Effective task assignment strategies for distributed systems under highly variable workloads

A thesis submitted for the degree of
Doctor of Philosophy

James Andrew Broberg B.App.Sc. (Hons.),
School of Computer Science and Information Technology,
Science, Engineering, and Technology Portfolio,
RMIT University,
Melbourne, Victoria, Australia.

9th November, 2006

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged.

James Andrew Broberg
School of Computer Science and Information Technology
RMIT University
9th November, 2006

Acknowledgments

I would like to extend my sincere thanks to my two supervisors, Professor Zahir Tari, and Professor Panlop Zeephongsekul, for their support and guidance over the period of my candidature.

I wish to thank the School of Computer Science and Information Technology at RMIT University for their material support (via scholarships and resources), providing an excellent work environment for me to pursue my research interests.

I would like to thank all the staff and research students in the Distributed Systems and Networking (DSN) discipline (formerly the eCDS Activity Unit) for providing a peer group where support and feedback are readily available.

I would like to thank my fellow PhD students (past and present), in particular Yaniv Bernstein, Peter Dimopolous, Nalaka Gooneratne, Kwong Lai, Panu Phinjaroenphan, Kazi Sakib and Abhinav Vora.

I wish to extend my sincere thanks and gratitude to Cathy Xia, Li Zhang and Zhen Liu for acting as my mentors during my internship at IBM T.J. Watson Research Center in 2005. The experience was extremely valuable and fruitful.

Finally, I wish to thank my partner, Amy McDowell, and my family, for their patience, support and good humour during this period.

Credits

Portions of the material in this thesis have previously appeared or are accepted to appear in the following publications:

- James Broberg, Panlop Zeephongsekul, Zahir Tari, Approximating General service distributions as Bounded Hyper-exponential, IEEE International Symposium on Computers and Communication (ISCC), To appear, June 2007 [Chapter 6]
- J. Broberg, Z. Tari and P. Zeephongsekul, Task assignment with work-conserving migration, Parallel Computing, Volume 32, Issues 11-12, Performance Evaluation of Communication Networks for Parallel and Distributed Systems, December 2006, Pages 808-830. ISSN: 0167-8191 [Chapter 5]
- Anthony Bedford, Panlop Zeephongsekul, James Broberg, Peter Dimopoulos, Zahir Tari, Queueing theory applications to communication systems: control of traffic flows and load balancing, Springer Handbook of Engineering Statistics, Hoang Pham (Ed.), ISBN: 1-85233-806-7, July 2006 [Chapters 2 and 3]
- Zahir Tari, James Broberg, Albert Y. Zomaya, Roberto Baldoni, A Least Flow-Time First Load Sharing Approach for Distributed Server, Journal of Parallel and Distributed Computing (JPDC), 832-842, Volume 65, Issue 7, ISSN: 0743-7315, July 2005 [Chapters 2 and 3]
- James Broberg, Zahir Tari, Panlop Zeephongsekul, Task Assignment based on Prioritising Traffic Flows, Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS 2004), 415-430, LNCS 3544, ISBN: 3-540-27324-7, June 2005 [Chapter 4]
- James Broberg, Zahir Tari, Panlop Zeephongsekul, Task Assignment based on Prioritising Traffic Flows, RMIT School of CS IT Technical Reports for 2004, TR-04-5, May 2004 [Chapter 4]
- James Broberg, Zahir Tari, Task Assignment based on Estimating the Service Requirement, RMIT School of CS IT Technical Reports for 2004, TR-04-1, February 2004 [Chapters 2 and 3]

- Bin Fu, James Broberg, Zahir Tari, Task Assignment for Overloaded Systems., IEEE International Symposium on Computers and Communication (ISCC), 1119-1125, June 2003 [Chapters 2 and 3]

This work was supported by the Distributed Systems and Networking (DSN) group at RMIT University, and ARC Discovery Grant no. DP0346545 awarded by the Australian Research Council (ARC) for 2003-2005.

The thesis was typeset using the LaTeX 2_ε document preparation system.

Contents

List of Figures

List of Tables

Abstract

The demand on networked computing resources is constantly growing for many application domains. Users of batch and scientific computing clusters are putting increasing demand on these resources as they endeavour to analyse and solve larger and more complex problems in diverse areas like Engineering, Life Sciences and Aerospace. Such problems can have highly variable demands with regard to the computing resources they require. With respect to so-called *Internet* applications, users are placing more demand on web servers and clusters that provide these services, as we satisfy more of our daily needs on-line. These services include (but are not limited to) e-commerce, online banking, social networking and acquisition and sharing of different multimedia file formats (such as photos, documents, music and movies). The demand for these services (and the resources required) can be transient at times, with frequent occurrences of 'flash crowds' where demand spikes unexpectedly. The demand they put on computing resources is also highly variable, placing unique requirements on system designers wanting to provide adequate service to as many customers as possible.

This thesis is concerned with analysing and improving the performance of distributed systems under highly variable workloads. We focus on improving the performance of distributed systems by creating task assignment policies that address the needs of modern computing workloads. Much of the past research in the area of task assignment (or scheduling) assumes that the workload is less variable, with the service distribution typically characterised by an exponential distribution.

Extensive recent studies have shown that modern computing workloads are highly variable, and are distributions that represent them are 'heavy-tailed'. Such distributions are characterised by numerous small tasks (or requests) with small service requirements, and few large tasks with disproportionally large service requirements. These characteristics make intelligent task assignment that enables effective utilisation of resources extremely challenging. In light of these findings we focus our efforts on the analysis, mathematical modelling and

improvement of task assignment policies under such highly variable workloads.

The first two issues that are considered involve creating more effective task assignment policies under two specific application domains; batch computing, and web serving clusters. Two policies (TAPTF and TAPTF-WC) are devised that endeavour to maximise the performance and utilisation of a distributed system in their respective application domains, by intelligently addressing the negative effect on performance that highly variable workloads cause.

We then consider some of the problems associated with the modelling and analysis of queueing systems that incorporate General service distributions. This is addressed by approximating such distributions as Hyper-exponential. We also re-computed our TAPTF model to utilise a Hyper-exponential (or Bounded Hyper-exponential) service distribution. This allows the TAPTF model to be used with nearly any General service distribution (e.g. Pareto, Bounded Pareto, Log-normal, Weibull), simply by first approximating it as Hyper-exponential or Bounded Hyper-exponential. As a result, TAPTF can be utilised under a wider range of potential workloads.

Finally, the issue of obtaining advanced queueing metrics is examined. Many queueing metrics (such as the variance of key measurements such as waiting time and slowdown) are difficult to obtain analytically. As such, we investigate techniques to obtain these metrics via simulation. A simulation framework to record important queueing metrics is presented, allowing us to measure such metrics that were too difficult to compute via analytical means. Several issues regarding simulating highly variable workloads are identified, and the variance of key metrics such as expected waiting time and slowdown are obtained.

Chapter 1

Introduction

Traditionally, distributed computing environments that serviced high-demand networked applications such as scientific computing and high-volume web sites relied on a singular (and typically very powerful) machine. These ranged from high-powered servers and mainframes up to multi-CPU 'super-computers'. Typically, these machines were significantly more expensive and powerful than commodity PCs. When the resource limit of these machines are reached and they can no longer satisfy the demand for the service they provide, they must be upgraded (often at great expense) or replaced outright with a faster machine.

With personal computers becoming more affordable and more powerful, the price/performance benefits of a traditional single-server approach have rapidly diminished. The dedicated server hardware of today is no longer significantly more powerful that a commodity PC. As such, the usage of a 'cluster' of commodity computers has become more prevalent in recent times. Such clusters are popular due to their scalable and cost effective nature - often providing more computing resources at a significantly lower cost compared to traditional mainframes or 'super-computers'. They also provide other benefits, such as redundancy and increased reliability. The applications of such systems include super-computing clusters [Schroeder and Harchol-Balter, 2004], so-called web 'farms' and content delivery networks (CDN's) serving high profile and high volume web-sites [Dilley et al., 2002], among other applications.

Figure 1.1 illustrates a common cluster configuration. In this configuration, there are a number of hosts waiting to service a user task or 'job'. This could be a request for a file on a web page, or a complex computation to be performed. These tasks arrive at a central dispatcher, and are dispatched to hosts according to a *task assignment policy*. This policy assigns tasks to hosts subject to a specific set of rules, with a typically goal being to maximise

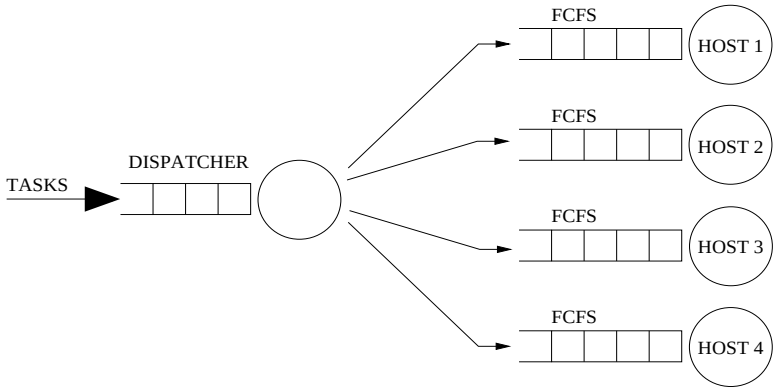


Figure 1.1: Distributed Server Model.

performance. How one measures performance varies depending on the application domain. When a task arrives at the dispatcher, it is placed in a queue, waiting to be dispatched in first-come-first-served (FCFS) order. When it is directed to a particular host it is then placed in a queue there, waiting to be served in a FCFS manner.

The choice of which task assignment policy to utilise can significantly effect the perceived performance and server throughput. A poorly chosen policy could assign tasks to already overloaded servers, while leaving other servers idle, drastically reducing the performance of the distributed system. One major aim of a task assignment policy is to distributed tasks such that all available system resources are utilised. However, the ideal choice of task assignment policy still an open question for many contexts.

Making this choice even more difficult is the highly variable nature of modern distributed computing workloads. Numerous recent studies have shown that these environments exhibit a wide range of task sizes, often spanning many orders of magnitude. These so-called 'heavy-tailed' workloads have been found to exist in a number of distributed computing environments - observed in transmission duration's of files and file sizes stored on servers [Crovella et al., 1998b; Crovella and Bestavros, 1997; Downey, 2001], as well as unix process lifetimes [Harchol-Balter and Downey, 1997] on servers. Such workloads are characterised by many small tasks and, with lower probability, disproportionally large (and disruptive) tasks. Often, 1% of tasks make up 90% of the workload. These conditions demand new task assignment techniques to improve performance in distributed systems, as most classical approaches to task assignment are based on the assumption of less variable, exponential workloads. Task assignment policies based on such an assumptions cannot effectively handle highly variable

workloads, as the performance of these policies is directly proportional to the variability of the workload. Policies based on an exponential workload assumption are not designed to reduce the variance of the task size distribution in any way.

We measure the performance of a given task assignment policy via a number of different metrics. System designers may place importance on one metric over others depending on the application domain. These metrics include:

- Mean Waiting Time - Waiting time is measured from the point when a task enters the system until it begins being serviced. This is the most commonly used metric.
- Mean Flow Time - Flow time refers to the waiting time plus the service time. This is a measure of the entire time a task spends in the system, from entering to leaving.
- Mean Slowdown - The slowdown refers to the waiting time divided by the service time (e.g. the size of the task). This metric attempts to capture the notion that a task's waiting time should be commensurate with its size. That is, a small task should only wait a short time to be serviced, whereas a larger task can absorb a larger waiting time while still maintaining good slowdown.

As well as considering mean metrics, we are also concerned with the integrity of these metrics. How well do these metrics represent the experience of the majority of tasks? As such, we also wish to consider the variance of each of the above three metrics.

1.1 Research Questions

There has been extensive research regarding task assignment (or scheduling) policies in the last 40 years. In the past much of this research has been focused on less variable, exponential workloads. In recent years more attention has been focused on distributed computing environments that experience highly variable, 'heavy-tailed' workloads. Task assignment policies have been devised that attempt to address the negative characteristics of these heavy-tailed workloads in order to maintain good performance. Nonetheless, it remains a relatively new area of research and there are still significant improvements that must be achieved for modern distributed computing services in order to provide acceptable performance. As such, we focus on the modelling and improvement of task assignment policies under realistic conditions of these highly variable workloads. The first two research questions relate to the creation of improved task assignment policies for two specific application domains under highly variable,

heavy-tailed workloads. It is critical to have an effective and appropriate task assignment policy to maximise the performance and utilisation of a distributed system. Advanced analytical modelling of such task assignment policies under such workloads can be difficult to perform, and even impossible in some circumstances. The next research question pertains to utilising techniques aimed at simplifying the modelling of task assignment policies under such highly-variable environments. The final question explores the methodology needed to perform more advanced analysis of these task assignment policies.

A) How can we improve task assignment policies for batch computing environments?

Scientific and batch computing have specific and unique requirements that differ from other applications. Tasks are often CPU bound and have extremely high memory requirements that precludes the use of work-conserving migration [Harchol-Balter, 2002; Milojicic; et al., 2000]. We cannot simply rely on existing task assignment policies (based on an assumption of less variable workloads) if we wish to provide acceptable performance in such systems. Task assignment policies must be devised that improve the performance by specifically dealing with the unique characteristics of highly variable workloads. Task migration can be utilised where appropriate (as tasks are not interactive), but work-conserving migration is often unfeasible due to the enormous memory requirements of some tasks.

B) How can we improve task assignment policies for interactive computing environments?

Policies created for batch computing are not suited to the interactive nature of serving web content. Tasks can be CPU bound (e.g. a CGI script) or network bound (e.g. a large file request), or both. Task assignment policies that deal with the negative effect of highly variable workloads while ensuring not to unduly delay or interrupt the processing of tasks are required. Task migration is available to use if beneficial, and should be work-conserving (i.e. no work is lost). Under these constraints, we need to devise a task assignment policy that is specifically suited to these application domains in order to maximise the performance and improve the experience of end-users.

C) How can we simplify the analysis of distributed systems under highly-variable workloads?

The general probability distributions used to characterise the highly-variable (or 'heavy-tailed') workloads experienced by distributed systems can make queueing theory analysis difficult, or even impossible in some circumstances. Ultimately, this can prevent us from gaining a greater insight to the behaviour of task assignment policies, restricting the view of system performance to mean metrics only. This can hide many subtleties in the behaviour of the system, such as the variance in performance experienced by tasks, and makes enforcing specific quality of service targets very difficult. Such analysis is often trivial for queueing systems that utilise exponential service distributions. As such, it would be ideal to model the highly variable workloads we are concerned with as a sum of weighted exponential distributions (known as 'hyper-exponential'), whilst still maintaining the important characteristics of the original 'heavy-tailed' distribution. We are especially concerned with the accurate modelling of *bounded* service distributions, which are commonly used in modern queueing system analysis.

D) How can we obtain advanced performance metrics of task assignment policies?

We need to model the important characteristics of waiting time and slowdown (such as the variance) to fully understand the experience of tasks in the distributed system. Such a model can be useful if we wish to eventually provide Quality of Service (QoS) guarantees to customers. Unfortunately, these performance metrics cannot always be computed analytically, even if the problem is grossly simplified. However we can endeavour to obtain these metrics via extensive and careful simulation, giving us greater insight into the experience of the majority of tasks in the queueing system.

1.2 Limitations of existing solutions

In this section we briefly highlight the limitations of existing solutions in addressing the requirements of the research questions posed in Section 1.1. A more extensive evaluation of these existing solutions is presented in Chapter 2.

1.2.1 Task assignment policies for batch computing environments

The issue of task assignment in a distributed system has been the subject of extensive research. Much of the classical research in this area has focused on the assumption that the workload does not show much variation, and is typically exponential in nature. Policies such as Random, Round Robin and certain dynamic policies such as Least Loaded First (LLF) and Shortest Queue First (SQF) were formulated based on these assumptions. However, when faced with highly variable workloads these policies perform poorly [Harchol-Balter et al., 1999], with the variation having a negative effect on all performance metrics. More recently, size-based policies have been formulated to directly address the negative effects of highly variable workloads by grouping like-sized tasks together (creating a less variable experience for them) [Harchol-Balter et al., 2003c; Crovella et al., 1998a; Schroeder and Harchol-Balter, 2004]. However many of these size-based policies assume that task sizes are known *a priori* at the dispatcher, which is often not the case. The TAGS [Harchol-Balter, 2002] policy is unique in that it makes no such assumption of prior knowledge of a task's size. TAGS shows good performance under highly variable workloads but generates significant wasted processing (from migrating many tasks without conserving any prior processing done) and often leaves servers idle. It also loses its effectiveness as the workloads become moderately less variable, and the system load increases.

1.2.2 Task assignment policies for interactive computing environments

Interactive computing environments such as high-volume web serving clusters face unprecedented workloads that can be highly variable and bursty in nature. Incidents of 'flash-crowds' often occur due to surges in interest for products, sporting events or breaking news [Arlitt and Jin, 2000; Iyengar et al., 1999]. Surprisingly, many web serving platforms still depend on traditional policies (e.g. Random or Weighted Random) and rudimentary metrics (e.g. Queue Length, number of TCP connections). Due to the real-time nature of such application domains these techniques are attractive as they can be computed efficiently. Despite this, these policies were not designed to effectively handle highly variable workloads, and as such are not ideally suited to these conditions. Significant improvement could be gained if we apply modern queueing theory analysis (based on realistic assumptions of heavy-tailed workloads) to this application domain. However, we cannot simply apply task assignment policies that are suited to batch and scientific computing to this application domain, as they have different assumptions and user requirements.

1.2.3 Analysis of distributed systems under highly-variable workloads

Highly-variable workloads have been observed in many computing environments, from batch computing clusters, Internet (ftp and http) and LAN traffic [Crovella et al., 1998b; Crovella and Bestavros, 1997]. These are often characterised mathematically by heavy-tailed or Power-Law distributions like Pareto or Bounded Pareto. However, these distributions can make certain advanced $M/G/1$ queueing analysis difficult, and even impossible in some circumstances. Exponential approximations of heavy-tailed workloads could potentially be used in such circumstances [Feldmann and Whitt, 1997], but their accuracy needs to be rigorously verified to ensure they still possess the characteristics of the original highly variable Pareto workload models before we can utilise them with confidence in our task assignment models. In particular, we wish to ensure these techniques can accurately approximate *bounded* service distributions, which are commonly used in queueing analysis and are used extensively in this thesis.

1.2.4 Advanced performance metrics of task assignment policies

It is important to understand all of the performance metrics of task assignment policies - including higher-order measures such as variance. These are necessary if these approaches are to be deployed, to be able to develop Quality of Service bound for customers, especially in commercial environments where this knowledge is essential. Deriving these metrics analytically can be challenging, and even impossible in many circumstances due to the mathematical difficulties involved when computing advanced queueing metrics that utilise a Pareto service distribution. Evidently, these metrics must be obtained via other means (such as simulation).

1.3 Contribution

In response to the research questions posed in Section 1.1, the following contributions are made:

1.3.1 Task assignment based on prioritising traffic flows

We present a task assignment policy, Task Assignment based on Prioritising Traffic Flows (TAPTF) [Broberg et al., 2005], designed specifically to address the performance issues caused by highly variable workloads in batch and scientific computing facilities. TAPTF is a size-based partitioning approach that utilises dual queues at each host, and attempts to group

like-sized tasks together to maximise performance and minimise the variance in waiting time experienced by tasks. Each host has specific size ranges assigned to them, and only service tasks within those size ranges. Tasks that exceed these ranges avoid unduly delaying other tasks by being migrated to another host once they exceed these size ranges. However, any work done on a task is abandoned if the task is migrated. TAPTF addresses several limitations of existing approaches in this problem domain. It is designed specifically to deal with the negative impact of highly variable workloads. It does not assume any *a priori* knowledge of a task's size at the dispatcher, unlike many recent size-based approaches. TAPTF also minimises the wasted processing that affects policies like TAGS, reducing the number of non work-conserving migrations that occur, and improving utilisation at all hosts.

1.3.2 Task assignment with work conserving migration

Interactive application domains such as high-volume web serving have different requirements to batch and scientific application domains. For instance, it is not reasonable from a user's perspective to migrate a task while it is being serviced, starting it from scratch at another host. As such we introduce Task Assignment with Work Conserving Migration (TAPTF-WC) [Broberg et al., 2006]. TAPTF-WC has all the features of TAPTF to mitigate the effects of highly variable workloads, while also providing work-conserving migration, making it suitable for more interactive applications. So, unlike traditional approaches that are still frequently used in this application domain, TAPTF-WC is specifically designed to handle the highly variable workloads that are now commonly experienced.

1.3.3 Approximating General service Distributions

Prony's method [Feldmann and Whitt, 1997] is a technique used to approximate a general distribution (such as Pareto or Log-normal) as a sum of exponential distributions, known as a Hyper-exponential. Exponential distributions are highly desirable to use in queueing theory analysis due to their tractable nature, with transforms and higher moments easy to compute. Bounded Pareto distributions are commonly used to accurately describe highly variable workloads, but how suitable is an unbounded Hyper-Exponential approximation of such a distribution? To address this issue we introduce a new class of distribution, the Bounded Hyper-exponential, and update Prony's method to fit a Bounded Pareto directly to a Bounded Hyper-exponential distribution. We find that the Bounded Hyper-exponential is much better suited to fitting heavy-tailed workloads, where the original distribution itself is

bounded. The resulting fit is significantly more accurate, with certain specific mathematical properties of interest matching perfectly. The accuracy of this approximation is verified by integrating the Hyper-exponential and Bounded Hyper-exponential service distributions into our original TAPTF model, and comparing queueing metrics. This also improves the utility and potential application of the TAPTF model, by allowing it be utilised with a wider range of General service distributions (Pareto, Bounded Pareto, Log-normal, Weibull, etc.).

1.3.4 Obtaining advanced performance metrics via simulation

Extensive simulation is performed, comparing one of our proposed approaches, TAPTF, against existing task assignment policies for validation purposes. These simulations are important for two key reasons. First, we wish to contrast our simulation results against the analytical results obtained previously, for both the proposed approaches and existing techniques. Second, as obtaining the variance of waiting time and slowdown analytically is extremely difficult we measure it via simulation to understand the behaviour of tasks in our system. Mean metrics are measured (for waiting time, flow time and slowdown), as well as the variance of each metric. We measure global, per host and per queue metrics. These advanced metrics give us a more detailed picture of the behaviour of tasks in the system. and can ultimately provide us with approximate bounds on performance. Ultimately these simulations will provide further insight into the problem of task assignment under highly variable workloads.

1.4 Thesis Structure

The remainder of the thesis is organised as follows:

- Chapter 2 provides the necessary background into the core concepts of this thesis. This includes background on queueing theory analysis of distributed systems, important characteristics of 'heavy-tailed' workloads and a description of the different metrics used to measure performance.

- Chapter 3 analyses the related work in task assignment policies, specifically focuses on their performance under highly variable workloads. Classical policies such as Random and Round Robin are examined, as well as more recent dynamic and size-based policies. The strengths and weaknesses of these policies in dealing with highly-variable workloads are highlighted.

- In Chapter 4 we present an improved task assignment policy for high variable batch computing workloads. The policy, Task Assignment based on Prioritising Traffic Flows (TAPTF), is validated by a rigorous mathematical model based on the fundamentals of queueing theory, and an analytical comparison is presented, comparing TAPTF against classical, dynamic and size-based task assignment policies. TAPTF shows significant improvements in performance under a variety of workload conditions over existing policies.

- In Chapter 5 we present an improved task assignment policy for highly variable interactive computing workloads. The Task Assignment with Work Conserving Migration (TAPTF-WC) policy is designed specifically for applications were work-conserving migration is feasible (and has negligible cost). TAPTF-WC is supported by a extensive mathematical model, and analytical comparisons are presented, comparing TAPTF-WC against classical and modified size-based approaches (that support work-conserving migration). TAPTF-WC shows improved performance over existing policies under a wide range of workload conditions.

- In Chapter 6 we investigate the effectiveness of Prony's method [Feldmann and Whitt, 1997] in approximating a Bounded Pareto service distribution (frequently used to describe distributed computing workloads) by a weighted sum of exponential distributions (known as a Hyper-exponential). Hyper-exponential distributions can be used to simplify queueing theory analysis of distributed systems, and often can enable the computation of more advanced queueing metrics, providing greater insight into the operation of our system. We modify Prony's method to fit directly to a new class of distribution, known as a Bounded Hyper-exponential, to provide even more accurate results. Furthermore, we re-evaluate the TAPTF model to handle Hyper-exponential and Bounded Hyper-exponential service distributions. Our modified Prony's method is validated by substituting the unbounded and bounded Hyper-exponential workload approximations into our updated TAPTF model, and comparing the results. Our updated TAPTF models also widen the application of TAPTF to nearly any long-tailed General service distribution (via approximating it using the original or modified Prony's method). We also highlight some of the desirable properties of Hyper-exponential and Bounded Hyper-exponential distributions that make them so amenable to analysis in queueing systems.

- In Chapter 7 we obtain further performance bounds via simulation. Through extensive simulation we obtain measurements of the mean and variance of important queueing metrics (to validate against analytical models), as well as obtaining per host and per queue performance metrics exhibited by TAPTF and other policies.
- Finally, in Chapter 8 we summarise the main contributions of this thesis, and suggest areas of future research that could be investigated.

Chapter 2

Background

In this chapter we cover the required background knowledge in queueing theory needed to understand the various models presented in the later chapters, as well as reviewing recent workload measurements to support the remainder of the thesis. In Section 2.1 we provide a brief introduction into the fundamentals of queueing theory, explaining key concepts and terminology. Section 2.2 describes the metrics that are commonly used to measure the performance of queueing systems, as well as how they are computed for the type of queueing systems we are interested in. Section 2.3 discusses some recent studies that analyse the characteristics of computing workloads, and the implications of those findings with regards to the design and operation of distributed systems are examined in Section 2.3.1.

2.1 A Queueing Theory Primer

Queueing is a part of every day life, and can be found occurring in computer and telephone networks, checkout counters at supermarkets or even waiting in traffic at an intersection. In order to understand the behaviour of such queues (as well as the experience of the entities residing in them) some kind of predictive model would be extremely valuable. Analytical models based on Queueing Theory can often very accurately model the mechanics of the scenarios described above. Queueing theory provides a stochastic and probabilistic approach to studying the operation and behaviour of queues. There have been several seminal books published on Queueing Theory - with the first key work written by Saaty [1961]. Saaty lists over 900 papers in his bibliography, indicating that the field was already well established by this point. Other key works include Kleinrock's two volumes on Queueing Theory [Kleinrock, 1975a;b] which are still utilised to this day in generating new queueing analysis. Countless

other more recent texts continue to cover and expand on this increasingly important field of study [Gross and Harris, 1998; Stallings, 2002; Ross, 2002].

2.1.1 Queue Characteristics

The characteristics of a queueing system are commonly described using Kendall's notation [Kendall, 1953] in the form $(A/B/C)$, where:

- A is the arrival pattern of customers;
- B is the service pattern of customers;
- C is the number of servers;

In more recent years Kendall's notation has been expanded to $(A/B/C/D/E)$, with the notation now including the following parameters:

- D is the system capacity or buffer size;
- E is the service discipline.

Despite these additions it is most common to see queueing systems described in the original Kendall notation, $(A/B/C)$.

Arrival Pattern

This refers to the distribution of the arrivals of tasks into a system. Some commonly measured distributions are:

$$\begin{aligned} \text{Exponential M:} f(t) &= \lambda e^{-\lambda t}, \ t \geq 0, \ \lambda > 0. \\ \text{Deterministic} \quad D & \\ \text{Erlangian with k stages } E_k : f_k(t) &= \frac{\lambda(\lambda t)^{k-1} e^{-\lambda t}}{(k-1)!}, \ t \geq 0, k = 1, 2, \ldots, \infty \end{aligned}$$

When the exact distribution is unspecified, it is described as G or GI, 'general' or 'general and independent'. The symbol M is used where the distribution is exponential, referring to the fact that in such cases the arrival pattern has the *Markov* property. This is a desirable property to have in a queueing system as Markovian queueing systems are analytically more tractable than other types of queues (e.g. General). Specifically, the Laplace transforms of such distributions are trivial to compute.

Service Pattern

This describes the service distribution. In modern communication analysis, this is often interchangeably referred to as the task size, job or workload distribution. It could refer to the distribution of the size of files being requested on a web server or even the distribution of CPU time required by tasks in a batch computing facility. The most commonly used distributions that represent the service pattern are the same as described for the arrival pattern. As with the arrival pattern, exponentially distributed service patterns are better than General distributions from a tractability standpoint. However, many recent measurements of modern communication workloads (such as web requests and unix process lifetimes) show that 'heavy-tailed' General distributions (such as Pareto) are a more accurate representation of the service pattern. The service pattern has a critical effect on the performance of a queueing system, as we will discover in Chapter 3.

Number of Servers

This parameter describes the number of entities that provide service to customers in the queueing system. In modern communication systems this service could be CPU time (e.g. executing a CGI script or other process), network bandwidth (e.g. a file server serving a file to a customer) or some other resource of interest (and combinations thereof). These servers could be homogeneous or heterogeneous.

System Capacity

Classical Queueing Theory assumes that there are no restrictions on the number of customers in the queueing system at a given time - that is, the queues are unbounded. Communication networks (such as those existing on the web) typically have restrictions on the number of customers waiting in a queue for service at a given time, both at the Operating System level and the Application level, where requests are ignored or 'dropped' once these limits are reached. If the queues are unbounded in a given queueing system and tasks arrived faster than could be serviced, then the queue lengths would constantly increase, approaching infinity.

Service Disciplines

Once customers are in a queue, we must decide on which ones to service. The rules that govern this choice are called *service disciplines*. They are also often referred to as scheduling

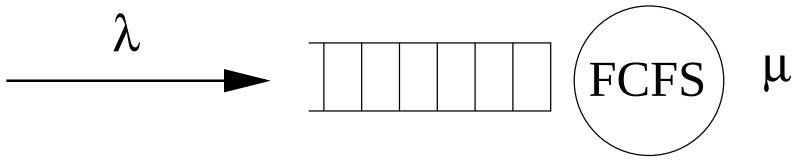


Figure 2.1: Single Server Queue

policies in the literature. The most common service discipline is First-Come-First (FCFS) (also known as First-In-First-Out, FIFO). As the name suggests, customers are serviced in order of their arrival. Intuitively this would seem the fairest and most logical choice, but other service disciplines are frequently utilised in queueing systems. A Last-In-First-Out (LIFO) service discipline services the last customer to arrive in a queue. Shortest-Remaining-Processing-Time (SRPT) services the customer with the shortest expected service time in the queue first (assuming that the service time is known in advance). Priority queueing disciplines are also commonly used, where a server processes customers in a queue in order of their priority or class. These service disciplines (and others) are explained in more detail in Section 3.1.

2.1.2 Single Server Queue

The simplest example of a queueing system is a single server queue. Figure 2.1 shows tasks arriving at a rate λ into a queue, waiting for service. The server will service tasks in a FCFS order, at a rate μ. Let us assume that the arrival pattern follows a *Poisson* [Cao et al., 2002] distribution and as such is Markovian, and that the queue length is unbounded. If the service distribution was exponential, we would classify this system using Kendall's notation as $M/M/1/\infty/FCFS$, or simply $M/M/1$. If the service distribution was a General Distribution (such as Pareto), we would describe it as $M/G/1/\infty/FCFS$, or $M/G/1$ for short.

2.2 Queueing Metrics

The cluster configuration depicted in Figure 1.1 (Chapter 1) is well suited to analysis via queueing theory. Armed with some basic knowledge about our system of interest, such as the arrival rate, λ, and distribution of service requirements, $f(x)$, we can obtain the expected performance metrics of the system. With these metrics, we can evaluate the performance of

different task assignment policies, and make an informed judgement regarding which policy is best to employ.

As such, some common metrics are used in order to directly compare the relative performance of the various task assignment policies. We consider the *mean waiting time*, *mean flow time*, and the *mean slowdown* of each task assignment policy. The waiting time refers to the time a task spent waiting in queues to be processed. The flow time is the sum of the waiting time and the service time. Slowdown refers to the waiting time divided by its processing time. We also consider second-order statistics such as the variance of each of our metrics to gauge the relative confidence in the observed means.

2.2.1 $M/G/1$ queues

Consider for a moment that each host in our basic distributed system (depicted in Figure 1.1) is a $M/G/1$/FCFS queue, where the arrival process into the system has rate λ. X represents the service time distribution, and ρ represents the utilisation ($\rho = \lambda E(X)$). W denotes a task's waiting time in the queue, F its flow time, S its slowdown, and Q is the queue length on arrival. Using a well known result in queueing theory [Kleinrock, 1975a], the expected values of these metrics are as follows:

$$\begin{aligned} E(W) &= \frac{\lambda E(X^2)}{2(1-\rho)} && \text{(Pollaczek-Khinchin formula)} \\ E(F) &= E(W) + E(X) \\ E(S) &= E(\frac{W}{X}) = E(W) \cdot E(X^{-1}) \\ E(Q) &= \lambda E(W) && \text{(Little's formula)} \end{aligned}$$

The system load (ρ) over an entire distributed system is defined as,

$$\rho = \frac{\lambda E(X)}{n}$$

where λ represents the outside task arrival rate into the system, $E(X)$ represents the mean task size and n represents the number of hosts in the system. When ρ is below 1, the system is stable and under-loaded. When ρ is greater than 1, the system is overloaded and more tasks enter the system than leave the system.

The expected waiting time, $E(W)$, is the most commonly used metric to describe the performance of a queueing system. Indeed, the main aim of many task assignment policies is

to minimise waiting time. We can see the expected waiting time is directly proportional to the second moment of the service distribution. The flow time metric $E(F)$ is also commonly used, describing the end-to-end time a task (or customer) spends in the queueing system. This is considered by some [Tari et al., 2005] to be a better representation of the experience of tasks in the queueing system. Slowdown ($E(S)$) is a more recently devised metric, that considers the notion of 'fairness'. Slowdown normalises the waiting time by considering the size (or service requirement) of a task. Common sense dictates that if a task has a small service requirement, then it should not have to wait an unduly long time to be serviced. Conversely, if a task has a large service requirement, they would expect their task to take a longer time, and thus can absorb a longer waiting time (proportional to their service time). Essentially we wish that a tasks waiting time be proportional to its service requirement.

2.3 Workload measurements of distributed systems

Numerous recent studies have shown that distributed computing environments exhibit a wide range of task sizes, often spanning many orders of magnitude. These so-called 'heavy-tailed' workloads have been found to exist in a number of computing environments. Researchers have found that a number of workloads measured on the World Wide Web exhibit heavy tails, including file requests by users, files transmitted via the network, transmission duration's of files and files stored on servers [Crovella et al., 1998b; Crovella and Bestavros, 1997]. Further examples of observed heavy-tailed workloads include the size of files stored in Unix file systems [Irlam, 1993], and the Unix process CPU requirements measured at UC Berkley [Harchol-Balter and Downey, 1997]. Motivated by such measurements, WWW traffic generating tools such as SURGE [Barford and Crovella, 1998] have been developed to more accurately 'stress-test' servers by generating realistic heavy-tailed traffic. More recently, traffic measurements of the 1998 World Cup [Arlitt and Jin, 2000] and the 1998 Winter Olympics [Iyengar et al., 1999] have exhibited some heavy-tailed characteristics.

E-commerce and e-payment workloads also present unique challenges to system designers as poor management of these workloads can directly correlate to lost income for these service providers. Indeed, prior research on workload characterisation of such sites found that the small portion of customers that exhibit the longest sessions also had the smallest buying ratio [Menasce et al., 1999]. That is, the longer a session took, the less likely the customer bought an item from an e-commerce site. Given that E-commerce sites are often heavily personalised and database driven, lengthy sessions can impose a burden on the

hosting site for little gain. Work by Neto et al. characterised web access patterns from a major US broadband provider for home and SOHO (Small office/home office) customers for commercial web services [Neto et al., 2004]. The durations of these sessions were found to be highly variable, follow lognormal and lognormal-pareto hybrid distributions for home and SOHO customers respectively. As more web content becomes dynamic and personalised in these systems, the server infrastructure required to satisfy these requests can increase dramatically [Arlitt et al., 2001]. Secure communication using HTTPS/SSL can also dominate the CPU on an E-commerce web server, increasing the computational cost over a non-SSL transaction by a factor of five to seven [Kant et al., 2000].

There are significant questions raised by these findings with regards to task assignment policies, as much of the existing work in the area was formulated under an assumption of a less variable, exponentially distributed workload. These findings have had a major impact on the way we traditionally design task assignment policies. In order to maximise the performance in modern communication systems that face these highly variable conditions, we must devise task assignment strategies that specifically deal with such 'heavy-tailed' workloads.

Description	α
Unix process CPU requirements [Harchol-Balter and Downey, 1997]	1.0
Sizes of files transmitted over the internet	$1.1 - 1.3$
[Crovella et al., 1998b; Crovella and Bestavros, 1997]	
1998 World Cup Web Site File Size Distribution [Iyengar et al., 1999]	1.37

Table 2.1: Some measurements of heavy-tailed traffic

2.3.1 Heavy-tailed properties

The so-called 'heavy-tailed' distributions have very high variance, where 1% of tasks can take 50% of the computing resources. They can be characterised by the function,

$$Pr\{X > x\} \sim x^{-\alpha},$$

where $0 \leq \alpha \leq 2$. The α parameter describes the variation of the distribution. Any set of tasks that is said to follow a heavy-tailed distribution is described as having the following properties [Harchol-Balter, 1999; 2002; Harchol-Balter et al., 1999]:

1. Decreasing failure rate. That is, the longer a task has run, the longer it is expected to continue running.

2. Infinite variance, and if $\alpha \leq 1$, infinite mean.

3. The property that a very small fraction (less than 1%) of the very largest jobs make up a large fraction (half) of the workload. This is commonly referred to as the *heavy-tailed property*. It is this property that makes the load very difficult to balance effectively.

For the purpose of analysis, we assume that the task sizes show some maximum (but large) value. This is a reasonable assumption in many cases, such as a web server, which would have some largest file. A *Bounded Pareto* distribution is therefore used, which has an lower and upper limit on the task size distribution. The probability density function for the Bounded Pareto $B(k, p, \alpha)$ is:

$$f(x) = \frac{\alpha k^\alpha}{1 - (k/p)^\alpha} x^{-\alpha-1}, \quad k \leq x \leq p \tag{2.1}$$

where α represents the task size variation, k is the smallest possible task, and p is the largest possible task. By varying the value of α we can observe distributions that exhibit moderate variability ($\alpha \approx 2$) to high variability ($\alpha \approx 1$). Typical measured values of the α parameter are between 0.9 - 1.3 [Crovella et al., 1998b; Crovella and Bestavros, 1997; Harchol-Balter, 1999], with an empirically measured mean of $\alpha \approx 1.1$.

Throughout this thesis we use the parameter α to capture the variance of the probability distribution of the service times. The direct relationship between this parameter and common representations of variability, such as the coefficient of variation, $C^2 = E\{X^2\}/E\{X\}^2$ and the second moment of the service distribution, $E\{X^2\}$, is demonstrated in Figures 3.1 and 3.2 respectively, and motivates its use for the remainder of the thesis. As such, we are confident that it is the most appropriate metric to quantify the variation in the workloads we examine in this thesis, and it has been used extensively in (similar) prior work by other researchers on workload modelling [Harchol-Balter and Downey, 1997; Crovella and Bestavros, 1997; Crovella et al., 1998b; Iyengar et al., 1999] and task assignment policies. [Harchol-Balter et al., 1999; Crovella et al., 1998a; Schroeder and Harchol-Balter, 2004; Harchol-Balter, 2002]. It allows us to generate a wide range of workloads with differing degrees of variation, approximating diverse workloads in a manner that is consistent with empirical measurements from supercomputing, batch, web (e.g. ftp/http) and commercial computing facilities.

2.3.2 Implications for designing Task Assignment Policies

Based on the workload measurements described in Section 2.3, and the properties of heavy-tailed workloads (described in Section 2.3.1) it is clear that task assignment policies for modern queueing systems need to mitigate the negative effects of heavy-tailed workloads if they are to be effective. Traditional task assignment policies were devised with the assumption of a less variable, exponential service distribution in mind. The characteristics of the exponential distribution are fundamentally different to that of heavy-tailed distributions, and task assignment policies based on the exponential assumption cannot expected to provide adequate performance under heavy-tailed workloads.

As such, the focus of remainder of this thesis is the effect of highly variable workloads on task assignment policies in distributed systems. These workloads consist mostly of small tasks, with extremely large tasks appearing with non-negligible and totally independent probability.

This should not be confused with autocorrelation, which is an entirely different phenomena to variability, with the former being the statistical measure of the relationship between a random event and itself at different time lags [Brockwell and Davis, 1986]. A positive autocorrelation can manifest itself as a large observation, which is consistently followed by another large observation, or a small observation consistently followed by another small observation at a given time lag. A negative correlation can be a large observation being followed by a small observation, or vice versa, at a given time lag. These type of observations are common in time-series style analysis, which is not utilised in this thesis.

For our purposes, when using $M/G/1$ queuing analysis the service distribution itself is stationary and memory-less. As such, there is no autocorrelation between task sizes at different time lags l.

We quantify the negative effects of heavy-tailed workloads in the next chapter. Additionally, the performance of classical task assignment policies as well as dynamic and size-based approaches will be analysed under heavy-tailed, highly variable workloads.

Chapter 3

Related Work

In this chapter we consider some of the important existing task assignment policies for distributed systems. We examine their strengths and weaknesses, particularly in dealing with realistic, highly variable workloads. In Section 3.1 we explore the different scheduling policies utilised in computing systems, and investigate the effect they have on common queueing metrics. Section 3.2 reviews the different techniques used to measure and disseminate load information on a computer system. We examine their effectiveness in being good load indicators and the importance of how frequently the information is disseminated to nodes in a distributed system. In Section 3.3 we examine the role of process migration in distributed systems, identifying how the mechanism works in modern distributed systems, and under what circumstances it is appropriate to use. Section 3.4 examines classical task assignment policies, including static policies such as Random and Round Robin, as well as Dynamic and Central-Queue policies. Section 3.5 considers more recently devised size-based policies such as the SITA (Size Interval Task Assignment) variants and TAGS (Task Assignment based on Guessing Size), illustrating how they address highly variable workloads, and highlighting areas where they can be improved. In Section 3.6 we compute some analytical queueing results of the task assignment policies discussed in this chapter under heavy-tailed, highly variable workloads, highlighting the negative effects these workloads have on performance.

3.1 Scheduling policies

As described in Section 2.1, the service (or scheduling) discipline dictates the manner and order in which tasks in a queue are serviced. Depending on the nature and characteristics of a computer system, only non pre-emptive scheduling (described in Section 3.1.1) may be

supported. This means once a task is in service, it cannot be interrupted (i.e. pre-empted) from service, rather it must run to completion. Alternatively, pre-emptive scheduling may be available (described in Section 3.1.2), where a task in service can be interrupted by an incoming or existing task in the queue.

Depending on the nature of the workload experienced by a computing system, the choice of scheduling policy can have a significant effect on the mean performance metrics of tasks, as well as the variance and consequently our confidence in these metrics [Conway et al., 1967; Kleinrock, 1975a;b; Wierman and Harchol-Balter, 2003; 2005]. Ultimately a scheduling policy will be chosen according to the problem domain, the hardware and the software limitations (e.g. operating system) of the computing system in question.

3.1.1 Non pre-emptive scheduling policies

Non pre-emptive scheduling policies dictate than once a task is in service, it cannot be interrupted by any other arriving task, or an existing task in the service queue. This is advantageous as it still allows the order of waiting tasks to be manipulated, whilst the costs associated with bring tasks in and out of service (and maintaining and saving state information) is avoided. From an analytical standpoint, queueing systems with non pre-emptive scheduling are typically easier to formulate and optimise.

First-In-First-Out (FIFO)

First-In-First-Out, as the name suggests, services tasks to completion in the order they arrive. It is often referred to in the literature as First-Come-First-Served (FCFS). A large volume of existing queueing results (such as the Pollaczek-Khinchin formula) assume a FIFO scheduling discipline. Intuitively it is considered the most *fair* scheduling policy, where a task is rewarded for arriving earlier than another task by being serviced before it.

Last-In-First-Out - (LIFO)

Under a Last-In-First-Out discipline, the last task to enter the queue is the first candidate for service. It is otherwise referred to as Last-Come-First-Served (LCFS) in scheduling literature. LIFO policies are analogous to stack data structures, where elements are added to the stack, and removed (i.e. popped) from the top of the stack rather than the bottom. One could imagine that there is a potential, especially under high loads, for tasks to languish at the bottom of the stack for lengthy periods of time. Indeed this turns out to be the case - while

the mean queueing metrics for FIFO and LIFO are equivalent, the second moment (and consequently the variance) of the LIFO policy are worse, having a greater dependence on the system load [Wierman and Harchol-Balter, 2003; 2005].

Random Selection for Service - (RSS)

A Random Selection for Service discipline selects tasks waiting in the queue for service in a purely random fashion. That is, each n customers in a queue have a $1/n$ chance of being selected for service. The queueing results for the RSS scheduling discipline have been found to be identical that of FIFO scheduling [Spirn, 1980]. This is expected, as Conway et al. have found that all non pre-emptive scheduling disciplines that do not make use of task sizes (such as FIFO, LIFO and RSS) have the same distribution on the number of tasks in the system [Conway et al., 1967]. Consequently, we can categorically state (according to the Pollaczek-Khinchin formula) that the expected (average) queueing metrics of these policies are equivalent.

Shortest Job First - (SJF)

Shortest Job First, otherwise known as Shortest Job Next (SJN), selects a task for service from all tasks waiting in a queue that has the smallest service requirement (e.g. execution time). A SJF scheduling discipline is desirable as it maximises the throughput - that is, the number of tasks that can be serviced to completion in a given period of time. However, there have been long held views that SJF policies can cause service starvation for tasks with larger service requirements if smaller tasks continually arrive. Also, its applications are limited as it requires knowledge (or accurate estimations) of the service requirements of all tasks waiting in the queue.

3.1.2 Pre-emptive scheduling policies

Pre-emptive scheduling policies allow arriving tasks (or tasks already queued) to *pre-empt* or interrupt a task currently being serviced, regardless of whether it has finished or not. The current task being processed is brought out of service in exchange for another task. Such techniques can often reduce the variance in the waiting times of tasks, depending on the particular pre-emptive scheduling policy used. For instance, a smaller task can interrupt a long running task so it is not waiting a disproportionate amount of time to be serviced.

However, additional overhead is incurred by swapping unfinished tasks in and out of service, and maintaining the appropriate state information so they may eventually resume.

Processor Sharing - (PS)

A processor sharing scheduling discipline (otherwise known as time sharing) givens a quantum of service to each task waiting in the queue in a round-robin fashion [Kleinrock, 1967]. Processor sharing queues have a number of desirable properties. They are considered to be *fair* in that all tasks have the same expected slowdown [Ward and Whitt, 2000; Wierman and Harchol-Balter, 2003]. Also, the average time a task spends in a system depends only on the mean of the task size distribution, and not on other higher moments. Intuitively these results make sense, as a processor sharing discipline addresses many of the issues associated with FCFS queueing. As tasks are given a fixed slice of service, larger tasks cannot block smaller tasks behind it in the queue for lengthy periods of time, as can occur in non pre-emptive scheduling policies. This also assists smaller tasks without requiring that the service time is known in advance, as they only require a small number of slices to complete service.

Shortest Remaining Processing Time - (SRPT)

A Shortest Remaining Processing Time scheduling discipline is a pre-emptive policy that services the task with the least remaining processing time first. It requires that the service requirement be known in advance to be utilised effectively. A SRPT discipline has long since been proved to be optimal in minimising the mean response time in a queue [Schrage, 1968; Winston, 1977; Smith, 1978]. Despite this, its usage is not widespread due to the common perception (and teaching) that it favours shorter tasks, and can disadvantage or even starve longer tasks [Tanenbaum, 1995; Stallings, 1995; Silberschatz and Galvin, 1998]. This perception of unfairness by the SRPT policy toward larger tasks has been challenged in recent research [Bansal and Harchol-Balter, 2001]. The authors analyse the SRPT scheduling discipline using standard $M/G/1$ queueing analysis, compare the results against the PS scheduling policy, which is considered *fair* in that it provides the same expected slowdown for all tasks. Bansal et al. contend that the notion that SRPT scheduling is unfair is largely unfounded, and demonstrate that under moderate system load (independent of task size distribution), all tasks prefer SRPT scheduling to PS. Under higher load, this observation still holds for heavy-tailed task size distributions. In light of these findings, the SRPT scheduling policy has since been implemented in a web server [Harchol-Balter et al., 2003c;

Schroeder and Harchol-Balter, 2006]. By simply changing the order in which web requests where scheduled, the authors demonstrated improved delay and mean response time whilst having minimal adverse effects on larger requests.

