldots, C_T) P(C_1, \ldots, C_T) \\
&\quad = \frac{1}{P(C_t, \ldots, C_T)} \sum_{c_1, \ldots, c_{t-1}} P(O_l \mid C_l) \ldots P(O_T \mid C_T) P(C_1, \ldots, C_T) \\
&\quad = \frac{1}{P(C_t, \ldots, C_T)} P(O_l \mid C_l) \ldots P(O_T \mid C_T) \sum_{c_1, \ldots, c_{t-1}} P(C_1, \ldots, C_T) \\
&\quad = \frac{1}{P(C_t, \ldots, C_T)} P(O_l \mid C_l) \ldots P(O_T \mid C_T) [P(C_t, \ldots, C_T)] \\
&\quad = P(O_l, \ldots, O_T \mid C_l, \ldots, C_T)
\end{aligned}$$

which is the RHS as required. □

Property 2 For $t = 1, 2, \ldots, T$, we have:

$$P(O_1, \ldots, O_T \mid C_t) = P(O_1 \ldots, O_t \mid C_t) P(O_{t+1}, \ldots, O_T \mid C_t) \tag{A.2}$$

Proof Using the mutual independence of $O_1, \ldots, O_T$, given $C_1, \ldots, C_T$, the LHS is:

$P(O_1, \ldots, O_T \mid C_t)$

$$= \frac{1}{P(C_t)} \sum_{c_1,\ldots,c_{t-1}} \sum_{c_{t+1},\ldots,c_T} P(C_1, \ldots, C_T) P(O_1, \ldots, O_t \mid C_1, \ldots, C_T) P(O_{t+1}, \ldots, O_T \mid C_1, \ldots, C_T)$$

$$= \frac{1}{P(C_t)} \sum_{c_1,\ldots,c_{t-1}} \sum_{c_{t+1},\ldots,c_T} P(O_1, \ldots, O_t, C_1, \ldots, C_T) P(O_{t+1}, \ldots, O_T \mid C_1, \ldots, C_T)$$

Using equation (A.1), we obtain:

$$= \frac{1}{P(C_t)} \sum_{c_1,\ldots,c_{t-1}} \sum_{c_{t+1},\ldots,c_T} P(O_1, \ldots, O_t, C_1, \ldots, C_T) P(O_{t+1}, \ldots, O_T \mid C_t, \ldots, C_T)$$

Summing over $\{c_{t+1}, \ldots, c_T\}$, we obtain:

$$= \frac{1}{P(C_t)} \sum_{c_1,\ldots,c_{t-1}} P(O_1, \ldots, O_t, C_1, \ldots, C_T) P(O_{t+1}, \ldots, O_T \mid C_t)$$

Summing over $\{c_1, \ldots, c_{t-1}\}$ gives us:

$$= \frac{1}{P(C_t)} P(O_1, \ldots, O_t, C_t) P(O_{t+1}, \ldots, O_T \mid C_t)$$

$$= P(O_1, \ldots, O_t \mid C_t) P(O_{t+1}, \ldots, O_T \mid C_t)$$

which is the RHS as required. □

Property 3 For $t = 1, 2, \ldots, T$, we have:

$$P(O_t, \ldots, O_T \mid C_t) = P(O_t \mid C_t) P(O_{t+1}, \ldots, O_T \mid C_t) \tag{A.3}$$

Proof We sum the RHS expression of equation (A.2) over the set $\{o_1, \ldots, o_{t-1}\}$ and obtain:

$$\begin{aligned} P(O_t, \ldots, O_T \mid C_t) &= \sum_{o_1,\ldots,o_{t-1}} P(O_1, \ldots, O_t \mid C_t) P(O_{t+1}, \ldots, O_T \mid C_t) \\ &= P(O_t \mid C_t) P(O_{t+1}, \ldots, O_T \mid C_t) \end{aligned}$$