Pre-emptive Shortest-Job-First (PSJF)

The pre-emptive Shortest-Job-First scheduling policy, like its non pre-emptive equivalent, aims to execute the task with the shortest service requirement. However, PSJF guarantees this behaviour at all times by allowing tasks that arrive to pre-empt the task currently being serviced if it is smaller. PSJF provides near optimal response time for tasks in a single queue, close to that of the SRPT policy, which is known to be optimal as described previously. However, for certain classes of tasks (e.g. larger tasks), PSJF can discriminate against them and provide unpredictable queueing performance [Harchol-Balter et al., 2003c; Schroeder and Harchol-Balter, 2006]. This is unsurprising given that PSJF favours smaller tasks over larger tasks at all times.

Pre-emptive Last-Come-First-Serve (PLCFS)

A pre-emptive Last-Come-First-Serve policy is simply a pre-emptive analogue to the LCFS described previously. At all times, a PLCFS policy ensures that the last arriving task is being serviced. Any running tasks will be pre-empted from service by a newly arriving task as necessary. Like Processor Sharing, PLCFS has been found to be both fair and provide predictable service times to all tasks, no matter what their size is [Wierman and Harchol-Balter, 2003; 2005].

Least-Attained-Service (LAS)

The Least Attained Service scheduling discipline is a pre-emptive policy that does not require any knowledge of a task's service requirement. LAS works by giving service to the task in the queue that has received the least service overall, compared to all other waiting tasks. Newly arriving tasks always pre-empt the current task in service, and is processed until the next task arrives or it has received an amount of service that is equivalent to the task it pre-empted, depending on which occurs first. A LAS policy is appealing, as it has been shown to provide a mean response time that is comparable to that achieved by SRPT, whilst not requiring knowledge of task sizes needed by SRPT [Rai et al., 2003]. In the same study, LAS was also shown to provide finite mean response time for the majority of tasks (up to

the 99th percentile) for tasks at an overload of $\rho = 2.0$. Further work has demonstrated that LAS (and LAS variants) are very effective as scheduling policies over bottleneck links in packet switched networks, significantly besting FIFO scheduling under a variety of load conditions [Rai et al., 2004].

3.2 Load index

The availability and utilisation of resources provided by a computer are typically characterised in the form of a load index. A load index could provide a measure of utilisation of a single resource on a computer system, such as the load on the CPU. Alternatively, it can be a combined measure of multiple resources and metrics like CPU load, memory utilisation, disk paging and the number of running processes. This information can be utilised to identify a node that is lightly loaded, making it a candidate to have tasks migrated there from a heavily load node (discussed in Section 3.3), or identifying a target node for a dynamic load distribution policy (discussed in Section 3.4). In Section 3.2.1 we discuss the types of measures and metrics that are commonly used to construct load indexes in computer systems. Section 3.2.2 describes some techniques to utilised to disseminate this information in an effective and timely manner among distributed systems.

3.2.1 Load index measurement

The type of load measurements and metrics used in computing systems depend very much on the operating system and the application domain. Work by Ferrari and Zhou attempts to formalise this by considering the notion that a *correct* load index for a given application domain (and process mix making up the workload) should ensure that the response time for a task must be a function of the load index [D. Ferrari, 1985; Ferrari and Zhou, 1986]. In this investigation, the authors consider the use of an linear combination of mean queue lengths (such as background processes) and coefficients describing the resource requirements of the next task waiting to be assigned in order to represent the perceived load at each machine. The intent of computing the index in this manner is to ensure the response time for a given application (with specific resource needs) is minimised. Ultimately this means that the indexes are computed on a per application basis as needed, considering the varying resource requirements of different applications. The authors note that it is unclear how suitable this measure would be in highly dynamic distributed computing environments.

Further work by Ferrari and Zhou evaluates the effectiveness of a variety of different load

indices as a means to perform load balancing in a distributed system [Ferrari and Zhou, 1988]. The authors emphasise the need for a given host to be viewed as a collection of resources, and for a given load index to represent that fact. Experiments were performed on a distributed system, utilising the instantaneous CPU queue length, averaged CPU queue length, averaged file and paging I/O and memory queue lengths (e.g. buffer space, page table) as load indices in an individual and combined manner. Other important factors in providing an effective load index were identified, including the averaging interval for measurement of a given resource queue length, and the exchange interval, representing the frequency of dissemination of the load index to other hosts. The importance of such load dissemination policies are investigated in the next section. The authors found that for the workload used in the experiments (consisting of common non-interactive unix tasks), using the instantaneous CPU queue length proved the most effective to improve the overall response times of jobs, besting several other hybrid load indices.

Kunz considers the influence of differing load descriptors for a load balancing scheme for a general purpose distributed system [Kunz, 1991]. A number of requirements are imposed for such a system, namely that no assumptions are made regarding the specifics of the underlying network, no *a priori* knowledge of incoming tasks is assumed, the system is dynamic, physically distributed and has cooperative scheduling (as defined in prior research [Casavant and Kuhl, 1988]) and that mean response time of a task is the key performance metric of interest. The choice of load descriptor in such systems obviously affects the ability of a system to meet these requirements. Kunz examines the utility of singular workload descriptors, as well as more complex combinations of workload descriptors, in providing efficient and effective load information for load balancing schemes. The single descriptors include the number of tasks in the run queue, available memory, rate of CPU context switches, rate of system calls, 1-minute load average and the amount of free CPU time. Combination metrics utilise multiple individual metrics which are multiplied by a coefficient to either normalise or weight a given metric. From the individual descriptors, the number of tasks in the run queue proved to be the best metric to minimise response time. No improvements were found when utilising combined descriptors over simply utilising the one-dimensional workload descriptors.

Work by Cardellini investigates the different state estimations (i.e. load index) utilised by DNS-based load control policies in distributed web clusters [Cardellini et al., 1998]. Choosing an effective load index in such a scenario is made more challenging by the fact that DNS cannot be relied upon to control all incoming requests due to caching of DNS data at both the ISP and client level. Indeed, it can have control over as little as 5% of requests in many

instances [Cardellini et al., 2002]. The fact that web requests are typically non-uniformly distributed and highly variable provides additional difficulties. Such indices must also consider the reliability of the load index, given that it can only be updated during a DNS TTL (Time-To-Live) interval. The authors cater for this limitation by estimating the *hidden load weight*, which considers the number of requests that a web server will receive during a TTL interval. This load measure can potentially estimate the number of request sessions, the number of page requests or the number of hits that occur during a specific interval. The authors consider load control policies using these state estimators that are updated at static TTL intervals [Colajanni et al., 1998b] and dynamic TTL intervals [Colajanni et al., 1998a].

A Two-tier Round Robin (RR2) approach was also examined, that utilises a static interval threshold-based approach to group client domains into two classes, *normal* and *hot*. Client domains are placed in either class by considering their hidden load weight index relative to the threshold. This two-tier approach is essentially a weighted round robin approach that attempts to avoid hot client domains from being constantly assigned to the same server, spreading requests over a larger (weighted) proportion of available servers and reducing the chance of overload. The hidden load weight index is used more directly in the Dynamically Accumulated Load (DAL) and Minimum Residual Load (MRL) policies, using a predictive model to gauge the expected load on servers (based on previous assignments it has made) to maintain a more accurate load index of each of its member servers. An Adaptive TTL (AdpTTL) approach was also evaluated, that utilises a load index that considers both the hidden load weight and the server capacity of member servers, making it suitable for heterogeneous cluster environments. This index is used to assign dynamic TTL values that are inversely proportional to the hidden load weight in an effort to achieve more fine-grained and consistent load balancing.

Cardellini found that under theoretical conditions there was little difference between the load estimates used (sessions, requests or hits) for the RR2, MRL, DAL or AdtTTL policies, with each one providing significantly better performance than the baseline round robin approach. Under realistic and dynamic experimental conditions, utilising the number of hits in the computation of the hidden load weight metric achieved the best (or near best) results for the RR2, MRL, DAL or AdtTTL policies.

3.2.2 Load index interpretation and dissemination

Mitzenmacher considers how useful old load information is when choosing where to direct newly incoming tasks in a distributed system [Mitzenmacher, 1997; 2000]. Given that it is typically unfeasible to have instantaneous global load information available at all times, Mitzenmacher investigates the use of load information that is only periodically updated. Given that load can change rapidly, it is often unclear what is the most appropriate way to utilise old load information. The author extensively models and evaluates different models for old load information. Mitzenmacher considers a theoretical *bulletin board* approach - where global information is centrally located at a bulletin board. However, in reality this information could potentially contain old (and inaccurate) information. To choose where an incoming task is assigned, one could choose d servers at random (i.e. a subset), check their load information from the bulletin board, and assign the task to the server with the least load. Alternatively the load of all servers could be checked, and a task assigned to the server with the least load. Providing the information on the bulletin board is up to date, the subset approach performs well and has lower overhead than checking the load of all servers. In more centralised distributed systems, where a centralised bulletin board of all load information is feasible, assigning tasks to the least loaded server can provide better results, and is indeed optimal from a mathematical standpoint in many situations.

The way load is disseminated and interpreted in a bulletin board model must still be addressed. Mitzenmacher suggests a *periodic update* model, where the bulletin board is updated with new load information every T seconds. As such, the update times would be 0, T, $2T$, ... The time between these updates is considered a phase, with phase i ending at time iT. Two key issues exist for this type of approach - the number of servers that are considered (e.g all servers or a subset) when choosing where tasks are assigned, and the length of the update phase T. If the update interval T is kept short, then choosing the shortest queue (according to the load information currently available) works well - as few new tasks arrive in the interval and the *old* load information remains relatively accurate. As T increases, instance's of *herd behaviour* can occur as tasks are assigned to the same small subset of servers that appear lightly loaded, eventually leading to overload. In such instances (as T grows larger and approaches ∞) simply assigning tasks at random to servers usually performs better.

Alternatively, a *continuous update* model can be used, where the bulletin board is updated in a continuous fashion but remains T seconds behind the true global state at all times. This

models the common scenario where there is transfer delay between when the load information is propagated and when it is available to use by tasks. As a result, tasks use information that is T seconds old when making a choice on its destination. Two scenarios were investigated - where the continuous update interval T is a fixed constant, and where T is replaced by X, a random exponentially distributed value. Significantly different results where observed in each instance. In the case of a fixed T, assigning tasks to the least loaded server performs poorly, even with relatively small T values. Choosing randomly amongst a small subset of least loaded servers performed significantly better over a wider range of T values. Using a randomly distributed update interval X produced surprisingly good results for both shortest queue and subsets of shortest queue assignments. Mitzenmacher attributes this to the fact that tasks entering the system at approximately the same time will have differing views of the system and consequently make different choices. This avoids the herd behaviour commonly experienced by other load balancing approaches and improves utilisation and performance. A *individual update* model is also briefly considered, where servers update their load information at different, independently distributed times. However, this model was found to perform similarly to the standard continuous update model.

Several interesting results are drawn from Mitzenmacher's work, but we note that it is limited to systems where arrivals and service distributions are exponentially distributed (i.e. $M/M/c$). As such, these findings cannot be directly mapped to systems with highly variable arrivals or service distributions. Indeed, the conclusions would likely be substantially different due to the highly variable nature of heavy-tailed workloads and their propensity to unbalance the load in distributed systems.

Research by Dahlin attempts to get more utility out of stale load information by improving the way it is interpreted and utilised in distributed systems [Dahlin, 2000]. Dahlin proposes *load interpretation* (LI) algorithms that consider not just the last reported load information, but also the age of that information and the rate that tasks arrive, which can potentially make the load information inaccurate.

Dahlin's work depends heavily on the prior work of Mitzenmacher [Mitzenmacher, 1997; 2000] regarding the means of obtaining and updating the load information itself (i.e. periodically, continuously or individually updating the load bulletin board). Rather, he focuses on devising load interpretation models that use the information in such load update models effectively.

The load interpretation models consider two key elements in their use of available load information. First, the magnitude of load imbalance between servers is considered. Second,

and most importantly, the interpretation of load information depends on its age and also the arrival rate of tasks into the system.

Under simulation, it was found that if load information was fresh (as the update interval, T, or arrival rate, λ, are low) then the load interpretation algorithms sent requests to servers with recently reported low load. In such circumstances the performance was found to match that of aggressive Least-Loaded-First (LLF) algorithms, and bested both random subset and purely random algorithms.

When faced with old load information, the proposed LI algorithm tended to distribute tasks more uniformly rather than relying too heavily on stale information, as the perceived least loaded server could have since been overloaded. Thus the performance demonstrated was comparable if not better than truly random algorithms, and better than algorithms that naively use stale load information.

When dealing with load information of a modest age (neither especially fresh or stale) the load interpretation model significantly outperformed other algorithms tested by up to 60%.

Several other key findings were obtained. In the load interpretation models considered, underestimating the arrival rate λ severely affects the performance of the load interpretation (LI) algorithm. However, overestimating the arrival rate has little harm. As such, the author recommends assuming a λ that maximises throughput (i.e. system load of 1.0). If the load is legitimately higher than this then no algorithm with help. Dahlin finds that even when limited load information is made available (such as with a 'k-subset' approach described previously [Mitzenmacher, 1997; 2000]), with the aid of an effective load interpretation algorithm such information can be used to outperform algorithms that use no load information under exponential workloads.

3.3 Process Migration

A process is an Operating System construct that represents an instance of a running computer application. A process typically has other resources associated with it, characterised by data, program stack, register contents and other state descriptors, depending on the Operating System it is running on. Process migration is the act of transferring a process between two logical entities (a source and destination) during execution. These entities could be different CPU's on the same machine, or different physical nodes on a network. If the migration is to be work-conserving, state information must also be transferred during migration of a process.

We often use the term task as a generalisation of the process concept.

The mechanics of work-conserving migration depends on the Operating System and the implementation details, but it can be described generally as follows [Milojicic; et al., 2000]:

1. Based on some migration criterion, a migration request is issued to a remote node. After negotiation between the source and destination node, the migration request is accepted.

2. A process is suspended from its execution, and detached from the source node. It is now in a migratory state.

3. Arriving messages are temporarily queued until the communication channels can be redirected to the destination node and delivered successfully.

4. The process state is extracted, which depending on implementation details can consist of memory and register contents, communication state (such as open files) and the kernel context.

5. The process is recreated on the destination node, in readiness for the relevant state information to be transferred.

6. State information is transferred and associated to the newly recreated process on the destination node.

7. Forwarding references must be created to direct communication to the newly recreated process on the destination node.

8. The newly recreated process resumes execution from the point at which it was halted on the source node.

The more recent growth in popularity of broadly accessible virtualisation solutions (such as Xen [Barham et al., 2003], VMWARE [Adams and Agesen, 2006]) and services (Amazon Cloud [Amazon.com, Inc, 2007], PlanetLab [Bavier et al., 2004]) provides an interesting framework where a lightweight virtual machine (VM) image can be a unit of execution (i.e. instead of a 'task' or process) and migration [Nelson et al., 2005]. These developments, described by Nelson et al., have allowed migration of unmodified applications encapsulated in virtual machines, even if these applications are unaware of the mechanics (described above) behind it. This migration can be achieved in a work-conserving fashion without the running

application nor any dependant clients or external resources being aware that it has occurred. The VMWARE management capabilities makes this possible encapsulating the state of the VM, such as CPU, networking, memory and I/O while the virtual machine is still running. It can transfer open network connections due to the layer of indirection provided by the VMWARE Virtual Machine layer. Physical memory is often the largest overhead in terms of state that must be migrated. Stopping a VM to save and transfer this state can cause a lengthy downtime, making the migration process far from transparent. As such, this is handled *in situ* by a virtual memory layer [Waldspurger, 2002], allowing memory state to be transferred whilst a VM is still running by iteratively pre-copying memory to the destination node and marking it as temporarily inaccessible to the source.

Nelson et al. found that for a variety of CPU, I/O and memory bound workloads, VM migration in a local cluster can be fast and transparent for the applications themselves and any dependent clients and external resources. In most causes downtime was kept under a second, short enough to avoid a noticeable lapse in service by clients. If the VMs are simply batch workloads, this is even less of an issue. Even if migration takes several seconds, this must be contrasted with the probable runtime of a virtual machine, which could be in the order of minutes or hours. As such, the benefits of migration would depend on the nature of the virtual machine workload.

Non work-conserving migration is often achieved simply via remote invocation. A process is stopped on a source node, and restarted (from scratch) at the destination node. This can often be a less expensive operation due to the fact that no state information needs to be transferred from the source node to the destination node. Clearly there will be a trade-off between losing any work done prior to migration, and avoiding the cost of state transfer in work-conserving migration, especially if a process has a large state that needs to be transferred. This balance depends on the characteristic's of tasks in a given application domain, and the efficiency of any migration mechanism that might be used.

The decision to migrate a process is typically justified by the assumption that it can complete its execution faster by accessing resources on another node. It bases this assumption on some comparative measure of the *load* at the source node and the destination node. The goodness of this assumption depends on how accurate the load measure is for the application domain the process is derived from [Kunz, 1991]. The differing representations of load, as well as load collection and dissemination policies have been covered previously in Section 3.2.

One measure of load, process lifetime, has been found to be an effective indicator of when it is prudent to employ process migration. Naturally, it is rarely beneficial to migrate a short

running tasks as the benefit does not outweigh the migration costs. Early work in this area (predating much of the analysis of computing workloads discussed in Section 2.3) found that it was possible to predict a process's expected lifetime with reasonable accuracy based on how long it has already lived [Cabrera, 1986; 1999].

The question of whether process migration actually provides benefits has been found to depend very much on the characteristics of the workload the process is derived from [Nuttall and Sloman, 1997]. This is highlighted by conflicting research regarding the benefit of migrating running processes in distributed systems. Analysis performed by Eager suggested that that performance gains obtained by work-conserving process migration are only modest in the many cases, and that there are no conditions where such migration yields major performance gains [Eager et al., 1988]. Conversely, more recent research has been conducted that disputes this, and endeavours to quantify the relationship between process lifetime and acceptable migration costs [Harchol-Balter and Downey, 1997].

3.4 Classical load distribution policies

The problem of optimal task assignment in a distributed system has been a well researched area for many years. Most of the so-called 'classical' approaches were created under the assumption of a Markovian service process (e.g. exponentially distributed service times). Many of these policies are still widely used, due to their simplistic nature and ease of implementation. In this section we explore some of these policies, evaluate their strengths and weaknesses, and identify some known results.

3.4.1 Random and Round Robin

Classical task assignment policies such as *Random* and *Round-Robin* [Silberschatz and Galvin, 1998] have traditionally been used in distributed systems, and are still widely used for many application domains. Under the Random policy, tasks are statically assigned to each back-end server with equal probability. Using a Round-Robin policy, tasks are assigned to servers in a cyclical fashion. Both policies equalise the expected number of tasks at each server, and are frequently used as a base line to compare with other task distribution policies. Tasks are assigned with no consideration of each host's load or the distribution of task sizes. Despite this, Random and Round-Robin are still commonly used in many scheduling environments (most likely due to ease of implementation). It has been shown previously [Harchol-Balter et al., 1999] that Random and Round-Robin both have similar performance characteristics.

Weighted variants of Random and Round-Robin are popular task assignment policies, particularly when a distributed system contains heterogeneous hosts [Tang and Chanson, 2000; He et al., 2004]. In these instances weighting the task assignment so that more powerful hosts receive a larger share of tasks can result in significant improvement over standard Random and Round-Robin policies. Random load distribution policies have also been combined with Dynamic policies, where a task is assigned to the least loaded server, selecting only from a random subset of the available servers [Mitzenmacher, 2001].

3.4.2 Dynamic

Dynamic policies intelligently assign tasks based on a representation of the current load at each host. The LLF (Least-Loaded-First) approach assigns tasks to the server with the least amount of work remaining, attempting to achieve instantaneous load balance. The work remaining can be approximated by the queue length (Shortest-Queue) [Winston, 1977; Nelson and Philips, 1989], or assuming the tasks service requirement is known *a priori*, the cumulative work remaining in the queue (Least-Work-Remaining). By keeping the load balanced, the waiting time in queue can be reduced. It is known that balancing the load minimises the mean response time [Crovella et al., 1998a; Schroeder and Harchol-Balter, 2004] in the type of distributed system that we consider in this thesis. A Dynamic policy exhibits good performance as the task size distribution becomes more uniform, but when it is close to the empirically measured workloads (where $\alpha \approx 1$) and highly variable, newer size-based approaches perform better. Despite this, a number of caveats exist. First, the 'best' performance is not always obtained by balancing the load, particularly if you are interested in different measures of performance, such as the mean slowdown. Second, balancing the load is not always practical, as often you are depending on approximate measures of the load, such as the queue length. Under highly variable workloads (where the difference between 'small' and 'large' tasks can be enormous) it is highly probable that the length of a queue can be an unreliable indicator of the actual load at a host. Third, there is additional complexity and overhead to update and disseminate load information. This issue was covered in detail in Section 3.2. As such, it is easy to imagine how it is a bad policy to depend only on the *number* of tasks in the queue at each back-end host, and the effect on performance that can result from using such information to base task assignment choices on.

3.4.3 Central Queue

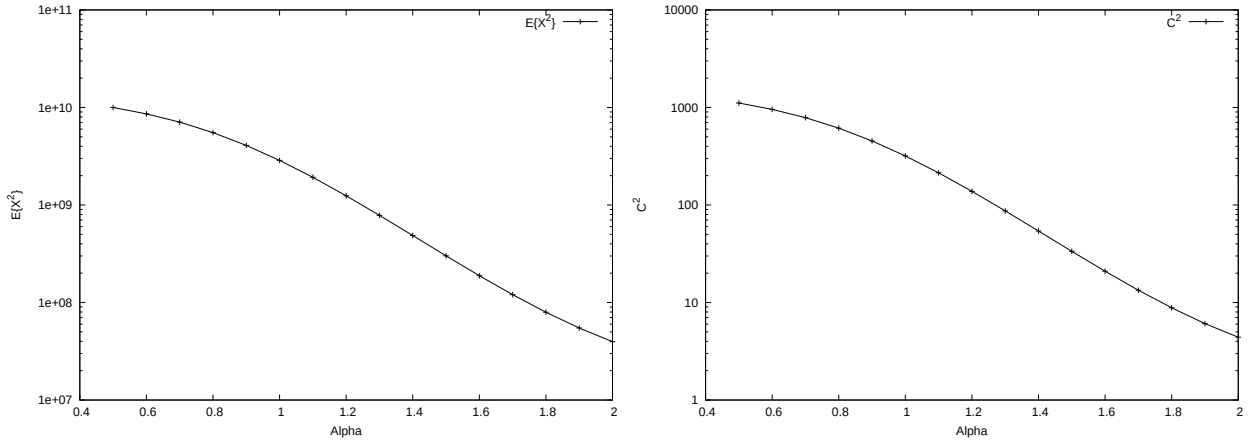
The Central-Queue policy holds tasks in a queue at the dispatcher until a host is idle. Such a policy has proved to be equivalent to a Least-Work-Remaining policy, showing that equivalent performance can be obtained without any prior knowledge of a task's size [Harchol-Balter, 2002; Harchol-Balter et al., 1999]. However, while both exhibit similarly good performance under an exponential workload, the performance of a Central-Queue policy is equally poor under more realistic conditions of heavy-tailed workloads. Recently, two variations of the Central-Queue policy have been proposed - Cycle Stealing with Immediate Dispatch (CS-ID) and Cycle Stealing with Central Queue (CS-CQ) [Harchol-Balter et al., 2003b;a]. CS-ID immediately dispatches tasks to a back-end server, whilst CS-CQ holds tasks in a central queue at the dispatcher until a host is idle. Both policies are evaluated against a Dedicated policy (much like the size-based policies outlined in the next section). In a Dedicated policy, one host is 'dedicated' to servicing all short jobs, while the other host services long jobs. Both CS-ID and CS-CQ follow a similar arrangement, but can steal cycles from an idle host if available (and it is prudent to do so). For example, if a short job arrives and the short host is busy whilst the long host is idle, the short job can be dispatched to the long host to improve utilisation. Both CS-ID and CS-CQ show improvement over a Dedicated policy in many areas (notably for short tasks), The application of these policies are limited to domains where *a priori* knowledge of a tasks size is known, and in the case of CS-CQ, there needs to be constant feedback between the dispatcher and the back end hosts to notify the dispatcher of an idle host.

3.4.4 Known Results

The Round-Robin policy results in a slightly less variable arrival stream than the Random policy. Despite this, the performance of the Random and Round-Robin policies have been shown to be roughly equivalent [Harchol-Balter et al., 1999].

Under a $M/M/c$ assumption, a Shortest-Queue policy has been shown to be optimal with respect to maximising the number of jobs completed by some time t [Weber, 1978]. Under the same assumptions, Nelson and Phillips claim that the Central-Queue (and therefore Least-Work-Remaining) policy is optimal [Nelson and Philips, 1989; 1993].

A Least-Work-Remaining policy is not analytically tractable under $M/G/c$ queueing systems. Nonetheless this policy has been shown to be equivalent to a Central Queue policy [Harchol-Balter et al., 1999], for which their exists known approximations [Sozaki and



(a) (b)

Figure 3.1: The second moment of a Bounded Pareto distribution ($E\{X\} = 3000$, $p = 10^7$) is shown in (a), where α is varied from 0.5 to 2.0. The squared coefficient of variation ($C^2 = E\{X^2\}/E\{X\}^2$) is shown in (b).

Ross, 1978; Wolff, 1989]:

$$E\{Q_{M/G/c}\} \quad = \quad E\{Q_{M/M/c}\}.\frac{E\{X^2\}}{E\{X\}^2}$$

where X is the service requirement distribution, and Q represents the queue length with subscripts representing its queue characteristics.

3.4.5 Limitations

The task assignment policies listed above are not suited to highly variable task size distributions. Consider the metrics shown in Section 2.2, and the approximation for the Least-Work-Remaining policy listed previously. What is immediately apparent is that all metrics depend on the second moment of the service requirement distribution, $E\{X^2\}$. For the Dynamic (e.g. Least-Work-Remaining) policy, we can see that all metrics will depend on the squared coefficient of variation ($C^2 = E\{X^2\}/E\{X\}^2$). C^2 characterises the variability of a probability distribution. Distributions such as Erlang are considered to have low variance, whereas distributions like Pareto and Hyper-exponential are considered highly variable. Figure 3.1 shows an example of one such highly variable distribution (where the y-axis is on a log scale). This is consistent with the type of workloads found in recent measurements, as discussed in

Section 2.3. We can see that as α decreases, the variation (as represented by the second moment of the distribution, and the squared coefficient of variation) explodes. Clearly, as the variability of the service time distribution increases, the performance of the distributed system will decrease rapidly using these classical load balancing policies.

3.5 Size-based load distribution policies

In the previous chapter we highlighted substantial recent research showing the frequent occurrence of heavy-tailed workloads in many distributed computing environments. The characteristics of these heavy-tailed workloads make them very challenging to manage using traditional load distribution policies. Indeed, many of these policies were created under the assumption of $M/M/c$ workloads - for instance where the distribution of service requirements follows an exponential distribution. To deal with 'heavy-tailed' $M/G/c$ workloads, new load distribution techniques need to be employed. In particular, they must address the characteristics of these workloads, such as their highly variable nature, that cause such poor performance under traditional load distribution policies. In recent years there have been several load distribution techniques designed to do just that, specifically created to exploit the characteristics of heavy-tailed workloads. They can be broadly classified as *size-based* policies - where the workload is partitioned into distinct size ranges, with each size range associated to a specific server. For example, you may have a two server system where one server processes 'small' tasks, while another server processes 'large' tasks.

These size-based policies can be further classified by what knowledge they assume is known at the dispatcher. Some policies assumes that a task's size is known *a priori* at the dispatcher, and as such can assign the task directly to the server that is responsible for servicing tasks in that range. This obviously restricts the application of these policies to domains where exact (or reasonably accurate) *a priori* knowledge of a task's size is available.

Other size-based policies have less restrictive assumptions regarding what information is available at the dispatcher. Policies such as TAGS [Harchol-Balter, 2002] assume no knowledge of a task's size at the dispatcher. They do however require knowledge of the distribution of task sizes.

3.5.1 SITA-E/V/U - Known task size

SITA-E (Size Interval Task Assignment with Equal Load) [Harchol-Balter et al., 1999] is a sized-based approach proposed by Harchol-Balter et al. that associates a unique size range

with each host in the distributed system. These size ranges are chosen specifically to equalise the expected load received at each host. Whilst proving effective under conditions of high task size variability, SITA-E is not the best policy in circumstances of lower task size variability (with a dynamic policy being more suitable).

SITA-V (Size Interval Task Assignment with Variable Load) [Crovella et al., 1998a] intentionally operates the hosts in a distributed system at different loads, and directs smaller tasks to lighter-loaded servers. The authors note that depending the performance metrics of interest, the conventional notion of balancing the load on all hosts may not result in optimal performance, especially when the size distribution is heavy-tailed. SITA-V, like SITA-E, assigns tasks to a given host based on their size. However, SITA-V exploits the heavy-tailed property of the task size distribution by running the vast majority of tasks (i.e. the small tasks) on lightly-loaded hosts, while running the minority of tasks (the larger sized tasks) on the heavily-loaded hosts, thus preventing small tasks getting 'stuck' behind large tasks and allowing them to be processed quickly. Mean slowdown is reduced, and the throughput is not adversely affected, but it can result in an increase in mean waiting time - which is expected, since minimal mean waiting time is known to occur when load is balanced.

Obviously there is a trade-off between improving mean slowdown and degrading mean waiting time, and the authors recognise this issue and identify the two contributing factors - the variability of tasks (as represented by α) and the overall system utilisation.

From the simulation results presented by the author [Crovella et al., 1998a] SITA-V shows itself to be a worthy choice when task sizes are highly variable ($\alpha \leq 1$), showing significant improvements in slowdown by factors of 1000 or more (using two hosts) over an equal load policy. When task variation is close to the empirically measured mean ($\alpha \approx 1.1$) or greater, the reduction in waiting time in processing the majority of small tasks on the lightly-loaded server can no longer outweigh the penalty of load imbalance on the other heavily-loaded server. This is due to the average size of small tasks increasing.

A size-based approach that is specifically suited for batch computing environments under super-computing workloads is SITA-U (Size Interval Task Assignment with Unbalanced load) [Schroeder and Harchol-Balter, 2004]. SITA-U purposely unbalances the load among the hosts while also being 'fair' - achieving the same expected slowdown for all jobs. The authors collected real trace data from super-computing centres and used it in their simulation, comparing SITA-U against Random, Least-Work-Left and SITA-E. Two variations of SITA-U are considered: SITA-U-opt, where service requirement cutoffs are chosen to minimise mean slowdown, and SITA-U-fair, where service requirement cutoffs are chosen to maximise

fairness. The simulation results showed that both variations of SITA-U performed better under a range of load conditions - with system load varied between 0.1 and 0.8. SITA-U-fair achieved significant performance gains over the range of load 0.5 - 0.8, demonstrating an improvement of 4 - 10 times with regards to mean slowdown, and from 10 - 100 with regards to variance in slowdown.

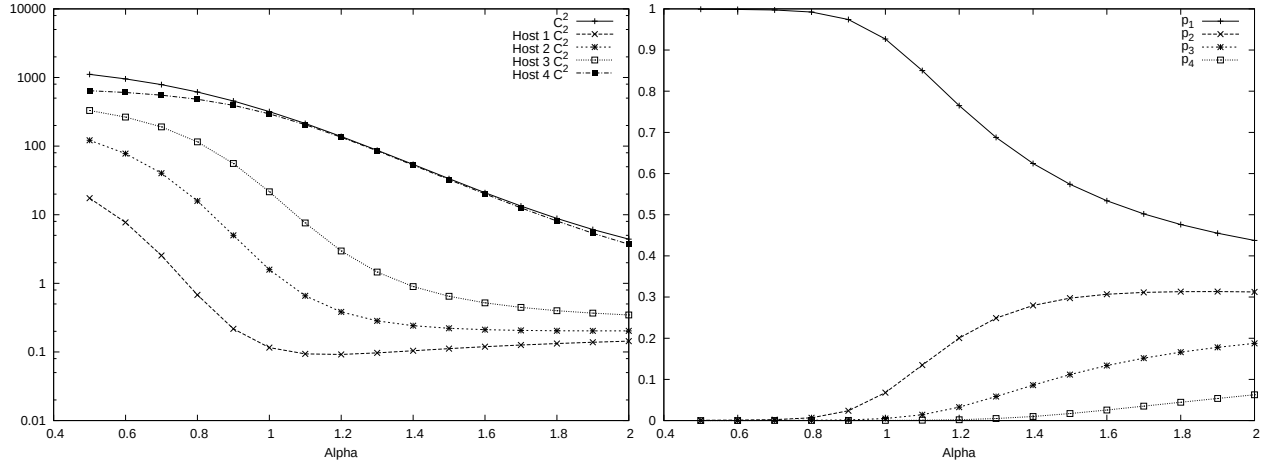
Most size-based policies perform well under very high task size variation, but their advantage over existing approaches is reduced as variation decreases. Most importantly, the application of the task assignment policies listed above is limited by the assumption that the service requirement of each task is known *a priori*, which is frequently not the case.

3.5.2 TAGS - Unknown task size

The size-based approaches considered thus far all assume that the exact service requirements is known at the dispatcher in advance. Often this is not the case - in many environments a tasks service requirement is not known until execution time on a given host. Task Assignment based on Guessing Size (TAGS) [Harchol-Balter, 2002] assumes no prior knowledge of a tasks service requirement. Like SITA-V, TAGS is slightly counter-intuitive in that it unbalances the load, and also considers the notion of 'fairness' - that all tasks should experience the same expected slowdown. The TAGS approach works by associating a processing time limit with each host. Tasks are executed on a host up until the designated time limit associated with that host - if the task has not completed by this point, it is killed and restarted from scratch at the next host. These cutoffs are a function of of the distribution of task sizes and the outside arrival rate, and can be computed to optimise certain metrics, such as waiting time or slowdown.

The design of the TAGS policy purposely exploits properties of the heavy-tailed distribution, such as decreasing failure rate - where the longer a task has run, the longer it is expected to run - and the fact that a tiny fraction (less that 1%) of the very longest tasks can make up over half the load.

Like other size-based approaches, under higher loads and less variable conditions, TAGS does not perform so well. TAGS gains much of its performance by exploiting the heavy-tailed property, by moving (in a two host example) the majority of the load onto host 2, allowing the vast majority of small tasks to be processed quickly on host 1. TAGS also suffers under high loads due to excess - the extra work created by restarting many jobs from scratch. As noted by the author [Harchol-Balter, 2002], "...overall excess increases with load because excess is



(a) (b)

Figure 3.2: The squared coefficient of variation experienced at each host in a 4 host SITA-E system is shown in (a). The fraction of tasks assigned to each host is shown in (b)

proportional to λ (task arrival rate), which is in turn proportional the [overall system] load, ρ."

3.5.3 Known results

It is easy to illustrate how such size based approaches are effective at counteracting the variance that can effect the performance of traditional approaches to load distribution. Consider a size-based policy that partitions the workload between the back end hosts. For the sake of brevity we restrict our illustration to policies where tasks sizes are known *a priori* at the dispatcher, such as SITA-E/V-U [Harchol-Balter et al., 1999; Crovella et al., 1998a; Schroeder and Harchol-Balter, 2004]. Nonetheless, similar analysis can be applied to size-based policies which assume no such knowledge, but still utilise similar techniques to reduce variance.

A size based policy assigns unique size ranges to each back-end host (eg. $k = s_0 < s_1 < s_2 < ... < s_n = p$). For a two host system, Host 1 would service tasks sized between s_0 and s_1. Host 2 would handle the remaining tasks, sized between s_1 and s_2.

Let p_i equal the fraction of tasks whose destination (where it will run to completion) is Host i. That is, tasks whose size is between s_{i-1} and s_i.

This is given by:

$$\begin{aligned} p_i &= P(s_{i-1} \leq X \leq s_i) \\ &= \frac{\alpha k^\alpha}{1-(k/p)^\alpha} \int_{s_{i-1}}^{s_i} x^{-\alpha-1} dx \\ &= \frac{k^\alpha}{1-(k/p)^\alpha} (s_{i-1}^{-\alpha} - s_i^{-\alpha}). \end{aligned} \tag{3.1}$$

Let us now consider only those tasks that are dispatched to and run-to-completion at Host i. Let $E(X_i^j)$ be the jth moment of the distribution of tasks that are dispatched to Host i's queue. We have:

$$E(X_i^j) = \int_{s_{i-1}}^{s_i} x^j f(x) dx \tag{3.2}$$

$$= \begin{cases} \frac{\alpha s_{i-1}^\alpha (s_{i-1}^{j-\alpha} - s_i^{j-\alpha})}{(j-\alpha)(1-(\frac{s_{i-1}}{s_i})^\alpha)} & \text{if } j \neq \alpha \\ \frac{s_{i-1} s_i}{s_i - s_{i-1}} (\ln s_i - \ln s_{i-1}) & \text{otherwise.} \end{cases} \tag{3.3}$$

Consider the following example. We shall consider a four host system, utilising the SITA-E task assignment policy. The arrival rate is Poisson and our service time distribution follows a Bounded Pareto distribution. As described previously, SITA-E chooses its size ranges in order to equalise the expected load assigned to each back-end host. Figure 3.2(a) shows the squared coefficient of variation experienced by each back end host. We can see a significant reduction in variation that has been achieved by partitioning the workload and assigning it to different hosts - effectively grouping like-sized tasks together. Indeed, the first two hosts have a C^2 that is less than one unit from $\alpha = 1.1$ to $\alpha = 2.0$. Nonetheless, we can see that the variation at the latter hosts are still quite high, approaching the value of C^2 of the task size distribution itself (i.e. before partitioning it). This is not a problem in itself, as we can see in Figure 3.2(b). We can see that the vast majority of tasks are processed by the lower hosts, and predominately the first host. These hosts have a significantly lower variance, and given they process nearly all tasks they dominate the overall system metrics such as expected waiting time and slowdown.

Recall from Section 2.2 that all our system metrics depend on the second moment of the task size distribution, $E[X^2]$. We can see now that by reducing the variance experienced by the majority of tasks (by minimising $E[X_i^2]$ at each Host i), size-based task assignment policies can significantly improve a system's performance over traditional techniques under heavy-tailed workloads.

3.5.4 Limitations

While size-based approaches have shown promising results, they have fundamental limitations in common. One problem with size-based approaches is that they can be easily unbalanced. As the task assignment policies are dictated by size, they do not consider the load of the host they are assigning to, and can easily assign a large task to an already overloaded server, while leaving other hosts under-utilised. Also, the assumption of *a priori* knowledge of task sizes limits their applications significantly. We cannot assume we have this knowledge for many application domains.

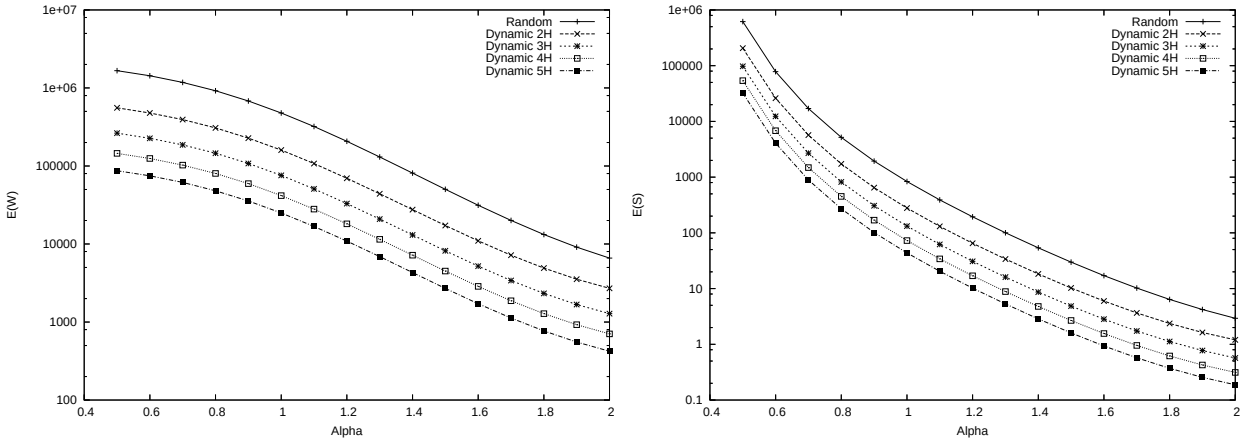
Like other size-based approaches, TAGS can be also limited by its static nature. If the characteristics of the task size distribution changes significantly, its performance will suffer - but to a lesser extend than SITA-E/SITA-V. While it is true that the cutoffs can be re-computed, this will incur significant overhead and would not be a desirable function to perform regularly. TAGS by its very nature can produce significant excess (wasted processing) by constantly restarting tasks that exceed processing limits at a given host. Also, as all tasks enter the system at the first host, it can potentially become overloaded. Other hosts can be under-utilised as a result.

There are potentially significant further gains that could be obtained if we can take the strengths of size-based approaches (such as negating the effects of highly-variable workloads) while addressing the weaknesses of these approaches (such as the assumption of *a priori* knowledge, and wasted processing from restarting many tasks).

3.6 Performance under heavy-tailed workloads

Let us consider the performance of a distributed server cluster, where our arrival rate is Poisson and our service time distribution follows a Bounded Pareto distribution (as discussed in Section 2.3.1). We set p (our largest task) to equal 10^7, and vary k (our smallest task) to keep the distributional mean, $E(X)$, fixed at 3000. These parameters are used in order to focus on the effect of the changing variance in the service time distribution.

Figure 3.3 depicts the expected waiting time and slowdown for a distributed system utilising either the Random or Dynamic policy. The system load is kept constant at $\rho = 0.5$ while the number of hosts is varied from 2 to 5. Alpha is varied from 0.5 to 2.0, ranging from extreme task size variation to moderate task size variation. The performance metrics are identical for the Random policy, regardless of the number of hosts. We can see that under the same conditions, when the number of hosts increases, the performance metrics of



(a) E(W) (b) E(S)

Figure 3.3: Performance of a distributed system with system load of 0.5. The number of hosts is increased while the system load *is kept constant at 0.5. The expected waiting time and slowdown are depicted in (a) and (b) respectively.*

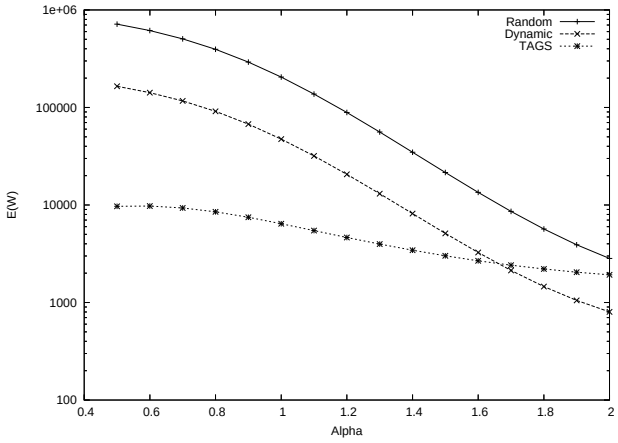
the Dynamic policy improves. This intuitively makes sense, as the Dynamic policy is known to improve utilisation, and reduces the probability of a host being idle. The Random policy simply performs random splitting on the arrival stream, and gives no consideration to the load on each back end host when assigning tasks.

Figure 3.4 shows a comparison of the Random, Dynamic, and TAGS policies in a two host distributed system. It is worth pointing out that the Dynamic policy (either Least-Work-Remaining or Shortest-Queue) cannot strictly be compared to the other policies listed, as it depends on state information (such as the cumulative workload or the queue length) being available at the dispatcher at all times. Furthermore, the cost of getting that information is not modelled. Regardless, we include it here for completeness and interest's sake. The system load is varied, showing performance where $\rho = 0.3$ (low load), $\rho = 0.5$ (moderate load) and $\rho = 0.7$ (high load).

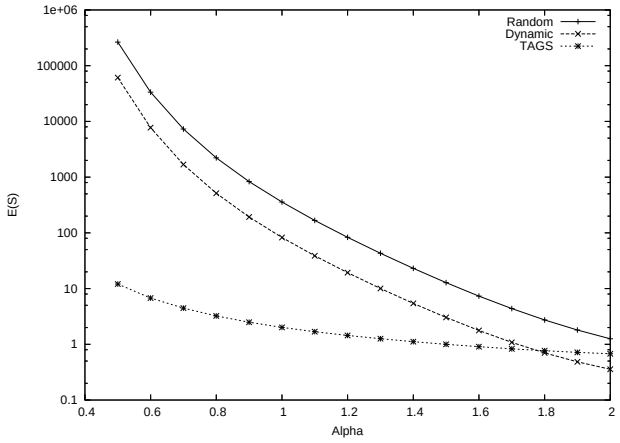
As discussed in Section 3.4.5 we can see that the performance metrics of both the Random and Dynamic policies are dependant on the variation of the task size distribution. As the variation increases (when α decreases) the expected waiting time and slowdown increase rapidly. Also, we note that the scale of improvement shown by the Dynamic policy over the Random policy decreases slightly as the system load increases. As the system load increases, there is a lower probability of a host being idle, even under the Random policy. As such the

Dynamic policy has less scope for improvement.

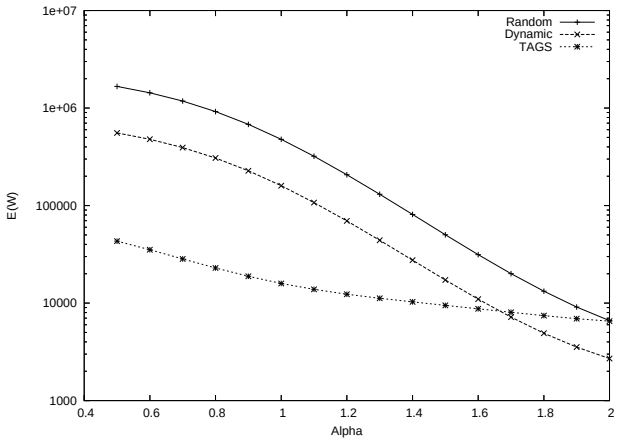
We can see an enormous improvement for the size-based task assignment policy shown (TAGS), especially under conditions of high and extreme task size variations. Such policies by their very nature reduce the variance of task sizes at each host, by partitioning the workload amongst each host. This has the effect of grouping similarly sized tasks together at the queues of each host, consequently reducing the variance at each host and improving performance metrics such as mean waiting time and slowdown.



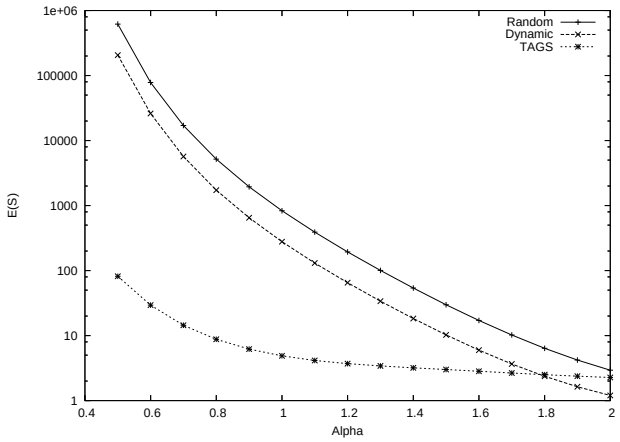
(a) E(W), $\rho = 0.3$



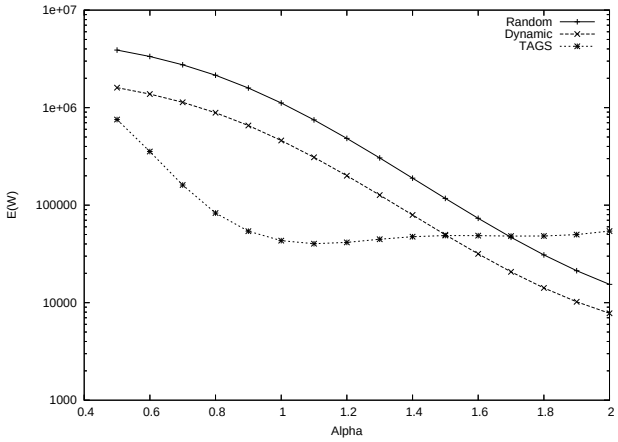
(b) E(S), $\rho = 0.3$



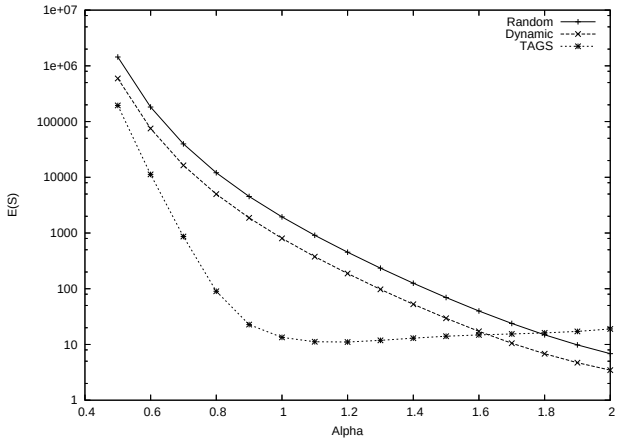
(c) E(W), $\rho = 0.5$



(c) E(S), $\rho = 0.5$



(d) E(W), $\rho = 0.7$



(e) E(S), $\rho = 0.7$

Figure 3.4: Performance of a two host distributed system with system load of 0.3, 0.5 and 0.7. The expected waiting time and slowdown are depicted under each load scenario.

Chapter 4

Task Assignment based on Prioritising Traffic Flow

In batch and scientific distributed computing domains, tasks are typically CPU-bound and often have very high memory requirements, precluding the use of work-conserving migration. The size of tasks in such systems are rarely known in advance, and can be difficult to estimate. These conditions pose challenging requirements to distributed system designers, especially when the workloads they experience are highly variable, as described in Section 2.3. Most existing task assignment policies perform poorly under such conditions, unable to deal with the negative effects of 'heavy-tailed' workloads (as demonstrated in Chapter 3). This chapter proposes a new load distribution approach, called TAPTF (Task Assignment based on Prioritising Traffic Flows) [Broberg et al., 2005] which deals with the inherent limitations of existing approaches under these particular application domains.

TAPTF improves performance under heavy-tailed workloads for certain classes of traffic by controlling the influx of tasks to each host. Tasks can potentially be dispatched to any host, rather than just the first host, as per the TAGS approach. This becomes crucial under two important scenarios. First, when task size variation decreases it becomes harder to exploit the so-called heavy-tailed properties. That is, the workload becomes more uniformly distributed and thus improved performance is gained from spreading the incoming tasks over multiple hosts. Second, when the system load is high, the first host (which in a TAGS system receives *all* incoming tasks) can become overloaded resulting in a decrease in the overall performance metrics (described in Section 2.2). Under a scenario of moderate to low variation *and* high system load, this technique is especially beneficial as observed in the

analytical comparison in Section 4.2.

TAPTF also introduces multiple queues with processing time limits ('cut-offs'). Each host has an ordinary queue which receives tasks directly from the dispatcher. Each host (excluding the first) also has a restart queue that receives tasks from the host above it. The use of dual queues (combined with cut-offs at each host) enables service differentiation at each host, allowing small tasks to be executed quickly without being delayed by larger tasks. To achieve this, tasks that exceed the cut-off on a given host are migrated to the next host's restart queue (to be restarted from scratch).

Unlike SITA-E/V/U and other size-based policies, TAPTF assumes no knowledge of the service requirements of incoming tasks. We are particularly interested in the areas that TAPTF can improve over TAGS, a policy that performs well when there is no pre-emption and task sizes are not known *a priori*. TAPTF is supported by a rigorous analytical model, based on fundamentals of queueing theory and priority queues which covers a wide spectrum of potentially different workload scenarios. This gives us great insight into the behaviour of tasks in the distributed system under the TAPTF policy. A detailed description of the TAPTF model is presented in Section 4.1. Section 4.2 gives an analytical comparison of TAPTF with existing approaches. Section 4.3 provides a detailed discussion of the analytical comparisons performed in Section 4.2. We conclude this chapter with some closing thoughts on the usefulness of the TAPTF approach in Section 4.4.

4.1 The Proposed Model - TAPTF

In this section we propose a new task assignment policy called TAPTF - Task Assignment based on Prioritising Traffic Flows - to address the limitations of existing approaches in dealing with certain classes of traffic. An overview of TAPTF is given in Section 4.1.1, detailing the motivation behind the TAPTF approach. Differences between it and existing models are highlighted, and important features are explained. In Section 4.1.2 the techniques used by TAPTF to improve performance are outlined. Section 4.1.3 provides a conceptual view of the proposed model. The important parameters associated with our model are defined and computed in Section 4.1.4. Section 4.1.5 describes how the task size cut-offs are chosen for each host.

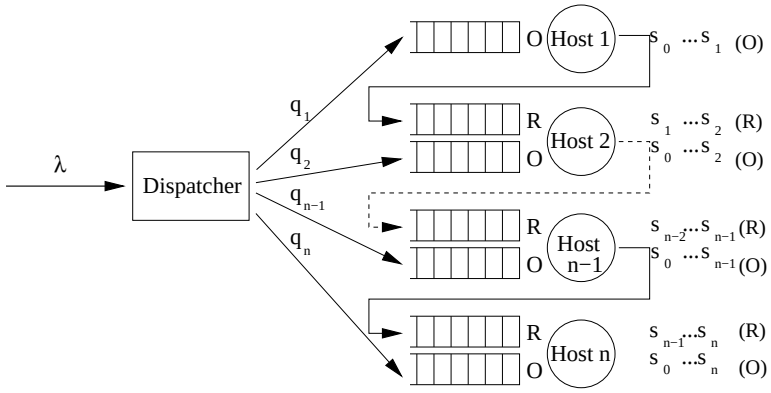


Figure 4.1: Illustration of the TAPTF model.

4.1.1 Motivation

Harchol-Balter's TAGS approach [Harchol-Balter, 2002], while seemingly counter-intuitive in many respects, proved to be a very effective task assignment policy for distributed systems. As such, TAGS provides an excellent point of comparison for any new task assignment policy operating under similar constraints. As described in Section 3.5.2, the TAGS policy has a number of desirable properties - the most important being that it does not assume any prior knowledge of the service requirement of incoming tasks, while still maintaining good performance. The TAGS policy performs admirably under realistic highly variable conditions, exploiting the heavy-tailed nature that is consistent with many computing workloads. Despite this, TAGS can produce significant *excess* at the back-end hosts - wasted processing that a task incurs (and the corresponding load placed on a host) when it has been placed in the incorrect queue and is subsequently restarted after exceeding the processing limit associated with a host. A task that is assigned incorrectly is penalised by being stopped, placed at the end of the next host's queue and restarted from scratch (upon reaching the front of that queue). These shortcomings are justified by the fact that, by the very nature of the heavy-tailed workload distribution, the tasks that are penalised can absorb the additional waiting and processing time *for the greater good*. Nonetheless, this is wasteful, but how can the efficiency be improved while still maintaining good performance? Three keys areas needed to be addressed:

- Reducing the variance of tasks that share the same queue.
- Reducing the penalty of wasted processing (*excess*) on the back-end hosts - caused by

tasks that do not complete their processing in time, and are restarted at another host ('hand-offs').

- Reducing the penalty on restarted tasks (given that a task could potentially be restarted multiple times).

The TAPTF policy was formulated to address these key issues. From Figure 4.1 we can see the introduction of dual queues at each host - an Ordinary (O) queue and a Restart (R) queue. Tasks can enter the system at any host. The influx of tasks that are assigned to each host is controlled by the fraction q_i. These additions allow the TAPTF model to be flexible, providing the means to manipulate the behaviour of the TAPTF model depending on the characteristics of the workload.

4.1.2 Techniques

In Section 4.1.1 a number of shortcomings of the TAGS model were identified that needed to be addressed. As such, TAPTF was designed in order to improve on these key areas. The reasoning behind the techniques that TAPTF uses to address the shortcomings of existing approaches are outlined as follows:

Variance reduction

It is desirable to reduce the effect of the task size variation, which has a significantly detrimental effect on performance as variability increases (as illustrated by the Pollaczek-Khinchin formula in Section 2.2). TAPTF reduces the variance in the sizes of tasks that share the same queue by the use of dual queues (an Ordinary (O) queue and a Restart (R) queue) and migration, in an effort to group like-sized tasks together. This is done in order to minimise the chance of a short task being stuck behind a long task in the same queue.

Reduce the number of hand-offs

The *excess* - extra work created by restarting many tasks from scratch - needs to be minimised. TAPTF attempts to reduce the amount of 'hand-offs' by placing as many tasks in the most appropriate queue (that is, their final destination) in the first instance as possible - reducing the penalty on both hosts and tasks. This is achieved in two interrelated ways. First, by manipulating the fraction of tasks (q_i) that is dispatched to each host, which has a follow-on effect of increasing the number of tasks that are correctly assigned to a suitable

host - that is, where they can run-to-completion. Secondly, the reason that it can enter the system (and potentially finish) at *any* host is due to the lower boundary cut-off of each Ordinary (O) queue being k, the smallest possible task size. In the TAGS system, a task that needs to be processed at Host i (e.g. its size is between s_{i-1} and s_i) must migrate from Host 1 to Host i. In TAPTF for the same task, there is a probability q_i that it will be directly dispatched there (an ideal choice), and a probability $q_i + q_{i+1} + ... + q_n$ that it be assigned to Host i or higher - where it will not be subjected to any hand-offs. This becomes more important as task size variation decreases.

Reducing the effect of hand-offs

While the number of hand-offs are reduced, they still will occur. Knowing this, we can minimise the detrimental effect of the hand-offs that do occur (specifically, on waiting time and slow down) by migrating restarted tasks via restart queues to their final destination. Depending on whether the goal is to optimise the overall performance metrics or to be 'fair' to the penalised tasks, the tasks could be fast tracked to their final destination by giving them priority of service at each host (over tasks in the ordinary queue received from the dispatcher). Note the default behaviour is the opposite, where tasks in the Ordinary queue have priority of service over tasks in the Restart queue.

Consider for a moment that each host in our distributed system is a $M/G/1$ FCFS queue (see Section 2.2). It can be seen that all performance metrics are dependent on $E(X^2)$, the second moment of the task size distribution. $E(X^2)$ is proportional to the variance in the size of tasks sharing the same queue. Even after we generalise the Pollaczek-Khinchin formula to priority queues (required by our dual queue implementation) this still holds true, as shown in Section 4.1.4. We can infer that reducing the variance in the service requirements of tasks at each host can improve performance, reducing the chance of a smaller task being stuck behind a significantly longer task.

4.1.3 A Conceptual view of the TAPTF model

As seen in Figure 4.1, tasks arrive at a central dispatcher, following a Poisson process with rate λ. The dispatcher assigns tasks (in a First-In-First-Out manner) to one of the n hosts (say, Host i, where $1 \leq i \leq n$) at random with probability q_i. For the purpose of analysis, we note that the arrival stream to host Host i is also a Poisson process with rate λq_i.

Due to the heavy-tailed characteristics of the task size distribution (as discussed in Sec-

tion 2.3.1), we assume that the distribution of task sizes (that is, the service distribution) follows a bounded Pareto Distribution $B(k, p, \alpha)$. A 'cut-off' (s_i) is assigned to each host in the distributed system. Specifically, tasks are processed on hosts with the following conditions:

- Host i's O queue deals only with tasks whose sizes are in the range $[k, s_i], 1 \leq i \leq n$
- Host i's R queue deals only with tasks whose sizes are in the range $[s_{i-1}, s_i], 1 < i \leq n$

where $k = s_0 < s_1 < s_2 < s_3 < \ldots < s_n = p$. These cut-offs can be computed in order to minimise certain measurable quantities such as mean waiting time or mean slowdown time. Further information on how the cut-offs are chosen is provided in Section 4.1.5.

Each host (excluding Host 1) provides two queues, an ordinary queue and a restart queue (denoted by O and R respectively). All tasks in the O and R queues are served on a First-Come-First-Served (FCFS) basis. Tasks sent to a given host from the dispatcher join that host's O queue. After a task has moved to the front of the queue it can begin to be processed. If the processing time of a task on a given host exceeds the assigned cut-off limit, the task is stopped, and moved to the restart (R) queue belonging to the next host. This process is repeated until these tasks run to completion at their final (correct) destination. Tasks waiting in an O queue have priority of service over those in the R queue at a given host. However, a task which is being served from the R queue will not be pre-empted from service by the arrival of a task into the O queue at a given host. This is the default behaviour of the TAPTF policy (and is denoted as TAPTF-O in the figures in Section 4.2).

One way the TAPTF model differs from TAGS is the fixed lower size boundaries at each host ($k = s_0$), so that all tasks with sizes *less than or equal* to a fixed cut-off point can be potentially be processed on a particular host. This means that a task can be dispatched to *any* host initially without being first dispatched to Host 1 (as per the TAGS approach) while preserving the property that a task's service demand is not known *a priori*. In addition, TAPTF uses dual queues at each host in order to speed up the flow of shorter tasks, allowing smaller tasks to be processed quickly in the ordinary queue and migrating larger tasks out of the way, allowing them to group together in the restart queues at subsequent hosts.

4.1.4 Mathematical Preliminaries for the TAPTF model

In this section, we define and compute all the important parameters associated with the TAPTF model. The main objective is to use them to determine the optimal cut-off points

n	Number of hosts in the system
$B(k, p, \alpha)$	Bounded Pareto task size distribution
k	Lower bound of task size distribution
p	Upper bound of task size distribution
$f(x)$	Probability density function for $B(k, p, \alpha)$
α	Heavy-tailed parameter
s_i	Task size cut-off for Host i
q_i	Fraction of tasks dispatched to Host i
λ	Outside task arrival rate into system
ρ	System load
p_i	Fraction of tasks whose final destination (the host it runs to completion on) is either Host i or its predecessors
h_{iO}	Fraction of tasks that *visit* Host i's ordinary (O) queue
h_{iR}	Fraction of tasks that *visit* Host i's restart (R) queue.
h'_{iO}	Fraction of tasks whose *final* destination is Host i's ordinary (O) queue
h'_{iR}	Fraction of tasks whose *final* destination is Host i's restart (R) queue
$E(X_{iO}^j)$	jth moment of the distribution of tasks whose final destination is Host i's ordinary (O) queue
$E(X_{iR}^j)$	jth moment of the distribution of tasks whose final destination is Host i's restart (R) queue
$E(hostX_{iO}^j)$	jth moment of the distribution of tasks who spent time in Host i's ordinary (O) queue
$E(hostX_{iR}^j)$	jth moment of the distribution of tasks who spent time in Host i's restart (R) queue
λ_{iO}	Arrival rate into Host i's ordinary (O) queue
λ_{iR}	Arrival rate into Host i's restart (R) queue
ρ_{iO}	Load at Host i's ordinary (O) queue
ρ_{iR}	Load at Host i's restart (R) queue
$E(hostW_{iO})$	Expected waiting time for a task at Host i's ordinary (O) queue
$E(hostW_{iR})$	Expected waiting time for a task at Host i's restart (R) queue
$E(W_{iO})$	Expected waiting time of a task whose final destination is Host i's ordinary (O) queue
$E(W_{iR})$	Expected waiting time of a task whose final destination is Host i's restart (R) queue
$E(S_{iO})$	Expected slowdown at Host i's ordinary (O) queue
$E(S_{iR})$	Expected slowdown at Host i's restart (R) queue

Table 4.1: Notation for TAPTF Model

(where $k = s_0 < s_1 < s_2 < \ldots < s_{n-1} < s_n = p$) corresponding to the minimum mean waiting time or slowdown for tasks entering the distributed system. It is worth noting that that the results below reduce to that obtained for the TAGS policy [Harchol-Balter, 2002] when $q_1 = 1$ and there are no ordinary queues. The notation for the TAPTF model is given in Table 4.1.

Recall an important observation from Section 2.3.1 - that many computing workloads have been found to have 'heavy-tailed' characteristics. For the purpose of analysis we use a Bounded Pareto distribution, which still exhibits the requisite properties that are consistent with 'heavy-tailed' workloads. The probability density function for the Bounded Pareto $B(k, p, \alpha)$ is shown in Equation (2.1) in Section 2.3.1.

We now wish to define p_i, the fraction of tasks whose final destination (that is, the host it runs to completion on) is either Host i or its predecessors. This is given by:

$$\begin{aligned} p_i &= P(X \leq s_i) \\ &= \frac{\alpha k^\alpha}{1-(k/p)^\alpha} \int_k^{s_i} x^{-\alpha-1} dx \\ &= \frac{1-(k/s_i)^\alpha}{1-(k/p)^\alpha}. \end{aligned} \tag{4.1}$$

The fraction of tasks that *visit* Host i's ordinary (O) queue, h_{iO}, is simply q_i, as the O queue on a given host only contains tasks received directly from the dispatcher, therefore:

$$h_{iO} = q_i \tag{4.2}$$

We define h_{iR} as being the fraction of jobs that *visit* Host i's restart (R) queue. As can be observed from Figure 4.1, h_{1R} is undefined as there is no restart queue at Host 1. Clearly $h_{2R} = q_1(1-p_1), h_{3R} = q_1(1-p_2) + q_2(1-p_2), \ldots$ and in general, for $2 \leq i \leq n$:

$$\begin{aligned} h_{iR} &= \sum_{j=1}^{i-1} q_j(1-p_{i-1}) \\ &= (1-p_{i-1}) \sum_{j=1}^{i-1} q_j. \end{aligned} \tag{4.3}$$

The fraction of tasks whose *final* destination is Host i's ordinary (O) queue is denoted by h'_{iO}, and is simply the product of q_i, the probability of a task being assigned to Host i, and

p_i, the fraction of tasks whose final destination is either Host i or its predecessors. Thus:

$$h'_{iO} = q_i p_i \tag{4.4}$$

The fraction of tasks whose *final* destination is Host i's restart (R) queue is denoted by h'_{iR}. Clearly $h'_{2R} = q_1(p_2 - p_1), h'_{3R} = q_1(p_3 - p_2) + q_2(p_3 - p_2), \ldots$ and in general, for $2 \leq i \leq n$:

$$\begin{aligned} h'_{iR} &= \sum_{j=1}^{i-1} q_j(p_i - p_{i-1}) \\ &= (p_i - p_{i-1}) \sum_{j=1}^{i-1} q_j. \end{aligned} \tag{4.5}$$

Consider those tasks that finish up in Host i's queues - that is, they have run-to-completion at Host i. Let $E(X_{iO}^j)$ and $E(X_{iR}^j)$ be the jth moment of the distribution of tasks whose final destination is Host i's ordinary (O) queue and Host i's restart (R) queue respectively. We have:

$$E(X_{iO}^j) = \frac{1}{p_i} \int_k^{s_i} x^j f(x) dx \tag{4.6}$$

$$= \begin{cases} \frac{\alpha s_i^j ((\frac{k}{s_i})^\alpha - (\frac{k}{s_i})^j)}{p_i(j-\alpha)(1-(\frac{k}{p})^\alpha)} & \text{if } j \neq \alpha \\ \frac{\alpha k^\alpha \ln(s_i/k)}{p_i(1-(\frac{k}{p})^\alpha)} & \text{otherwise} \end{cases} \tag{4.7}$$

and

$$E(X_{iR}^j) = \frac{1}{p_i - p_{i-1}} \int_{s_{i-1}}^{s_i} x^j f(x) dx \tag{4.8}$$

$$= \begin{cases} \frac{\alpha(s_i^j(\frac{k}{s_i})^\alpha - s_{i-1}^j(\frac{k}{s_{i-1}})^\alpha)}{(p_i - p_{i-1})(j-\alpha)(1-(\frac{k}{p})^\alpha)} & \text{if } j \neq \alpha \\ \frac{\alpha k^\alpha \ln(s_i/s_{i-1})}{(p_i - p_{i-1})(1-(\frac{k}{p})^\alpha)} & \text{otherwise.} \end{cases} \tag{4.9}$$

Now, consider the tasks that spent time in Host i's queues. Let $E(hostX_{iO}^j)$ and $E(hostX_{iR}^j)$ be the jth moment of the distribution of tasks who spent time in Host i's ordinary (O) queue and Host i's restart (R) queue respectively. Note that the fraction of tasks which leave Host i's O queue is $q_i - q_i p_i$, therefore we have:

$$E(hostX_{iO}^j) = \frac{h'_{iO}}{h_{iO}} E(X_{iO}^j) + \frac{h_{iO} - h'_{iO}}{h_{iO}} s_i^j. \tag{4.10}$$

Similarly,

$$E(hostX_{iR}^{j}) = \frac{h'_{iR}}{h_{iR}}E(X_{iR}^{j}) + \frac{h_{iR} - h'_{iR}}{h_{iR}}s_i^j. \tag{4.11}$$

Let the arrival rate into Host i's ordinary (O) and restart (R) queue be denoted by λ_{iO} and λ_{iR} respectively. Then, based on similar principles to (4.2) and (4.3),

$$\lambda_{iO} = \lambda q_i \tag{4.12}$$

$$\text{and} \quad \lambda_{iR} = \lambda h_{iR}. \tag{4.13}$$

The loads at Host i's O and R queue are:

$$\rho_{iO} = \lambda_{iO}E(hostX_{iO}) \tag{4.14}$$

$$\text{and} \quad \rho_{iR} = \lambda_{iR}E(hostX_{iR}) \tag{4.15}$$

respectively.

In the TAPTF model, we can choose to prioritise tasks that are in the restart queue (specifically 'hand-offs' that are received from the host above it) over tasks in the ordinary queue. Alternatively, we can choose to give tasks in the ordinary queue (which are received directly from the dispatcher) priority of service over tasks in the restart queue (which in effect becomes a *low* priority queue). As mentioned in Section 4.1.3, the latter is the default behaviour of the TAPTF model, as it provides the best performance.

The next set of results, concerning the expected waiting times for tasks in Host i's ordinary (O) and restart (R) queues, relies on some key facts:

- Tasks in the O queue have priority of service over tasks in the R queue
- A task in service at the R queue will not be pre-empted from service by a task which subsequently arrives into the O queue
- Within each queue, tasks are processed on a FCFS basis.
- Finally, we will have to assume that tasks that arrive into the R queues form Poisson processes.[1]

[1] As discussed earlier, arrival streams into O queues are Poisson processes. This is not the case with arrivals into R queues, which (as noted by Harchol-Balter) are less bursty (i.e. more uniformly random), than those of a Poisson process. If the task sizes are exponentially distributed, then the output of a queue will be Poisson but not otherwise.

Let $E(hostW_{iO})$ and $E(hostW_{iR})$ be the expected waiting time for a task at Host i's ordinary (O) and restart (R) queue respectively. Then using a classical result on non pre-emptive priority queue by A. Cobham [Cobham, 1953] we obtain:

$$E(hostW_{iO}) = \frac{\lambda_{iO} E(hostX_{iO}^2)}{2(1-\sigma_1)} \tag{4.16}$$

$$\text{and} \quad E(hostW_{iR}) = \frac{\lambda_{iR} E(hostX_{iR}^2)}{2(1-\sigma_1)(1-\sigma_2)} \tag{4.17}$$

where $\sigma_1 = \rho_{iO}$ and $\sigma_2 = \rho_{iO} + \rho_{iR}$.[2] The above results assume that $0 < \sigma_2 < 1$

We will now obtain formulae for the two main operational measures of our distributed system: the expected waiting time and slowdown for tasks entering the system. Let $E(W_{iO})$ and $E(W_{iR})$ be the expected waiting time of a task whose final destination is Host i's O and R queue respectively. We have, for $1 \leq i \leq n$

$$E(W_{iO}) = h'_{iO} E(hostW_{iO}) \tag{4.18}$$

and, for $2 \leq i \leq n$,

$$E(W_{iR}) = h'_{iR} \sum_{m=1}^{i-1} q_m [E(hostW_{mO}) + \sum_{j=m+1}^{i} E(hostW_{jR})]. \tag{4.19}$$

The explanation for the last equation is as follows: for a task whose final destination is Host i's R queue, it must have first entered a Host j's O queue where $1 \leq j \leq i-1$ and then migrates down successive R queues where it finally runs-to-completion at Host i's R queue. Along the way, the task accumulates waiting times as it navigates its way to its final destination.

Let $E(S_{iO})$ and $E(S_{iR})$ be the expected slowdown at Host i's O and R queue respectively. As explained in Section 2.2, the slowdown is characterised by the waiting time divided by the processing time. We consider the slowdown to be a critical metric as it is desirable for a tasks waiting time to be proportional to its processing time - that is, a task with minimal processing requirements should only wait a small amount, but a task with greater processing requirements can absorb a longer waiting time. Then, by definition,

$$E(S_{iO}) = E(W_{iO}) E(1/X_{iO}) \tag{4.20}$$

$$\text{and} \quad E(S_{iR}) = E(W_{iR}) E(1/X_{iR}). \tag{4.21}$$

[2] These results are generalisation of the famous Pollaczek-Khinchin formula to priority queues.

4.1.5 Choosing the cut-offs

Like most size-based (or similar) policies, the performance of TAPTF is critically dependent on the choice of cut-offs used. From Section 3.5.2 we recall that cut-offs refer to the size-range associated with each host. The cut-offs can be chosen to optimise for mean waiting time, or mean slowdown. In order to optimise for mean waiting time, the load must be balanced more evenly amongst the host. To optimise for mean slowdown, load unbalancing techniques are employed, especially under conditions of high task size variation. We have chosen to optimise for both mean waiting time and more importantly, mean slowdown, as it is desirable for a tasks delay to be proportional to its service requirement.

The cut-offs for TAPTF are chosen in a similar fashion as the TAGS approach. The cut-offs for TAPTF and TAGS are a function of the task size distribution (in our case defined by the Bounded Pareto $B(k, p, \alpha)$) and the task arrival rate into the distributed system, λ. These parameters can be determined by observing the distributed system for a period of time. Optimal mean waiting time and mean slowdown for the case of two and three hosts can be obtained for TAGS, given the above parameters, by solving for the optimal values of the cut-offs using *Mathematica* [Wolfram Research, 2003] (as described in the appendix of Harchol-Balter's work on TAGS [Harchol-Balter, 2002]). For the case of four or more hosts, the cut-offs need to be tuned by hand, but the same results can be achieved by following some simple rules of thumb [Harchol-Balter, 2002].

Using the mathematical results described in Section 4.1.4, we too can work towards obtaining optimal cut-off points (s_i) for each of our hosts in the TAPTF system. Since our aim is to produce a task assignment policy that minimises the overall expected waiting time or slowdown respectively (depending on our goals), the following optimisation problems need to be addressed:

$$\text{Problem I} \qquad \begin{aligned} &\text{Minimize} \sum_{i=1}^{n} E(W_{iO}) + \sum_{i=2}^{n} E(W_{iR}) \\ &\text{Subject to} \, \rho_{iO} + \rho_{iR} < 1, 1 \leq i \leq n. \end{aligned}$$

$$\text{Problem II} \qquad \begin{aligned} &\text{Minimize} \sum_{i=1}^{n} E(S_{iO}) + \sum_{i=2}^{n} E(S_{iR}) \\ &\text{Subject to} \, \rho_{iO} + \rho_{iR} < 1, 1 \leq i \leq n. \end{aligned}$$

Now the optimisation problem has been defined, we can choose to optimise for mean

waiting time (described by Problem I), mean slowdown (described in Problem II) or a combination of the two.

As described above, the choice of cut-offs depend on the task size variability. From Section 2.3.1 we recall that the lower the α parameter, the higher the variability, and the smaller the percentage of tasks is that makes up 50% of the load. TAGS (and subsequently TAPTF, which can behave like TAGS by setting $q_1 = 1.0$ when prudent) can exploit this property of the heavy-tailed distribution by running all (or the vast majority) of the (small) tasks on the first host, leaving them under light to moderate load, while the largest tasks filter down to be eventually processed by the latter hosts.

As the variability decreases (α increases) we can no longer exploit the heavy-tailed property so easily. The average size of the tasks we consider 'small' slowly gets bigger as α increases. As such we have to choose our cut-offs accordingly, as well as manipulating the fraction of tasks that are assigned to the latter hosts. We still exploit the heavy-tailed property by processing larger jobs on the latter hosts, but we are not unbalancing the load to the extent we could when variability was higher ($\alpha \leq 1$). As α approaches 2.0, the task size variation is lower, and the other hosts have to start pulling their weight in order to maintain good mean waiting time and slowdown. TAPTF exploits this knowledge to provide better performance in those areas.

4.2 Analytical Comparison

In order to gauge the usefulness of the TAPTF approach, an analytical comparison with TAGS and Random was performed. Random is included as a baseline, whereas TAGS provides the best point of comparison as it operates under similar constraints to TAPTF, where no *a priori* knowledge of a task's service requirement is assumed. These approaches were evaluated under a variety of conditions and their performance compared using the most important of the metrics discussed in Section 2.2 - mean waiting time and mean slowdown.

A range of α values were considered, from 0.5 to 2.0, demonstrating a wide range of task size variation, from extreme task size variation ($\alpha \approx 0.5$) to low task size variation ($\alpha \approx 2.0$), and everything in between. To observe the effect of changing variance, the mean of the Bounded Pareto distribution is fixed at 3000, and the the maximum value p is set at 10^7. In order to keep the mean fixed, the minimum value k is varied as the α parameter changes.

Each α value was evaluated for different system loads (ρ) - 0.3 (low load), 0.5 (moderate load) and 0.7 (high load). These comparisons were performed for two and three host systems,

α	q_1	q_2
1.0	0.98	0.02
1.1	0.97	0.03
1.2	0.96	0.04
1.3	0.95	0.05
1.4	0.93	0.07
1.5	0.90	0.10
1.6	0.87	0.13
1.7	0.84	0.16
1.8	0.80	0.20
1.9	0.76	0.24
2.0	0.73	0.27

(a) $E(W)$

α	q_1	q_2
1.0	1.00	0.00
1.1	0.99	0.01
1.2	0.99	0.01
1.3	0.98	0.02
1.4	0.96	0.04
1.5	0.94	0.06
1.6	0.91	0.09
1.7	0.88	0.12
1.8	0.84	0.16
1.9	0.80	0.20
2.0	0.76	0.24

(b) $E(S)$

Figure 4.2: Distribution of tasks in TAPTF - 2 Hosts, $\rho = 0.3$

after which we could no longer find optimum s_i values with the computational resources available to us. This is not a big problem in itself as noted in prior research [Harchol-Balter, 2002], as an n Host distributed system (where $n > 2$) with a system load ρ can always be arranged in such a way to provide performance that is comparable or even better than the best performance of a two or three host system (where n is a multiple of two or three respectively) with system load ρ. For instance, a 4 Host system (with two subsystems containing 2 hosts each) will behave identically to a standard 2 Host system. That is, the performance characteristics of one subsystem in this scenario will be the same as the whole 2 Host system. The same would apply to a 6 Host system, (with two subsystems containing 3 hosts) and a standard 3 Host system. This holds true for any task assignment policy.

The analytical comparison was performed in Mathematica 5.0 [Wolfram Research, 2003], using the mathematical preliminaries discussed in the Section 4.1.4. The generalised TAPTF mathematical model is also used to model the behaviour of TAGS by setting $q_1 = 1.0$ (and subsequently q_2 ... q_n to equal 0) - negating the dual queues and multiple entry points and making it behave identically to TAGS. For each scenario, optimum cut-offs are found with respect to mean waiting time and mean slowdown for both TAPTF and TAGS using the NMinimize function in Mathematica to produce the best (and fairest) comparison. NMinimize searches numerically for the s_i values in each instance that produce local minima for

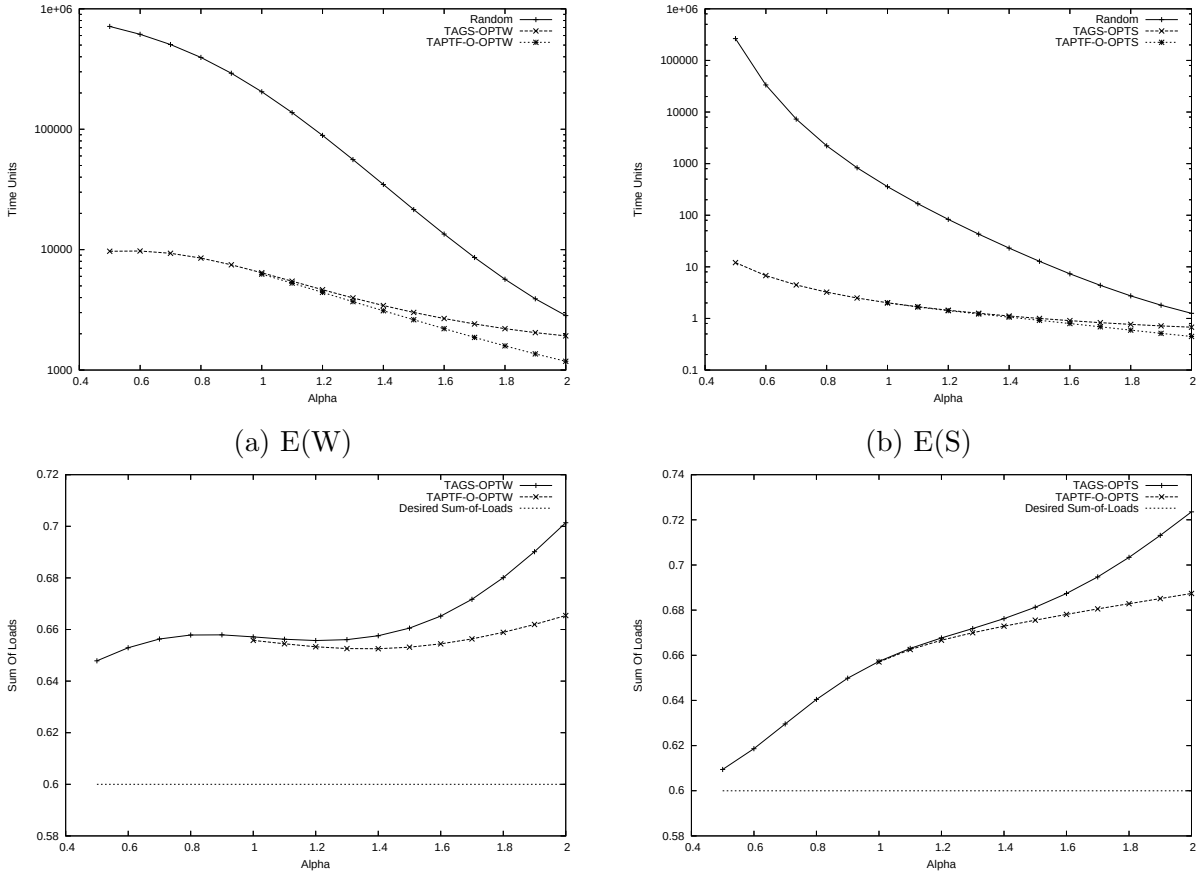
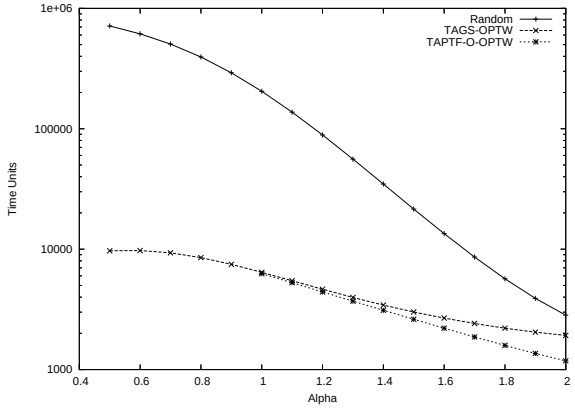


Figure 4.3: Performance of a two host distributed system with system load of 0.3. The expected waiting time and slowdown are depicted in (a) and (b) for policies optimised for these respective metrics. Likewise, corresponding load comparisons (desired versus actual Sum-Of-Loads) are shown in (c) and (d).

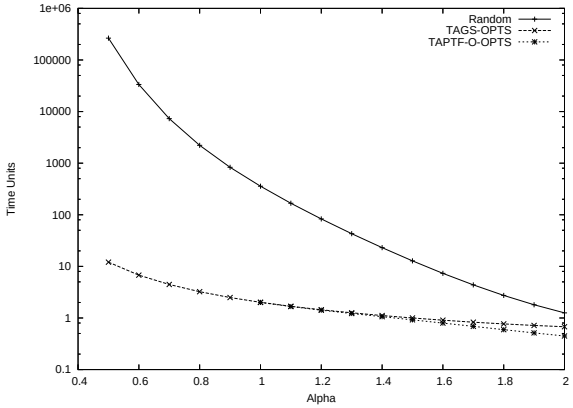
the expected waiting time, $E(W)$ and the expected mean slowdown, $E(S)$.

The choice of q_i parameters has a significant effect on the performance of the TAPTF policy. In the case of 2 hosts, we can search numerically for the combinations of s_i *and* the q_i values that result in the best results (i.e. local minima) for the expected waiting time and slowdown. In the case of 3 hosts, we must tune the q_i parameters by hand. This is not as problematic as it seems, as we can use our intuition regarding the required spread of tasks as well as the near-optimal results obtained in the 2 host scenarios to guide our choices.

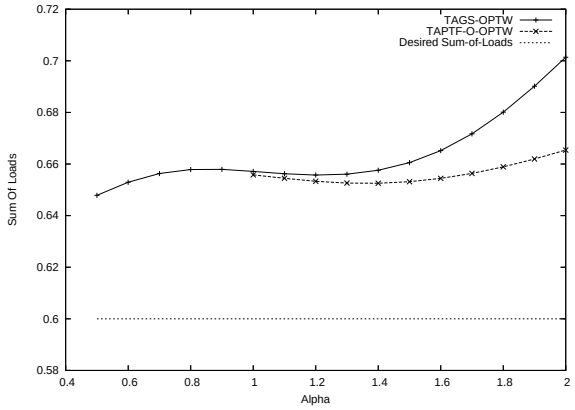
Task assignment policies that assume *a priori* knowledge of task sizes (e.g. SITA-E/V/U)



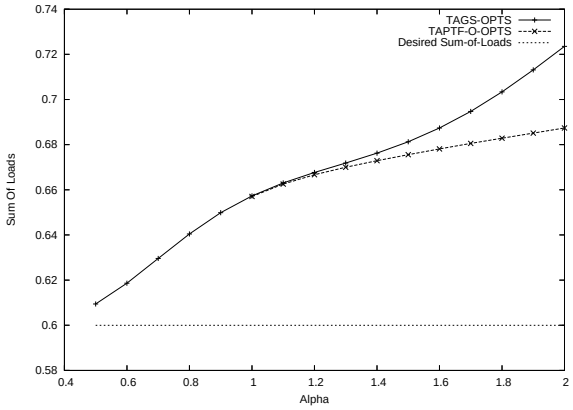
(a) E(Q) - OPTW



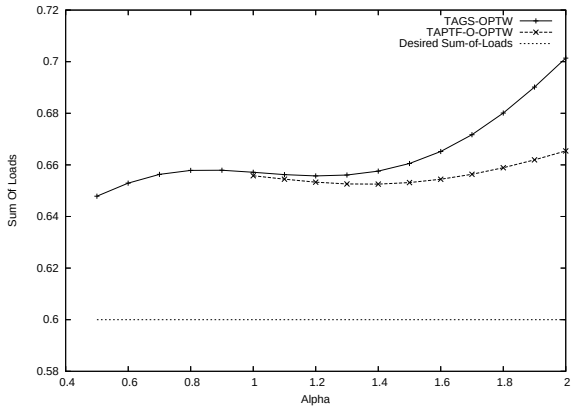
(b) E(Q) - OPTS



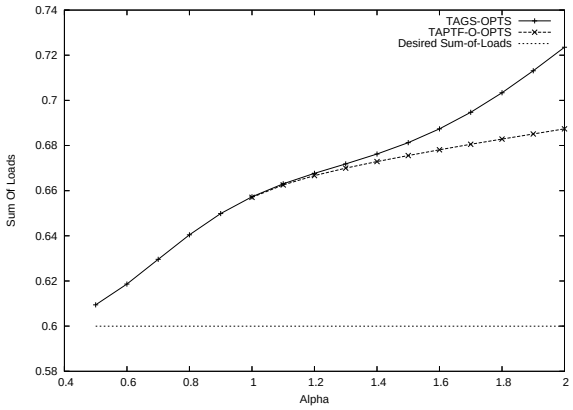
(c) E(X) - OPTW



(d) E(X) - OPTS



(e) ρ - OPTW



(f) ρ - OPTS

Figure 4.4: Per queue metrics for a two host distributed system with system load of 0.3. The expected queue length are depicted in (a) and (b), while the expected task sizes are depicted in (c) and (d). Corresponding load comparisons for each queue are shown in (e) and (f).

α	q_1	q_2
0.9	0.98	0.02
1.0	0.97	0.03
1.1	0.96	0.04
1.2	0.94	0.06
1.3	0.91	0.09
1.4	0.88	0.12
1.5	0.85	0.15
1.6	0.81	0.19
1.7	0.78	0.22
1.8	0.74	0.26
1.9	0.71	0.29
2.0	0.68	0.32

(a) $E(W)$

α	q_1	q_2
0.9	1.00	0.00
1.0	0.99	0.01
1.1	0.99	0.01
1.2	0.98	0.02
1.3	0.96	0.04
1.4	0.93	0.07
1.5	0.90	0.10
1.6	0.86	0.14
1.7	0.82	0.18
1.8	0.78	0.22
1.9	0.74	0.26
2.0	0.71	0.29

(b) $E(S)$

Figure 4.5: Distribution of tasks in TAPTF - 2 Hosts, $\rho = 0.5$

are not evaluated in this section, as we are motivated by a more pessimistic (and less restrictive) view of the distributed (cluster) model, where this information is not guaranteed to be available. The Least-Work-Remaining and Central-Queue policies (which have been shown to be equivalent) are omitted for two key reasons. First, these policies do not fit the assumptions of the problem domain discussed in Section 4.1.1. Second, while these policies are considered by many as being suitable for conditions of low to moderate variation, previous work [Harchol-Balter, 2002] has shown only a moderate increase in performance over Random under a similar evaluation to that performed in this chapter. The same study showed the TAGS algorithm outperforming a Least-Work-Remaining policy in nearly all scenarios (both low and high variation) considered. Thus, it is prudent to focus our attention on TAGS.

In the interests of clear and meaningful results, comparisons of mean waiting time and mean slowdown are performed using the respective TAPTF and TAGS policies optimised for that metric, as described in Section 4.1.5. The Random policy is included as a baseline for comparative purposes in each instance. It is worth noting that the expected waiting time and slowdown graphs are presented on a log scale for the y-axis.

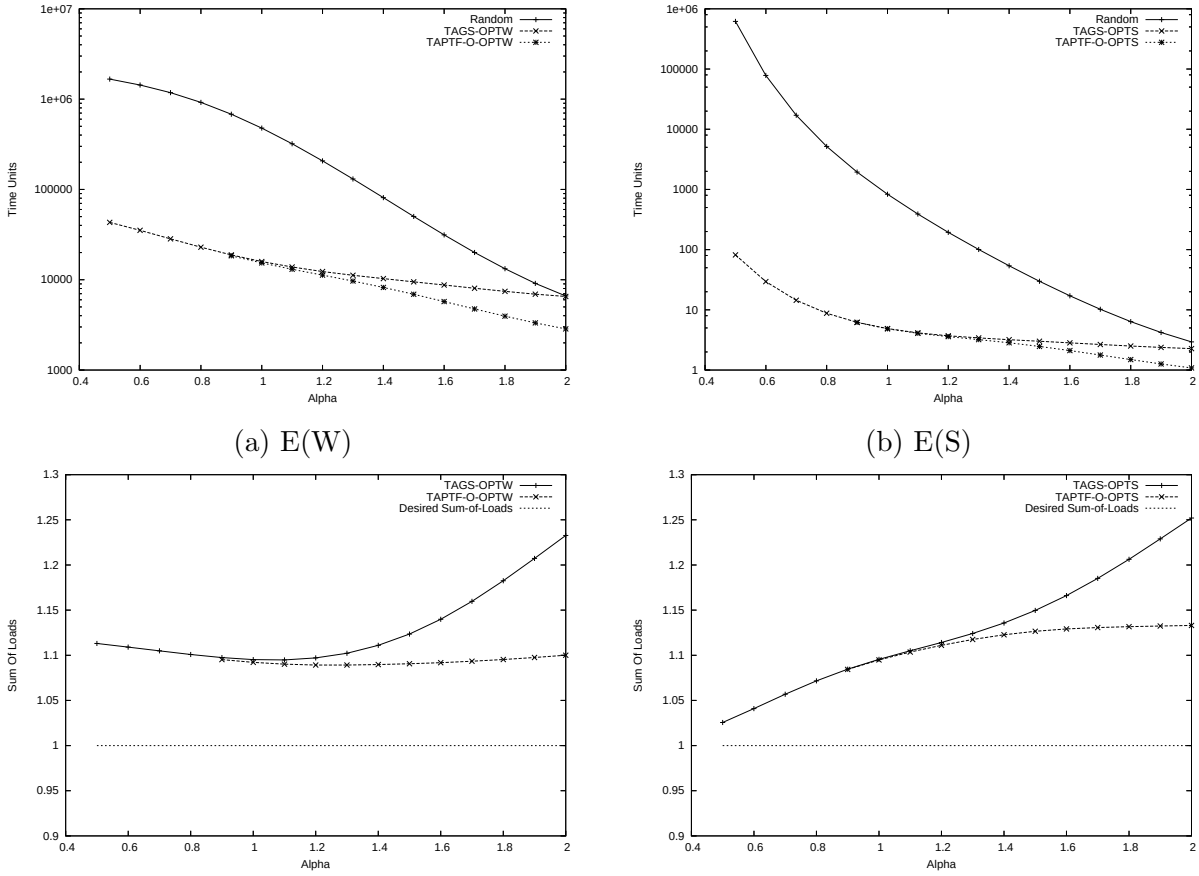


Figure 4.6: Performance of a two host distributed system with system load of 0.5. The expected waiting time and slowdown are depicted in (a) and (b) for policies optimised for these respective metrics. Likewise, corresponding load comparisons (desired versus actual Sum-Of-Loads) are shown in (c) and (d).

4.2.1 Two Hosts

An analytical comparison of TAGS and TAPTF in a two host distributed system is presented in this section. The results show the performance metrics for a system load of 0.3 (Figure 4.3), 0.5 (Figure 4.6) and 0.7 (Figure 4.8). Results for TAPTF are only shown where they are better than TAGS, as TAPTF can reduce to TAGS (and achieve identical performance) as described in Section 4.1 and Section 4.2.