which is the RHS as required. □

A.2 Mathematica algorithm M/M/1-DPS

An example screenshot of the Mathematica algorithm is given in figure A.1. This numerical algorithm initialises different weights and rates for two job classes and outputs four response time moments (sufficient for the purpose of approximating density functions), but easily extends to higher moments as it is fast in its numerical iterations. The pseudocode along with explanations on how the algorithm was derived is given in algorithm 4.

```
NumMoms[as_:{0.25, 0.75}, λs_:{0.5, 0.5}, μs_:{1.5, 1.5}, mom_:4] :=
 Block[{}, K = Length[as]; Λ = Plus @@ λs;
  inc[li_, pos_, inc_:1] := MapIndexed[(If[First[#2] == pos, #1 + inc, #1]) &, li];
  unit[i_] := inc[ConstantArray[0, K], i];
  (λ# = λs[[#]]) & /@ Range[K];
  (μ# = μs[[#]]) & /@ Range[K];
  (α# = as[[#]]) & /@ Range[K];
  (p# = λs[[#]] / Λ) & /@ Range[K];
  ps = λs / Λ;
  zs = Array[Subscript[z, #] &, K];
  PD[li_] := (Composition @@ (Function[x, D[x, {z#, li[[#]]}]] & /@ Range[K]));
  R = r @@ zs;
  Q = q @@ zs;
  (T# = t# @@ Join[{s}, zs]) & /@ Range[K];
  liststates[pop_] := Select[PadLeft[IntegerDigits[#, pop + 1], K] & /@ Range[0, (pop + 1)^K - 1],
    (pop == Plus @@ #) &];
liststates[0] = {PadLeft[{0}, K]};
  Rlhs = Sum[(μi αi (1 - zi) - Λ αi zi (1 - ps.zs)) PD[unit[i]][R], {i, 1, K}] - Λ (1 - ρ) (1 - ps.zs);
  Q = Sum[αi zi PD[unit[i]][R], {i, 1, K}] + 1 - ρ;
  (Tlhs# = μ# (Q - T#) - Sum[(αj - α#) ((s + λs.(1 - zs)) zj - μj (1 - zj)) PD[unit[j]][T#], {j, 1, K}] -
       (s + λs.(1 - zs)) T#) & /@ Range[K];
  cumsub0 = cumsubvars[0] = {};
  Do[(tsubn,-1,# = tcumsubn,m,# = tcumsubvars[n, m, #] = {}) & /@ Range[1, K], {n, 0, mom}, {m, -1, mom}];
  (Tcumsub-1,# = {}) & /@ Range[1, K];
Do[Reqs[i] = ((PD @ #) [Rlhs]) & /@ (liststates[i]) /. Thread[zs → 1] /. ρ → Λ Sum[pi - μi, {i, 1, K}] /.
      pK → 1 - Sum[pi, {i, 1, K - 1}];
subvars[i] = Select[((PD @ #) [R]) & /@ (liststates[i]) /. Thread[zs → 1], (Not[MemberQ[cumsubvars[i - 1], #]]) &];
    cumsubvars[i] = cumsubvars[i - 1] ~Join~ subvars[i];
rsubi = Solve[Thread[Reqs[i] == 0] /. cumsubi-1 // Simplify, subvars[i]] // First;
    cumsubi = cumsubi-1 ~Join~ rsubi, {i, 1, mom + 1}]
Do[tsubvars[n, m, j] = Select[(D[(PD @ #) [Tj], {s, n}]) & /@ (liststates[m]) /. Thread[zs → 1] /. s → 0,
      (Not[MemberQ[tcumsubvars[n, m - 1, j], #]]) &];
    tcumsubvars[n, m, j] = (tcumsubvars[n, m - 1, j] ~Join~ tsubvars[n, m, j]), {j, 1, K}, {m, 0, mom},
    {n, 0, mom}];
Do[
   Teqs[n, m, j] = (D[(PD @ #) [Tlhsj], {s, n}]) & /@ (liststates[m]) /. Thread[zs → 1] /. s → 0 /. cumsubn+1 /.
      ρ → Λ Sum[pi - μi, {i, 1, K}] /. pK → 1 - Sum[pi, {i, 1, K - 1}];
   tsubn,m,j = Solve[Thread[Teqs[n, m, j] == 0] /. tsubn,m-1,j, tsubvars[n, m, j]] // Flatten;
   tcumsubn,m,j = tcumsubn,m-1,j ~Join~ tsubn,m,j, {j, 1, K}, {m, 0, mom}, {n, 0, mom}];
{-Together[t1^(1,0,0)[0, 1, 1] // (Composition @@ (Function[x, (x /. tcumsub#,mom,1)] & /@ Range[0, 1]))],
   Together[t1^(2,0,0)[0, 1, 1] // (Composition @@ (Function[x, (x /. tcumsub#,mom,1)] & /@ Range[0, 2]))],
   -Together[t1^(3,0,0)[0, 1, 1] // (Composition @@ (Function[x, (x /. tcumsub#,mom,1)] & /@ Range[0, 3]))],
   Together[t1^(4,0,0)[0, 1, 1] // (Composition @@ (Function[x, (x /. tcumsub#,mom,1)] & /@ Range[0, mom]))]}
 ]
```