Figures 4.3(a) and 4.3(b) show the mean waiting time and slowdown respectively under a low system load ($\rho = 0.3$). From our analysis the TAGS policy achieves better mean

α	q_1	q_2
0.9	0.97	0.03
1.0	0.95	0.05
1.1	0.92	0.08
1.2	0.88	0.12
1.3	0.84	0.16
1.4	0.79	0.21
1.5	0.76	0.24
1.6	0.72	0.28
1.7	0.69	0.31
1.8	0.67	0.33
1.9	0.64	0.36
2.0	0.62	0.38

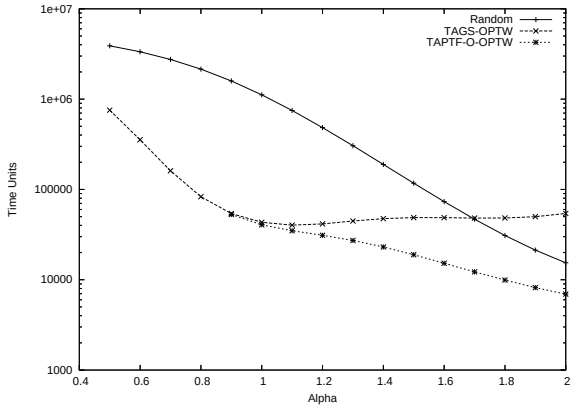
(a) $E(W)$

α	q_1	q_2
0.9	1.00	0.00
1.0	0.99	0.01
1.1	0.98	0.02
1.2	0.95	0.05
1.3	0.91	0.09
1.4	0.86	0.14
1.5	0.81	0.19
1.6	0.77	0.23
1.7	0.73	0.27
1.8	0.70	0.30
1.9	0.67	0.33
2.0	0.64	0.36

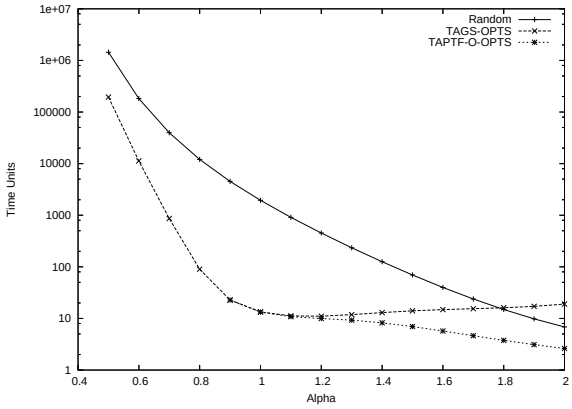
(b) $E(S)$

Figure 4.7: Distribution of tasks in TAPTF - 2 Hosts, $\rho = 0.7$

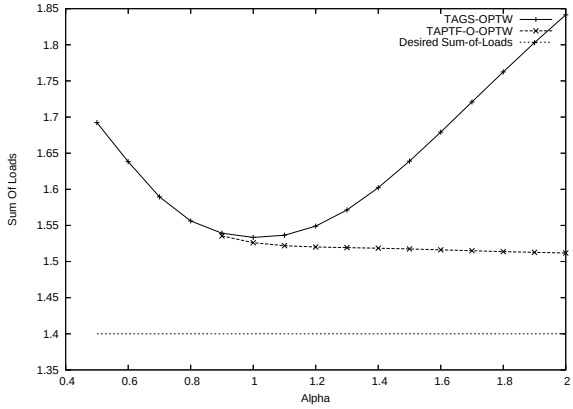
waiting time and slowdown under conditions ranging from extreme to high variation (where α is between 0.5 and 1.0). The areas where the TAPTF policy improves on TAGS are highlighted on the graphs. It can be observed that in conditions of moderate to low variation (where α is between 1.1 and 2.0), the TAPTF policy achieves better performance with respect to mean waiting time and slowdown. This performance increase can be attributed to the use of dual queues and by assigning tasks to all servers (or a subset thereof) rather than feeding all tasks into the first host, as per the TAGS approach. Figure 4.2 gives a breakdown of the fraction of tasks dispatched to Host 1 (denoted by q_1) or Host 2 (denoted by q_2). From the table we can see that as variation increases (and α decreases) TAPTF approaches TAGS-like behaviour for optimal performance. We can see when optimised for waiting time (where $\alpha = 1.0$), almost all tasks (98%) are dispatched to Host 1. As variation increases further (where α is between 0.5 and 1.2) TAGS-like behaviour produces the best results. Conversely, when variation decreases it pays to assign some tasks to the second host. As the variation decreases (and α approaches 2.0) we can afford to assign more tasks to the second host. Figures 4.3(c) and 4.3(d) again highlight the effect of decreasing variance on TAGS - as α decreases, the amount of excess load generated by the TAGS policy increases significantly, while the TAPTF maintains consistent load. As the fraction assigned to Host



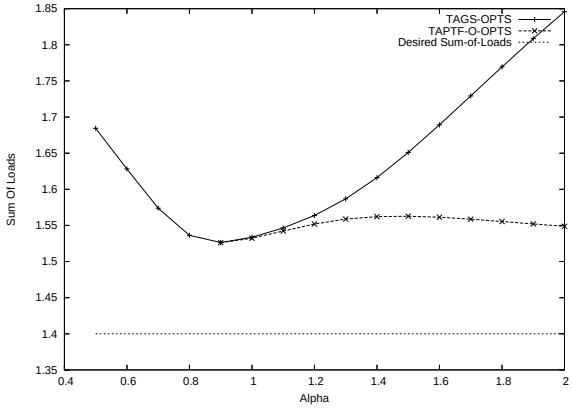
(a) E(W) - $\rho = 0.7$



(v) E(S) - $\rho = 0.7$



(c) Sum-of-Loads



(d) Sum-of-Loads

Figure 4.8: Performance of a two host distributed system with system load of 0.7. The expected waiting time and slowdown are depicted in (a) and (b) for policies optimised for these respective metrics. Likewise, corresponding load comparisons (desired versus actual Sum-Of-Loads) are shown in (c) and (d).

2 (q_2) increases, so to does the factor of improvement over TAGS, both in expected waiting time and slowdown in addition to system load.

Figures 4.6(a) and 4.6(b) depict the mean waiting time and slowdown respectively under a moderate system load ($\rho = 0.5$). Under this increased system load we can see that the performance of the TAPTF policy is better than TAGS over a larger range of task variation scenarios (where α is between 1.0 and 2.0) than under a system load of 0.3. Figure 4.5 gives a breakdown of the fraction of tasks dispatched to each back-end host in the distributed system. As the system load has increased, a greater fraction of tasks are now being dispatched to

the second host in order to maintain better performance than TAGS with respect to mean waiting time and mean slowdown. From Figures 4.6(c) and 4.6(d) a significant rise in the system load (and thus excess) can be observed as the task size distribution becomes less variable. Conversely, the TAPTF maintains a consistent system load over all observed areas. Again we see that as the fraction assigned to Host 2 (q_2) increases (and α approaches 2.0) the factor of improvement in all metrics over TAGS gets larger.

Figures 4.8(a) and 4.8(b) show the mean waiting time and slowdown respectively under a high system load ($\rho = 0.7$). The TAPTF policy betters TAGS over a larger range of task variation scenarios than occurred under low load (with TAPTF demonstrating lower mean waiting time and slowdown where α is between 0.9 and 2.0). It can be observed that TAGS suffers significantly under a high system load. As highlighted in Figure 4.7 we are seeing an increased fraction of tasks dispatched to the second host in order to maintain superior performance to the TAGS policy. From Figures 4.8(c) and 4.8(d) we can observe a sharp increase in system load (and subsequently excess) where $\alpha < 1.0$. It can be seen that as α approaches 2.0 the factor of improvement over TAGS increases in all metrics. Indeed, under this scenario of high system load (where $\rho = 0.7$), we have observed the biggest factor of improvement yet over TAGS.

α	q_1	q_2	q_3
0.9	1.0	0.0	0.0
1.0	1.0	0.0	0.0
1.1	0.9	0.1	0.0
1.2	0.8	0.2	0.0
1.3	0.75	0.25	0.0
1.4	0.7	0.3	0.0
1.5	0.6	0.4	0.0
1.6	0.6	0.4	0.0
1.7	0.6	0.3	0.1
1.8	0.6	0.3	0.1
1.9	0.5	0.4	0.1
2.0	0.5	0.4	0.1

(a) $\rho = 0.3$

α	q_1	q_2	q_3
0.9	0.95	0.05	0.0
1.0	0.9	0.1	0.0
1.1	0.8	0.2	0.0
1.2	0.75	0.25	0.0
1.3	0.7	0.3	0.0
1.4	0.7	0.3	0.0
1.5	0.6	0.4	0.0
1.6	0.6	0.3	0.1
1.7	0.5	0.4	0.1
1.8	0.5	0.4	0.1
1.9	0.5	0.3	0.2
2.0	0.5	0.3	0.2

(b) $\rho = 0.5$

α	q_1	q_2	q_3
0.9	0.95	0.05	0.0
1.0	0.9	0.1	0.0
1.1	0.8	0.2	0.0
1.2	0.8	0.2	0.0
1.3	0.7	0.3	0.0
1.4	0.6	0.3	0.1
1.5	0.6	0.3	0.1
1.6	0.5	0.3	0.2
1.7	0.5	0.3	0.2
1.8	0.5	0.3	0.2
1.9	0.5	0.3	0.2
2.0	0.4	0.3	0.3

(c) $\rho = 0.7$

Figure 4.9: Distribution of tasks in TAPTF - 3 Hosts

4.2.2 Three Hosts

An analytical comparison of TAGS and TAPTF in a three host distributed system is presented in this section. The results show the performance metrics for a system load of 0.3 (Figure 4.10), 0.5 (Figure 4.11) and 0.7 (Figure 4.12).

Figures 4.10(a) and 4.10(b) show the mean waiting time and slowdown respectively under a low system load ($\rho = 0.3$). It can be observed from the graphs that TAPTF performs better over a large range of α values, showing improved performance with respect to mean waiting time and slowdown where α is between 1.1 and 2.0. Figure 4.9(a) gives an indication of how TAPTF distributed the load more intelligently as the task size variation decreases. As the variation decreases a significant amount of tasks are dispatched to the second host (denoted by q_2), and as α approaches 2.0 we can see more tasks being dispatched to the third and final host (denoted by q_3). The final host in a TAGS system typically processes only the largest tasks - as variation decreases this practise is shown to be poor, as demonstrated by TAPTF's superior performance. Figures 4.10(c) and 4.10(d) highlight the benefit of the TAPTF approach under high to low variation (where α is between 1.1 and 2.0) showing consistent system loads while TAGS exhibits a sharp increase. As α approaches 2.0, the TAGS policy is producing significant excess load, which is a worrying sign under such a low arrival rate into the distributed system.

The mean waiting time and slowdown under a moderate system load ($\rho = 0.5$) are depicted in Figures 4.11(a) and 4.11(b) respectively. As the system load has increased it can be seen from the graphs that TAPTF shows improvement over a larger range of α values (where α is between 0.9 and 2.0). From Figure 4.9(b) it can be observed that in most cases a larger fraction of tasks are now being assigned to the second and third hosts (than under a system load of 0.3). It is worth noting that optimum values for the cut-offs (s_i) for TAGS could not be found for α values of 0.5 or 0.6, suggesting that it was impossible (or at least not computationally feasible) to find cut-offs that could keep the load below 1.0 at each host. From Figures 4.11(c) and 4.11(d) we can see that the increased arrival rate has a detrimental effect on the total load in the TAGS system. A sharp increase in load (and corresponding excess) can be observed as α approaches 2.0, while the TAPTF maintains consistent load over the same area.

Under a high system load ($\rho = 0.7$), the mean waiting time and slowdown are depicted in Figures 4.12(a) and 4.12(b). Similar problems to those experienced under a system load of 0.5 occurred when finding cut-offs for many α values under the three host, $\rho = 0.7$ scenario for

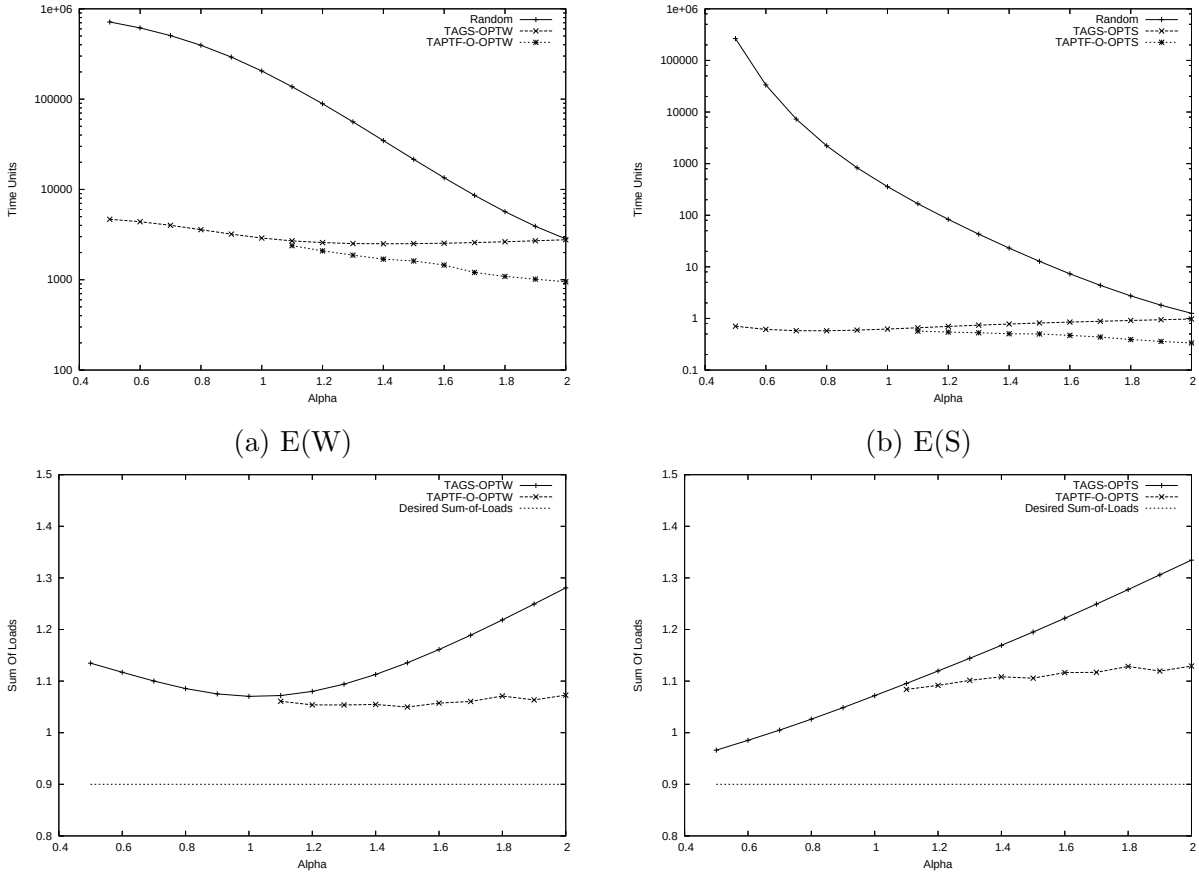


Figure 4.10: Performance of a three host distributed system with system load of 0.3. The expected waiting time and slowdown are depicted in (a) and (b) for policies optimised for these respective metrics. Likewise, corresponding load comparisons (desired versus actual Sum-Of-Loads) are shown in (c) and (d).

TAGS. That is, it was impossible to find optimum cut-offs that satisfied the requirement that the load must be below 1.0 at all hosts. This is confirmed when looking at the corresponding Sum-Of-Load measurements shown in Figures 4.12(c) and 4.12(d), showing the Sum-Of-Loads approaching 3.0 (indicating that some or all of the hosts are overloaded) where α is less than 0.8 or greater than 1.3. Figure 4.9(c) shows the fraction of tasks (q_i) allocated to each back-end server. We can see to handle the increased system load, a larger proportion of tasks are being assigned to the second and third host on average to cope. Indeed, when α is 2.0, each back-end host is allocated a fairly equal share of the incoming tasks (where

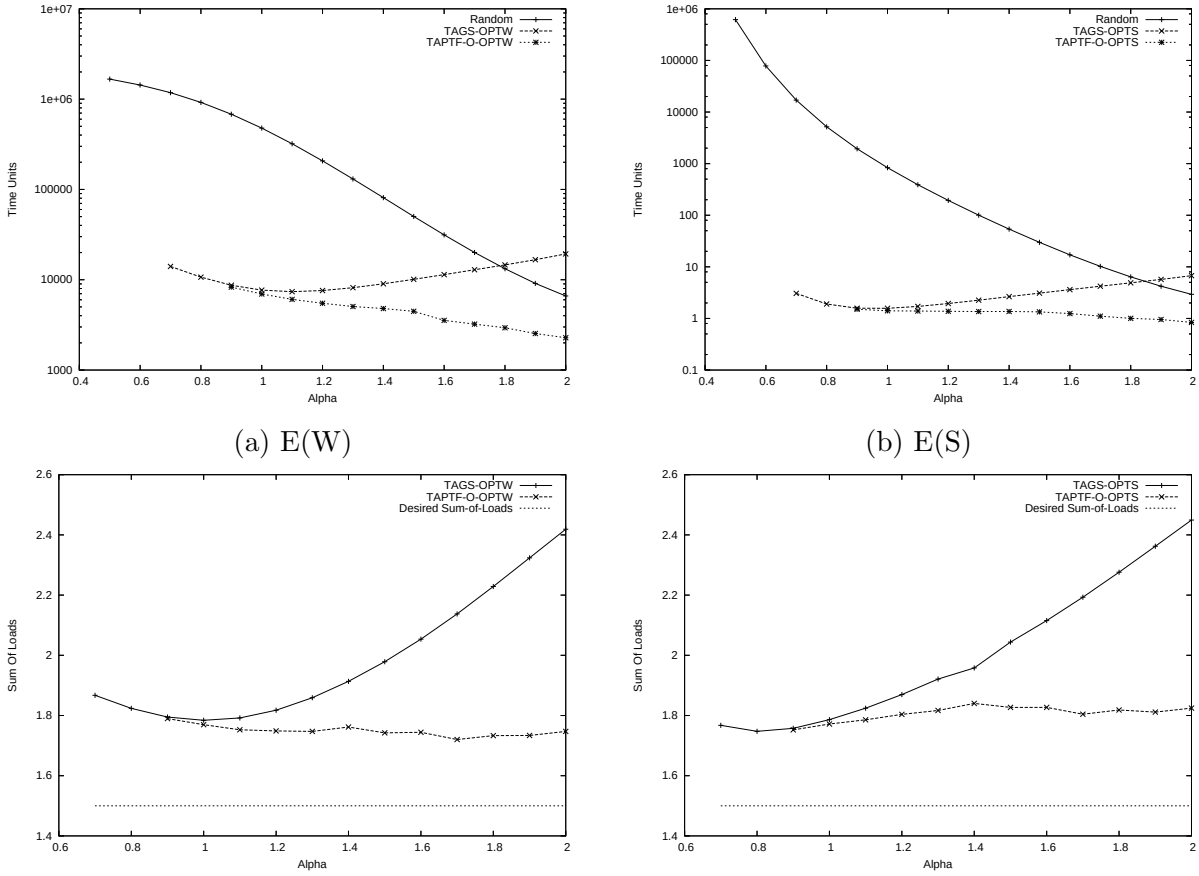


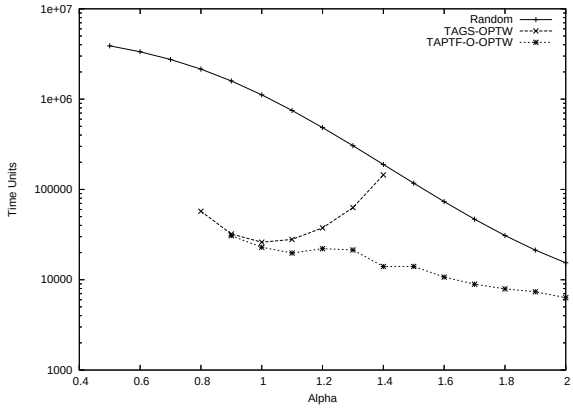
Figure 4.11: Performance of a three host distributed system with system load of 0.5. The expected waiting time and slowdown are depicted in (a) and (b) for policies optimised for these respective metrics. Likewise, corresponding load comparisons (desired versus actual Sum-Of-Loads) are shown in (c) and (d).

$q_1 = 0.4, q_2 = 0.3$ and $q_3 = 0.3$). Again it can be observed that, as the system load has increased, the range of α values where TAPTF outperforms TAGS is still similarly large - where α is between 0.9 and 2.0.

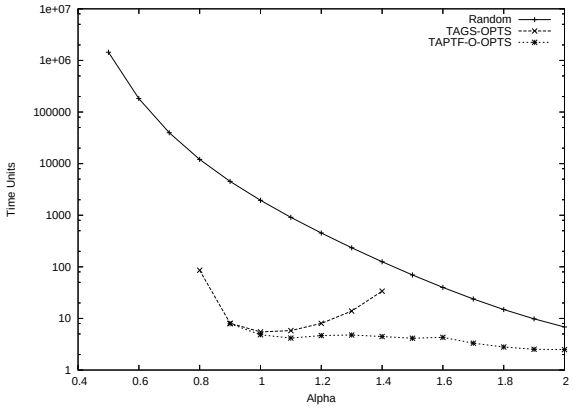
4.3 Discussion

In this section we discuss the implications of the results presented in Section 4.2 - both for the two host (Section 4.2.1) and three host (Section 4.2.2) scenarios.

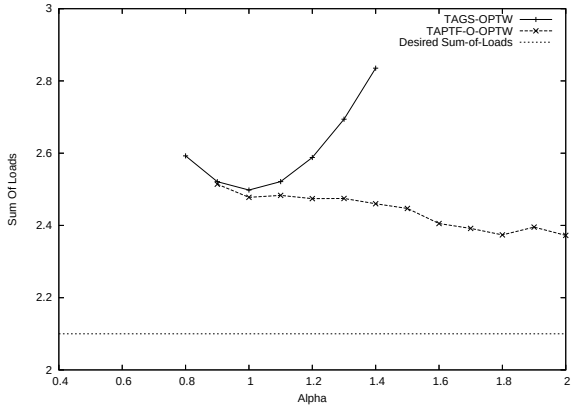
An analytical representation of the Random load distribution policy was included as



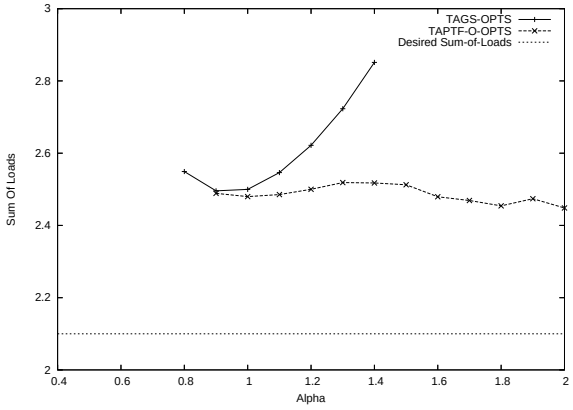
(c) E(W)



(d) E(S)



(c) Sum-of-Loads



(d) Sum-of-Loads

Figure 4.12: Performance of a three host distributed system with system load of 0.7. The expected waiting time and slowdown are depicted in (a) and (b) for policies optimised for these respective metrics. Likewise, corresponding load comparisons (desired versus actual Sum-Of-Loads) are shown in (c) and (d).

a baseline for comparison against TAGS and TAPTF. As discussed in previous work by Mor Harchol-Balter [Harchol-Balter, 2002] and illustrated by the Pollaczek-Khinchin formula shown in Section 2.2, all performance metrics for the Random policy are directly proportional to the variance of the task size distribution. As such, as the task size variation increases, and α decreases, the expected mean waiting time and slowdown explode exponentially in all the scenarios examined.

From the figures presented in Section 4.2.1 and Section 4.2.2, it is clear that TAGS (or at least TAGS-like behaviour) is the best policy under conditions of extreme to very high

variation. As mentioned previously, TAPTF is an adaptable task assignment policy, which can behave identically (and reduces analytically) to TAGS (i.e. set q_1 to 1.0) when it is prudent with regards to obtaining the best performance for a given scenario. In effect, the TAPTF policy encompasses TAGS ability to exploit a highly variable task size distribution, as well as remaining flexible enough to handle instances of lower variation and higher system loads by virtue of its many parameters that can be manipulated where required.

In areas of lower variation (and even low system load) we can see the benefit of dispatching tasks to hosts other than the first (highlighted by Figures 4.2, 4.3(a) and 4.3(b)). It is clear that as variation decreases, it pays to dispatch a growing proportion of tasks to the second host. This is largely due to the fact that we can no longer exploit the heavy-tailed property of the task size distribution, as the variation between the sizes of tasks decreases, and the average size of so-called *small* tasks increases.

The benefit of TAPTF over the TAGS policy becomes even more apparent as the system load increases (shown in Figures 4.5, 4.6(a) and 4.6(b)). TAGS suffers to a greater extent under higher loads, as an increase in excess (wasted processing caused by hand-offs, shown in Figures 4.6(c) and 4.6(d)) and growing average queue lengths combine to have a detrimental effect on TAGS performance under conditions of moderate to low task size variation. It can be observed that as the system load increases, the task variation range where the TAPTF policy betters TAGS becomes larger, and the factor of that improvement (in both mean waiting time and slowdown) increases. For example, consider the two host case. Consider the results shown in Figures 4.3(a), 4.6(a) and 4.8(a), depicting the mean waiting time under system loads of 0.3, 0.5 and 0.7 respectively. TAPTF betters TAGS when $\alpha \geq 1.3$ under a low system load of 0.3. With a moderate system load of 0.5, TAPTF betters TAGS when $\alpha \geq 1.1$. When the system load is high (0.7), TAPTF exhibits superior performance than TAGS when $\alpha \geq 1.0$. Similarly, consider when $\alpha = 2.0$ in each of these scenarios. Under a system load of 0.3, TAPTF exhibits an factor of improvement of approximately 1.5 over TAGS. Under a system load of 0.5, TAPTF shows an improvement of 2.7 over TAGS. When the system load is 0.7, TAPTF shows a substantial improvement over TAGS - by a factor of 6.6.

Section 4.2.2 summarises some interesting results for the three host scenario. We are particularly interested in the performance characteristics of TAGS and TAPTF for three hosts, as there is far greater flexibility with regards to choosing the cut-offs (s_i values) at each host in order to exploit the heavy-tailed property successfully. This increased flexibility is largely due to the obvious fact that there is an additional size partition, but also as a

consequence it is easier to keep the load below 1 at the respective hosts (both facts noted by Mor Harchol-Balter in her previous work on TAGS). This turns out to be true for TAPTF but not necessarily for TAGS, which suffers under heavy system loads.

In light of these changed conditions, we found that in some cases, as variation increases (and α decreases), the mean slowdown for the TAGS policy actually improves - to a certain point. Consider Figures 4.10(a) and 4.10(b), depicting a two host system under a low system load of 0.3. We observe a fairly flat and consistent response from the TAGS policy for the expected mean waiting time and slowdown over the range of α values shown. Slowdown gradually decreases as α approaches 0.7, then increases slightly as α reaches 0.5. This, as mentioned above, is because as the variation of tasks sizes becomes larger, TAGS can increasingly exploit the heavy-tailed property of such a distribution through choosing effective cut-offs that enable small tasks to be processed quickly, while ensuring large tasks are moved to latter hosts and do not unduly delay smaller tasks. This ensures good results with regards to overall metrics like mean waiting time and slowdown under conditions of extreme to highly variable task size distributions.

Despite the different behaviour exhibited for the three host scenario, TAGS is still bettered by the TAPTF policy under conditions ranging from high to low task size variation due to the same factors as under the two host scenario. Again we see the benefits achieved by dispatching a proportion of tasks to all hosts, not just the first. This is especially true as the system load increases - so to does the factor of improvement of TAPTF over TAGS. The advantages of the generic and flexible TAPTF model are highlighted in Figures 4.9(a) to 4.9(c) (and subsequently Figures 4.10 to 4.12). In several instances (Figures 4.11 and 4.12) it was not actually possible to find optimum cut-offs for TAGS that satisfied the constraint that the load must remain below 1 at all hosts.

Consider the case for three hosts where the system load is a (moderate) 0.5. We can see from Figures 4.11(a) and 4.11(b) that TAPTF clearly betters TAGS over a range of α values from 0.9 to 2.0. When α equals 2.0, the factor of improvement of TAPTF over TAGS is approximately 8 for both mean waiting time and slowdown. Under the same scenario, TAGS produces 3 times more excess load than the TAPTF policy - a significant amount of wasted processing.

4.4 Conclusion

In this chapter we have presented a new approach to task assignment in a distributed system, TAPTF (Task Assignment based on Prioritising Traffic Flows). TAPTF is a flexible policy that addresses the shortcomings of existing approaches (outlined earlier in this chapter) to task assignment. TAPTF demonstrated improved performance (both in mean waiting time and mean slowdown) in key areas where the TAGS and Random policies suffer. Most significantly, TAPTF exhibited improved performance under low to high task size variation and high system load by reducing the excess associated with a large number of restarts and by intelligently controlling the influx of tasks to each back-end host. We found for two and three host scenarios that as system load increases the range of α parameters where an improvement was shown, and the magnitude of that improvement increased. Given that TAPTF can encompass the best characteristics of existing approaches, as well as improving on them in what are considered critical scenarios of heavy traffic load and highly variable task sizes, we consider TAPTF to be a worthy policy for load distribution in environments where tasks are not pre-emptible and task sizes are not known *a priori*.

Chapter 5

Task Assignment with Work-Conserving Migration

The classical problem of task assignment in a distributed system is a critical and well researched area. A common configuration of such systems utilise a centralised dispatcher, directing incoming requests to homogeneous back-end hosts that offer mirrored services. Tasks are then serviced first-come-first-served at each host, and are not pre-emptible. Often, migration is available but is not work-conserving (any work done prior to migration is lost). Such a scenario is consistent with many batch and super-computing facilities where typically a task's memory requirement is so enormous that features like pre-emption and work-conserving migration are not feasible or even practical to implement.

Numerous task assignment policies have been proposed to address this particular problem [Crovella et al., 1998a; Harchol-Balter et al., 1999; Harchol-Balter, 2002], which are described in detail in Chapter 3. One of our important contributions was the TAPTF policy - Task Assignment based on Prioritising the Traffic Flow [Broberg et al., 2005], presented in Chapter 4. The TAPTF policy demonstrated improved performance, both analytically and through rigorous simulation, in key areas where existing policies are weak. Despite this, the TAPTF policy was not ideally suited to certain applications due to its design - particularly in the manner it stops and restarts tasks from scratch. This makes it unsuitable in its current form for certain interactive or real-time applications, like high-volume web serving from a web server cluster.

Thus, we consider a similar problem scenario to that described above but with an important modification - that work-conserving migration is available with low to negligible costs.

This is consistent with (but not limited to) many popular web serving architectures, known as web clusters or farms. A common configuration of these local and geographically distributed web clusters is where centralised dispatchers direct all incoming requests to back-end hosts, which can also redirect requests amongst themselves if prudent to do so [Cardellini et al., 2003; Aversa and Bestavros, 2000]. Transparent, negligible cost work-conserving migration is possible (and indeed highly desirable) due to the minimal state information that needs to be transferred for most web requests.

The architectures themselves have been the subject of significant research [Cardellini et al., 1999; 2002], with numerous commercial implementations [Cisco Systems, 1997] and service providers [Dilley et al., 2002]. As such, their utility and application are generally well defined. Analytical solutions to this particular problem are generally not as well defined due to the added complexity in mathematically modelling such scenarios.

We choose to focus on the utility of a work-conserving TAPTF policy in distributing requests via a centralised dispatcher in a locally distributed web-server system. We are also particularly interested in analysing the benefit of allowing back-end web servers to redirect requests when prudent. As such, we a present a task assignment policy specifically suited to these environments (such as high volume web serving clusters) where local centralised dispatchers are utilised to distribute tasks amongst back-end hosts offering mirrored services, with work-conserving migration available between hosts. The TAPTF-WC (Task Assignment based on Prioritising Traffic Flows with Work-Conserving Migration) policy was specifically created to exploit such environments. TAPTF-WC exhibits consistently good performance over a wide range of task distribution scenarios due to its flexible nature, spreading the work over multiple hosts when prudent, and separating short task flows from large task flows via the use of dual queues. Tasks are migrated in a work conserving manner, reducing the penalty associated with task migration found in many existing policies such as TAGS and TAPTF which restart tasks upon migration. We find that the TAPTF-WC policy is well suited for load distribution under a wide range of different workloads in environments where task sizes are not known *a priori* and work-conserving migration is available. The practical implementation aspects of such scheduling and redirection techniques are discussed in Section 5.1, but are not the focus of this chapter. Rather we focus our attention on a performance analysis of the proposed policy using the fundamentals of queueing theory.

The rest of the chapter is organised as follows. Section 5.1 covers the background required for the remainder of the chapter. In Section 5.2 the most relevant related work is presented. A detailed description of the TAPTF-WC model is provided in Section 5.3. Section 5.4 gives

an analytic comparison of TAPTF-WC with existing approaches. In Section 5.5 we briefly consider a scenario where the migration costs are non-negligible - that is, where a migration occurs a fixed or proportional cost. Section 5.6 provides a detailed discussion of the analytical comparisons performed in Sections 5.4 and 5.5. in Section 5.7 we conclude this chapter with some closing thoughts on the insight gained during the evaluation, and consider the benefits of the TAPTF-WC approach.

5.1 Background

This section provides an overview of the practical implementation aspects for task assignment policies in web server clusters. This will help the reader understand the importance of effective task assignment for web server clusters. An overview of the practical implementation aspects for this task assignment policy in web server clusters is provided in Section 5.1.1.

5.1.1 Web Server Clusters

To address the issue of improving performance in high volume web sites, many distributed architectures have been proposed to improve the users experience - such as response time to retrieve a file or the throughput. These architectures attempt to provide expandable resources in order to solve the issue of scalability - how to service the growing number of users and the expanding bandwidth available to them. With network bandwidth increases at a rate twice as fast as server capacity and dynamically generated content accounting for a larger percentage of web content being requested, the server side will be a bottleneck now and into the future [Cardellini et al., 2002].

There are many options available to an architect of such systems to address these scalability issues. One is to *scale-up*, improving a node that is already available to you. You can achieve this in two ways. First, there is the notion of *hardware scale-up* [Bill Devlin and Spix, 1999], where more resources (such as disk, memory and CPU's) are added to your node as needed. This can be useful as a temporary solution but it does not scale very far (considering the consistent increase in Internet traffic) and it does not solve the issue of reliability that a set of servers offering mirrored services could achieve.

Another technique is referred to as *software scale-up* where improvements are made in the performance and efficiency of a node at the software level. This could be achieved by making the operating system more efficient, reducing overheads at the web server application level [Pai et al., 1999; Doolittle and Nelson, 1991] or improving the request scheduling

policy [Bansal and Harchol-Balter, 2001]. Again, this can only achieve limited performance gains and does not address the issue of scalability nor reliability.

To cope with increasing demand and to provide some level of fault tolerance and redundancy, we *scale-out* [Bill Devlin and Spix, 1999] by adding additional nodes to our web server architecture. One widely-used technique is to *global scale out*, locating nodes in different geographical locations. This has a number of desirable properties, such as offering faster local mirrors (depending on a user's location) and providing large scale redundancy, removing any single point of failure. Despite this, it is very common to address the issue of load distribution in such architectures by exploiting the DNS mechanism, attempting to route requests intelligently during the address resolution phase (i.e. by using the authoritative DNS server, A-DNS, in conjunction with simple task assignment policies such as Random/Round-Robin). Unfortunately this proves to be a very course-grained approach due to caching of DNS information at local names servers and even the client itself. Measurements have shown that the A-DNS only controls a small percentage of requests, as low as 5% of total requests reaching the system [Colajanni et al., 1998b; Cardellini et al., 2002]. As such, it is not an entirely effective way to distribute the workload in a distributed web server system.

More fine-grained control is needed to effectively distribute the load in a distributed web-server system, given the empirical evidence we have about the characteristics of the workload and the knowledge of the negative performance implication that poor task assignment choices can cause. As such, we consider the notion of *local scale out* where we have a local group of back-end web-servers that can potentially service any request, with the responsibility of request assignment placed on a front-end web server or dispatching device, or even the back-end web-servers themselves (with requests being broadcast or multicast to each server).

We chose to focus on task assignment policies that are suited to local scale-out architectures. They are the best mechanism to effectively distribute the workload in a distributed web-server system, providing the most control with regards to dispatching or routing incoming requests.

There are a number of differing arrangements that can constitute a local web-server cluster. One common arrangement is the virtual web cluster, where one IP address (the Virtual IP, VIP) is visible to clients. This IP is not assigned to a particular front-end server or device, rather it is shared by each server node. Another arrangement is the traditional locally distributed web system, where each server node has a unique IP that is visible to all clients.

In particular we focus on the cluster-based web system, which has a number of desirable

features that make it appealing over the virtual web cluster or a distributed web system. The cluster-based arrangement has a single IP address and thus a single point of contact. It can provide fine-grained control of request routing. Most importantly, it requires no special reconfiguration of clients or servers - it integrates well with current protocols, standards and clients. This is crucial as it is highly desirable to make any complex request routing and redirection transparent to the client. This architecture makes second level routing possible via HTTP redirection at the application level, or via TCP hand-offs or splicing at the IP protocol level [Adhya et al., 2002; Cohen et al., 1999; Maltz and Bhagwat, 1998; Rosu and Rosu, 2002; Spatscheck et al., 2000]. The efficiency of these techniques have improved considerably over the last 7 years, to the point where the overhead placed on cluster resources is minimal. An extensive evaluation of the implementation aspects of such techniques can be found in a recent survey of web cluster technology [Cardellini et al., 2002]. An architecture containing a centralised dispatcher with second level routing provides an ideal platform to implement the work-conserving TAPTF model, TAPTF-WC, introduced in this chapter.

5.2 Related Work

An exhaustive evaluation of general purpose task assignment policies can be found in Chapter 3 as well as in existing literature [Harchol-Balter et al., 1999; Harchol-Balter, 2002; Broberg et al., 2005]. Indeed, many commercial web load balancing solutions depend on traditional load distribution techniques such as (weighted) Random and Round-Robin, as well as Shortest-Queue-First assignment policies. These policies were shown in Chapter 3 to perform poorly under high intensity, highly variable workloads, suggesting that modern techniques that address the negative characteristics of these workloads are required. We now highlight some techniques that are specifically focused on task assignment in web server clusters.

Two recent policies that have been proposed specifically to deal with load balancing in clustered web servers are EQUILOAD [Ciardo et al., 2001] and ADAPTLOAD [Riska et al., 2002c]. Under the EQUILOAD policy, back-end servers continuously monitor the incoming workload they receive, and periodically re-negotiate their agreement on the size of requests to be allocated to them. A methodology is provided to characterise web workload, fitting them with phase-type distributions that closely resemble the original distribution. The characterisation can be done both off-line (e.g. on a complete trace) or online (periodically examining workload seen thus far, and adjusting its fitting). The policy is based solely on the

distribution of incoming task sizes. EQUILOAD is not truly adaptive - special events may drastically alter the relative popularity of web server document(s) causing the boundaries chosen to be no longer optimal.

Riska notes that a robust scheduling policy must consider arrival rate of incoming tasks and the distribution of their service requirement, and that any changes in observed burstiness in average arrival rate should trigger a change in policy parameters to adopt to the new arrival rate [Riska et al., 2002c]. As such, the ADAPTLOAD policy was formulated. ADAPTLOAD uses workload history to adapt the boundaries. Simulations indicate knowledge of finite workload can be used as a good indicator of future behaviour. The ADAPTLOAD policy examines the last K requests to build a discrete data histogram needed to determine boundaries for the allocation of the next K requests. The authors found that K should be neither too small (as ADAPTLOAD needs a statistically significant sample) nor too large (since it needs to adapt to fluctuations). If a significant proportion of workload consists of a few popular files it may not be possible to select N distinct boundaries (i.e. for each server) and still ensure each interval corresponds to an equal amount of load received at each server. Thus, the authors introduced probabilistic boundaries to combat this issue. A probability p_i is assigned to each boundary point s_i, expressing the portion of requests for file size s_i to be served by server i. The remaining portion $1 - p_i$ of requests for this file size is served by server $i + 1$ or additional servers.

ADAPTLOAD is compared to Join Shortest Weighted Queue (JSWQ), where "the length of each queue in the system is weighted by the size of the queued requests" [Riska et al., 2002c] - essentially a Least-Work-Remaining approach. Under low load, JSWQ does better than ADAPTLOAD, which can direct a request to a server that is busy even when an idle server is available (due to pre-computed boundaries). Under periods of transient overload ADAPTLOAD outperforms JSWQ, achieving lower average slowdowns and returning to acceptable overload levels quicker. ADAPTLOAD manages consistently small slowdowns for nearly all classes of requests. Like other size-based approaches, EQUILOAD and ADAPTLOAD are limited to applications where task sizes are known in advance.

5.3 The Proposed Model - TAPTF-WC

In this section we propose a new task assignment policy called TAPTF-WC - Task Assignment based on Prioritising Traffic Flows with Work-Conserving Migration - to address the limitations of existing approaches in dealing with certain classes of traffic.

5.3.1 Motivation

We are motivated by the need for a flexible task assignment policy that provides good performance for a local cluster with a centralised dispatcher. No assumptions are made regarding *a priori* knowledge of an individual task's size (rather just requiring a broad knowledge of the *distribution* of task's service requirement). This precludes the usage of size-based policies (such as SITA-E/V/U [Harchol-Balter et al., 1999; Crovella et al., 1998a; Schroeder and Harchol-Balter, 2004]) that assume precise knowledge of a tasks service requirement upon arrival at the dispatcher. We also consider the potential of back-end hosts to re-route tasks (in a work conserving fashion) in order to further improve performance of the system. As such, policies like TAGS and TAPTF cannot fully exploit such capabilities, due to their non work-conserving nature when migrating tasks. This scenario is consistent with many common cluster web serving architectures, as described in Section 5.1. It also needs to deal with the commonly experienced scenario of highly variable workloads, which many existing task assignment policies handle poorly.

5.3.2 Conceptual view of the TAPTF-WC model

We consider an extension of the TAPTF policy, called TAPTF-WC - Task Assignment based on Prioritising Traffic Flows with Work-Conserving Migration. TAPTF has a number of desirable characteristics (as described in Section 5.2) that make it an ideal base on which to build a work-conserving task assignment policy. Unlike TAPTF (and TAGS) where tasks are restarted from scratch if they exceed the cut-off at a given host, TAPTF-WC conserves each portion of work it completes at a given host. Upon migration to a new host, TAPTF-WC resumes work from where it ceased processing before migration. This is consistent with distributed systems (such as but not limited to the web cluster environment described in Section 5.1) that support work-conserving migration.

In TAPTF-WC, arrivals of tasks to the dispatcher follow a Poisson process with rate λ. The dispatcher then assigns tasks to each of the n hosts, call these Host i, $1 \leq i \leq n$, at random with probability q_i respectively. Using a well known property of the Poisson process, the arrival stream to Host i is also a Poisson process with rate λq_i.

Due to the heavy-tailed characteristic of the task size distribution, we will assume that task sizes (service distribution) follow a bounded Pareto Distribution $B(\alpha, k, p)$.

We also make the following assignment of loads to hosts:

- Tasks that run-to-completion at Host i's O queue are those whose initial (original) sizes

are in the range $[k, s_i]$ and remaining size is $< s_i$,

- Tasks that run-to-completion at Host i's R queue (where $1 < i \leq n$) are those whose initial (original) sizes are in the range $[s_{i-1}, s_i]$ and remaining size is $< s_i - s_{i-1}$,

where $k < s_1 < s_2 < s_3 < \ldots < s_n = p$. The aim is to compute the cut-offs (s_i values) in order to minimise critically important performance metrics such as mean waiting time or mean slowdown.

Like TAPTF each host, except for Host 1, accommodates two queues, the ordinary (O) queue and the restart (R) queue. Tasks sent to a host from the Dispatcher join the O queue. If the remaining size of a task does not fall within the correct range (that is, it exceeds the processing limit s_i associated with some Host i), they are moved to the R queue belonging to next host down the line. Unlike the standard TAPTF approach, any computational work done at a given host is conserved when it is moved to the next host's restart (R) queue. This process is repeated until these tasks can run-to-completion. Tasks kept in the O queues (received directly from the dispatcher) have priority of service over those in the R queue. However, a task which is being served in the R queue will not be pre-empted from service by an arrival of a task into the O queue. All tasks in the R and O queues are served on a First-Come-First-Serve basis.

The above model differs from TAGS in that we have set the boundaries of task sizes at each host so that all tasks with remaining sizes *less than or equal* to a fixed cut-off point are processed by the host. This means that a task can be dispatched to *any* hosts initially without being first dispatched to Host 1 as in TAGS in order to preserve the property that job's service demand is not known *a priori*. TAPTF-WC (like TAPTF) uses dual queues at each host in order to speed up the flow of shorter tasks.

Shown in Figure 5.1 is the TAPTF-WC system for 4 hosts. Note the size ranges associated with each Host. 'Original size' refers to the initial size of tasks that will run-to-completion at that host. 'Remaining' refers to the tasks with remaining processing time less than the cut-off that will run-to-completion at that host.

5.3.3 Mathematical Preliminaries for the TAPTF-WC model

In this section, we define and compute all the important parameters associated with the TAPTF-WC model. As in the analysis of TAPTF in the previous chapter, the main objective of this process is to use these parameters to determine the optimal cut-off points $k < s_1 <$

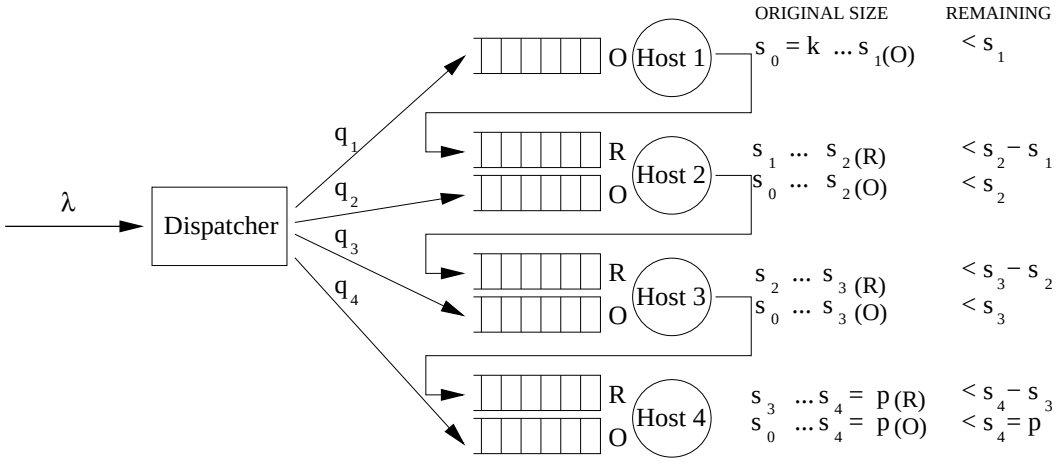


Figure 5.1: TAPTF-WC With 4 Hosts

$s_2 < \ldots < p$ that allow us to minimise the mean waiting time or slowdown for tasks entering the distributed system, in order to provide the best overall performance for our system.

In Section 2.3.1 we found that our task size distribution (whose probability density function is denoted here by $f(x)$) follows a Bounded Pareto distribution $B(k, p, \alpha)$, where α represents the task size variation, k is the smallest possible task, and p is the largest possible task.

Much of the analysis that follows is similar to that undertaken in the previous chapter. However there are some subtle yet critical differences that are made in order to model work-conserving migration. Indeed, we must consider any processing that may have already occurred at other hosts when modelling the behaviour of each host in the queueing system. We present the entire TAPTF-WC model (including some repeated definitions from Chapter 4) for the sake of completeness and comprehension.

Let p_i be the probability that a task's *original* size will be less than the cut-off assigned to Host i. That is, the fraction of tasks whose final destination is either Host i or its predecessors. p_i will be used later to calculate those tasks that start and finish at the same host.

$$p_i = P(X \leq s_i) \tag{5.1}$$

$$= \frac{\alpha k^\alpha}{1-(k/p)^\alpha} \int_k^{s_i} x^{-\alpha-1} dx \tag{5.2}$$

$$= \frac{1-(k/s_i)^\alpha}{1-(k/p)^\alpha}. \tag{5.3}$$

The Ordinary (O) queue only accepts tasks that are received directly from the dispatcher.

As such, the fraction of tasks that *visit* Host i's ordinary (O) queue, denoted as h_{iO}, is simply q_i:

$$h_{iO} = q_i. \tag{5.4}$$

The fraction of jobs that *visit* Host i's Restart (R) queue is denoted as h_{iR}. h_{1R} is not defined as there is no Restart queue at Host 1. Therefore, for $2 \leq i \leq n$:

$$\begin{aligned} h_{iR} &= \sum_{j=1}^{i-1} q_j(1 - p_{i-1}) \\ &= (1 - p_{i-1}) \sum_{j=1}^{i-1} q_j. \end{aligned} \tag{5.5}$$

The expression h'_{iO} denotes the fraction of tasks whose *final* destination is Host i's ordinary (O) queue. Evidently this is product of the probability of a task being assigned to Host i (q_i), and the fraction of tasks whose original size is less than the cut-off at Host i (p_i). Therefore:

$$h'_{iO} = q_i p_i \tag{5.6}$$

The expression h'_{iR} denotes the fraction of tasks whose *final* destination is Host i's restart (R) queue. Clearly h'_{1R} is undefined as their is no Restart queue at Host 1. Therefore, where $2 \leq i \leq n$:

$$\begin{aligned} h'_{iR} &= \sum_{j=1}^{i-1} q_j(p_i - p_{i-1}) \\ &= (p_i - p_{i-1}) \sum_{j=1}^{i-1} q_j. \end{aligned} \tag{5.7}$$

Now let us focus on the tasks that finish (i.e. complete their final amount of processing) in Host i's queues. Let $E(X_{iO}^j)$ and $E(X_{iR}^j)$ be the jth moment of the distribution of the *remaining sizes* of tasks whose final destination is Host i's O queue and Host i's R queue respectively. Given that TAPTF-WC is work-conserving, we are careful to condition on the distribution of task's remaining sizes at Host i, not their initial (original) size. For $E(X_{iO}^j)$ they are one and the same, given that the tasks that finish in an Ordinary queue started in the same queue, this we do not need to track any previous work done at other hosts.

We have

$$E(X_{iO}^j) = \frac{1}{p_i}\int_k^{s_i} x^j f(x)dx \tag{5.8}$$

$$= \begin{cases} \frac{\alpha s_i^j((\frac{k}{s_i})^\alpha - (\frac{k}{s_i})^j)}{p_i(j-\alpha)(1-(\frac{k}{p})^\alpha)} & \text{if } j \neq \alpha \\ \frac{\alpha k^\alpha \ln(s_i/k)}{p_i(1-(\frac{k}{p})^\alpha)} & \text{otherwise} \end{cases} \tag{5.9}$$

For $E(X_{iR}^j)$ we are conditioning on the distribution of task's remaining sizes by considering the work already done (s_{i-1}).

$$E(X_{iR}^j) = \frac{1}{p_i - p_{i-1}}\int_{s_{i-1}}^{s_i} (x - s_{i-1})^j f(x)dx \tag{5.10}$$

$$= \frac{\alpha k^\alpha}{1-(\frac{k}{p})^\alpha}\frac{1}{p_i - p_{i-1}}\sum_{z=0}^{j}(\frac{j!}{z!(j-z)!})({s_{i-1}}^{j-z})(-1)^{j-z}\int_{s_{i-1}}^{s_i} x^{z-\alpha-1}dx \tag{5.11}$$

$$= \begin{cases} \frac{\alpha k}{(1-(\frac{k}{p}))(p_i-p_{i-1})}[\ln(s_i/s_{i-1}) + (\frac{s_i}{s_{i-1}} - 1)] & \text{if } j = \alpha = 1 \\ \frac{\alpha k^2}{(1-(\frac{k}{p})^2)(p_i-p_{i-1})}[\ln(s_i/s_{i-1}) + \frac{1}{2}(1 - \frac{s_i}{s_{i-1}}) + 2(\frac{s_i}{s_{i-1}} - 1)] & \text{if } j = \alpha = 2 \\ \frac{\alpha k}{(1-(\frac{k}{p}))(p_i-p_{i-1})}[s_{i-1}(\frac{1}{s_i} + 1) - 2s_{i-1}\ln(s_i/s_{i-1}) + s_i - s_{i-1}] & \text{if } j = 2, \alpha = 1 \\ \frac{\alpha k^\alpha}{(1-(\frac{k}{p})^\alpha)(p_i-p_{i-1})}\sum_{z=0}^{j}(\frac{j!({s_{i-1}}^{j-\alpha})(-1)^{j-z}((\frac{s_i}{s_{i-1}})^{z-\alpha}-1)}{(z!(j-z)!)(z-\alpha)}) & \text{otherwise.} \end{cases} \tag{5.12}$$

Now, we consider those tasks that spend time in Host i's queues (regardless of whether they complete their processing there or not). We let $E(hostX_{iO}^j)$ and $E(hostX_{iR}^j)$ be the jth moment of the distribution of tasks who spent time in Host i's O queue and Host i's R queue respectively. Therefore:

$$E(hostX_{iO}^j) = \frac{h'_{iO}}{h_{iO}}E(X_{iO}^j) + \frac{h_{iO} - h'_{iO}}{h_{iO}}s_i^j. \tag{5.13}$$

Similarly,

$$E(hostX_{iR}^j) = \frac{h'_{iR}}{h_{iR}}E(X_{iR}^j) + \frac{h_{iR} - h'_{iR}}{h_{iR}}(s_i - s_{i-1})^j. \tag{5.14}$$

We let λ_{iO} and λ_{iR} represent the arrival rate into Host i's ordinary (O) and restart (R) queue respectively. Then, based on similar logic to h_{iO} and h_{iR}:

$$\lambda_{iO} = \lambda q_i \tag{5.15}$$

$$\text{and} \quad \lambda_{iR} = \lambda h_{iR}. \tag{5.16}$$

The loads at Host i's O and P queue are:

$$\rho_{iO} = \lambda_{iO} E(hostX_{iO}) \tag{5.17}$$

$$\text{and} \quad \rho_{iR} = \lambda_{iR} E(hostX_{iR}) \tag{5.18}$$

respectively.

The next set of results, concerning the expected waiting times for tasks in Host i's ordinary (O) and restart (R) queues, relies on the same key facts described in the analysis of TAPTF (Chapter 4):

- Tasks in the O queue have priority of service over tasks in the R queue
- A task in service at the R queue will not be pre-empted from service by a task which subsequently arrives into the O queue
- Within each queue, tasks are processed on a FCFS basis.
- Finally, we will have to assume that tasks that arrive into the R queues form Poisson processes.

The expected waiting time for tasks arriving at Host i's O and R queues is defined as $E(hostW_{iO})$ and $E(hostW_{iR})$ respectively. Utilising the same classic result employed in Chapter 4 for a non pre-emptive priority queue [Cobham, 1953] system we find that:

$$E(hostW_{iO}) = \frac{\lambda_{iO} E(hostX_{iO}^2)}{2(1-\sigma_1)} \tag{5.19}$$

$$\tag{5.20}$$

$$\text{and} \quad E(hostW_{iR}) = \frac{\lambda_{iR} E(hostX_{iR}^2)}{2(1-\sigma_1)(1-\sigma_2)} \tag{5.21}$$

$$\tag{5.22}$$

where $\sigma_1 = \rho_{iO}$ and $\sigma_2 = \rho_{iR} + \rho_{iO}$. The above results are dependent on $0 < \sigma_1 < 1$ and $0 < \sigma_2 < 1$ at all times, or they do not hold. We now wish to work toward obtaining the two system-wide metrics that are of most importance to us. That is, the expected waiting time and slowdown for tasks entering the system. We let $E(W_{iO})$ and $E(W_{iR})$ denote the expected waiting time of a task whose final destination is Host i's O and R queue respectively. Therefore, for $1 \leq i \leq n$

$$E(W_{iO}) = h'_{iO} E(hostW_{iO}) \tag{5.23}$$

and, for $2 \leq i \leq n$,

$$E(W_{iR}) = h'_{iR} \sum_{m=1}^{i-1} q_m [E(hostW_{mO}) + \sum_{j=m+1}^{i} E(hostW_{jR})]. \quad (5.24)$$

The waiting time $E(W_{iR})$ for a task whose final destination is a Restart queue must factor in any waiting time it may have accumulated as it migrates toward its final destination.

The expected slowdown experienced by tasks that finish at Host i is characterised by $E(S_{iO})$ and $E(S_{iR})$ respectively. Therefore,

$$E(S_{iO}) = E(W_{iO})E(1/X_{iO}) \quad (5.25)$$

$$\text{and} \quad E(S_{iR}) = E(W_{iR})E(1/X_{iR}). \quad (5.26)$$

5.3.4 A TAPTF-WC model with cost-based migration

While not the focus or application of the TAPTF-WC model, for completeness we will briefly consider a scenario where the act of work-conserving migration incurs a cost to the source host to save and transfer state, and the destination host to restart the task, rebuild the state information and resume processing. We will consider two scenarios. First, we consider the case where a task that restarts incurs a fixed cost, γ, that is borne by the source host it has migrated from and the destination host it has migrated to. Second, we consider a scenario where a task incurs a cost, β, that is proportional to the service requirement, X. Again, this cost can be placed on both the source and destination hosts. This is consistent with many of the systems and migration mechanisms described in Section 3.3.

Fixed cost migration

First let us consider a fixed migration cost, that occurs when a task is migrated to a restart queue. This fixed cost is represented by γ, where $\gamma > 0$. For generality, we will individually factor all cost that are incurred by the *source* host in halting a running task, saving and packaging its state information for transfer. We shall refer to this cost as γ_s, the migration cost. The cost of resuming a task, unpacking and recreating the transferred state information is placed upon the destination host. We will refer to this cost as γ_d, the resumption cost.

We need to redefine $E(X_{iR}^j)$, the *jth* moment of the distribution of the *remaining sizes* of tasks whose final destination is Host i's R queue. For $E(X_{iR}^j)$ we are now conditioning on the distribution of task's remaining sizes by considering the work already done (s_{i-1}) *and* the fixed resumption cost incurred from restarting the task, γ_d.

$$E(X_{iR}^j) = \frac{1}{p_i - p_{i-1}} \int_{s_{i-1}}^{s_i} (x - s_{i-1} + \gamma_d)^j f(x) dx \tag{5.27}$$

$$= \frac{\alpha k^\alpha}{1 - (\frac{k}{p})^\alpha} \frac{1}{p_i - p_{i-1}} \sum_{z=0}^{j} (\frac{j!}{z!(j-z)!})(s_{i-1} + \gamma_d)^{j-z}(-1)^{j-z} \int_{s_{i-1}}^{s_i} x^{z-\alpha-1} dx \tag{5.28}$$

$$= \begin{cases} \frac{k}{(1-(\frac{k}{p}))(p_i - p_{i-1})}[\ln(s_i/s_{i-1}) + (\frac{s_i}{s_{i-1}} - 1) + \gamma_d({s_{i-1}}^{-1} - s_i^{-1})] & \text{if } j = \alpha = 1 \\ \frac{2k^2}{(1-(\frac{k}{p})^2)(p_i - p_{i-1})}[\frac{(s_{i-1}-\gamma_d)}{2}(\frac{4s_i - s_{i-1} + \gamma_d}{{s_i}^2} - \frac{3s_{i-1}+\gamma_d}{{s_{i-1}}^2}) + \ln(\frac{s_i}{s_{i-1}})] & \text{if } j = \alpha = 2 \\ \frac{k}{(1-(\frac{k}{p}))(p_i - p_{i-1})}[s_i - s_{i-1} - (s_{i-1} - \gamma_d)^2({s_{i-1}}^{-1} - {s_i}^{-1}) & \\ \quad + 2(s_{i-1} - \gamma_d)\ln(\frac{s_{i-1}}{s_i})] & \text{if } j = 2, \alpha = 1 \\ \frac{\alpha k^\alpha}{1-(\frac{k}{p})^\alpha} \frac{1}{p_i - p_{i-1}} \sum_{z=0}^{j} (\frac{j!}{z!(j-z)!})(s_{i-1} + \gamma_d)^{j-z}(-1)^{j-z}(\frac{{s_i}^{z-\alpha} - {s_{i-1}}^{z-\alpha}}{z-\alpha}) & \text{otherwise.} \end{cases} \tag{5.29}$$

Let us consider those tasks that spend time in Host i's O queues. As defined previously, $E(hostX_{iO}^j)$ is the jth moment of the distribution of tasks who spent time in Host i's O queue. We must redefine this expression to consider the migration cost γ_s, which is incurred when a task is migrated to another host without completing its service in this queue:

$$E(hostX_{iO}^j) = \frac{h'_{iO}}{h_{iO}} E(X_{iO}^j) + \frac{h_{iO} - h'_{iO}}{h_{iO}} (s_i + \gamma_s)^j. \tag{5.30}$$

Now, we consider those tasks that spend time in Host i's R queue (regardless of whether they complete their processing there or not). As defined previously, $E(hostX_{iR}^j)$ is the jth moment of the distribution of tasks who spent time in Host i's R queue. We must redefine this expression to consider the resumption cost γ_d, which is incurred regardless of whether a task runs to completion or is migrated yet again without completing its service in this queue. In addition, the migration cost γ_s is incurred when a task is migrated without completing its service in this queue:

$$E(hostX_{iR}^j) = \frac{h'_{iR}}{h_{iR}} E(X_{iR}^j) + \frac{h_{iR} - h'_{iR}}{h_{iR}} (s_i - s_{i-1} + \gamma_s + \gamma_d)^j. \tag{5.31}$$

By allowing the migration costs at the source node and destination node to be modelled separately, we attempt to maintain the generality of the TAPTF-WC model. If γ_s and γ_d are set to equal zero, the original TAPTF-WC is obtained. By altering γ_s and γ_d we can represent a wide variety of application domains, where these costs might be roughly equivalent, or where the cost at one side (either source or destination) may dominate the other.

Proportional cost migration

Let us consider a proportional migration cost, that occurs when a task is migrated to a restart queue. We model this cost as a per-unit cost related to the original service requirement. Let us denote β as the per-unit cost factor, where $\beta > 0$. We again separately consider the migration costs that are incurred by the *source* host in halting a running task, saving and packaging its state information for transfer, which we denote as β_s. The resumption cost for recreating the state information and resuming a running task at the *destination* node is β_d

We again need to redefine $E(X_{iR}^j)$. For $E(X_{iR}^j)$ we are now conditioning on the distribution of task's remaining sizes by considering the work already done (s_{i-1}) *and* the resumption cost incurred, $\beta_d x$, where x is the task's original size.

$$E(X_{iR}^j) = \frac{1}{p_i - p_{i-1}} \int_{s_{i-1}}^{s_i} (x - s_{i-1} + \beta_d x)^j f(x) dx \tag{5.32}$$

$$= \frac{\alpha k^\alpha}{1 - (\frac{k}{p})^\alpha} \frac{1}{p_i - p_{i-1}} \sum_{z=0}^{j} (\frac{j!}{z!(j-z)!})(s_{i-1})^{j-z}(-1)^{j-z} \int_{s_{i-1}}^{s_i} \frac{(x + \beta_d x)^z}{x^{\alpha+1}} dx \tag{5.33}$$

$$= \begin{cases} \frac{\alpha k}{(1-(\frac{k}{p}))(p_i - p_{i-1})}[\ln(s_i/s_{i-1}) + (\frac{s_i}{s_{i-1}} - 1) + \beta_d \ln \frac{s_i}{s_{i-1}}] & \text{if } j = \alpha = 1 \\ \frac{2k^2}{(1-(\frac{k}{p})^2)(p_i - p_{i-1})}[(1+\beta_d)^2(\ln(\frac{s_i}{s_{i-1}})) - \frac{{s_{i-1}}^2}{2{s_i}^2} & \\ + \frac{2(s_{i-1}+\beta_d s_{i-1})}{s_i} - \frac{3}{2} - 2\beta_d] & \text{if } j = \alpha = 2 \\ \frac{\alpha k}{(1-(\frac{k}{p}))(p_i - p_{i-1})}[(1+\beta_d)^2(s_i - s_{i-1}) & \\ - 2(1+\beta_d)(s_{i-1})(\ln(\frac{s_i}{s_{i-1}})) - \frac{{s_{i-1}}^2}{s_i} + s_{i-1}] & \text{if } j = 2, \alpha = 1 \\ \frac{\alpha k^\alpha}{1-(\frac{k}{p})^\alpha} \frac{1}{p_i - p_{i-1}} \sum_{z=0}^{j} {}^jC_z {s_{i-1}}^{j-z}(-1)^{j-z} \frac{{s_i}^{-\alpha}((1+\beta_d)s_i)^z - {s_{i-1}}^{-\alpha}((1+\beta_d)s_{i-1})^z}{z-\alpha} & \text{otherwise.} \end{cases} \tag{5.34}$$

Let us consider those tasks that spend time in Host i's O and R queues (regardless of whether they complete their processing there or not). $E(hostX_{iO}^j)$ and $E(hostX_{iO}^j)$ are the jth moments of the distribution of tasks who spent time in Host i's O and R queues respectively. We must redefine these expressions to consider the source migration (β_s) and destination resumption (β_d) costs.

In the O queues, $\beta_s x$ is incurred when a task exceeds the cut-off associated with the host it is processing on. We first need to find the expected original size of tasks that *visit* Host i's O queue but do not run to completion there, thus incurring a migration cost to transfer the task.

$$E(\bar{X}_{iO}) = \frac{h_{iO} - h'_{iO}}{h_{iO}} \int_{s_i}^{s_n} x f(x) dx \tag{5.35}$$

$$= \frac{\alpha k^\alpha}{1 - (\frac{k}{p})^\alpha} \frac{h_{iO} - h'_{iO}}{h_{iO}} \int_{s_i}^{s_n} x^{-\alpha} dx \tag{5.36}$$

$$= \begin{cases} \frac{\alpha k^\alpha}{1-(\frac{k}{p})^\alpha} \frac{h_{iO}-h'_{iO}}{h_{iO}} \ln(\frac{s_n}{s_i}) & \text{if } \alpha = 1 \\ \frac{\alpha k^\alpha}{1-(\frac{k}{p})^\alpha} \frac{h_{iO}-h'_{iO}}{h_{iO}} \frac{{s_n}^{1-\alpha} - {s_i}^{1-\alpha}}{1-\alpha} & \text{otherwise.} \end{cases} \tag{5.37}$$

The cost to resume a task, β_d, is incurred regardless of whether a task runs to completion or is migrated yet again without completing its service in the R queue. Unfortunately it is not as straight forward as the fixed cost migration scenario. We must first find the expected original size of tasks that *visit* Host i's R queue (and incur a restart cost) but do not run to completion there. We denote this expected value as $E(\bar{X}_{iR})$, where:

$$E(\bar{X}_{iR}) = \frac{h_{iR} - h'_{iR}}{h_{iR}} \int_{s_i}^{s_n} x f(x) dx \tag{5.38}$$

$$= \frac{\alpha k^\alpha}{1 - (\frac{k}{p})^\alpha} \frac{h_{iR} - h'_{iR}}{h_{iR}} \int_{s_i}^{s_n} x^{-\alpha} dx \tag{5.39}$$

$$= \begin{cases} \frac{\alpha k^\alpha}{1-(\frac{k}{p})^\alpha} \frac{h_{iR}-h'_{iR}}{h_{iR}} \ln(\frac{s_n}{s_i}) & \text{if } \alpha = 1 \\ \frac{\alpha k^\alpha}{1-(\frac{k}{p})^\alpha} \frac{h_{iR}-h'_{iR}}{h_{iR}} \frac{{s_n}^{1-\alpha} - {s_i}^{1-\alpha}}{1-\alpha} & \text{otherwise.} \end{cases} \tag{5.40}$$

We can now easily find both $E(hostX^j_{iO})$ and $E(hostX^j_{iR})$, where:

$$E(hostX^j_{iO}) = \frac{h'_{iO}}{h_{iO}} E(X^j_{iO}) + \frac{h_{iO} - h'_{iO}}{h_{iO}} (s_i + \beta_s E(\bar{X}_{iO}))^j. \tag{5.41}$$

and,

$$E(hostX^j_{iR}) = \frac{h'_{iR}}{h_{iR}} E(X^j_{iR}) + \frac{h_{iR} - h'_{iR}}{h_{iR}} (s_i - s_{i-1} + \beta_s E(\bar{X}_{iR}) + \beta_d E(\bar{X}_{iR}))^j. \tag{5.42}$$

We again allow the migration costs at the source node and destination node to be modelled separately. If β_s and β_d are set to equal zero, the original TAPTF-WC model is obtained.

5.3.5 Choosing the cut-offs

Now that we have computed the important parameters for the TAPTF-WC policy, we can use them to find the 'best' cut-offs for our system. Like TAPTF and other size-based policies before it, the choice of these cut-offs used in TAPTF-WC can have an enormous effect on the performance of a distributed system. The principles in finding the best cut-offs are the same as described in our previous analysis of TAPTF [Broberg et al., 2005], found in Chapter 4, so we will briefly summarise the key issues. The optimal cut-offs for TAPTF-WC are influenced by the task size distribution (Bounded Pareto $B(k, p, \alpha)$) and the task arrival rate into the distributed system, λ. Given the above parameters and utilising the mathematical preliminaries described in Section 5.3.3, we can attempt to obtain optimal cut-offs (s_i) for each of our hosts in the TAPTF-WC system.

In order to produce a task assignment policy which minimises the expected waiting time and slowdown at each host, the following optimisation problems need to be addressed:

$$\text{Problem I} \qquad \text{Minimize} \sum_{i=1}^{n} E(W_{iO}) + \sum_{i=2}^{n} E(W_{iR}) \tag{5.43}$$

$$\text{Subject to} \, \rho_{iO} + \rho_{iR} < 1, 1 \leq i \leq n. \tag{5.44}$$

$$\text{Problem II} \qquad \text{Minimize} \sum_{i=1}^{n} E(S_{iO}) + \sum_{i=2}^{n} E(S_{iR}) \tag{5.45}$$

$$\text{Subject to} \, \rho_{iO} + \rho_{iR} < 1, 1 \leq i \leq n. \tag{5.46}$$

Using the optimisation problems described above, we can elect to optimise for the best mean waiting time (represented by Problem I) or mean slowdown (represented by Problem II).

Optimal mean waiting time and mean slowdown for the case of two and three hosts can be obtained, using the model found in Section 5.3.3, by solving for the optimal values of the cut-off (s_i) values and task splitting fractions (q_i) using *Mathematica* [Wolfram Research, 2003]. For the three host case, s_i's can be numerically found but the q_i's must be tuned by hand. For the case of four or more hosts, all parameters (s_i and q_i) would need to be tuned by hand.

We recall that the lower the α parameter, the higher the variability, and the smaller the percentage of tasks is that makes up 50% of the load. From the previous chapter we noted TAGS (and subsequently TAPTF, which could operate identically to TAGS by setting

$q_1 = 1.0$ when prudent) can exploit this property of the heavy-tailed distribution by running all of the (small) tasks on the first host, leaving them under light to moderate load, while the largest tasks filtered down to be eventually processed by the latter hosts. Despite operating under new assumptions, it is often prudent for the TAPTF-WC policy to act in a similar fashion - especially under extremely variable workloads.

As such, when TAPTF-WC sets $q_1 = 1.0$ it is effectively behaving as a work-conserving version of the TAGS policy. TAGS in its original form did not account for work-conserving migration, as it was most suited to batch and super-computing facilities where work-conserving migration is not guaranteed to be available, nor used for performance reason. We denote this special case of TAPTF-WC as TAGS-WC (Task Assignment based On Guessing Size - Work Conserving). In subsequent sections we will utilise TAGS-WC as a useful point of comparison with TAPTF-WC.

5.4 Analytical Comparison

An analytical comparison of the TAPTF-WC approach with a work-conserving TAGS policy (TAGS-WC) was performed in order to ascertain the performance of these respective policies. This provided the best point of comparison as both policies under similar constraints (i.e. no *a priori* knowledge of a task's service requirement, no pre-emption, work-conserving migration available). It is these very constraints that preclude a direct EQUILOAD/ADAPTLOAD [Ciardo et al., 2001; Riska et al., 2002c] and size-based [Harchol-Balter et al., 1999; Crovella et al., 1998a; Schroeder and Harchol-Balter, 2004] policies.

Additionally, each work-conserving policy was compared to its original form (where work is not conserved) - TAGS to TAGS-WC and TAPTF to TAPTF-WC. Although a direct comparison is not of great use (as they operate under different assumptions), we are interested in quantifying benefits that work-conserving migration could provide, if available.

These approaches were evaluated under a variety of conditions and their performance compared using the most important of the metrics - mean waiting time and mean slowdown.

A large range of α values were considered, from 0.5 to 2.0, demonstrating a wide spectrum of task size variation. Each α value was evaluated for different system loads (ρ) - 0.3 (low load), 0.5 (moderate load) and 0.7 (high load). Performance metrics were computed for these load scenarios for both two and three host systems.

The analytical comparison was performed in Mathematica 5.0 [Wolfram Research, 2003], using the mathematical preliminaries presented in Section 5.3.3. The generalised TAPTF-

α	q_1	q_2
1.0	0.98	0.02
1.1	0.97	0.03
1.2	0.96	0.04
1.3	0.95	0.05
1.4	0.93	0.07
1.5	0.91	0.09
1.6	0.88	0.12
1.7	0.84	0.16
1.8	0.81	0.19
1.9	0.77	0.23
2.0	0.74	0.26

(a) $E\{W\}$

α	q_1	q_2
1.0	1.00	0.00
1.1	0.99	0.01
1.2	0.99	0.01
1.3	0.98	0.02
1.4	0.96	0.04
1.5	0.94	0.06
1.6	0.91	0.09
1.7	0.88	0.12
1.8	0.84	0.16
1.9	0.80	0.20
2.0	0.77	0.23

(b) $E\{S\}$

Figure 5.2: Distribution of tasks in TAPTF-WC - 2 Hosts, $\rho = 0.3$

WC mathematical model is also used to model the behaviour of a work-conserving TAGS policy, TAGS-WC, by setting $q_1 = 1.0$ (and subsequently q_2 ... q_n to equal 0). This has the effect of negating the dual queues and multiple entry points utilised by the TAPTF-WC policy, and making it behave like a work-conserving version of the existing TAGS policy. Optimum cut-offs are found with respect to mean waiting time and mean slowdown for both TAPTF-WC and TAGS-WC using the NMinimize function in Mathematica in order to produce the best (and fairest) comparison of task assignment policies. This is achieved by finding the s_i values in each instance that produce local minimum's for the expected waiting time, $E(W)$ and the expected mean slowdown, $E(S)$.

The choice of task distribution at the dispatcher is also critical, where q_i controls the fraction of tasks assigned to Host i from the dispatcher. In the case of two hosts, we can numerically find the best values at each host for both s_i *and* q_i, resulting in ideal conditions for our task assignment policy, TAPTF-WC, for $E(W)$ *or* $E(S)$. For the case of three hosts, we can solve for optimal s_i values but must manually tune the q_i parameters. We can use common sense regarding the necessary spread of tasks, using the results from the two host case and experimentation to find good parameters. In the three host cases, we elected to use single sets of q_i parameters that resulted in both good $E(W)$ *and* $E(S)$.

We reiterate that for the purpose of clear and meaningful results, all comparisons of mean

waiting time and mean slowdown that are performed in this section utilise the respective TAPTF-WC and TAGS-WC policies optimised for that metric. The expected waiting time and slowdown graphs are presented on a log scale for the horizontal axis.

5.4.1 Two Hosts

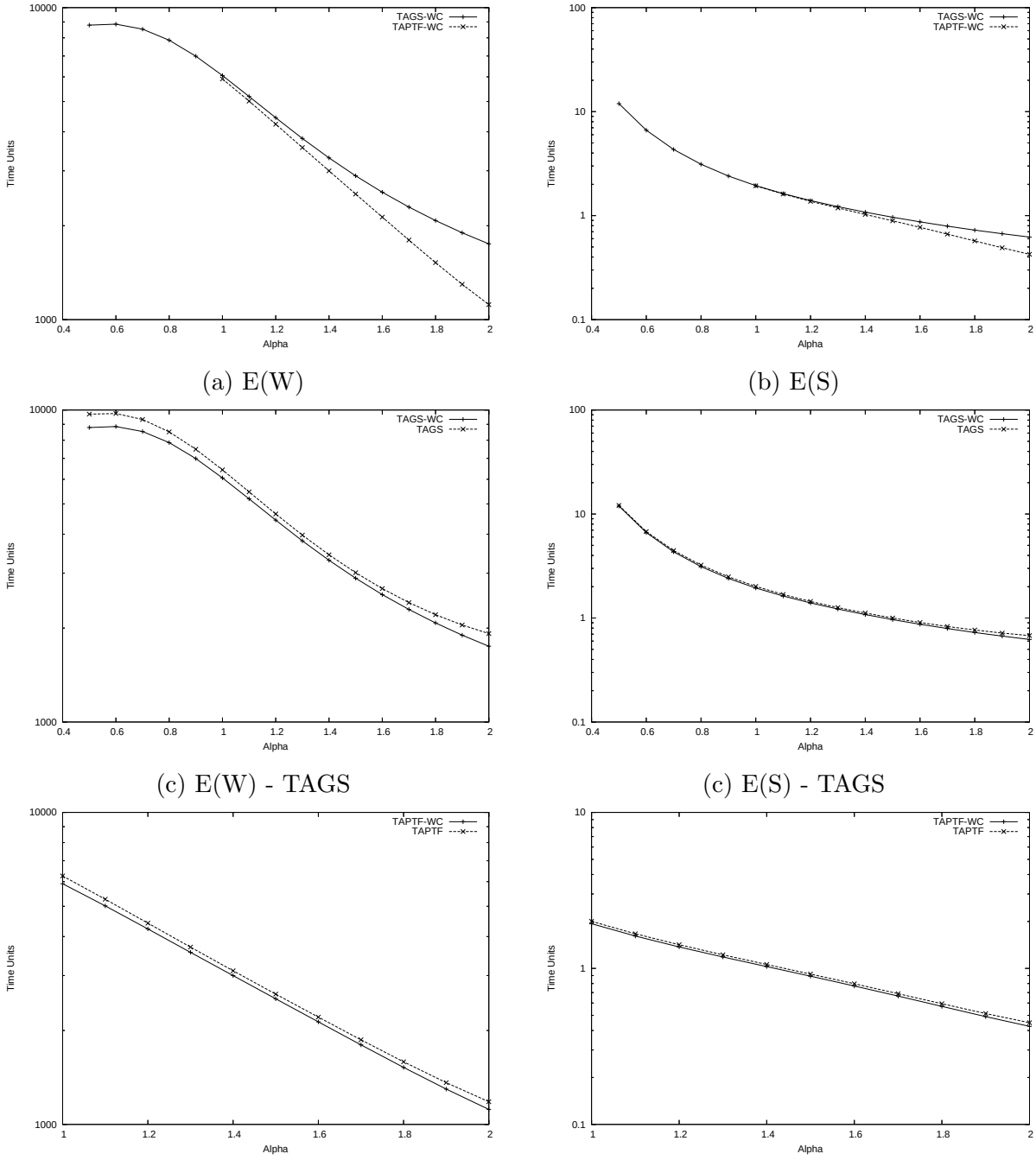
In this section we examine an analytical comparison of TAGS-WC and TAPTF-WC in a two host distributed system. A range of results are presented, showing the important performance metrics under system loads of 0.3 (Figure 5.3), 0.5 (Figure 5.5) and 0.7 (Figure 5.7). Results for TAPTF-WC are only shown where they are superior to TAGS-WC, as TAGS-WC is now a subset of the TAPTF-WC policy's behaviour, which can be enacted when prudent to do so for best performance.

From Figure 5.2 we observe that the optimal values of the q_i parameters follow similar trends to that exhibited by the TAPTF policy (in Chapter 4). Indeed, the q_i values are almost identical for the non work-conserving and work-conserving TAPTF models under this scenario. As the variation increases, less tasks are directed to the second host, until eventually no tasks are directly dispatched to that host, tending toward behaviour we done as TAGS-WC (i.e. a work-conserving TAGS model).

From Figures 5.3(a) and (b) we can see a reasonable improvement in expected waiting time (from $\alpha = 1.1$) and slowdown (from $\alpha = 1.3$) for the TAPTF-WC policy over the TAGS-WC incarnation under a low system load of 0.3. The magnitude of this improvement expands as variability decreases, approaching $\alpha = 2.0$.

Figures 5.3(c) and (d) depict the improvement that work-conserving migration (if available) can achieve in expected waiting time and slowdown for the TAGS policy. From the graphs we can see that the improvement is barely perceptible - this is expected under such low system load as the excess generated by the standard policy has little effect as the arrival rate and expected queue lengths are low. The choice of task assignment policy used is less critical under such a low load.

Figures 5.3(e) and (f) contrast the performance of the TAPTF policy and the work-conserving variation of it, denoted as TAPTF-WC. Like the TAGS policies, there is little appreciable difference in the performance of the two TAPTF variations, due to the low system load coupled with the fact that the TAPTF policy was designed in the first instance to reduce the amount of excess generated. This was achieved by using several techniques that minimising the amount of hand-offs where tasks are restarted from scratch.



(a) E(W)

(b) E(S)

(c) E(W) - TAGS

(c) E(S) - TAGS

(d) E(W) - TAPTF

(e) E(S) - TAPTF

Figure 5.3: Performance of a two host distributed system with system load of 0.3. The expected waiting time and slowdown are depicted in (a) and (b) for work-conserving policies optimised for these respective metrics. In (c), (d), (e) and (f) work-conserving and non work-conserving versions of TAGS and TAPTF are compared.

Figure 5.4 shows the optimal values of the q_i parameters for a two host case under a system load of 0.5. We can see that more tasks are assigned to the second host than was the case under a system load of 0.5. As expected, as the variation increases (α approaches 0) less and less tasks are assigned to the second host, until eventually host one receives all tasks from the dispatcher.

Figures 5.5(a) and (b) show the expected waiting time and slow down under a moderate system load of 0.5. The TAPTF-WC policy exhibits improved performance in expected waiting time (from $\alpha = 1.1$) and slowdown (from $\alpha = 1.2$) over the TAGS-WC policy. The magnitude of this performance increase grows as α approaches 2.0, with an improvement of approximately two times in expected waiting time and slowdown when $\alpha = 2.0$.

Again, as was the case under a low system load of 0.3, the magnitude of improvement of the TAGS-WC policy over the TAGS policy under a moderate system load of 0.5 is small. Figures 5.5(c) and (d) highlight this fact, despite the fact we can observe this improvement increasing slightly. Again this is unsurprising, despite the system load increasing it is still not enough to alter the performance metrics significantly.

Figures 5.5(e) and (f) contrast the performance metrics of the TAPTF policy with that of the TAPTF-WC policy. The characteristics of the graph are largely the same for both expected waiting time and slowdown, with no appreciable difference in performance. As described previously, the standard TAPTF policy already has measures in place to reduce the amount and effect of excess processing that exists due to non work-conserving migration, which explains the similarity between the results.

Figure 5.6 shows the ideal q_i parameters for a high system load of 0.7. The two trends that we previously observed continue. First, as system load increases, more tasks are assigned directly to the second host (for a given α value) from the dispatcher. Second, as the variation increases (and α decreases), less tasks are assigned to the second host, until eventually no tasks are assigned directly there from the dispatcher.

Figures 5.7(a) and (b) show the performance of TAPTF-WC and TAGS-WC under a high system load of 0.7. As α decreases (from $\alpha = 1.1$) we can see an increasing performance gain for TAPTF-WC over TAGS-WC in both expected waiting time and slowdown. When α reaches 2.0, TAPTF-WC shows a substantial improvement in performance, by factor of approximately 2.5 in expected waiting time and slowdown.

Under a high system load of 0.7 we are finally seeing a difference in performance between the standard TAGS policy, and the TAGS work-conserving policy, TAGS-WC. The excess processing generated by the TAGS policy now has a visibly adverse effect on performance,

α	q_1	q_2
0.9	0.98	0.02
1.0	0.97	0.03
1.1	0.96	0.04
1.2	0.94	0.06
1.3	0.92	0.08
1.4	0.89	0.11
1.5	0.86	0.14
1.6	0.82	0.18
1.7	0.79	0.21
1.8	0.76	0.24
1.9	0.73	0.27
2.0	0.70	0.30

(a) $E\{W\}$

α	q_1	q_2
0.9	1.00	0.00
1.0	0.99	0.01
1.1	0.99	0.01
1.2	0.98	0.02
1.3	0.96	0.04
1.4	0.94	0.06
1.5	0.90	0.10
1.6	0.87	0.13
1.7	0.83	0.17
1.8	0.79	0.21
1.9	0.76	0.24
2.0	0.73	0.27

(b) $E\{S\}$

Figure 5.4: Distribution of tasks in TAPTF-WC - 2 Hosts, $\rho = 0.5$

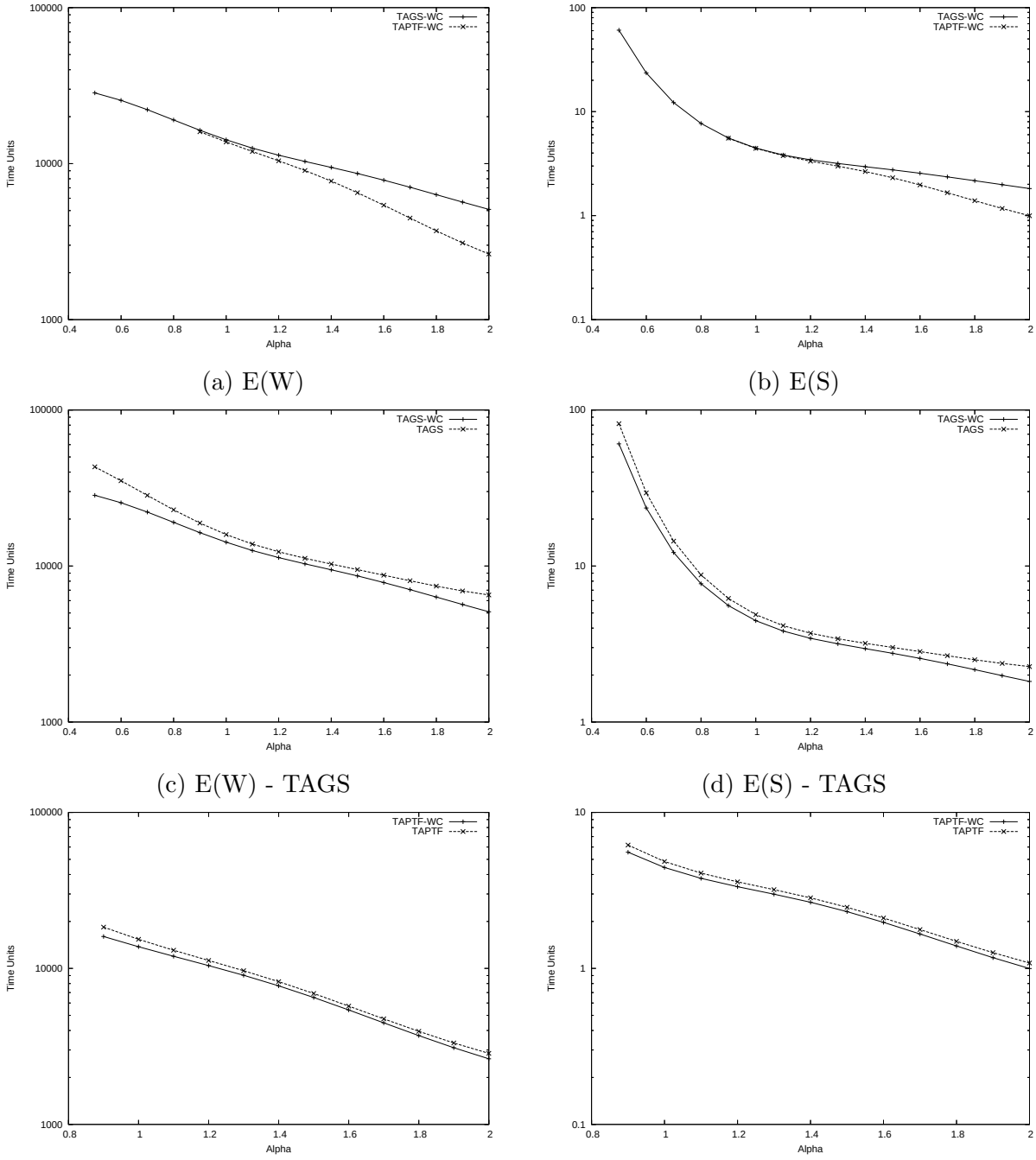
which can be seen when contrasting it with a work-conserving version of the same policy (shown in Figures 5.7(c) and (d)). When $\alpha = 2.0$, there is factor of approximately 3.5 difference between TAGS and TAGS-WC, showing the detrimental effect that excess processing causes when the arrival rate is higher and expected queue lengths increase.

Figures 5.7(e) and (f) show the performance metrics of the TAPTF policy compared with the TAPTF-WC policy. As expected the performance is still largely the same, for the same reasons given earlier in this section.

5.4.2 Three Hosts

In this section we present an analytical comparison of TAGS-WC and TAPTF-WC in a three host distributed system. The important performance metrics are shown under system loads of 0.3 (Figure 5.9), 0.5 (Figure 5.11) and 0.7 (Figure 5.13). As in the previous section, results for TAPTF-WC are only shown where they are superior to TAGS-WC.

Figure 5.8 shows the q_i parameters found (through experimentation) for a three host case, under a low system load of 0.3. These parameters are more coarse than those found (via numerical optimisation) in the two host cases. However, similar trends can be observed. We



(a) E(W)

(b) E(S)

(c) E(W) - TAGS

(d) E(S) - TAGS

(e) E(W) - TAPTF

(f) E(S) - TAPTF

Figure 5.5: Performance of a two host distributed system with system load of 0.5. The expected waiting time and slowdown are depicted in (a) and (b) for work-conserving policies optimised for these respective metrics. In (c), (d), (e) and (f) work-conserving and non work-conserving versions of TAGS and TAPTF are compared.

α	q_1	q_2
0.9	0.97	0.03
1.0	0.96	0.04
1.1	0.93	0.07
1.2	0.89	0.11
1.3	0.85	0.15
1.4	0.82	0.18
1.5	0.78	0.22
1.6	0.75	0.25
1.7	0.72	0.28
1.8	0.69	0.31
1.9	0.67	0.33
2.0	0.66	0.34

(a) $E\{W\}$

α	q_1	q_2
0.9	1.00	0.00
1.0	0.99	0.01
1.1	0.98	0.02
1.2	0.96	0.04
1.3	0.92	0.08
1.4	0.88	0.12
1.5	0.84	0.16
1.6	0.80	0.20
1.7	0.76	0.24
1.8	0.73	0.27
1.9	0.70	0.30
2.0	0.68	0.32

(b) $E\{S\}$

Figure 5.6: Distribution of tasks in TAPTF-WC - 2 Hosts, $\rho = 0.7$

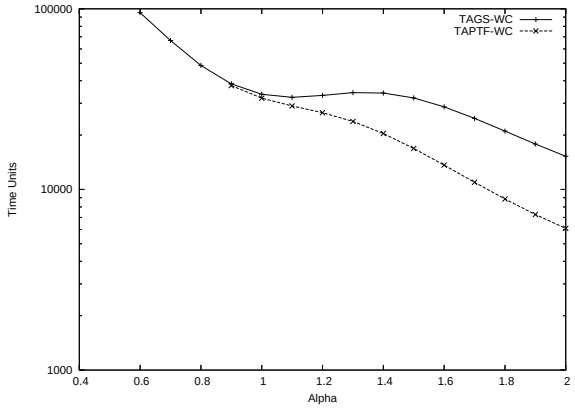
can see the majority of tasks are assigned to Host 1, while lesser proportions are assigned to Hosts 2 and 3 respectively. As variation increases, the majority of tasks are dispatched directly to Host 1, with a smaller proportion to Host 2 and no tasks are assigned to Host 3.

Figures 5.9(a) and (b) show the expected waiting time and slowdown of TAPTF-WC and TAGS-WC under a low system load of 0.3. Despite the low load, we can still see some significant improvements in these metrics over a wide range of α values, from high variation, where $\alpha = 0.9$, to lower variation, as α approaches 2.0. When $\alpha = 2.0$ TAPTF-WC improves on TAGS-WC by a factor of 2 in both expected waiting time and slowdown.

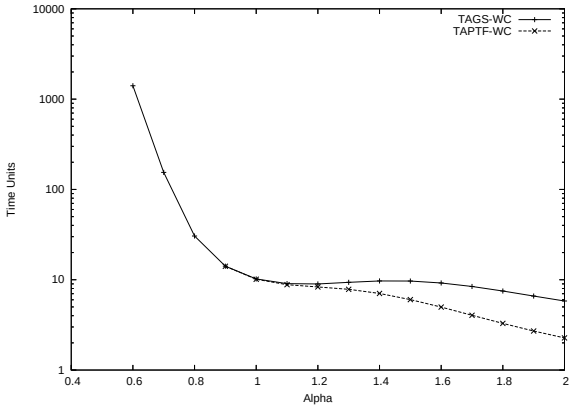
Despite the system load being low (0.3) the work-conserving TAGS policy TAGS-WC shows a clear improvement in performance over all areas examined (shown in Figures 5.9(c) and (d)). The TAGS-WC policy improves on the standard TAGS policy by an average factor of approximately 1.5 in both expected waiting time and slowdown.

Figures 5.9(e) and (f) depict a familiar picture - with the work-conserving policy TAPTF-WC showing only minor improvements over its non work-conserving original form in both expected waiting time and slowdown.

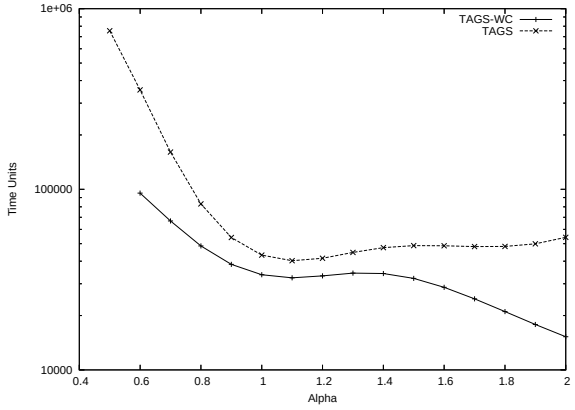
Figure 5.10 shows the q_i parameters found (again via experimentation) for a three host case, under a moderate system load of 0.5. We find in several cases (as expected) that for a



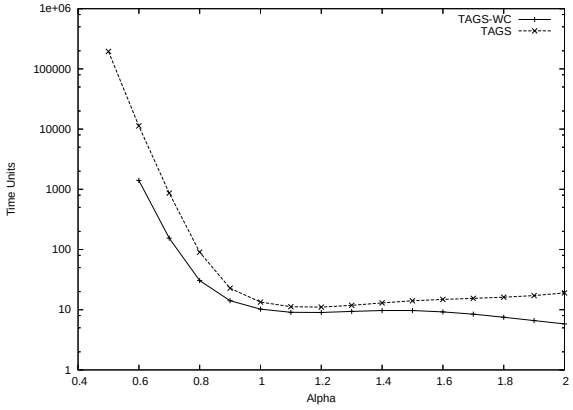
(a) E(W)



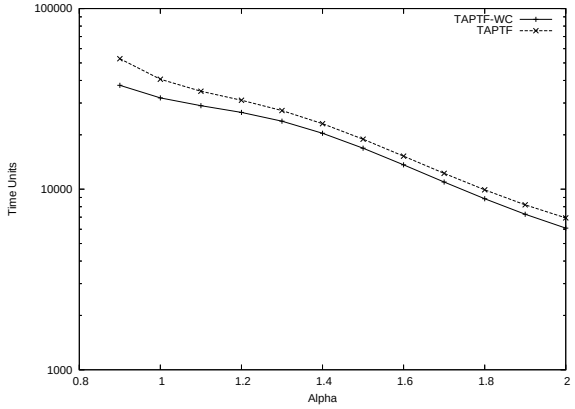
(b) E(S)



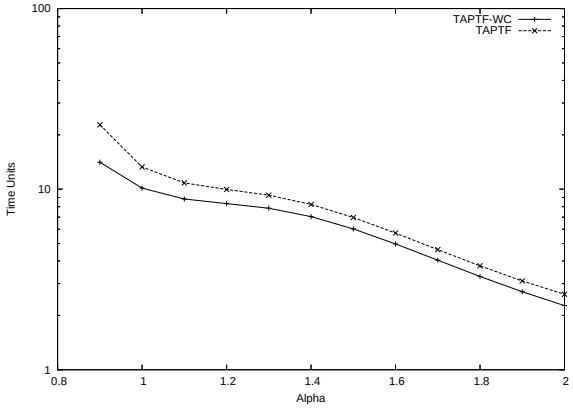
(c) E(W) - TAGS



(d) E(S) - TAGS



(a) E(W) - TAPTF



(b) E(S) - TAPTF

Figure 5.7: Performance of a two host distributed system with system load of 0.7. The expected waiting time and slowdown are depicted in (a) and (b) for work-conserving policies optimised for these respective metrics. In (c), (d), (e) and (f) work-conserving and non work-conserving versions of TAGS and TAPTF are compared.