Figure A.1: Mathematica code for response time moments in M/M/1-DPS queues [8].

A.3 Parametrisation of MMPP(4)/M/1-DPS queue

For the MMPP(4)/M/1-DPS model, the class 1 generator matrix has non-cyclic rates given by: $q_{13}^{(1)} = 1713.3$, $q_{24}^{(1)} = 26368.9$, $q_{31}^{(1)} = 1693.7$, $q_{32}^{(1)} = 19.6$, $q_{41}^{(1)} = 2573.7$, $q_{42}^{(1)} = 23796.3$, and $q_{12}^{(1)} = q_{14}^{(1)} = q_{21}^{(1)} = q_{23}^{(1)} = q_{34}^{(1)} = q_{43}^{(1)} = 0$. Similarly, the class 2 generator matrix rates are: $q_{12}^{(2)} = q_{34}^{(2)} = 0.04$, $q_{13}^{(2)} = 36.5$, $q_{14}^{(2)} = 0.025$, $q_{21}^{(2)} = q_{43}^{(2)} = 5.62$, $q_{23}^{(2)} = 3.51$, $q_{24}^{(2)} = 1338.3$, $q_{31}^{(2)} = 36.45$, $q_{32}^{(2)} = 0.063$, $q_{41}^{(2)} = 9.0$, and $q_{42}^{(2)} = 1333$. These MMPP rates

relate directly to state transition probabilities $p_i^{(s)}$ for $i = 1, \ldots, 4$, and $s = 1, 2$, which are defined as:

$$p_i^{(s)} = \frac{\sum_{j=1, i \neq j}^{4} q_{ji}^{(s)}}{\sum_{j=1, i \neq j}^{4} q_{ji}^{(s)} + \sum_{j=1, i \neq j}^{4} q_{ij}^{(s)}} \tag{A.4}$$

Further, it is possible to extend this technique beyond four MMPP phases. Typically, four phases are sufficient to capture the burstiness of the job arrival process whilst maintaining the computational complexity of training the model relatively low.

A.4 MAP-fitting with KPC-toolbox

We summarise the steps for fitting a MAP with the KPC-toolbox as follows:

1. Pre-process the timestamped packet traces collected at routers to obtain traces of packet inter-arrival times (in seconds). Then, calculate moments, autocorrelations and bicovariances from the traces using the KPC-toolbox.

2. Perform order selection for the MAP during pre-fitting to select the number of states. Using trace descriptors, run KPC-based fitting that minimises an objective cost function whilst matching autocorrelation.

3. Output the matrices D_0 and D_1. Compare original and MAP-fitted results of moments (up to three) and autocorrelation (for ten different lags). If matrix D_1 is diagonal, transform MAP(D_0, D_1) into an MMPP with the same order.