α	q_1	q_2	q_3
0.9	0.95	0.05	0.0
1.0	0.90	0.10	0.0
1.1	0.90	0.10	0.0
1.2	0.80	0.20	0.0
1.3	0.80	0.20	0.0
1.4	0.70	0.30	0.0
1.5	0.60	0.40	0.0
1.6	0.60	0.30	0.1
1.7	0.60	0.30	0.1
1.8	0.60	0.30	0.1
1.9	0.60	0.30	0.1
2.0	0.60	0.30	0.1

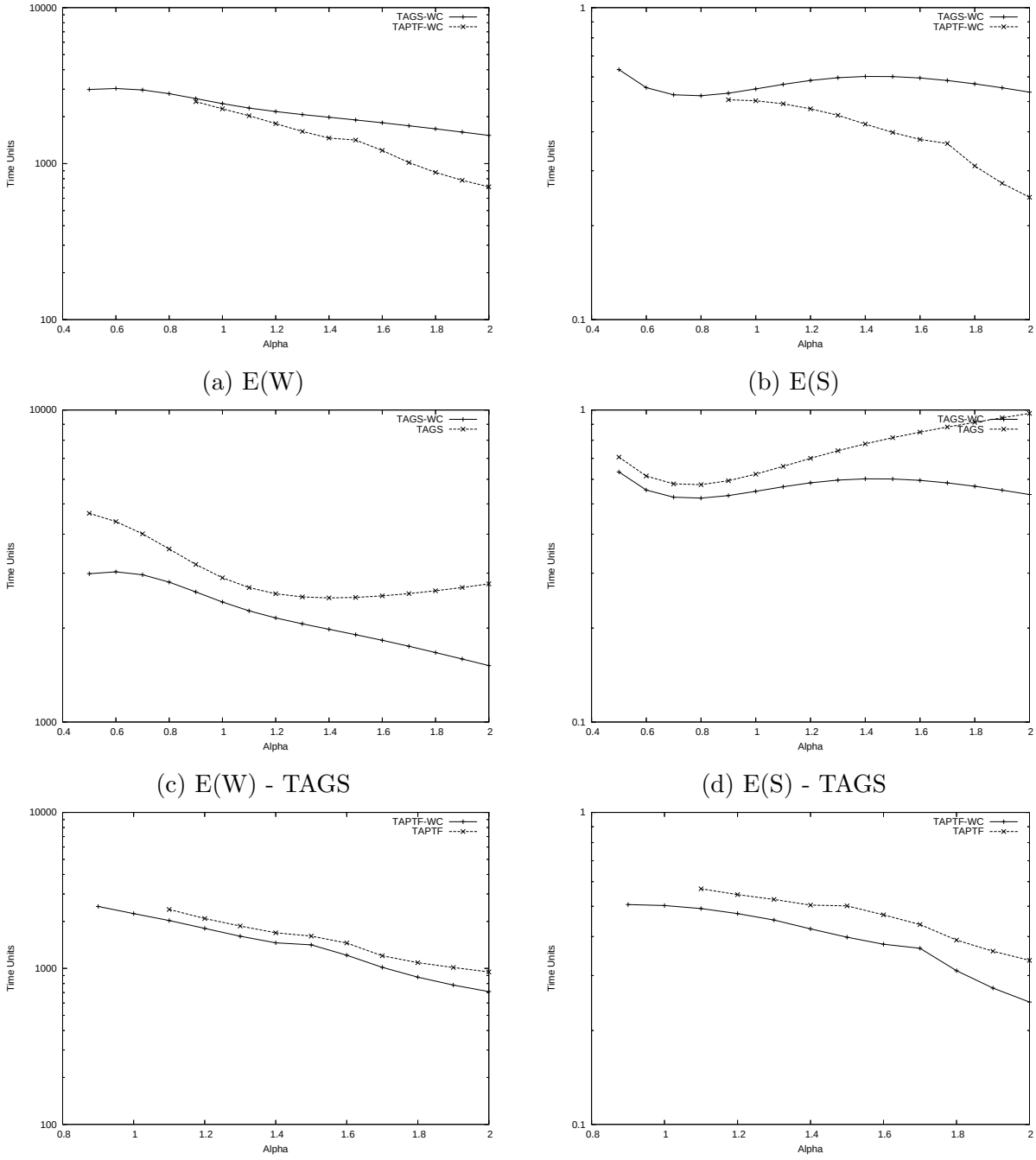
Figure 5.8: Distribution of tasks in TAPTF-WC - 3 Hosts, $\rho = 0.3$

given α value, the second host is receiving a greater proportion of tasks than was occurring under a system load of 0.3. The trends in q_i's as variation increases are consistent with all previous observations, with Host 1 receiving the vast majority of tasks, while Host 2 receives a less proportion, and Host 3 eventually receives no tasks directly from the dispatcher.

Substantial improvements can be observed in Figures 5.11(a) and (b), depicting the expected waiting time and slowdown respectively under a moderate system load of 0.5. We see that TAPTF-WC exhibits improved expected waiting time and slowdown from an α value of 0.9 (denoting a highly variable workload) to areas of lower variation in the workload (as α approaches 2.0). When $\alpha = 1.9$ we can observe that TAPTF-WC has an approximate improvement of 2.5 times over TAGS-WC. This is a substantial improvement under conditions of only moderate system load.

TAGS-WC exhibits a clear (and in some cases substantial) improvement over the standard TAGS policy under a moderate load of 0.5. Figures 5.11(c) and (d) show the expected waiting time and slowdown respectively. We can see that meaningful performance gains could be achieved if work-conserving migration is available to be utilised in a given distributed system. When $\alpha = 1.9$, TAGS-WC shows an improvement of approximately 4 times in both expected waiting time and slowdown.

Figures 5.11(e) and (f) show the largest difference in performance between TAPTF-



(a) E(W)

(b) E(S)

(c) E(W) - TAGS

(d) E(S) - TAGS

(e) E(W) - TAPTF

(f) E(S) - TAPTF

Figure 5.9: Performance of a three host distributed system with system load of 0.3. The expected waiting time and slowdown are depicted in (a) and (b) for work-conserving policies optimised for these respective metrics. In (c), (d), (e) and (f) work-conserving and non work-conserving versions of TAGS and TAPTF are compared.

α	q_1	q_2	q_3
0.9	0.95	0.05	0.0
1.0	0.90	0.10	0.0
1.1	0.80	0.20	0.0
1.2	0.75	0.25	0.0
1.3	0.70	0.30	0.0
1.4	0.60	0.40	0.0
1.5	0.60	0.30	0.1
1.6	0.60	0.30	0.1
1.7	0.60	0.30	0.1
1.8	0.60	0.30	0.1
1.9	0.60	0.30	0.1
2.0	0.60	0.30	0.1

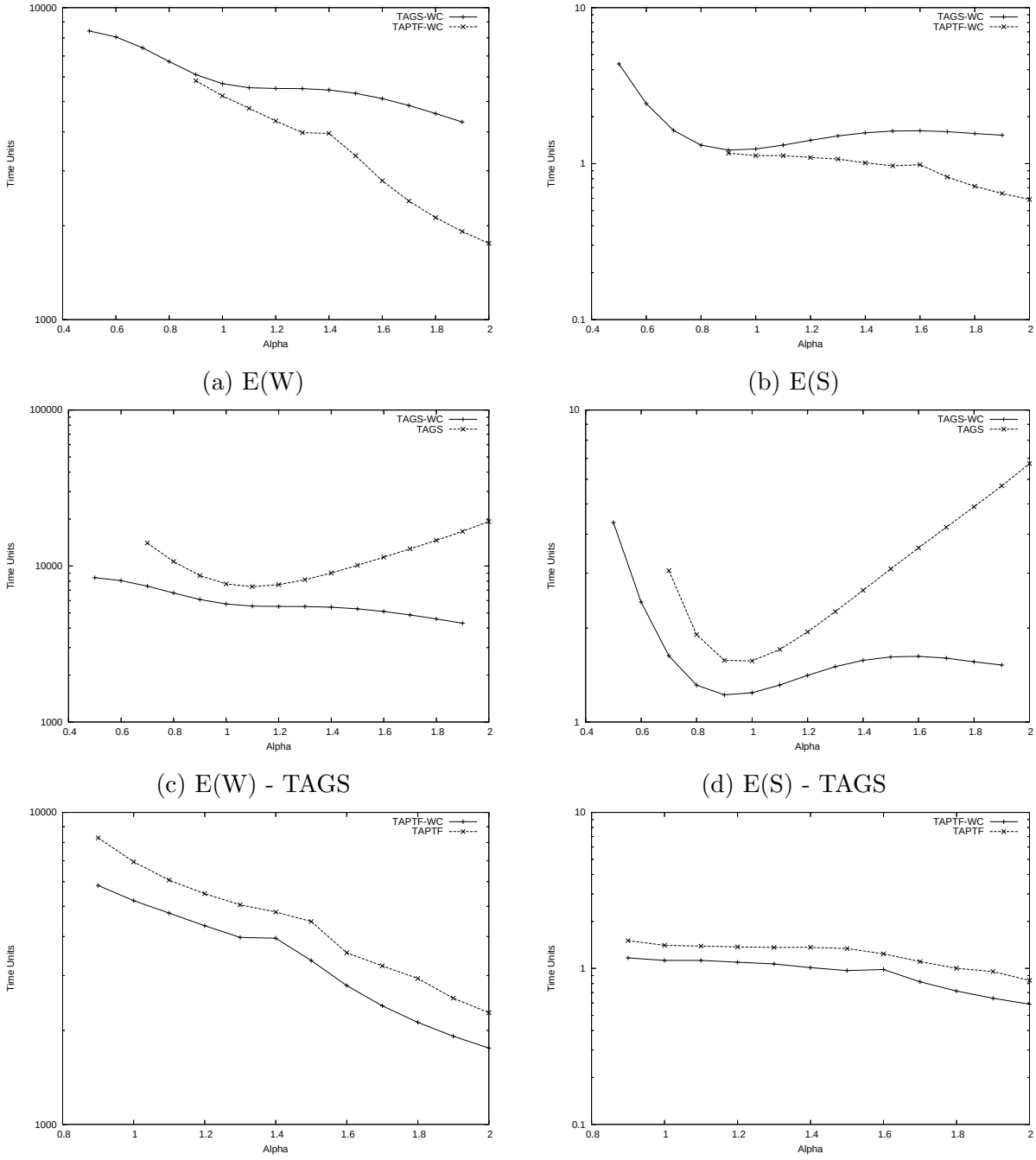
Figure 5.10: Distribution of tasks in TAPTF-WC - 3 Hosts, $\rho = 0.5$

WC and TAPTF observed so far. From the figures we can observe a consistent factor of improvement from the TAPTF-WC policy of 1.3 times in expected waiting time. This is over the entire range of α values observed, from $\alpha = 0.9$ (highly variable) to $\alpha = 2.0$ (low variation). Similarly, a factor of improvement of 1.4 times can be seen in expected slowdown, when comparing TAPTF-WC to TAPTF over the same range of α values.

The q_i parameters found for a three host system with a high system load of 0.7 are depicted in Figure 5.12. By contrasting the q_i parameters used here with those used for lower loads (under three hosts), we can see a subtle trend where it is beneficial to dispatch more tasks to the second and third hosts as system load rises.

Figures 5.13(a) and (b) show the expected waiting time and slowdown of of our three host system under a high load of 0.7. A clear benefit can be seen in both waiting time and especially slowdown, ranging from conditions of high variation ($\alpha = 0.9$) and expanding in magnitude as α approaches 2. As was the case under high load (for three hosts) in the previous chapter for TAGS, TAGS-WC is unable to operate in certain workload conditions. Specifically, when $\alpha > 1.6$ no suitable cut-offs can be found that keep the load below 1 at all hosts. When $\alpha = 1.6$ TAPTF-WC has an approximate improvement over TAGS-WC in waiting time and slowdown of a factor of 2.2 times.

Figures 5.13(c) and (d) highlights the enormous benefit that work-conserving migration



(a) E(W)

(b) E(S)

(c) E(W) - TAGS

(d) E(S) - TAGS

(e) E(W) - TAPTF

(f) E(S) - TAPTF

Figure 5.11: Performance of a three host distributed system with system load of 0.5. The expected waiting time and slowdown are depicted in (a) and (b) for work-conserving policies optimised for these respective metrics. In (c), (d), (e) and (f) work-conserving and non work-conserving versions of TAGS and TAPTF are compared.

α	q_1	q_2	q_3
0.9	0.95	0.05	0.0
1.0	0.9	0.1	0.0
1.1	0.8	0.2	0.0
1.2	0.7	0.3	0.0
1.3	0.6	0.4	0.0
1.4	0.6	0.3	0.1
1.5	0.6	0.3	0.1
1.6	0.6	0.3	0.1
1.7	0.6	0.3	0.1
1.8	0.6	0.3	0.1
1.9	0.5	0.3	0.2
2.0	0.5	0.3	0.2

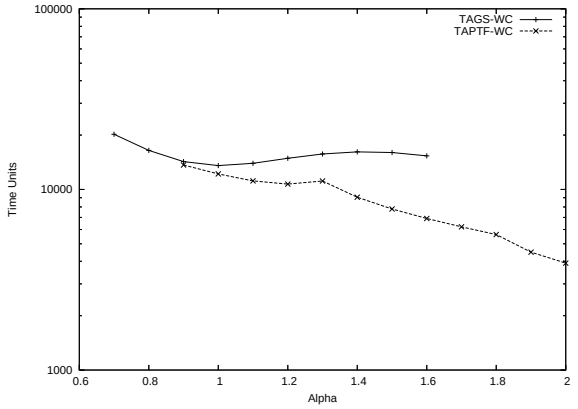
Figure 5.12: Distribution of tasks in TAPTF-WC - 3 Hosts, $\rho = 0.7$

(if available) would have for a TAGS policy under high system load. We can see both the waiting time and slowdown maintain a relatively flat response, and the addition of work-conserving migration allows TAGS-WC to service a wider range of workload variations than TAGS could.

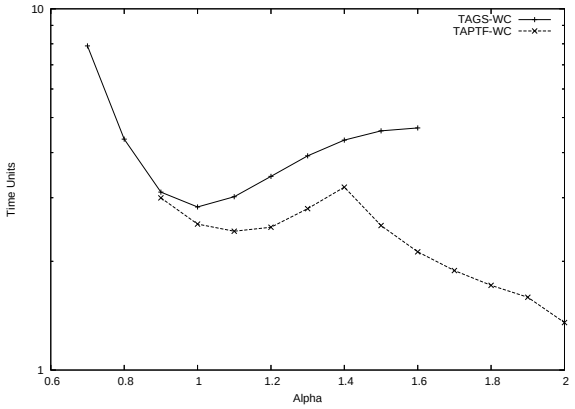
TAPTF and its work-conserving variant, TAPTF-WC, are shown Figures 5.13(e) and (f) for three hosts under a high system load of 0.7. We can observe that the improvement in waiting time and particularly slowdown has expanded as the system load increased. When $\alpha = 2$ we observed an improvement factor of 1.6 for waiting time and 1.84 in slowdown for the TAPTF-WC policy over the standard TAPTF policy.

5.5 Analytical Comparison - Cost-based migration

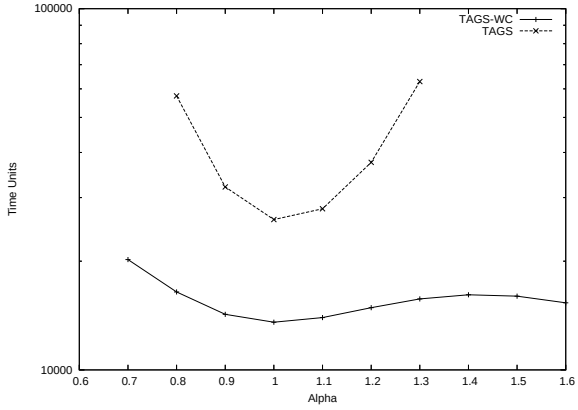
In this section we examine a selection of scenarios where the act of task migration incurs a cost to either or both of the source host and the destination host, as per the updated TAPTF-WC model described in Section 5.3.4. We consider the case where the migration costs are fixed, as well as where the costs are proportional to the original size of the task. In each instance we consider both the migration costs (incurred by the source host) and the resumption costs (incurred by the destination host). For brevity, in both instances we



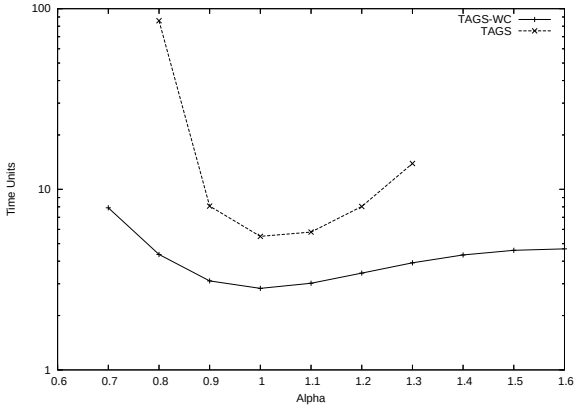
(a) E(W)



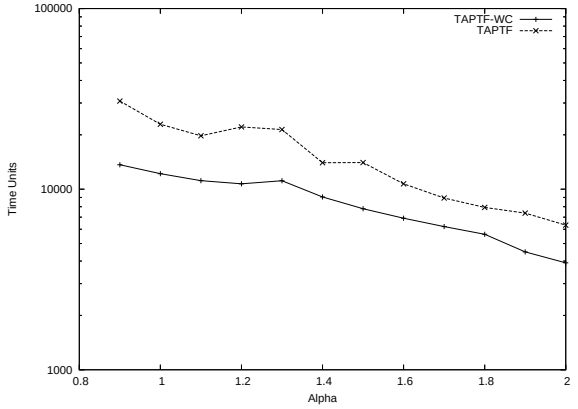
(b) E(S)



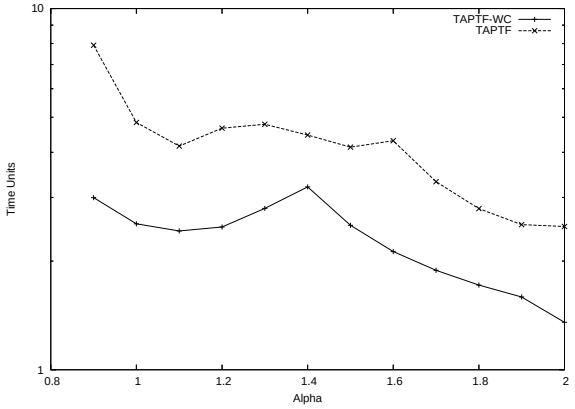
(c) E(W) - TAGS



(d) E(S) - TAGS



(e) E(W) - TAPTF



(f) E(S) - TAPTF

Figure 5.13: Performance of a three host distributed system with system load of 0.7. The expected waiting time and slowdown are depicted in (a) and (b) for work-conserving policies optimised for these respective metrics. In (c), (d), (e) and (f) work-conserving and non work-conserving versions of TAGS and TAPTF are compared.

only consider cases where the migration and resumption costs are equivalent. One could also easily model cases where the migration cost dominates the resumption cost, or vice versa - the combinations are too numerous to enumerate them all. The loads and task size distribution are varied as described in the previous section. We compare each TAGS-WC and TAPTF-WC model (and their cost-based variants) to the original TAGS and TAPTF models. While they are not necessarily directly comparable (as they are designed for different application domains), it provides an interesting point of reference to consider the benefits of work-conserving migration, if available, and the potentially negative effect of migration costs that *could* occur in certain circumstances.

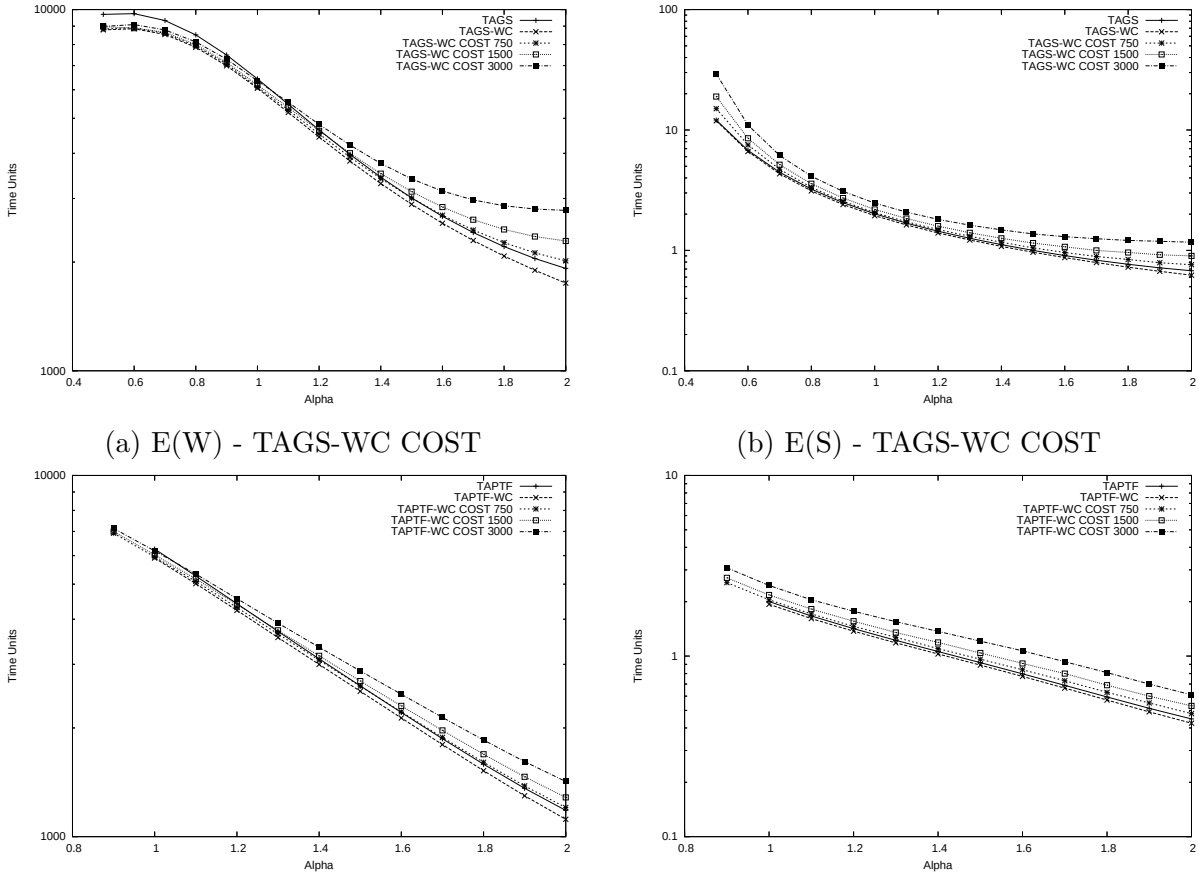
5.5.1 Fixed cost migration

In Figures 5.14, 5.15 and 5.16 we consider TAGS-WC and TAPTF-WC systems that incur a fixed cost for task migration, under system loads of 0.3, 0.5 and 0.7. We vary the fixed migration and resumption costs examining scenarios where $\gamma_s = \gamma_d = 750$, 1500 and 3000.

Figures 5.14(a) and (b) show the TAGS-WC policy, contrasted against the original TAGS as well as TAGS-WC models that incorporate fixed costs for each task migration, under a system load of 0.3. From the waiting time depicted in Figure 5.14(a) we can see that when $\alpha \geq 1.2$ any gains made by a work-conserving TAGS policy are countered by the fixed migration cost, with the original TAGS model performing marginally better than any of the fixed cost based TAGS-WC models. Conversely, we can see that as α approaches 0.5, work-conserving migration shows some minor improvement, even when incurring costs. Figure 5.14(b) depicts a comparison of slowdown, where the results show few surprises. As the migration cost for TAGS-WC increases, the slowdown gets incrementally worse.

Figures 5.14(c) and (d) show the expected waiting time and slowdown for TAPTF, TAPTF-WC and the TAPTF-WC fixed cost variants. There is only a marginal difference in performance in all instances, with both waiting time and slowdown becoming incrementally worse as the fixed migration cost increases. Even at the lowest fixed migration cost ($\gamma_d = \gamma_s = 750$), the original TAPTF model (which restarts tasks from scratch) performs better in all instances.

The expected waiting time and slowdown under a system load of 0.5 for the TAGS variants are shown in Figures 5.15(a) and (b). From Figure 5.15(a) we can see a trend emerging, where the cost based TAGS-WC policies suffer as the variation decreases (α approaches 2.0), whilst maintaining good expected waiting time as the variation increases (α approaches 0.5)



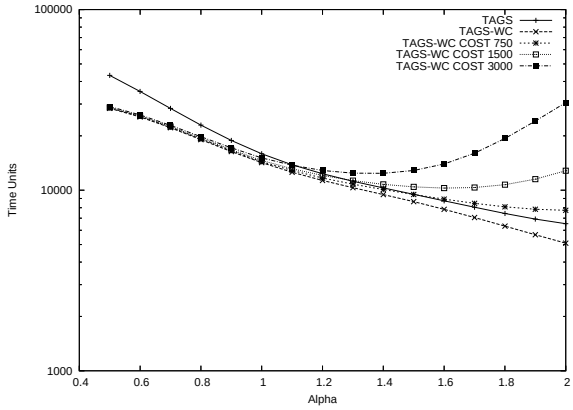
(a) E(W) - TAGS-WC COST

(b) E(S) - TAGS-WC COST

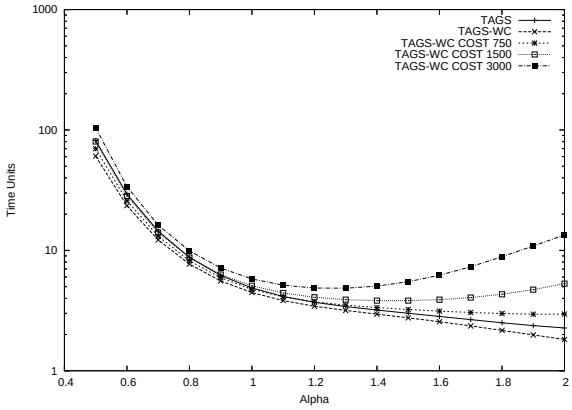
(c) E(W) - TAPTF-WC COST

(d) E(S) - TAPTF-WC COST

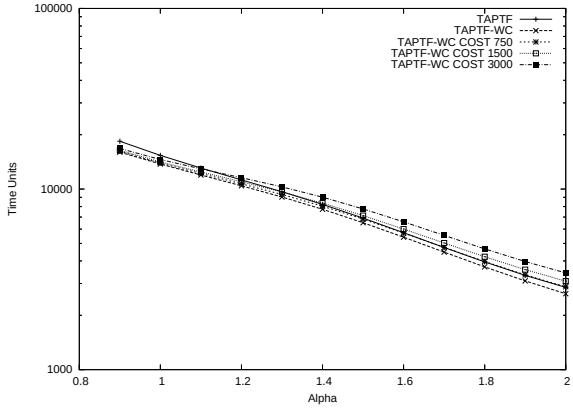
Figure 5.14: Performance of a two host distributed system with system load of 0.3 and a fixed migration cost, where $\gamma_s = \gamma_d$.



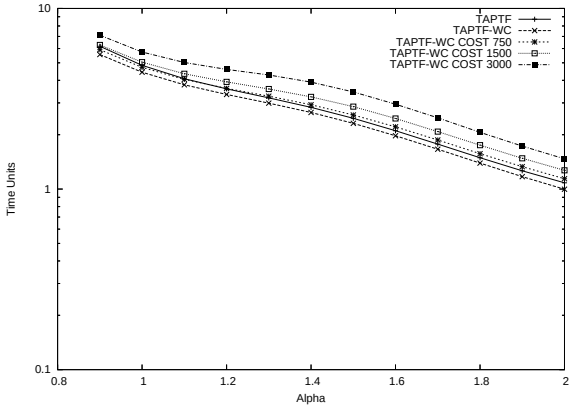
(a) E(W) - TAGS-WC COST



(b) E(S) - TAGS-WC COST

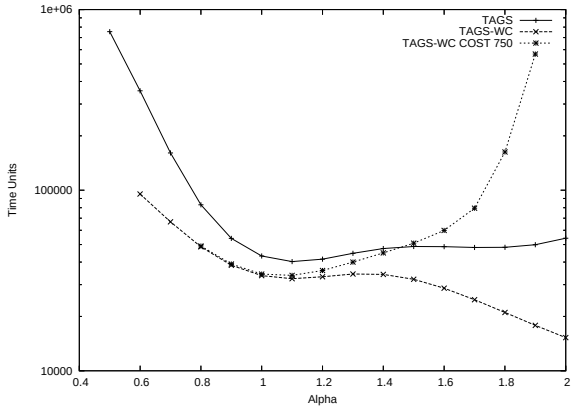


(c) E(W) - TAPTF-WC COST

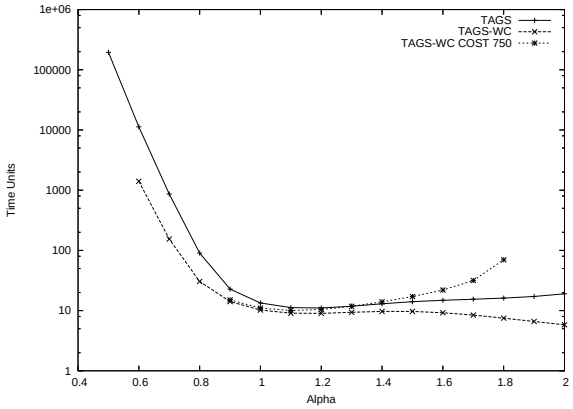


(d) E(S) - TAPTF-WC COST

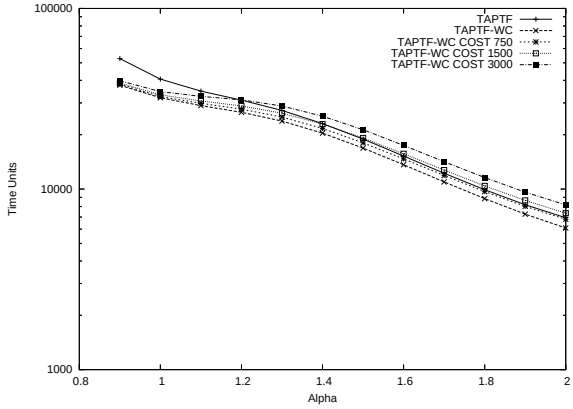
Figure 5.15: Performance of a two host distributed system with system load of 0.5 and a fixed migration cost, where $\gamma_s = \gamma_d$.



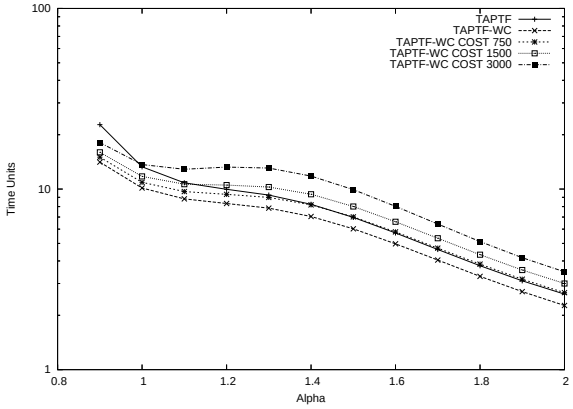
(a) E(W) - TAGS-WC COST



(b) E(S) - TAGS-WC COST



(c) E(W) - TAPTF-WC COST



(d) E(S) - TAPTF-WC COST

Figure 5.16: Performance of a two host distributed system with system load of 0.7 and a fixed migration cost, where $\gamma_s = \gamma_d$.

despite high fixed migration and resumption costs. With respect to the cost-based TAGS-WC models, the slowdown increases significantly as the task size variation decreases, especially where the fixed migration cost is high ($\gamma_d = \gamma_s = 3000$).

Figures 5.15(c) and (d) show the expected waiting time and slowdown for the TAPTF variants. Again we see little appreciable difference between them, with waiting time and slowdown increases gradually as the migration cost rises.

Figures 5.16(a) and (b) show the waiting time and slowdown of various TAGS models under a high system load of 0.7. In this case, the TAGS-WC cost model is only shown where $\gamma_d = \gamma_s = 750$. No parameters could be found for $\gamma_d = \gamma_s = 1500$ or 3000 as the load could not be kept below 1 at each host. However, we can still see a continuing trend, where the TAGS-WC cost-based model suffers as the task size variation decreases, as more and more tasks are migrated given the load must be shared more evenly to keep the load below 1 on both hosts. As costs are incurred for each of these migrations, the waiting time rapidly deteriorates. However, as the variation increases, the penalty TAGS incurs from restarting tasks from scratch is much higher than the fixed cost of migrating a task in a work-conserving fashion.

There are few surprises when considering the waiting time and slowdown for the TAPTF variants, depicted in Figures 5.16(c) and (d). However, we note that as the task size variation increases the original TAPTF model has the highest waiting time and slowdown as compared to TAPTF-WC and each of its fixed cost-based variants.

5.5.2 Proportional cost migration

In Figures 5.17, 5.18 and 5.19 we consider TAGS-WC and TAPTF-WC systems that incur a proportion cost for task migration, under system loads of 0.3, 0.5 and 0.7. The proportional migration and resumption costs are varied to examine scenarios where $\beta_s = \beta_d = 0.25$, 0.5 and 0.75. This equates to penalties on the source and destination hosts of 25%, 50% and 75% of a task's original service requirement respectively.

Figures 5.17(a) and (b) show the expected waiting time and slowdown for the TAGS variants under a low system load of 0.3. For both the expected waiting time and slowdown, as the per-unit migration costs increase, the metrics increase proportionally. We note that there is increased divergence as the task size variation increases and the per-unit costs increase. This makes intuitive sense as there tend to be more larger tasks, and as the migration costs are proportional to the task size, there is increased overhead to migrate these tasks.

The expected waiting and slowdown times for various TAPTF policies are shown Figures 5.17(c) and (d). The results become incrementally worse as the per-unit migration costs increases for both waiting time and slowdown. There is again some divergence as the task size distribution becomes more variable, in conjunction with increased per-unit migration cost. However, this is less pronounced than what was observed for the TAGS variants above.

The TAGS variants are examined under a moderate system load of 0.5 in Figures 5.18(a) and (b), with the TAPTF variants examined under the same conditions in Figures 5.18(c) and (d). Given that the migration costs are proportional to the task size, the order of performance of the various TAGS and TAPTF variants in comparison to each other remain unchanged. We begin to note the effect of the increased system load in conjunction with increasing migration costs for the TAGS-WC cost-based variant, as they struggle to keep the load below 1 at each host.

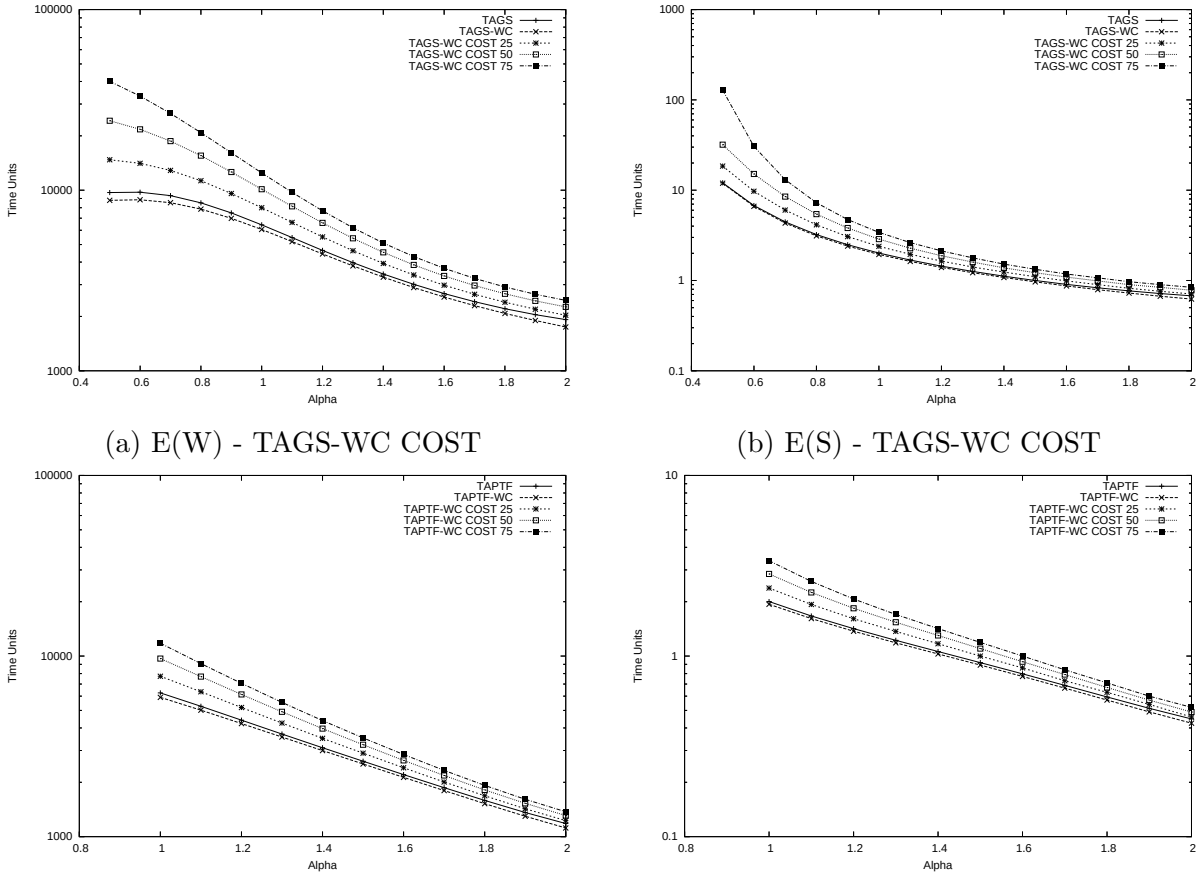
Figures 5.19(a) and (b) show the various TAGS policies under a high system load of 0.7. Consistent with what occurred under this system load with a fixed migration cost, optimal settings for TAGS-WC with proportional migration costs could only be computed where $\beta_s = \beta_d = 0.25$. In this particular case, results could only be obtained where $0.9 \leq \alpha \leq 2.0$. This is due to an inability to keep the load at both hosts under 1 in other instances. The results obtained for TAGS-WC where $\beta_s = \beta_d = 0.25$ are consistently worse than TAGS and TAGS-WC in nearly all cases.

Figures 5.19(c) and (d) show the results for the TAPTF variants, which are consistent with all previously observed results. We can see the combination of high load and high per-unit migration costs are beginning to take its toll on the TAPTF-WC cost-based variants shown. This is especially the case as the task size variation increases. Indeed, with larger tasks occurring more regularly, the cost of migrating these tasks is high.

5.6 Discussion

In this section we consider the implications of the results presented in Section 5.4 and Section 5.5. Results for the two host scenario were shown in Section 5.4.1, and the three host scenario were presented in Section 5.4.2. The results for a fixed migration cost model was presented in Section 5.5.1, whilst proportional migration cost results were shown in Section 5.5.2

Figure 5.3 shows a comparison of expected waiting time and slowdown for the two work-conserving policies we are interested in, TAPTF-WC and TAGS-WC, under a two host system



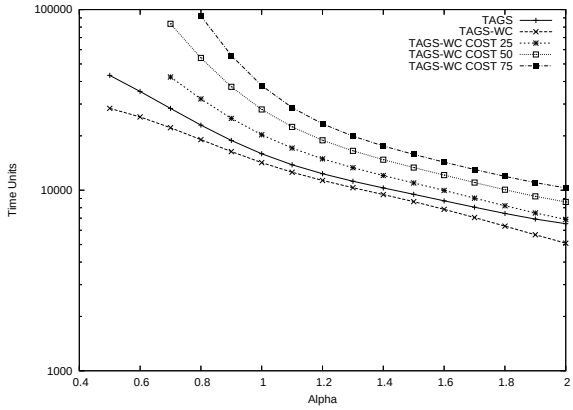
(a) E(W) - TAGS-WC COST

(b) E(S) - TAGS-WC COST

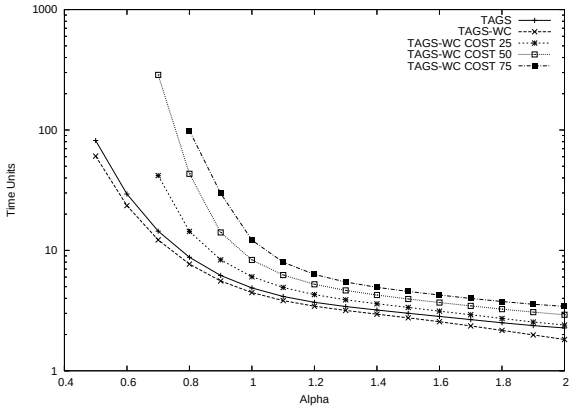
(c) E(W) - TAPTF-WC COST

(d) E(S) - TAPTF-WC COST

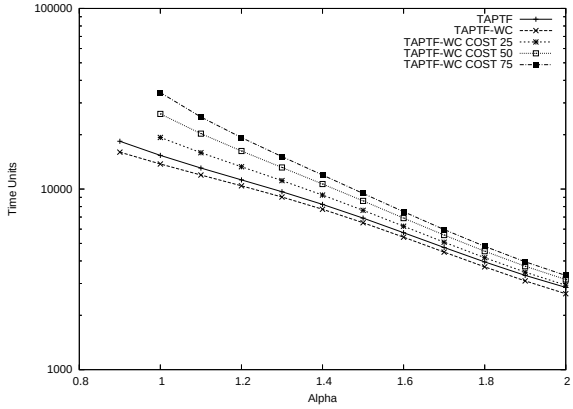
Figure 5.17: Performance of a two host distributed system with system load of 0.3 and a proportional migration cost, where $\beta_s = \beta_d$.



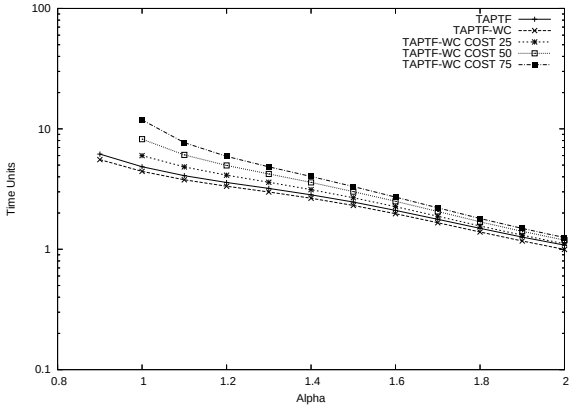
(a) E(W) - TAGS-WC COST



(b) E(S) - TAGS-WC COST

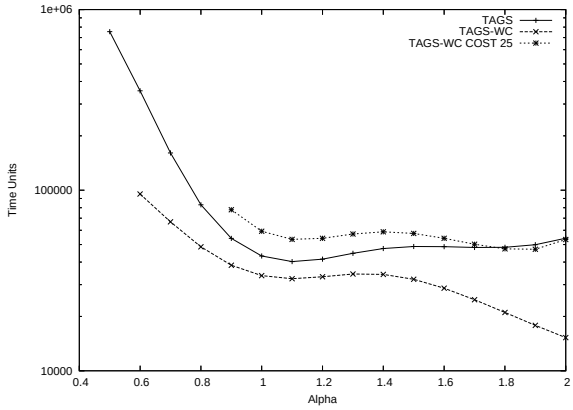


(c) E(W) - TAPTF-WC COST

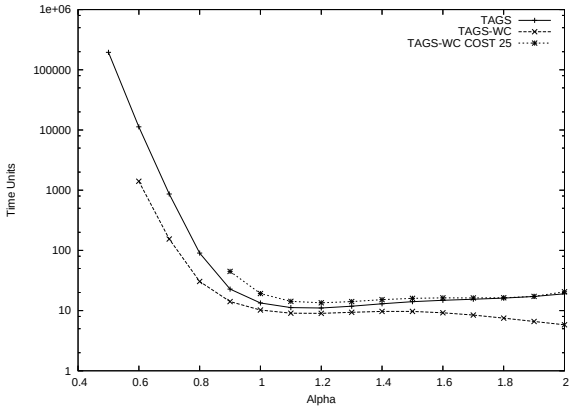


(d) E(S) - TAPTF-WC COST

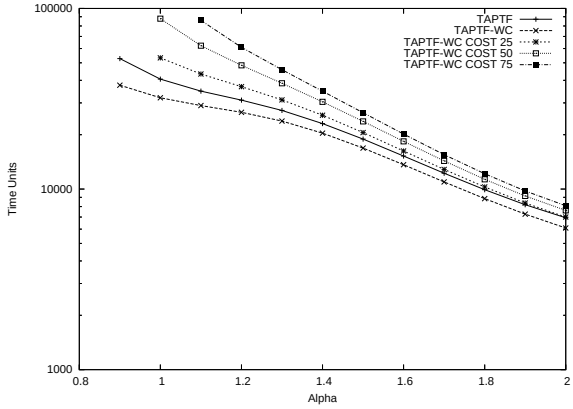
Figure 5.18: Performance of a two host distributed system with system load of 0.5 and a proportional migration cost, where $\beta_s = \beta_d$.



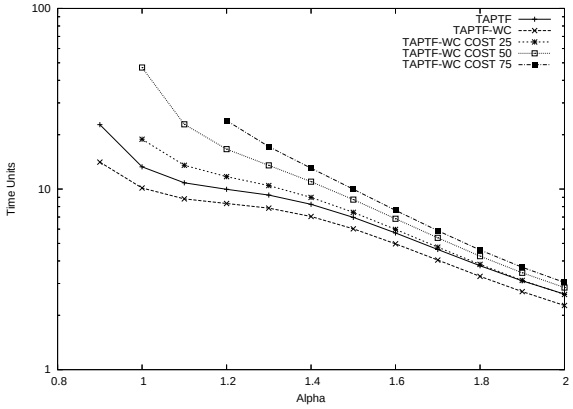
(a) E(W) - TAGS-WC COST



(b) E(S) - TAGS-WC COST



(c) E(W) - TAPTF-WC COST



(d) E(S) - TAPTF-WC COST

Figure 5.19: Performance of a two host distributed system with system load of 0.7 and a proportional migration cost, where $\beta_s = \beta_d$.

with a system load of 0.3. Under such a low system load we expected very little difference in performance, as even naive task assignment policies generally perform well. Here we see only a slight improvement in performance as the variation in task sizes decreases. This is because the TAPTF-WC policy capitalises on this reduction in variance by spreading the influx of tasks between the two hosts rather than only allowing tasks to enter at the first host.

The visible difference in performance between the work and non work-conserving TAPTF and TAGS policies is barely perceptible, again due to the low utilisation levels on all hosts. What excess that is generated by the non work conserving TAPTF and TAGS policies is not enough to significantly effect the performance metrics (which is immediately apparent when compared to their work-conserving variants).

Figure 5.5 shows the two policies under a moderate system load of 0.5. Since the utilisation level has increased we are seeing a more noticeable performance difference between the TAPTF-WC and TAGS-WC policies over a wider range of traffic variation - most obvious under moderate load, where TAPTF-WC is exploiting both hosts rather than allowing the first host to be disproportionally utilised as can be the case under the TAGS-WC policy. This results in better performance for the TAPTF-WC policy, because as variation decreases TAGS-WC can no longer exploit the 'heavy-tailed' properties of the task size distribution, and is in fact disadvantaged by its own design.

The difference between a system with work-conserving migration available to one without (illustrated again by contrasting the two variations of the TAPTF and TAGS policies) is more noticeable under the higher system load, but are still small. The characteristic shape of each metric graph are largely the same, with little divergence visible in comparisons for expected waiting time and slowdown.

The two host system is depicted under a high system load of 0.7 in Figure 5.7. Under this higher system load we can observe a substantial improvement in TAPTF-WC over the TAGS-WC policy, increasing in magnitude as α increases and variation decreases. Again these gains are made by exploiting the second (potentially idle) host more as variation decreases, rather than allowing the first host to be disproportionally loaded by making it the only point of entry for tasks. This is especially important under high loads, as utilisation has increased and the expected queue length is longer, requiring more intelligent use of the resources available to us to maintain good performance. Techniques that TAPTF-WC utilises such as enabling multiple entry points and separating short and long task flows help maintain these performance levels, even under high load.

Figure 5.9 depicts the expected waiting time and slowdown for TAPTF-WC and TAGS-

WC, under a three host system with a low system load of 0.3. Even under this low load we can see an appreciable improvement in performance shown by TAPTF-WC under a wide range of variation scenarios. This again can be attributed to the fact that we have more flexibility in our task assignment policy when comparing to a work-conserving TAGS approach, with TAPTF-WC able to utilise the available hosts more fully. This trend continues (with the performance improvement expanding) under a system load of 0.5.

Similar trends continue as we consider a three host system under a high system load of 0.7, depicted in Figure 5.13. TAPTF-WC shows a relatively larger performance improvement over TAPTF, but the most interesting result is the performance of TAGS-WC. We can see that without the burden of non work-conserving migration (and the wasted processing associated with it), TAGS-WC is more suited to processing high system loads than TAGS was. Over a wide range of workloads, TAGS-WC shows a relatively flat response for both waiting time and slowdown, until it too is eventually unable to service the workload (when $\alpha > 1.6$).

While the addition of work conservation only offers a small improvement between the two TAPTF variants under low and moderate load, the same is not true of the TAGS variants. TAGS by its very design can be a wasteful task assignment policy, due to only having a single entry point for all tasks. Under certain scenarios (under task size distributions of moderate to low variation) the number of hand-offs can be significant. The negative effect of these hand-offs are compounded when combined with high system loads. A significant amount of wasted processing occurs when tasks are migrated and restarted from scratch, with any prior work being lost. As such a TAGS policy had the most to gain by adding work-conserving migration. This is clearly depicted in Figures 5.9, 5.11 and 5.13.

Whilst not the focus of this chapter, we briefly explored the effect of a fixed cost migration (incurred by both the source host to migrate and the receiver host to resume) in Section 5.5. Fixed costs were examined where the migration and resumption costs were 750, 1500 and 3000 units. This equates to 25%, 50% and 75% of the mean task size respectively. These costs were chosen as they are quite high, with a migration cost that is equivalent to the mean being an extreme case. As we observed from Figures 5.14, 5.15 and 5.16 in a large number of cases you would achieve better mean waiting time and slowdown by restarting tasks from scratch when there is a large cost involved to migrate a task in a work-conserving fashion. However, the results are important for tasks that, due to the requirements of the application domain (e.g. interactive tasks, web requests) cannot simply be restarted from scratch, such as those described in 5.1.1. For the TAGS variants, some interesting cases occurred under higher loads. With respect to waiting time, a fixed-cost TAGS-WC maintained respectable

waiting time under extreme task size variation, as α approaches 0.5. Under a low system load, fixed-cost migration had little effect on the expected metrics for both the TAGS and TAPTF variants.

Proportional per-unit migration costs were examined where the migration and resumption costs were 25%, 50% and 75% of the original task size of the task to be migrated. The respective results found for both TAGS and TAPTF variants (in relation to each other) where consistent, with performance decreasing as the proportional migration cost increased. As with the fixed cost scenarios, costs of 25%, 50% and 75% would be considered high but provide an interesting view of the performance and benefit of migration under such crippling costs.

5.7 Conclusion

In this chapter we presented work-conserving extensions of the TAPTF policy, TAPTF-WC. We also presented work-conserving variants of the TAGS task assignment policy (denoted as TAGS-WC) as a special case of TAPTF-WC. We modelled TAPTF-WC (and TAGS-WC) where migration costs were negligible, fixed and proportional to the original service requirement of a task. Where migration costs were negligible, TAPTF-WC exhibited consistently good performance over a wide range of task distribution scenarios due to its flexible nature, spreading the work over multiple hosts when prudent, and separating short task flows from large via the use of dual queues. Tasks are migrated in a work-conserving manner, reducing the penalty associated with task migration found in existing policies such as TAGS and TAPTF which restart tasks upon migration. This makes the TAPTF-WC policy well suited to environments where work-conserving migration facilities are available, such as web server clusters (as described in Section 5.1). When comparing TAGS and TAPTF to their respective work-conserving variants, we noted that TAGS-WC showed a larger improvement. This was due to the more wasteful nature of the TAGS policy under scenarios of high system load and more moderate task size variation, which was rectified by adding work-conserving migration (TAGS-WC). A TAGS-WC approach is useful in conditions of high to extreme variation, being ideally suited (by design) to such scenarios. Conversely, TAPTF-WC is well suited to conditions of high to moderate and low variation, and high system loads. This is due to its ability to deal with both highly variable workloads, keeping small and large tasks from interfering with each other via dual queues and migration, and moderately variable workloads, exploiting idle hosts as the tasks become more uniformly sized. As such, we find

that the TAPTF-WC policy (encompassing TAGS-WC) is well suited for load distribution under a wide range of different workloads in environments where task sizes are not known *a priori* and negligible cost work-conserving migration is available.

Chapter 6

Approximating General service distributions

Exponential distributions have traditionally been used to model the traffic (e.g. inter-arrival and service distributions) experienced in computer networks. They are attractive as they are amenable to analysis typically utilised in queueing models. Modern traffic analysis has shown that many computing workloads are in fact 'heavy-tailed' and highly variable, and are well represented by general distributions such as Log-normal [Mitzenmacher, 2004; Downey, 2005] and Pareto. There has also been evidence suggesting long-tailed arrival patterns in some application domains. However, the use of such distributions can make an analytical analysis of some queueing metrics such as waiting time, busy period, slowdown, etc. difficult due to the fact that the Markovian properties of certain stochastic processes in queues are no longer in force. The use of Prony's method to fit a series of exponential distributions to the original General distribution can help avoid this problem, resulting in a Hyper-exponential distribution that represents the characteristics of the original distribution, but is more amenable to analysis. Bounded representations of general distributions (such as Bounded Pareto) are often used, but they suffer from similar problems. We show that by default Prony's method is not ideally suited to fitting bounded distributions, and present two ways of improving the fit: by normalising the Hyper-exponential resulting from Prony's method between the bounds of the distribution being approximated, and by re-evaluating Prony's method to fit directly to a Bounded Hyper-exponential. Following this, we re-evaluate our TAPTF model to utilise a Hyper-exponential or a Bounded Hyper-exponential service distribution. This opens up our TAPTF model, allowing it to handle additional service distributions that can be ap-

proximated using Hyper-exponential or Bounded Hyper-exponential distributions, such as Log-normal and Weibull. Fundamentally, this means the flexible TAPTF model can be used with a wider variety of computing workloads. We demonstrate the accuracy of using such Hyper-exponential approximations in queueing models by examining queueing metrics from updated Random and TAPTF queueing systems that utilise Bounded Hyper-exponential service distributions. Finally, we highlight a few of the mathematical properties of Hyper-exponential and Bounded Hyper-exponential distributions that makes them so attractive to use in analysis of queueing systems.

6.1 Introduction

Heavy-tailed workloads have been observed frequently in many computing and communications environments - such as files requested by users, files transmitted on a network, files stored on a server and transmission duration's of files [Crovella et al., 1998b; Crovella and Bestavros, 1997]. Traffic analysis of the 1998 World Cup showed that the workload was consistent with a heavy-tailed distribution, both when examined on a day by day basis [Ciardo et al., 2001] and over the 92 day period of the World Cup [Arlitt and Jin, 2000]. Consistent with these measurements, workload generating tools such as SURGE [Barford and Crovella, 1998] have been formulated to 'stress-test' web servers in a more realistic manner. This is achieved by generating web workload (i.e. requests) whose service requirement follows a heavy-tailed distribution. This phenomenon is covered in detail in Chapters 1 and 2.

'Heavy-tailed' distributions such as Pareto have very high variance, where 1% of tasks can take 50% of the computing resources. However, the Pareto distribution is unsuitable to use in queueing analysis, given that the expected value $E[X]$ is undefined when $\alpha \leq 1$, and the variance $var(x)$ is undefined when $\alpha \leq 2$.

For the purpose of analysis, it is often assumed that the task sizes show some maximum (but large) value. This is a reasonable assumption in many cases, such as a web server, which would have some largest file. A *Bounded Pareto* distribution is therefore used, which has an lower and upper limit on the task size distribution. The Bounded Pareto distribution has been used often in recent analysis of task assignment policies [Broberg et al., 2005; Harchol-Balter, 2002; Harchol-Balter et al., 1999]. The Bounded Pareto has all moments finite, however advanced analysis (such as computing the standard deviation and variance of common metrics such as waiting time and slowdown) is complex due to the difficulties in manipulating the Laplace transforms of these metrics.

It is well known that a complete analysis of the $M/G/c$ queueing system is a difficult problem due to the fact that the usual Markovian property underlying the birth and death queueing models can no longer be assumed. Analysis of the $M/G/1$ queues is even quite complex and various devices, such as the method of embedded Markov Chain, have to be resorted to in order to obtain the solutions of the systems. Analysis of $M/M/1$ and $M/M/c$ systems are computationally trivial by comparison (as described in previous research [Osogami and Harchol-Balter, 2003]). We will endeavour to approximate our 'heavy-tailed' Pareto distribution with a series of exponential distributions (known as a Hyper-exponential distribution) while still maintaining the important characteristics of the original distribution.

Once we have a General model of the current traffic (such as Pareto), we wish to convert it to a more analytically friendly representation. We fit our Pareto service distribution as series of Exponential distributions (known as Hyper-exponential), while still maintaining the important characteristics of the original distribution, such as the long tail. We do this using a technique known as 'Prony's Method' [Marple, 1986], which has been revisited in more recent work by Feldmann and Whitt [Feldmann and Whitt, 1997]. This is described in Section 6.2. In Section 6.3 we introduce the notion of a Bounded Hyper-exponential, and re-evaluate Prony's method in order to fit a Bounded Pareto directly to this distribution. In Section 6.4 we measure the quality of fit achieved, through visual inspection and a comparison of moments. We wish to represent our TAPTF $M/BP/1$ system as a $M/H_n/1$ or $M/BH_n/1$ systems, without adversely affecting the integrity and realism of our original model. This requires us to re-evaluate our model with a Hyper-exponential (or Bounded Hyper-exponential) service distribution, shown in Section 6.5. Section 6.6 depicts a comparison of queueing metrics (for Random and TAPTF properties systems), contrasting the original Bounded Pareto service models with approximated Bounded Hyper-exponential service models (computed in Section 6.5) in each instance. In Section 6.7 we explore some of the properties of Hyper-exponential and Bounded Hyper-exponential distributions (such as the higher moments and Laplace transforms) that makes them so amenable to use in queueing system analysis. In Section 6.8 we summarise our contributions, and explore further improvements that can be made regarding the techniques presented in this chapter.

6.2 Fitting General Distributions to Hyper-exponential

We wish to fit a series of exponential functions, known as a Hyper-exponential distribution, to a Pareto (or Bounded Pareto) distribution.

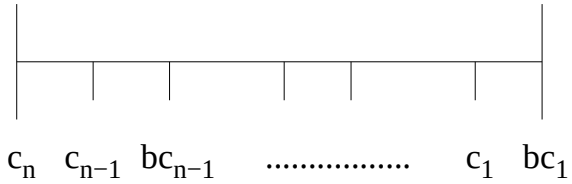


Figure 6.1: Prony's Method Matching Points

The Pareto distribution has a probability density function (p.d.f) of:

$$f(x) \quad = \frac{\alpha k^{\alpha}}{x^{\alpha+1}}, k > 0, x > k \tag{6.1}$$

The cumulative distribution function (c.d.f) is:

$$\begin{aligned} F(x) &= Pr[X \leq x] \\ &= 1 - (\frac{k}{x})^{\alpha} \end{aligned} \tag{6.2}$$

The complementary cumulative distribution function (c.c.d.f) is:

$$\begin{aligned} F^c(x) &= 1 - F(x) \\ &= (\frac{k}{x})^{\alpha} \end{aligned} \tag{6.3}$$

The Bounded Pareto distribution is bounded from below (by k), and above (by p). This is a realistic for many computing workloads as typically a task (e.g. a web request or CPU process) would have lower and (large) upper bound. It has a probability density function (p.d.f) of:

$$f(x) = \frac{\alpha k^{\alpha}}{1 - (k/p)^{\alpha}} x^{-\alpha-1}, k \leq x \leq p \tag{6.4}$$

The cumulative distribution function (c.d.f) is:

$$\begin{aligned} F(x) &= Pr[X \leq x] \\ &= \frac{1 - (k/x)^{\alpha}}{1 - (k/p)^{\alpha}} \end{aligned} \tag{6.5}$$

The complementary cumulative distribution function (c.c.d.f) is:

$$\begin{aligned} F^c(x) &= 1 - F(x) \\ &= \frac{(k/x)^{\alpha} - (k/p)^{\alpha}}{1 - (k/p)^{\alpha}} \end{aligned} \tag{6.6}$$

We wish to fit these distributions to a series of exponential functions, known as a *Hyper-exponential*. An n part Hyper-exponential has a probability density function (p.d.f) of:

$$h_n(x) = \sum_{i=1}^{n} P_i \lambda_i e^{-\lambda_i x}, \text{where } \sum_{i=1}^{n} P_i = 1 \tag{6.7}$$

The cumulative distribution function (c.d.f) is:

$$\begin{aligned} H_n(x) &= Pr[X \leq x] \\ &= \sum_{i=1}^{n} P_i (1 - e^{-\lambda_i x}) \end{aligned} \tag{6.8}$$

The complementary cumulative distribution function (c.c.d.f) for an n part Hyper-exponential is described as follows:

$$H_n{}^c(x) = \sum_{i=1}^{n} P_i e^{-\lambda_i x} \tag{6.9}$$

In order to fit a long tailed distribution to a mixture of exponential distributions, we utilise a technique known as Prony's method [Marple, 1986; Feldmann and Whitt, 1997]. Techniques for fitting long tailed data sets have also been devised previously [Riska et al., 2002a;b], but in this chapter we focus on fitting distributions, not experimental data measurements. The recursive fitting process begins at the tail, computing the pair (λ_1, P_1), then (λ_2, P_2) and continues until (λ_n, P_n). After a segment is fitted, it is subtracted from the original c.c.d.f, then the next segment is fitted to the remaining c.c.d.f. By doing this we are fitting an individual exponential component to each of the points (c_1, bc_1), (c_2, bc_2) up until c_n (as seen in Figure 6.1). The higher indexed components have a tail that decays more rapidly (and a higher λ parameter), and as such will not adversely affect the earlier fitted components. Finally, we compute the last pair of parameters, (λ_n, P_n).

It is assumed that the ratios c_i/c_{i+1} are sufficiently large, and the parameter b must be chosen such that $1 < b < c_i/c_{i+1}$, for all i. If the parameters λ_i are sufficiently separated, a good fit should be obtained. Once the procedure completes, we should have a n part Hyper-exponential c.c.d.f H^c that is larger than the original c.c.d.f, F^c (from the distribution we are fitting) at the matching points. That is:

$$H^c(xc_i) > F^c(xc_i), 1 \leq i < n \tag{6.10}$$

for $x = 1$ and b.

Given that F^c will mostly likely be a long-tailed distribution, Feldmann and Whitt [Feldmann and Whitt, 1997] note that there will be a t_0 such that:

$$F^c(t) \geq H^c(t) \tag{6.11}$$

for all $t \geq t_0$. As such it is important to choose a c_1 large enough that t_0 is beyond the "area of interest". In our case, we are particularly interested in the fitting of a Bounded Pareto distribution. Given that we are only interested in a finite region this poses no problems at all - we focus our fitting efforts solely within the lower and upper bounds of the original Bounded Pareto. Specifically, we set $bc_1 \approx p$ and $c_n \approx k$, where p and k are the upper and lower bounds respectively.

Feldmann and Whitt do not highlight the fact that this procedure can result in a poor fit at the base of the distribution. Specifically, the last pair of parameters (λ_n, p_n) are not computed to match c_n or bc_n, so the fit can deteriorate markedly at the base.

6.3 Fitting General Distributions to Bounded Hyper-exponential

The use of bounded General distributions has become common place in the analysis of modern computer systems. They are often more amenable to analysis than their unbounded counterparts, as they typically have all moments finite. They are also more realistic representations of the computing phenomena they are modelling, where tasks have finite service requirements, and are best modelled by finite distributions. Even so, it is often still desirable to represent these distributions as Markovian, to utilise the wide range of existing $M/M/c$ knowledge.

Motivated by this, we introduce the notion of a *Bounded Hyper-exponential* distribution. That is, we incorporate a lower and upper bound (as per the Bounded Pareto). The Bounded Hyper-exponential has a probability density function (p.d.f) of:

$$bh_n(x) = \sum_{i=1}^{n} \frac{P_i \lambda_i e^{-\lambda_i x}}{e^{-\lambda_i k} - e^{-\lambda_i p}} \tag{6.12}$$

The cumulative distribution function (c.d.f) is:

$$\begin{aligned} BH_n(x) &= Pr[X \leq x] \\ &= \sum_{i=1}^{n} P_i \frac{e^{-\lambda_i k} - e^{-\lambda_i x}}{e^{-\lambda_i k} - e^{-\lambda_i p}} \end{aligned} \tag{6.13}$$

The complementary cumulative distribution function (c.c.d.f) is:

$$\begin{aligned} {BH_n}^c(x) &= 1 - BH_n(x) \\ &= \sum_{i=1}^{n} P_i \frac{e^{-\lambda_i x} - e^{-\lambda_i p}}{e^{-\lambda_i k} - e^{-\lambda_i p}} \end{aligned} \tag{6.14}$$

We wish to fit a general distribution as a Bounded Hyper-exponential. We can achieve this in two ways. Obviously, this would be of most use if the original distribution is itself bounded - it makes sense to fit one bounded distribution to another. First, we could utilise Prony's method (as described in Section 6.2), then normalise the resulting Hyper-exponential over specific lower and upper bounds, creating a bounded (by normalisation) Hyper-exponential. Second, and more accurately, we could modify Prony's method in order to fit the General distribution directly to a Bounded Hyper-exponential distribution. We describe this updated process in the following section.

6.3.1 Updating Prony's Method

We choose λ_1 and P_1 to match the c.c.d.f $F^c(t)$ at the arguments c_1 and bc_1, solving the two equations:

$$P_1 \frac{e^{-\lambda_1 x c_1} - e^{-\lambda_1 p}}{e^{-\lambda_1 k} - e^{-\lambda_1 p}} = F^c(xc_1), \text{for x = 1 and b} \tag{6.15}$$

for P_1 and λ_1, assuming c_1, b, $F^c(c_1)$ and $F^c(bc_1)$ are known. We can then obtain:

$$\frac{e^{-\lambda_1 c_1} - e^{-\lambda_1 p}}{e^{-\lambda_1 b c_1} - e^{-\lambda_1 p}} = \frac{F^c(c_1)}{F^c(bc_1)} \tag{6.16}$$

Unfortunately we cannot isolate λ_1 as was possible previously when fitting to an unbounded Hyper-exponential. However it is trivially solved using a numerical solver. Once we solve for λ_1, we can then compute:

$$P_1 = F^c(c_1) \frac{e^{-\lambda_1 k} - e^{-\lambda_1 p}}{e^{-\lambda_1 c_1} - e^{-\lambda_1 p}} \tag{6.17}$$

As with the original Prony's method, we assume that λ_i will be sufficiently larger than λ_1 for all $i \geq 2$ so that the final approximation satisfies:

$$\sum_{i=1}^{n} P_i \frac{e^{-\lambda_i t} - e^{-\lambda_i p}}{e^{-\lambda_i k} - e^{-\lambda_i p}} \simeq P_1 \frac{e^{-\lambda_1 t} - e^{-\lambda_1 p}}{e^{-\lambda_1 k} - e^{-\lambda_1 p}}, \text{for t} \geq c_1 \tag{6.18}$$

Now, for $2 \leq i \leq n$, let:

$$F^c{}_i(xc_i) = F^c{}_{i-1}(xc_i) - \sum_{j=1}^{i-1} P_j \frac{e^{-\lambda_j x c_i} - e^{-\lambda_j p}}{e^{-\lambda_j k} - e^{-\lambda_j p}} \quad \text{for } x = 1 \text{ and } b \tag{6.19}$$

where $F^c{}_1(t) = F^c(t)$. We then solve the two equations:

$$P_i \frac{e^{-\lambda_i x c_i} - e^{-\lambda_i p}}{e^{-\lambda_i k} - e^{-\lambda_i p}} = F_i^c(xc_i), \text{for x} = 1 \text{ and b} \tag{6.20}$$

to obtain:

$$\frac{e^{-\lambda_i c_i} - e^{-\lambda_i p}}{e^{-\lambda_i b c_i} - e^{-\lambda_i p}} = \frac{F^c{}_i(c_i)}{F^c{}_i(bc_i)} \tag{6.21}$$

We numerically solve for λ_i, and then compute:

$$P_i = F^c{}_i(c_i) \frac{e^{-\lambda_i k} - e^{-\lambda_i p}}{e^{-\lambda_i c_i} - e^{-\lambda_i p}} \tag{6.22}$$

for $2 \leq i \leq k-1$. For the last parameter pair (λ_n, P_n) we note that P_n is determined by the condition:

$$P_n = 1 - \sum_{j=1}^{n-1} P_j \tag{6.23}$$

Given that:

$$P_n \frac{e^{-\lambda_n c_n} - e^{-\lambda_n p}}{e^{-\lambda_n k} - e^{-\lambda_n p}} = F^c{}_n(c_n) \tag{6.24}$$

where $F^c{}_n(c_n)$ is defined, we obtain λ_n from the equation:

$$\frac{e^{-\lambda_n k} - e^{-\lambda_n p}}{e^{-\lambda_n c_n} - e^{-\lambda_n p}} = \frac{P_n}{{F^c}_n(c_n)} \tag{6.25}$$

We can then numerically solve for λ_n. As with the original Prony's method, providing we obtain probability weights for each exponential segment ($P_i > 0$) and the parameters λ_i are well separated, we should obtain a good fit to the original distribution.

Once the procedure completes, we should have a n part Bounded Hyper-exponential c.c.d.f BH^c that is larger than the original c.c.d.f, F^c (from the distribution we are fitting) at the matching points. That is:

$$BH^c(xc_i) > F^c(xc_i), 1 \leq i < n \tag{6.26}$$

for $x = 1$ and b.

If F^c is a long-tailed distribution, then there will be a t_0 such that:

$$F^c(t) \geq BH^c(t) \tag{6.27}$$

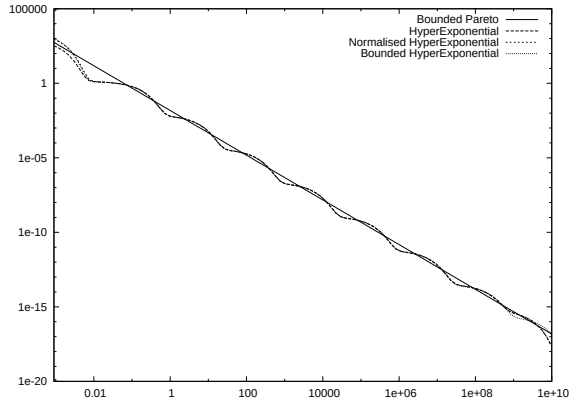
for all $t \geq t_0$.

Distribution	$E[X]$	$E[X^2]$
BPAR	3000.00	1e+13
HYP	2684.82	6.56336e+12
NHYP	2691.42	6.58528e+12
BHYP	3269.62	1.19397e+13

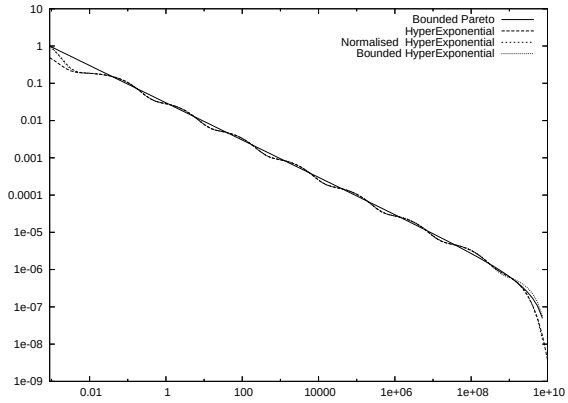
Table 6.1: Matching moments, $\alpha = 0.5$

6.4 Quality of fit

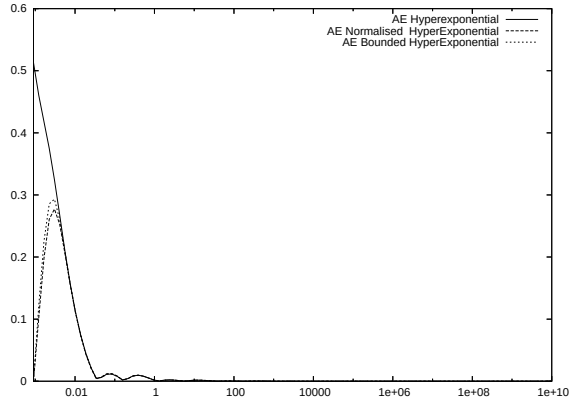
In this section we present four fitting examples, utilising the original Prony's method as well as our updated approach, suited to bounded input distributions. These are provided to give a glimpse of the trends we have observed from fitting many permutations and combinations of bounded input distributions to n-part Hyper-exponential distributions. We fitted to 10



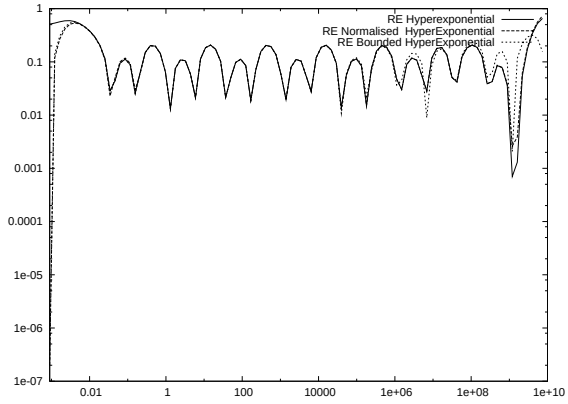
(a) p.d.f.



(b) c.c.d.f



(c) Absolute Error - c.c.d.f.



(d) Relative Error - c.c.d.f.

Figure 6.2: Fitting a Bounded Pareto Distribution ($E[X] = 3000$, $\alpha = 0.5$, $k = 0.0009$, $p = 10^{10}$) to a Hyper-exponential, Normalised Hyper-exponential and Bounded Hyper-exponential

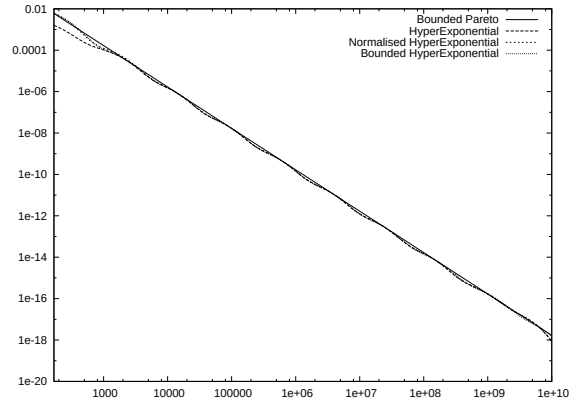
part Hyper-exponential in each instance, and used the same matching points (chosen so the ratio between each point is equal, as described by Feldmann and Whitt [Feldmann and Whitt, 1997]). Points c_1 and c_n were chosen to be as close to the upper and lower bounds (respectively) as possible. It is worth noting that in most instances significantly better fits can be obtained by alternate matching point allocations, but are not shown here to keep the comparison consistent.

Figure 6.2 illustrates a Bounded Pareto distribution ($E[X] = 3000$, $\alpha = 0.5$, $k = 0.0009$, $p = 10^{10}$) being approximated by a Hyper-exponential distribution (using Prony's method), a Normalised Hyper-exponential distribution (normalising the results from Prony's method), and a Bounded Hyper-exponential (using our updated Prony's method).

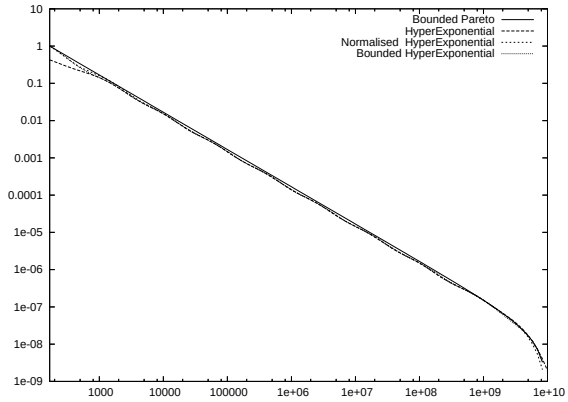
From examination of the p.d.f and c.c.d.f, it appears that the Hyper-exponential obtained via Prony's method is a poor fit at the base and the tail of the distribution, diverging significantly. Even the Normalised and Bounded fits diverge toward the base, before converging again. This is a known artifact stemming from the choice of matching parameters - given that c_n isn't a true matching point in Prony's method (as mentioned in Section 6.2). As the c_i parameters are evenly spaced (from c_1 to c_n) this leaves a 'hole' between the base of the distribution and c_{n-1}, the first true matching point. The Bounded Hyper-exponential distribution follows the curve of the tail effectively, whilst the Normalised and standard Hyper-exponential decays too rapidly. This is confirmed by the high relative error toward the tail of the distribution shown by these fits.

We notice a significant difference in the comparison of moments (depicted in Table 6.1). The Hyper-exponential obtained via Prony's method is within 89% and 66% of the first and second moments of the original distribution, respectively. The Normalised fit does not match much better, within 90% of the first moment, and 66% of the second moment. The Bounded Hyper-exponential fit (using our updated Prony's method) is significantly more accurate, within 92% of the first moment and 84% of the second moment. Accurate matching of the second moment is crucial if if these approximations are to be used in queueing analysis.

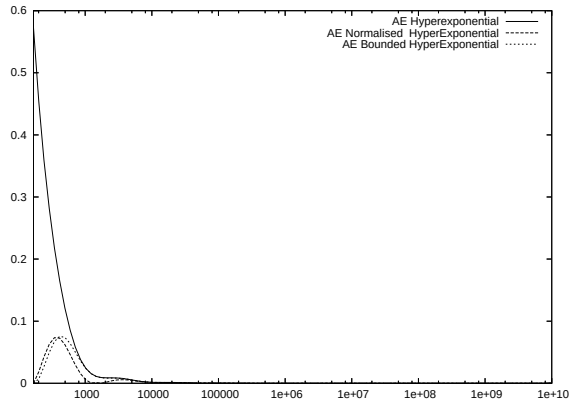
Figure 6.3 illustrates another Bounded Pareto distribution ($E[X] = 3000$, $\alpha = 1.0$, $k = 167.555$, $p = 10^{10}$), which we again approximate with various Hyper-exponential distributions. From the p.d.f and c.c.d.f we can observe that the standard fit obtained via Prony's method is poor at both the base, and to a lesser extent, the tail of the distribution. The Normalised and Bounded approximations provide a much tighter fit, especially at the critical areas of the base and tail of the distribution. This is confirmed by examining the absolute and relative error of these fits. We can see in the case of the Bounded Hyper-exponential fit,



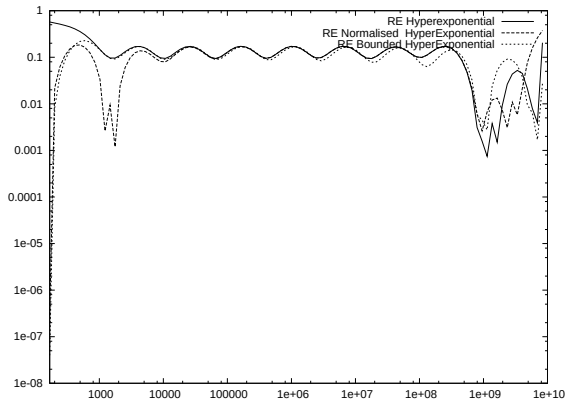
(a) p.d.f.



(b) c.c.d.f



(c) Absolute Error - c.c.d.f.



(d) Relative Error - c.c.d.f.

Figure 6.3: Fitting a Bounded Pareto Distribution ($E[X] = 3000$, $\alpha = 1.0$, $k = 167.555$, $p = 10^{10}$) to a Hyper-exponential, Normalised Hyper-exponential and Bounded Hyper-exponential

the absolute error is around 0.1 for the majority of the fit, and significantly less at the base and tail.

The comparison of moments (Table 6.2) shows that the first moments are similar in all fits. The Hyper-exponential is within 82% of the first moment, and 91% of the second moment. The Normalised fit is within 90% of the first moment, and 92% of the second moment, matching the moments more accurately. The Bounded fit is within 89% of the first moment, and 95% of the second moment.

Figure 6.3 illustrates a Bounded Pareto distribution ($E[X] = 3000$, $\alpha = 1.5$, $k = 1000.32$, $p = 10^{10}$) and it's corresponding Hyper-exponential approximations. Upon examination of the p.d.f. and c.c.d.f we can see that the Normalised and Bounded fits are quite close, while

Distribution	$E[X]$	$E[X^2]$
BPAR	3000	1.67555e+12
HYP	2482.59	1.52573e+12
NHYP	2691.64	1.5428e+12
BHYP	2675.74	1.60094e+12

Table 6.2: Matching moments, $\alpha = 1.0$

the unbounded Hyper-exponential fits poorly around the base of the distribution. This is especially evident when examining the absolute and relative errors at the base for the hyper-exponential fit, showing significant divergence from the original Bounded Pareto. We also observe that the Normalised and Bounded fits maintain a relative error of around 0.1 for the majority of the fit.

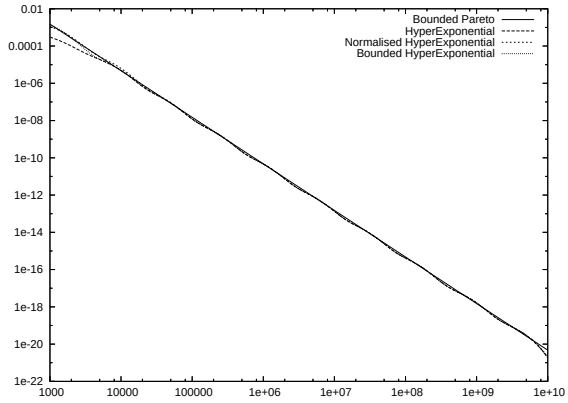
Table 6.3 depicts a comparison of the first and second moments of these distributions. The Hyper-exponential fit matches the first moment poorly, within 61% of the original Bounded Pareto. The second moment is matched more accurately, within 91%. The normalised fit is significantly better, fitting with 97% of the first moment and 90% of the second moment. The Bounded fit is also good, fitting with 94% of the first moment and 92% of the second moment.

Distribution	$E[X]$	$E[X^2]$
BPAR	3000.01	9.48839e+09
HYP	1850.16	8.55168e+09
NHYP	3093.6	8.57685e+09
BHYP	2811.19	8.69128e+09

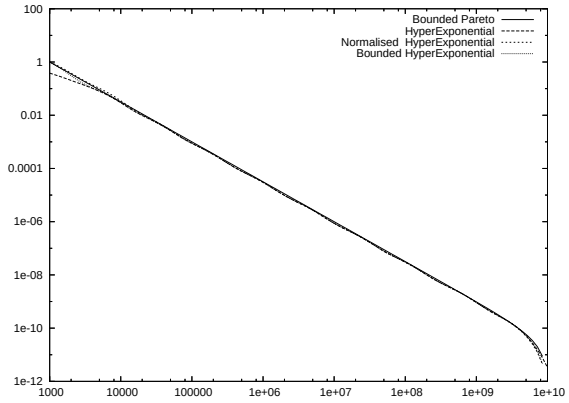
Table 6.3: Matching moments, $\alpha = 1.5$

Figure 6.5 illustrates a Bounded Pareto distribution ($E[X] = 3000$, $\alpha = 2.0$, $k = 1500$, $p = 10^{10}$) being approximated by a Hyper-exponential distribution (using Prony's method), a Normalised Hyper-exponential distribution (normalising the results from Prony's method), and a Bounded Hyper-exponential (using our updated Prony's method).

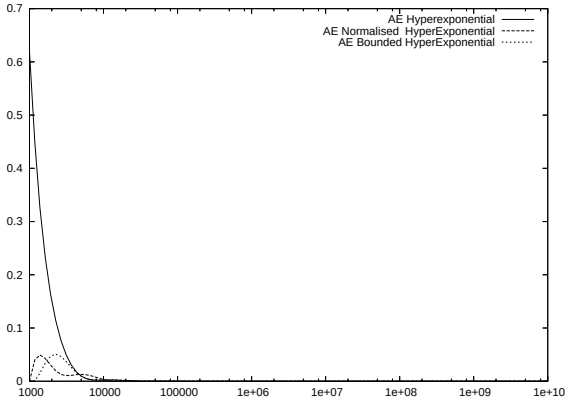
On initial visual inspection of both the p.d.f and the c.c.d.f, all 3 of our fitting attempts seem very accurate. We can notice some divergence of the Hyper-exponential fit at the base



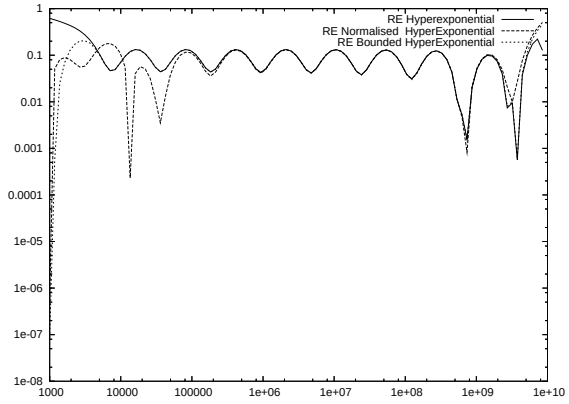
(a) p.d.f.



(b) c.c.d.f

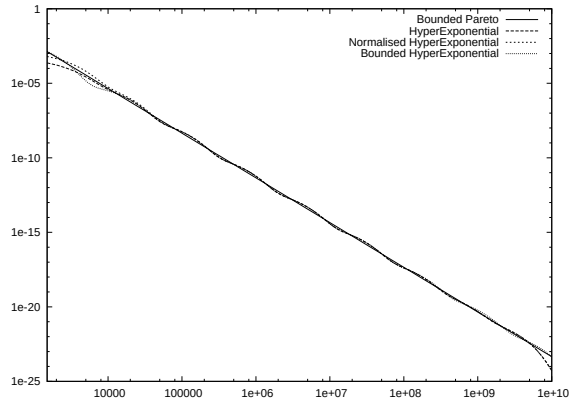


(c) Absolute Error - c.c.d.f.

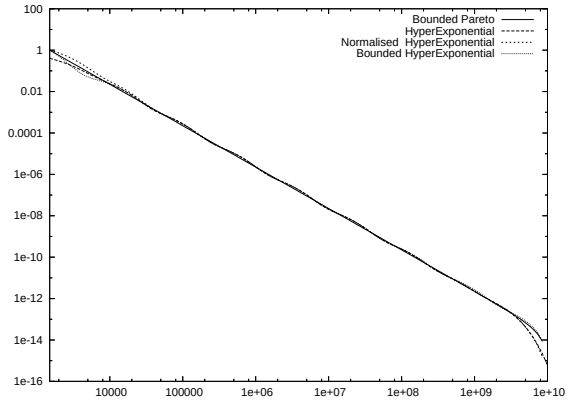


(d) Relative Error - c.c.d.f.

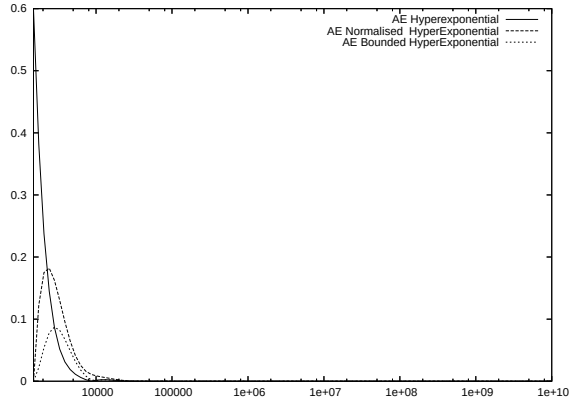
Figure 6.4: Fitting a Bounded Pareto Distribution ($E[X] = 3000$, $\alpha = 1.5$, $k = 1000.32$, $p = 10^{10}$) to a Hyper-exponential, Normalised Hyper-exponential and Bounded Hyper-exponential



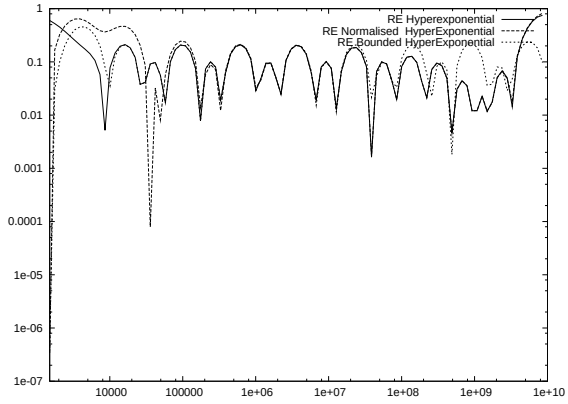
(a) p.d.f.



(b) c.c.d.f



(c) Absolute Error - c.c.d.f.



(d) Relative Error - c.c.d.f.

Figure 6.5: Fitting a Bounded Pareto Distribution ($E[X] = 3000$, $\alpha = 2.0$, $k = 1500$, $p = 10^{10}$) to a Hyper-exponential, Normalised Hyper-exponential and Bounded Hyper-exponential

Distribution	$E[X]$	$E[X^2]$
BPAR	3000.00	7.07068e+07
HYP	1525.93	6.57345e+07
NHYP	3416.07	7.40877e+07
BHYP	2825.25	7.01227e+07

Table 6.4: Matching moments, $\alpha = 2.0$

and the tail of the distribution, which is to be expected given that the original distribution is bounded, despite setting matching points near the lower and upper bounds. This is confirmed from examination of the relative and absolute errors, showing that the Hyper exponential is a very poor match for the original distribution at these points.

Normalising the results obtained in Prony's method obtains a better fit given that our original distribution is bounded. This is especially evident in the base of the distribution (near the lower bound) where the fit is improved greatly, which is highlighted by observing the absolute error. However it did not improve the fit along the shape of the tail, decaying more rapidly than the original distribution, illustrated by the deteriorating relative error in the c.c.d.f.

When comparing the moments of the original and fitted distributions (shown in Table 6.4), we find that the Hyper-exponential obtained by Prony's method is only 56% accurate when matching the first moment, and 93% accurate when matching the second moment. The Normalised result from Prony's method improves the match, approximating within 88% and 95% for the first and second moments respectively. Using our updated Prony's method results in a more accurate matching of moments, with an accuracy of 94% and 99% for the first and second moments respectively.

6.5 Re-evaluating the TAPTF $M/BP/1$ model as $M/H_n/1$ and $M/BH_n/1$

We need to update our original TAPTF model [Broberg et al., 2005] (presented in Chapter 4) to allow the use of a Hyper-exponential or Bounded Hyper-exponential distribution. This also improves the potentially applications of the TAPTF algorithm, allowing it to be utilised with any General service distribution that can be effectively approximated using the techniques described in Sections 6.2 and 6.3.Fortunately, only certain parameters need to be re-evaluated. Most of the definitions remain unchanged.

Recall that the probability density function (p.d.f) of a Hyper-exponential is:

$$h_n(x) = \sum_{i=1}^{n} P_i \lambda_i e^{-\lambda_i x}$$

As such, p_m, the fraction of tasks whose final destination (where it runs to completion on) is either Host m or its predecessor, is defined as follows:

$$\begin{aligned} p_m &= P(X \leq s_m) \\ &= \int_k^{s_m} \sum_{i=1}^{n} P_i \lambda_i e^{-\lambda_i x} \\ &= \sum_{i=1}^{n} P_i (e^{-\lambda_i k} - e^{-\lambda_i s_m}) \end{aligned} \tag{6.28}$$

We also need to evaluate the expected moments of tasks that run-to-completion at Host m. $E(X_{mO}^j)$ and $E(X_{mR}^j)$ are the jth moment of the distribution of tasks whose final destination is Host m's ordinary (O) queue and restart (R) queues respectively. We have:

$$\begin{aligned} E(X_{mO}^j) &= \frac{1}{p_m} \int_k^{s_m} x^j h_n(x) dx \\ &= \frac{1}{p_m} \sum_{i=1}^{n} \frac{P_i}{{\lambda_i}^j} (\sum_{l=0}^{j} {}^jC_l e^{-\lambda_i k} l! (\lambda_i k)^{j-l} - \sum_{l=0}^{j} {}^jC_l e^{-\lambda_i s_m} l! (\lambda_i s_m)^{j-l}) \\ &= \frac{1}{p_m} \sum_{i=1}^{n} \sum_{l=0}^{j} \frac{P_i}{{\lambda_i}^j} {}^jC_l l! \lambda^{j-l} (e^{-\lambda_i k} k^{j-l} - e^{-\lambda_i s_m} s_m^{j-l}) \\ &= \frac{1}{p_m} \sum_{i=1}^{n} \sum_{l=0}^{j} \frac{{}^jC_l P_i l!}{{\lambda_i}^l} (e^{-\lambda_i k} k^{j-l} - e^{-\lambda_i s_m} s_m^{j-l}) \end{aligned} \tag{6.29}$$

and

$$\begin{aligned} E(X_{mR}^j) &= \frac{1}{p_m - p_{m-1}} \int_k^{s_m} x^j h_n(x) dx \\ &= \frac{1}{p_m - p_{m-1}} \sum_{i=1}^{n} \sum_{l=0}^{j} \frac{{}^jC_l P_i l!}{{\lambda_i}^l} (e^{-\lambda_i s_{m-1}} s_{m-1}^{j-l} - e^{-\lambda_i s_m} s_m^{j-l}) \end{aligned} \tag{6.30}$$

All other parameters remain unchanged.

If we choose to use a Bounded Hyperexponential distribution, we must re-compute the same parameters. The probability density function (p.d.f) of a Bounded Hyperexponential is:

$$bh_n(x) = \sum_{i=1}^{n} \frac{P_i \lambda_i e^{-\lambda_i x}}{e^{-\lambda_i k} - e^{-\lambda_i p}}$$

As such, p_m is defined as follows:

$$\begin{aligned} p_m &= P(X \leq s_m) \\ &= \int_k^{s_m} \sum_{i=1}^{n} \frac{P_i \lambda_i e^{-\lambda_i x}}{e^{-\lambda_i k} - e^{-\lambda_i p}} \\ &= \sum_{i=1}^{n} \frac{P_i (e^{-\lambda_i k} - e^{-\lambda_i s_m})}{e^{-\lambda_i k} - e^{-\lambda_i p}} \end{aligned} \tag{6.31}$$

Again, we need to evaluate the expected moments of tasks that run-to-completion at Host i. We have:

$$\begin{aligned} E(X_{iO}^j) &= \frac{1}{p_m} \int_k^{s_m} x^j bh_n(x) dx \\ &= \frac{1}{p_m} \sum_{i=1}^{n} \frac{P_i}{{\lambda_i}^j (e^{-\lambda_i k} - e^{-\lambda_i p})} (\sum_{l=0}^{j} {}^jC_l e^{-\lambda_i k} l! (\lambda_i k)^{j-l} - \sum_{l=0}^{j} {}^jC_l e^{-\lambda_i s_m} l! (\lambda_i s_m)^{j-l}) \\ &= \frac{1}{p_m} \sum_{i=1}^{n} \sum_{l=0}^{j} \frac{P_i}{{\lambda_i}^j} {}^jC_l l! \lambda^{j-l} \frac{(e^{-\lambda_i k} k^{j-l} - e^{-\lambda_i s_m} s_m^{j-l})}{(e^{-\lambda_i k} - e^{-\lambda_i p})} \\ &= \frac{1}{p_m} \sum_{i=1}^{n} \sum_{l=0}^{j} \frac{{}^jC_l P_i l!}{{\lambda_i}^l} \frac{(e^{-\lambda_i k} k^{j-l} - e^{-\lambda_i s_m} s_m^{j-l})}{(e^{-\lambda_i k} - e^{-\lambda_i p})} \end{aligned} \tag{6.32}$$

and

$$\begin{aligned} E(X_{iR}^j) &= \frac{1}{p_m - p_{m-1}} \int_{s_{m-1}}^{s_m} x^j bh_n(x) dx \\ &= \frac{1}{p_m - p_{m-1}} \sum_{i=1}^{n} \sum_{l=0}^{j} \frac{{}^jC_l P_i l!}{{\lambda_i}^l} \frac{(e^{-\lambda_i s_{m-1}} {s_{m-1}}^{j-l} - e^{-\lambda_i s_m} s_m^{j-l})}{(e^{-\lambda_i k} - e^{-\lambda_i p})} \end{aligned} \tag{6.33}$$

All other parameters remain unchanged.

6.6 Comparing $M/BP/1$ and $M/BH_n/1$ queueing models

In Section 6.4 we found the Hyper-exponential approximations to be a good representation of the original general distributions we were trying to fit. In Section 6.6.1 we first compare a trivial 2 host random task assignment queueing system, using a Bounded Pareto and Bounded Hyper-exponential service distributions. Then in Section 6.6.2, using the re-evaluated model computed in Section 6.5 we can now compare the queueing results directly for a TAPTF queueing system, to check the accuracy when utilising approximated service distributions.

6.6.1 Random Queueing Model

In this section we compare some common queueing metrics for a two host distributed system under a system load of 0.5, with a Random task assignment policy. We observe the expected waiting time and slowdown for the system with a Bounded Pareto service distribution, as well as Normalised Hyper-exponential and Bounded hyper-exponential approximations of that service distribution. These specific service distribution examples are drawn directly from Section 6.4.

α	k	p	$E[W]$ BP	$E[S]$ BP	$E[W]$ BH	$E[S]$ BH	$E[W]$ NH	$E[S]$ NH
1.0	167.555	10^{10}	279258000	833333	266823000	850586	257133000	821085
1.5	1000.32	10^{10}	1581400	948.535	1448550	897.47	1429480	823.317
2.0	1500	10^{10}	11784.5	5.23754	11258.1	6.97184	11820.6	6.14309

Table 6.5: Comparing queueing metrics for Random

Table 6.5 shows the comparison of random queueing metrics for three scenarios, where the α parameter of the original Bounded Pareto service distribution is 1.0, 1.5 and 2.0. When α is 1.0, the expected waiting time is within 95% and 92% for the Bounded and Normalised Hyper-exponential fits respectively. The expected slowdown is approximated within 98% and 99% for the Bounded and Normalised fits respectively.

When α is 1.5, the expected waiting time is within 92% and 90% for the Bounded and Normalised Hyper-exponential fits respectively. The expected slowdown is approximated within 95% and 87% for the Bounded and Normalised fits respectively.

For the scenario where α is 2.0, the expected waiting time is within 95% and 90% for the Bounded and Normalised Hyper-exponential fits respectively. The expected slowdown is

approximated extremely accurately, within 95% and 99% for the Bounded and Normalised fits respectively.

6.6.2 TAPTF Queueing Model

We now compare the same metrics for a two host distributed system under a system load of 0.5, with the TAPTF task assignment policy. Again each workload scenario is drawn from Section 6.4. In each instance the optimal parameters for the TAPTF policy (i.e. the s_i and q_i values) are utilised. We utilise the updated TAPTF model that was computed in Section 6.5, that utilises a Bounded Hyper-exponential service distribution in place of the original Bounded Pareto service distribution.

α	k	p	$E[W]$ BP	$E[S]$ BP	$E[W]$ BH	$E[S]$ BH	$E[W]$ NH	$E[S]$ NH
1.0	167.555	10^{10}	216573	32.81	162133	27.30	158550	27.31
1.5	1000.32	10^{10}	32420.5	7.95	24715.3	6.19	37573.6	9.51
2.0	1500	10^{10}	3875.25	1.13	3203.5	1.21	5541.86	1.89

Table 6.6: Comparing queueing metrics for TAPTF

Table 6.6 shows the comparison of TAPTF queueing metrics for three scenarios, where the α parameter of the original Bounded Pareto service distribution is 1.0, 1.5 and 2.0. When α is 1.0, we observe a reasonable fit of within 75% and 73% for the Normalised and Bounded Hyper-exponential models. The slowdown is approximated more accurately, both within 83% of the original Bounded Pareto TAPTF model. When α is 1.5, we can observe a fit for the expected waiting time that is within 76% and 86% for the Bounded and Normalised fits respectively. The expected slow down is approximated 77% and 84% for the Bounded and Normalised fits respectively. In the case where α is 2.0, the expected waiting time is approximated with 83% and 70% for the Bounded and Normalised fits respectively. The expected slowdown is approximated within 93% and 60% respectively.

6.6.3 Discussion

In Section 6.6.1 we compared the queueing metrics for a 2 Host system with Random task assignment. We compared the expected waiting time and slow down for systems where the service distribution was Bounded Pareto, as well as Bounded and Normalised Hyper-exponential approximations of the Bounded Pareto distribution. In the scenarios examined,

we found the Hyper-exponential approximations to be a good representation of the original Bounded Pareto queueing model, with nearly all results being over 90% accurate.

Section 6.6.2 depicted a comparison of the queueing metrics for a 2 Host system utilising the TAPTF task assignment policy. We found the Bounded and Normalised Hyper-exponential TAPTF to be a reasonable approximation of the original TAPTF Bounded Pareto model. We found the TAPTF model to be significantly more sensitive to the quality of fit of the service distribution. This is not surprising given the model has more parameters that depend on the service distribution both directly and indirectly (such as p_i). We also note the accuracy of the Hyper-exponential approximation can be improved significantly by utilising a different technique to choose the matching points, particularly focusing on the fit around the base of the distribution. This is a known weakness with the Prony approach for this particular fitting application, as discussed previously in this chapter, which we plan on addressing in future work.

6.7 Applications of Hyper-exponential distributions in General queueing models

In the previous sections we observed that hyper-exponential approximations provided a good representation of the highly variable General workloads they attempted to characterise. In this section we use these hyper-exponential approximations to highlight some of the benefits for $M/G/1$ (where $G \equiv H_n$ or BH_n) queueing analysis. We are mostly interested in the application of H_n or BH_n service distributions, but many of the benefits can also apply for $G/G/1$ queueing models (where the arrival pattern is $G \equiv H_n$ or BH_n).

In particular, we would like to find some bounds (hard or otherwise) on the range of performance experienced by tasks in a queueing system. Ideally we would like to find specific Quality of Service bounds so that we can make promises to customers regarding the type of performance they can expect in the system (e.g. 95% of all customers will have an expected waiting time of less than time t). To achieve this, we need to find the probability distribution of waiting time in our queueing system. We also wish to compute the higher moments (and consequently the variance) of common metrics such as expected waiting time and slowdown, to observe how accurately they reflect the behaviour of the majority of tasks in the queueing system.

6.7.1 General Hyper-exponential analysis

We defined the p.d.f of our Hyper-exponential in Section 6.2 of this chapter. We now wish to obtain the Laplace transform $L_{h_n}(s)$ of the service distribution $h_n(t)$.

We proceed as follows:

$$L_{h_n}(s) = \int_0^\infty e^{-st} h_n(t).dt \tag{6.34}$$

$$= \int_0^\infty e^{-st} \sum_{i=1}^{n} P_i \lambda_i e^{-\lambda_i t}.dt \tag{6.35}$$

$$= \sum_{i=1}^{n} \frac{P_i \lambda_i}{\lambda_i + s} \tag{6.36}$$

We can trivially obtain the moments of the service distribution as follows:

$$E[X^n] = (-1)^n \frac{d^n L_{h_n}(s)}{ds}\bigg|_{s=0} \tag{6.37}$$

Using this procedure we find the first moment (mean), $E[X]$:

$$E[X] = (-1)\ \frac{d}{ds} \sum_{i=1}^{n} \frac{P_i \lambda_i}{\lambda_i + s}\bigg|_{s=0} \tag{6.38}$$

$$= \sum_{i=1}^{n} \frac{P_i}{\lambda_i} \tag{6.39}$$

The second moment, $E[X^2]$ is similarly easy to find:

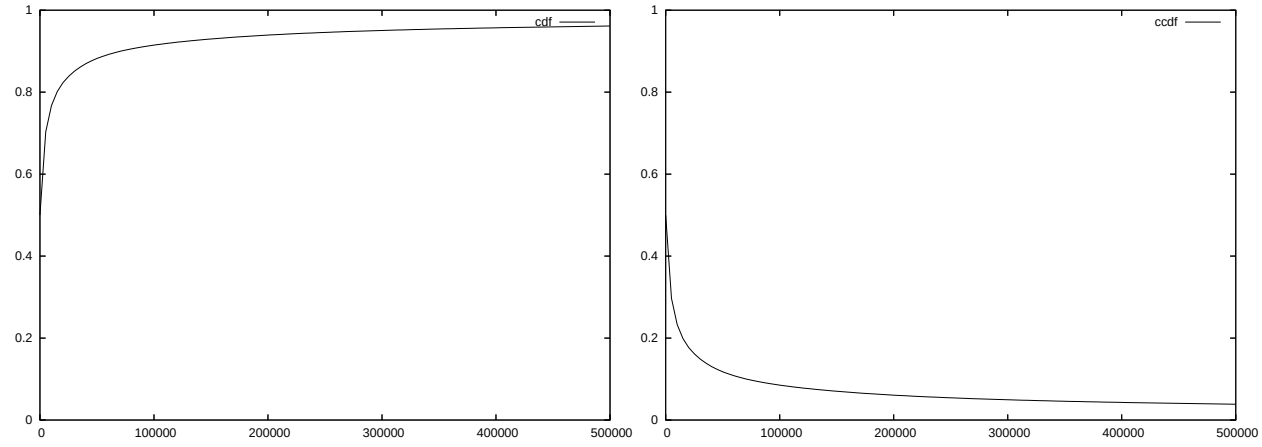
$$E[X^2] = (-1)^2 \frac{d^2}{ds} \sum_{i=1}^{n} \frac{P_i \lambda_i}{\lambda_i + s}\bigg|_{s=0} \tag{6.40}$$

$$= \sum_{i=1}^{n} \frac{2P_i}{{\lambda_i}^2} \tag{6.41}$$

We can also find $L_W(s)$, the Laplace transform of the waiting time. This known result [Kleinrock, 1975a], which is true for any $M/G/1$ queue[1], is defined as follows:

$$L_W(s) = \frac{s(1-\rho)}{s - \lambda + \lambda L_{h_n}(s)} \tag{6.42}$$

$$= \frac{s(1-\rho)}{s - \lambda + \lambda \sum_{i=1}^{n} \frac{P_i \lambda_i}{\lambda_i + s}} \tag{6.43}$$



(a) - $W(t)$ (b) - $W^c(t)$

Figure 6.6: The c.d.f of waiting time is shown in (a), for a hyper-exponential approximation of a bounded Pareto distribution with $\alpha = 1.0$, $k = 1000.32$ and $p = 10^{10}$. The corresponding c.c.d.f is shown in (b).

where $\rho = \lambda E[X]$, and X is the service time.

We can numerically invert this result in order to obtain $w(t)$, the p.d.f of the waiting time distribution. This is significant in that we can now perform some advanced analysis of the behaviour of system that was not possible before. From the p.d.f we can now obtain the cumulative distribution function (c.d.f) as follows:

$$\begin{aligned} W(t) &= Pr[T \leq t] \\ &= \int_0^t w(t).dt \end{aligned} \tag{6.44}$$

The complementary cumulative distribution function (c.c.d.f) is:

$$\begin{aligned} W^c(t) &= 1 - W(t) \end{aligned} \tag{6.45}$$

Using the c.d.f and the c.c.d.f we can now make concrete guarantees regarding the waiting time experienced by a certain percentage of tasks. Consider the example shown in Figure 6.6. The waiting time distribution for a $M/H_n/1$ queueing system is shown, where the system

[1]Provided the queueing discipline is First-Come-First-Serve and the system is work-conserving.

load ρ is 0.5. We can see from the c.d.f shown in Figure 6.6(a) that there is approximately a 90% probability of some waiting time T being less than 100000.

We can now also obtain the nth moments of the waiting time distribution in two ways. One is as follows:

$$E[W^n] = (-1)^n \frac{d^n L_W(s)}{ds}\bigg|_{s=0} \tag{6.46}$$

Alternatively we can compute the nth waiting time moments as follows:

$$E[W^n] = \int_0^\infty t^n w(t).dt \tag{6.47}$$

The first moment (IE. the mean) of the waiting time distribution can now be computed. We will proceed as follows, using $L_X(s)$ as the Laplace transform of the service distribution to maintain generality:

$$E[W] = (-1)\frac{d}{ds}\frac{s(1-\rho)}{s-\lambda+\lambda L_X(s)}\bigg|_{s=0} \tag{6.48}$$

$$= \frac{\lambda(\rho-1)(L_X(s)-1-sL'_X(s))}{(s-\lambda+\lambda L_X(s))^2}\bigg|_{s=0} \tag{6.49}$$

When we set $s = 0$ we obtain an indeterminate result (0/0), so we must first apply *l'Hopital's rule* twice to the numerator and denominator. This is a relatively straightforward process (albeit tedious and unwieldy if done by hand), giving:

$$E[W] = \frac{\partial_{s,2}(\lambda(\rho-1)(L_X(s)-1-sL'_X(s)))}{\partial_{s,2}((s-\lambda+\lambda L_X(s))^2)}\bigg|_{s=0} \tag{6.50}$$

$$= \frac{\lambda(\rho-1)(-L''_X(s)-sL'''_X(s))}{2(1+\lambda L'_X(s))^2+2(s-\lambda+\lambda L_X(s))L''_X(s)}\bigg|_{s=0} \tag{6.51}$$

We can now set $s = 0$ and after simplifying the expression, we obtain the famous *Pollaczek-Khinchin* formula for expected waiting time:

$$E[W] = \frac{(1-\rho)L''_X(0)}{2(1+\lambda L'_X(0))^2} \tag{6.52}$$

$$= \frac{\lambda E[X^2]}{2(1-\rho)} \tag{6.53}$$

The second moment of the waiting time distribution can be found in a similar fashion:

$$E[W^2] = (-1)^2 \frac{d^2}{ds} \frac{s(1-\rho)}{s-\lambda+\lambda L_X(s)}\bigg|_{s=0} \tag{6.54}$$

$$= \frac{\lambda(\rho-1)(2(-1+L_X(s)-sL'_X(s))(1+\lambda L'_X(s))+s(s-\lambda+\lambda L_X(s))L''_X(s)}{(s-\lambda+\lambda L_X(s))^3}\bigg|_{s=0} \tag{6.55}$$

As was the case with the first moment, we get an indeterminate result when $s = 0$. We have to apply *l'Hopital's rule* three times to the numerator and denominator (omitted here for brevity) and set $s = 0$ to obtain the following:

$$E[W^2] = \frac{\lambda(-1+\rho)(-3\lambda(L''_X(0)^2+2(1+\lambda L'_X(0))L'''_X(0))}{6(1+\lambda L'_X(0))^3} \tag{6.56}$$

$$= \frac{\lambda(-1+\rho)(-3\lambda E^2[X^2]+2(1-\rho)(-E[X^3])}{6(1-\rho)^3} \tag{6.57}$$

We can compute the variance in the waiting time as follows:

$$var[W] = E[W^2] - E[W]^2 \tag{6.58}$$

6.7.2 General Bounded Hyper-exponential analysis

The p.d.f of our Bounded Hyper-exponential was defined in Section 6.3 of the previous chapter. We now wish to obtain the Laplace transform $L_{bh_n}(s)$ of the service distribution $bh_n(t)$.

We proceed as follows:

$$L_{bh_n}(s) = \int_k^p e^{-st} bh_n(t).dt \tag{6.59}$$

$$= \int_k^p e^{-st} \sum_{i=1}^{n} \frac{P_i \lambda_i e^{-\lambda_i t}}{e^{-\lambda_i k} - e^{-\lambda_i p}}.dt \tag{6.60}$$

$$= \sum_{i=1}^{n} \frac{P_i \lambda_i (e^{-(s+\lambda_i)k} - e^{-(s+\lambda_i)p})}{(e^{-\lambda_i k} - e^{-\lambda_i p})(\lambda_i + s)} \tag{6.61}$$

We can now obtain the nth moments of the service distribution as follows:

$$E[X^n] = (-1)^n \frac{d^n L_{bh_n}(s)}{ds}\bigg|_{s=0} \tag{6.62}$$

Using this procedure we find the first moment (mean), $E[X]$:

$$E[X] = (-1)\frac{d}{ds}\sum_{i=1}^{n}\frac{P_i\lambda_i(e^{-(s+\lambda_i)k}-e^{-(s+\lambda_i)p})}{(e^{-\lambda_i k}-e^{-\lambda_i p})(\lambda_i+s)}\Bigg|_{s=0} \tag{6.63}$$

$$= \sum_{i=1}^{n}\frac{P_i\lambda_i e^{-(k+p)s}(e^{k(s+\lambda_i)}(1+p(s+\lambda_i))-e^{p(s+\lambda_i)}(1+k(s+\lambda_i)))}{(e^{k\lambda_i}-e^{p\lambda_i})(s+\lambda_i)^2}\Bigg|_{s=0} \tag{6.64}$$

$$= \sum_{i=1}^{n}\frac{P_i(e^{k\lambda_i}(1+p\lambda_i)-e^{p(\lambda_i}(1+k\lambda_i))}{(e^{k\lambda_i}-e^{p\lambda_i})\lambda_i} \tag{6.65}$$

The second moment, $E[X^2]$ is similarly easy to find:

$$E[X^2] = (-1)^2\frac{d^2}{ds}\sum_{i=1}^{n}\frac{P_i\lambda_i(e^{-(s+\lambda_i)k}-e^{-(s+\lambda_i)p})}{(e^{-\lambda_i k}-e^{-\lambda_i p})(\lambda_i+s)}\Bigg|_{s=0} \tag{6.66}$$

$$= \sum_{i=1}^{n}\frac{P_i(e^{k\lambda_i}(2+p\lambda_i(2+p\lambda_i))-e^{p(\lambda_i}(2+k\lambda_i(2+k\lambda_i)))}{(e^{k\lambda_i}-e^{p\lambda_i}){\lambda_i}^2} \tag{6.67}$$

We can also find $L_W(s)$, the Laplace transform of the waiting time. This known result is defined as follows:

$$L_W(s) = \frac{s(1-\rho)}{s-\lambda+\lambda L_{bh_n}(s)} \tag{6.68}$$

$$= \frac{s(1-\rho)}{s-\lambda+\lambda\sum_{i=1}^{n}\frac{P_i\lambda_i(e^{-(s+\lambda_i)k}-e^{-(s+\lambda_i)p})}{(e^{-\lambda_i k}-e^{-\lambda_i p})(\lambda_i+s)}} \tag{6.69}$$

Unfortunately, we cannot easily numerically invert this result in order to obtain $w(t)$, the p.d.f of the waiting time distribution.

However, We can still obtain the moments of the waiting time distribution, as follows:

$$E[W^n] = (-1)^n\frac{d^nL_W(s)}{ds}\Bigg|_{s=0} \tag{6.70}$$

We computed the first two moments of waiting time generally in Section 6.7.1. It is just a matter of substituting the appropriate moments of the Bounded Hyper-exponential service distribution into those expressions.

The variance in the waiting time can then be computed as follows:

$$var[W] = E[W^2]-E[W]^2 \tag{6.71}$$

Given that we can now obtain the mean *and* the variance of certain metrics (such as the waiting time), we can utilise Chebyshev's Inequality [Ross, 2002] to find upper and lower bounds on the probability of specific Quality of Service targets being met.

6.8 Conclusion

Prony's method (as described by Feldman and Whitt [Feldmann and Whitt, 1997]) is not well suited to fitting a Hyper-exponential to a bounded General distribution, nor was it expected to be. However, the recursive process that Prony's method utilises is extremely well suited to adaptation for our purposes. First we normalised the Hyper-exponential obtained by Prony's method in order to obtain a better fit when the original distribution is bounded. We then adapted Prony's method in order to fit a Bounded General distribution directly to a bounded Hyper-exponential distribution, providing an even better fit. The "area of interest" we fit to is simply between the lower and upper bounds of the original bounded distribution, resulting in a Bounded Hyper-exponential distribution with similar statistical properties to the original distribution it is approximating. We incorporated Hyper-exponential and Bounded Hyper-exponential service distributions into Random and TAPTF queueing models and showed that the metrics of interest were consistent with the original Bounded Pareto Random and TAPTF models. The Bounded Hyper-exponential distribution so obtained can then be utilised to find Laplace transforms of important queueing metrics such as waiting time and slowdown. We then demonstrated that since the transform of a series of exponential distributions can be presented as a rational function, the result can be readily differentiated to provide mean values and higher moments (and consequently the variance) of these queueing metrics for certain queueing systems.

Chapter 7

Obtaining advanced performance metrics via simulation

In Chapter 4 we explored methodologies for modelling and analysis of queueing systems under highly variable 'heavy-tailed' workloads. Despite many useful findings, we discovered that certain metrics were, for all practical purposes, extremely difficult if not impossible to compute for size-based task assignment policies like TAPTF.

For this reason, and also to validate the analytical results presented in previous chapters, we wish to perform rigorous simulation of TAPTF, as well as other existing task assignment approaches like TAGS and Random. We are still concerned with the mean metrics, but also wish to observe measurements that were difficult to compute analytically, such as the variance in the mean waiting time and slowdown. These measurements are particularly important due to the highly variable nature of the workloads we are simulating, as the mean metrics can often be misleading and highly unrepresentative of the experience of many tasks in the system.

7.1 Simulation framework

We utilised the OMNeT++ Discrete Event Simulation System [Varga, 2001] as a platform to perform comparative simulations of a variety of task assignment policies, under a wide range of loads and workloads. OMNeT++ is a public-source, highly modular framework that is commonly used to model communication networks, queueing networks, hardware architectures and business processes.

We developed add-on modules to OMNeT++ (in C++) to accurately simulate the oper-

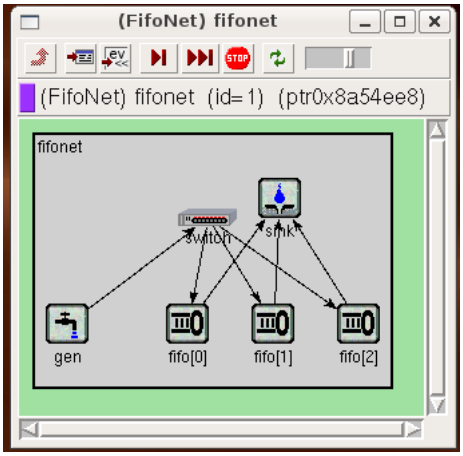


Figure 7.1: Random 3 Host OMNeT++ model

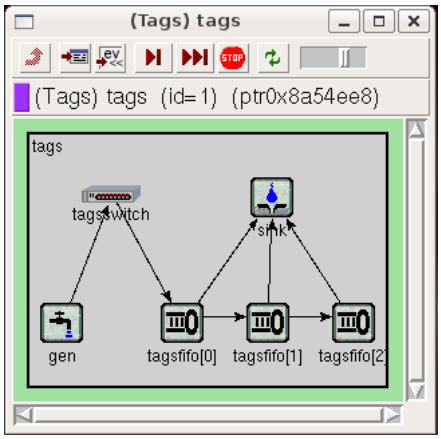


Figure 7.2: TAGS 3 Host OMNeT++ model

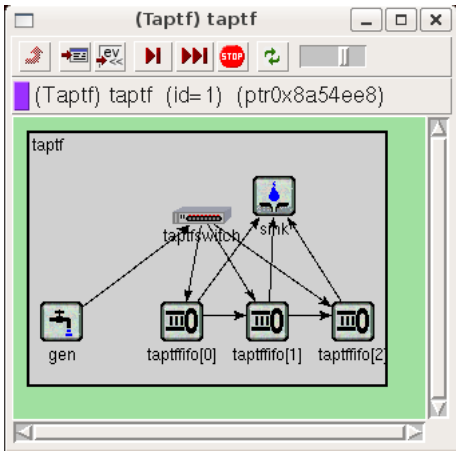


Figure 7.3: TAPTF 3 Host OMNeT++ model

ation of task assignment policies including Random, TAGS and TAPTF. These modules are depicted in Figures 7.1 - 7.3 respectively.

Each task assignment policy was modelled with a task generator, a dispatcher (i.e. switch) and a number of back-end hosts. The task generator generated tasks with a service requirement drawn from a Bounded Pareto distribution, and with inter-arrival times drawn from a Poisson distribution. The dispatcher implemented the logic of the task assignment policy it was modelling, directing tasks to specific or random back-end hosts where appropriate. The back-end hosts receive tasks from the dispatcher (and from other back-end hosts in the case of TAPTF and TAGS) and processes them according to the relevant task assignment policy.

Simulating these task assignment policies also allows us to collect a wide variety of queueing metrics - many of which are difficult to compute analytically. Due to the modular nature of OMNeT++ collecting these metrics was easy to achieve. By simply extending the standard message type that is utilised in OMNeT++, we were able to measure metrics on a per-task basis and collate them at the 'sink' at the end of a simulation run.

[caption=Custom Task in OMNeT++,label=mytask] message MyTask fields: int myNum; // Task number int myPriority = 0; // Used for TAPTF double mySize = 0.0; // Size generated from B(k,p,alpha) int hand-offs = 0; // Times task has been handed off double waste = 0.0; // Total wasted processing int recordMe = 0; // Task is warm-up or recorded int firstHost = 0; // Host where task is assigned int lastHost = 0; // Host where task runs-to-completion

Listing **??** shows the custom message (i.e a *Task*) that was defined. Each task in the system has a unique task number. In the case of TAPTF we also need to denote the priority

of a task - as tasks that are restarted are placed in the restart queue at the following host, which has priority over the ordinary queue at a given host. A task has a size than is generated from a Bounded Pareto distribution $BP(k, p, \alpha)$ as has been the case in previous chapters. We also count the number of hand-offs a given task experiences (where a task exceeds the cutoff at a given host, and is restarted from scratch at the next host), and the wasted processing generated from the hand-off. The hand-off and waste metrics only apply to TAPTF and TAGS. We typically run a given simulation for a lengthy warm-up period so the queues can reach a steady state before we take any measurements. During the warm-up period a task's recordMe variable is set to 0, and once the warm-up period has passed it is set to 1 so the sink knows it needs to measure this task. For the TAGS and TAPTF policies, we also track which host a task enters the system and also the host where a task runs-to-completion. Obviously for the TAGS policy, tasks always enter at the first host, but we are still interested in where they finish. For the TAPTF policy we can also measure which *queue* at a given host a task runs-to-completion in.

7.2 Simulation methodology

Accurately simulating highly variable workloads is a very challenging proposition. Indeed, we do not expect the simulation results to match the analytical results perfectly, but we do expect to see the same general performance trends observed in Chapters 3 and 4. The difficulty with simulating queueing systems with highly variable service distributions is the sheer amount of simulation runs required for the observed mean of the samples to converge to the analytical mean [Heidelberger, 1995].

The service requirements and inter-arrival times generated in the simulation follow a Bounded Pareto and Poisson distribution respectively, and are generated using a modern, high quality random number generator known as Mersenne Twister [Matsumoto and Nishimura, 1998]. The random number generator is seeded with unique seeds for each run so that all runs are repeatable, and task assignment policies are directly comparable (as they experience the exact same tasks and workload for a given load scenario).

The ideal simulation methodology would be as follows. Each experimental data point (for a given task assignment policy) is derived from n independent samples generated by the simulation platform. For each independent sample we run the queueing system for a lengthy warm-up phase. For the two host scenarios, the warm-up phase consists of 100,000 tasks, after which we measure the behaviour of task 100,001 (i.e. the waiting time and slowdown). For

three hosts, the warm-up phase runs for 150,000 tasks, after which task 150,001 is measured. From the n independent samples we can compute the sample mean as well as the variance, standard deviation and confidence intervals of the sample mean.

In statistics, a sample of $n = 30$ is usually considered sufficient (or statistically significant) for the Central Limit Theorem to apply (providing the variance of the distribution being measured is finite) [Ross, 2001]. For our purposes this is nowhere near sufficient, and we found that samples of 100, 300, 600, 1000 and even 3000 were often observed to provide unacceptable results in our experiments. For our purposes the more samples taken the better, but consideration must be taken as the experiments can take an unacceptable amount of time to run. Measuring 5000 samples, which we hoped would provided an acceptable balance between the time needed to run the simulations and the statistical significance of the results, still resulted in measurements that diverged significantly from the analytical results. It was not feasible to increase the number of independent samples any further.

Perhaps surprisingly, the divergence between the simulation and analytical results was most apparent as the variation in the workload dropped (i.e. $1 \leq \alpha \leq 2$). This was particularly unfortunate as this was the key area where TAPTF improved performance over the TAGS policy. However it makes sense given our knowledge of heavy-tailed distributions. As α increases, the probability of these 'rare' large events becomes smaller (but, we emphasise, is not negligible.) However there is a high probability of these 'rare' large events not occurring at all in a sample space of $n = 5000$, providing misleading results.

To get acceptable results, we needed to change our simulation approach to enable us to get a large number of samples quickly. We could not simply measure $500,000$ independent samples (for each data point required) as that would take an unfeasible amount of time. We observed that the majority of the simulation CPU time is taken up in the start-up phase (before any tasks have been generated), rather than when the simulation is actually running. Therefore instead of measuring a single task from each simulation run, we take a large sample of tasks from a given run, and collate our metrics from that. For instance, we can run a simulation for a warm-up period of $100,000$ tasks, then measure the next $500,000$ tasks.

The downside of this approach is that, strictly speaking, the samples are no longer independent (i.i.d) samples. This could be disputed however, as it has been argued in prior work on queueing systems that the dependencies of samples X_1, X_2, X_2 ... X_n becomes weaker as the samples become further apart. Despite these findings, we will refrain from computing confidence intervals from the samples.

The simulation itself is naturally *embarrasingly parallel*, and is essentially a parameter-

sweep style application. The simulation runs were 'farmed' out to eight machines to expediate the execution of the hundreds of thousands of runs that are aggregated to make up the results in this chapter. Despite this, due to the sheer number of runs required the simulations still took nearly two weeks to complete.

7.3 Simulation results

In this section we present detailed simulation results showing the performance of the TAPTF, TAGS and Random. We consider both two and three host configurations, and observe a wide range of workloads (where α is varied from 0.5 to 2.0). We also consider a variety of system loads (represented by ρ) ranging from low (0.3), moderate (0.5), and high (0.7).

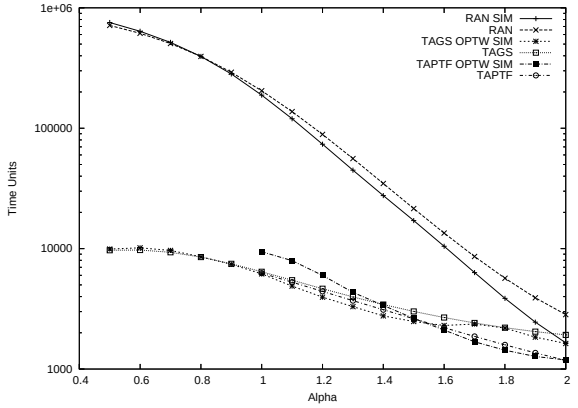
We focus on the mean metrics (such as mean waiting time and slowdown), as well as the variance those metrics. We also measure the waiting time and slowdown on a per host, and in the case of TAPTF, per queue basis. Finally, we measure the number of hand-offs generated in each scenario, as well as the corresponding wasted processing that occurs. For each graph, where expected waiting times are measured, the TAGS and TAPTF models that have parameters optimised for this metric are utilised. Likewise, where slowdown is shown, TAGS and TAPTF models that have optimised parameters for this metric are used.

7.3.1 Two Hosts

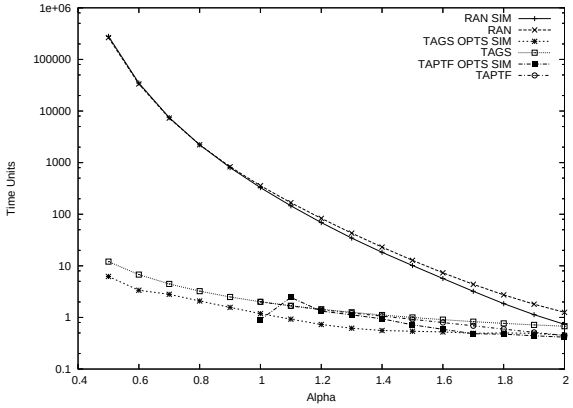
Figures 7.4(a) and (b) depict the expected waiting time for a two host distributed system under a low system load of 0.3. We can see that the results are largely consistent with the figures obtained analytically in Chapter 4. TAPTF shows a marginal improvement in the waiting time as the task size distribution becomes less variable.

Figures 7.4(c) and (d) show a different perspective. Here we observe the variance for both the waiting time and slowdown. With respect to the variance in waiting time, the results are clearly in favour of the TAPTF policy, over a wide range of α values (i.e. $\alpha > 1.1$). As a result we have more confidence in the mean metrics presented for TAPTF rather than TAGS in Figure 7.4(a). Consequently, but over a smaller region ($\alpha > 1.5$), we can make a similar claim regarding our confidence in the mean slowdown metrics.

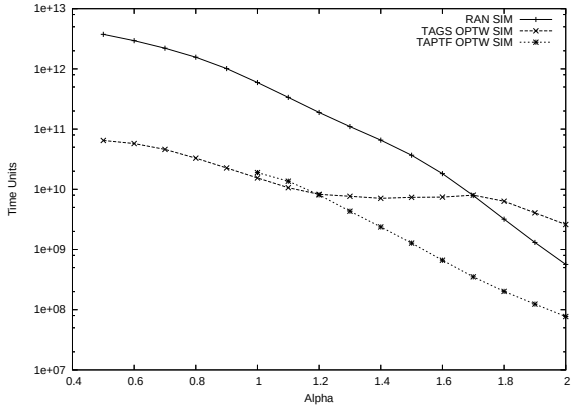
When we examine the results on a per host basis (Figures 7.4(e) and (f)), we can see as expected, that there is very little difference in the expected waiting time for each host under a Random task assignment policy. Random simply splits the incoming task stream and makes no effort to reduce the variance experienced by tasks. We note that the first TAGS host



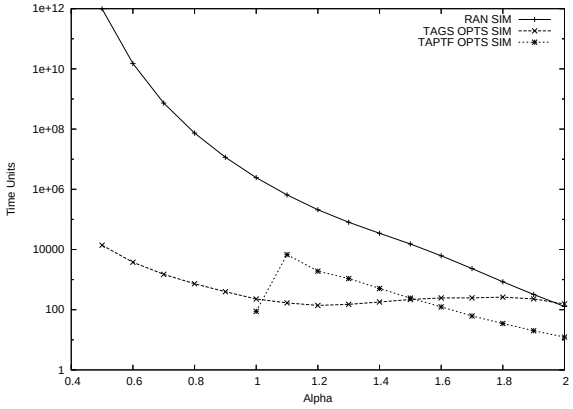
(a) E(W) - $\rho = 0.3$



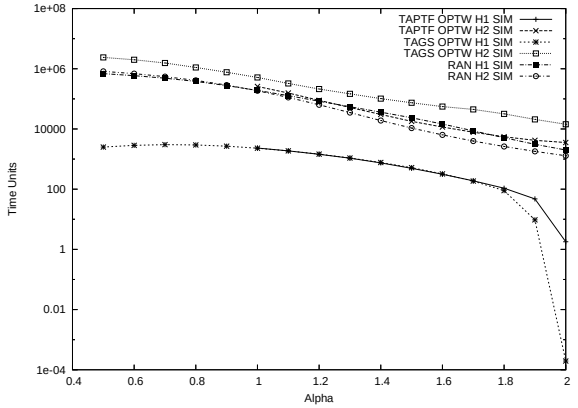
(b) E(S) - $\rho = 0.3$



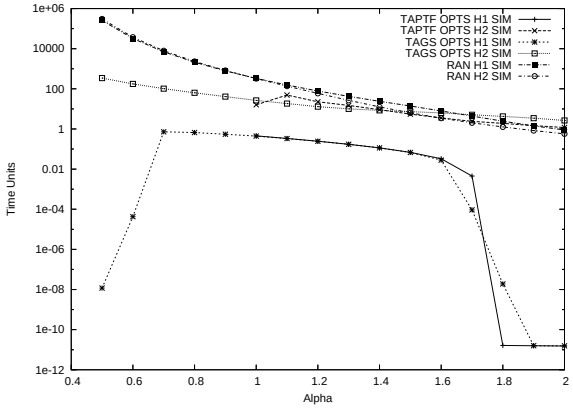
(c) Var(W) - $\rho = 0.3$



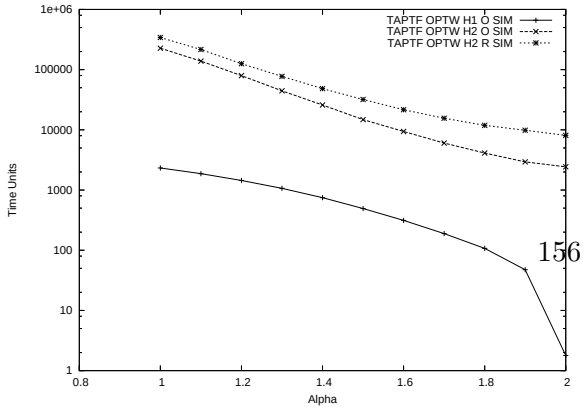
(d) Var(S) - $\rho = 0.3$



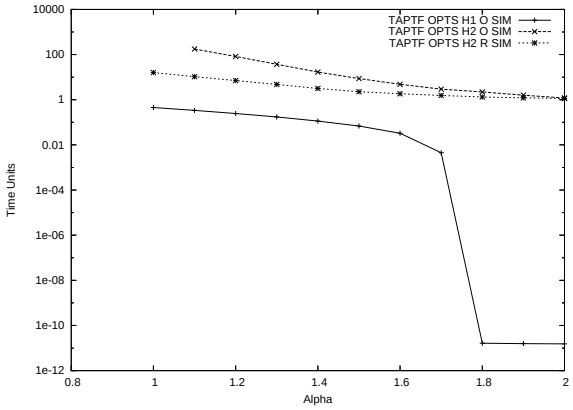
(e) E(W) per host - $\rho = 0.3$



(f) E(S) per host - $\rho = 0.3$

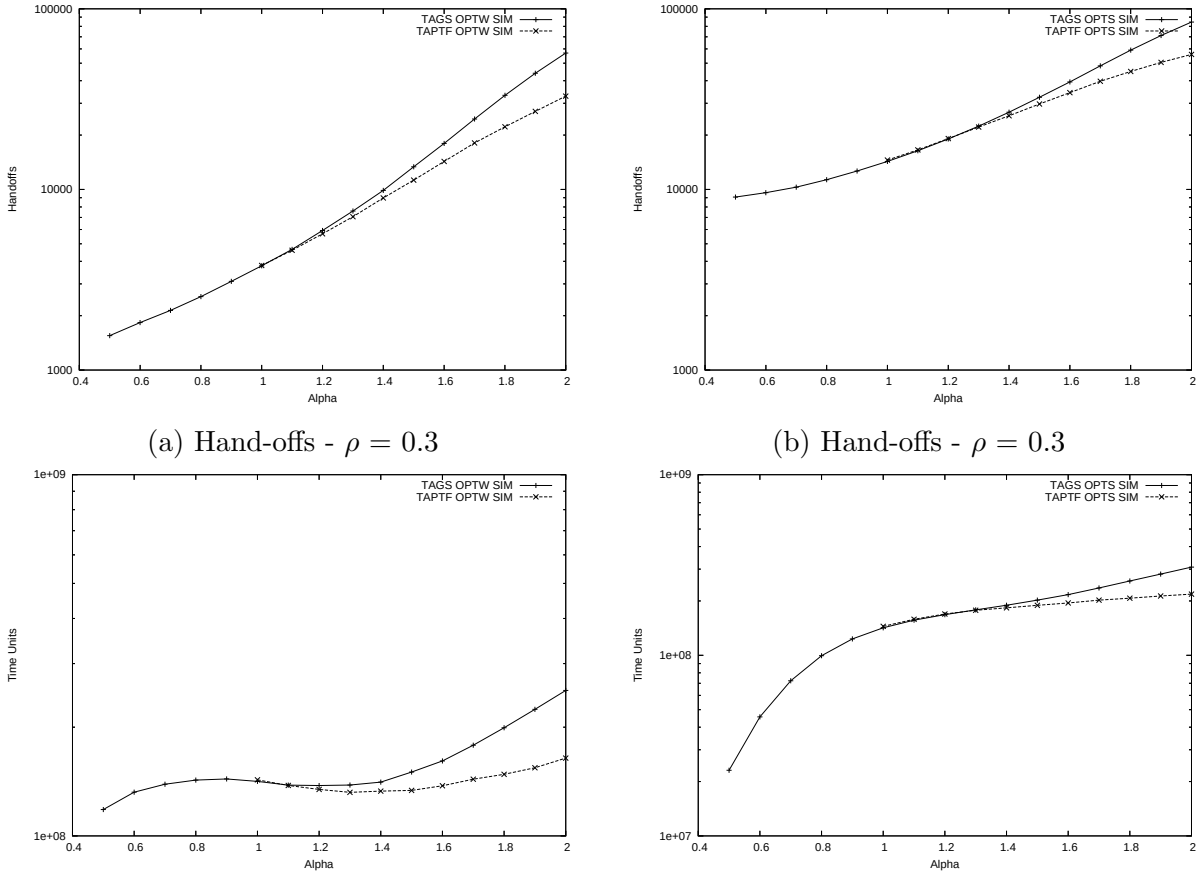


(g) E(W) per queue - $\rho = 0.3$



(h) E(S) per queue - $\rho = 0.3$

Figure 7.4: Performance of a two host distributed system with system load of 0.3



(a) Hand-offs - $\rho = 0.3$

(b) Hand-offs - $\rho = 0.3$

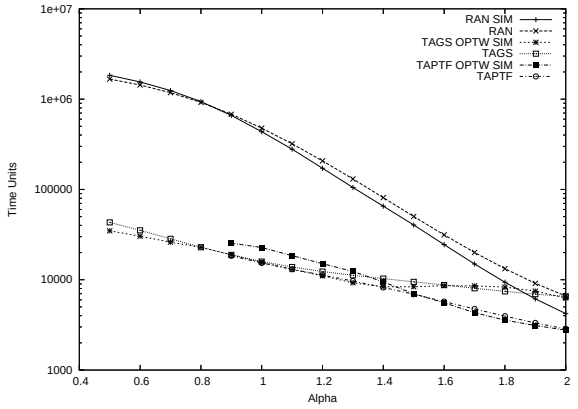
(c) Waste - $\rho = 0.3$

(d) Waste - $\rho = 0.3$

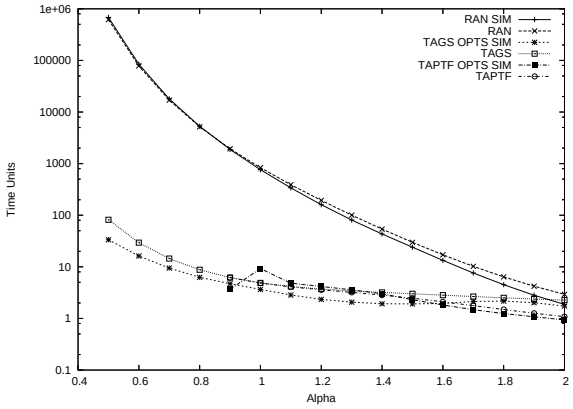
Figure 7.5: The number of hand-offs for systems optimised for waiting time or slowdown are shown in (a) and (b) respectively. The corresponding waste is shown in (c) and (d).

has the lowest expected waiting time, while the second TAGS host has the highest expected waiting time. This has little effect on the overall TAGS metrics as it processes the majority of tasks to completion on the first host. This only becomes detrimental as the distribution of service requirements becomes less variable, and the first host becomes overworked. The first TAPTF host also has similarly low expected waiting time to TAGS, while exhibiting slightly better expected waiting time at the second host. These trends are largely consistent when examining the per host slowdown.

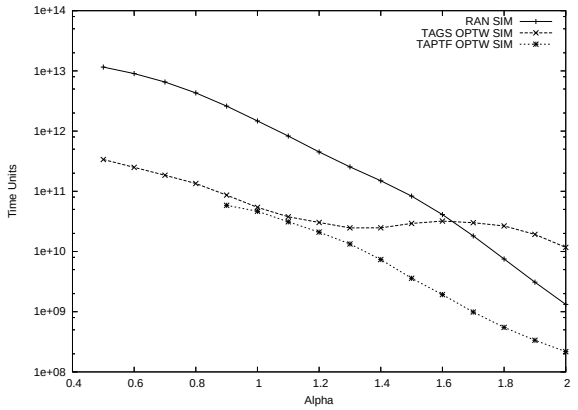
Figures 7.4(g) and (h) depicts the expected waiting time and slowdown for the TAPTF on a per queue basis. We can observe for the waiting time, that the expected values increase as we go from the first queue to the last queue. Unsurprisingly, the Ordinary queue at Host 1



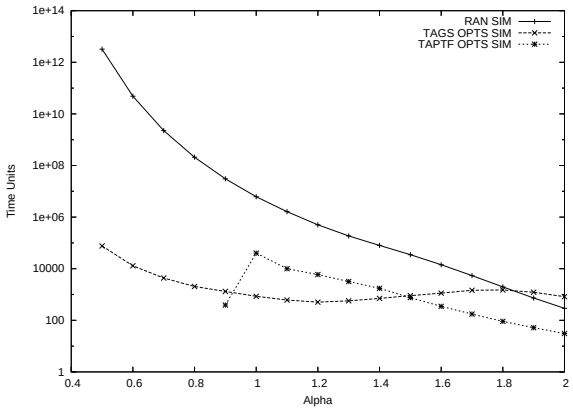
(a) E(W) - $\rho = 0.5$



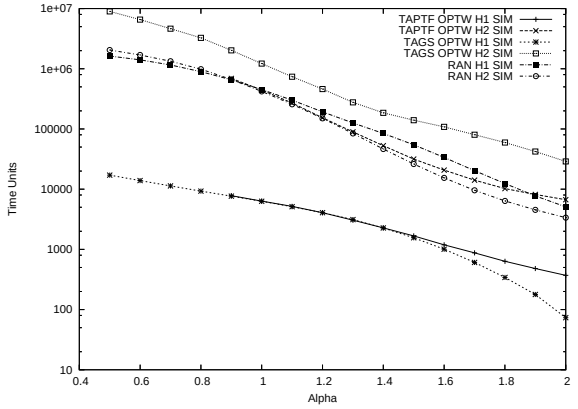
(b) E(S) - $\rho = 0.5$



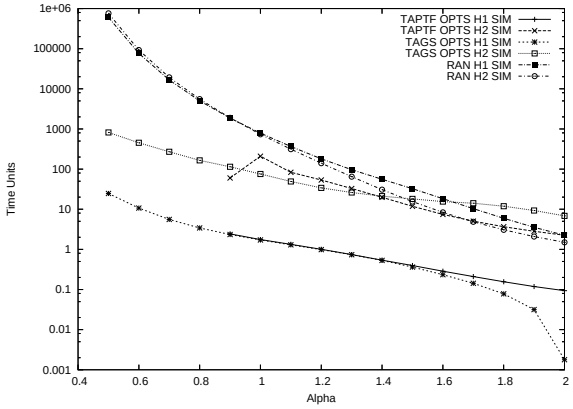
(c) Var(W) - $\rho = 0.5$



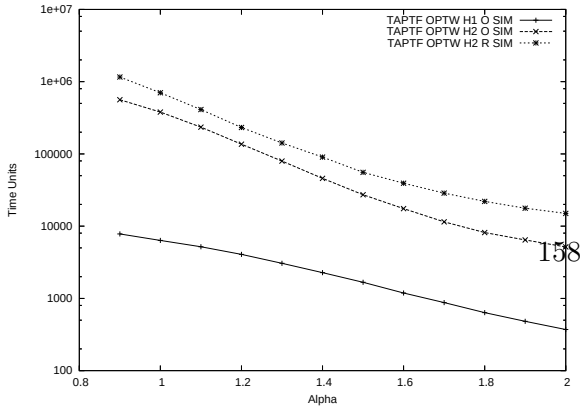
(d) Var(S) - $\rho = 0.5$



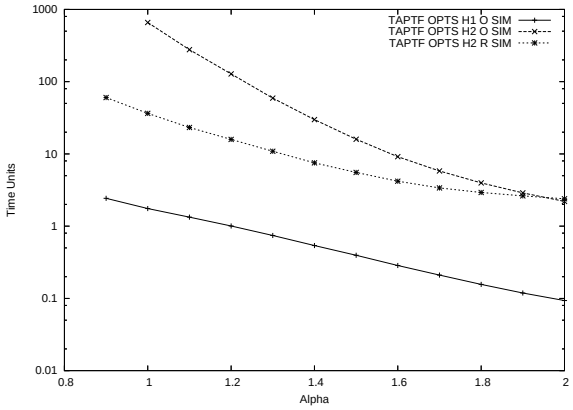
(e) E(W) per host - $\rho = 0.5$



(f) E(S) per host - $\rho = 0.5$

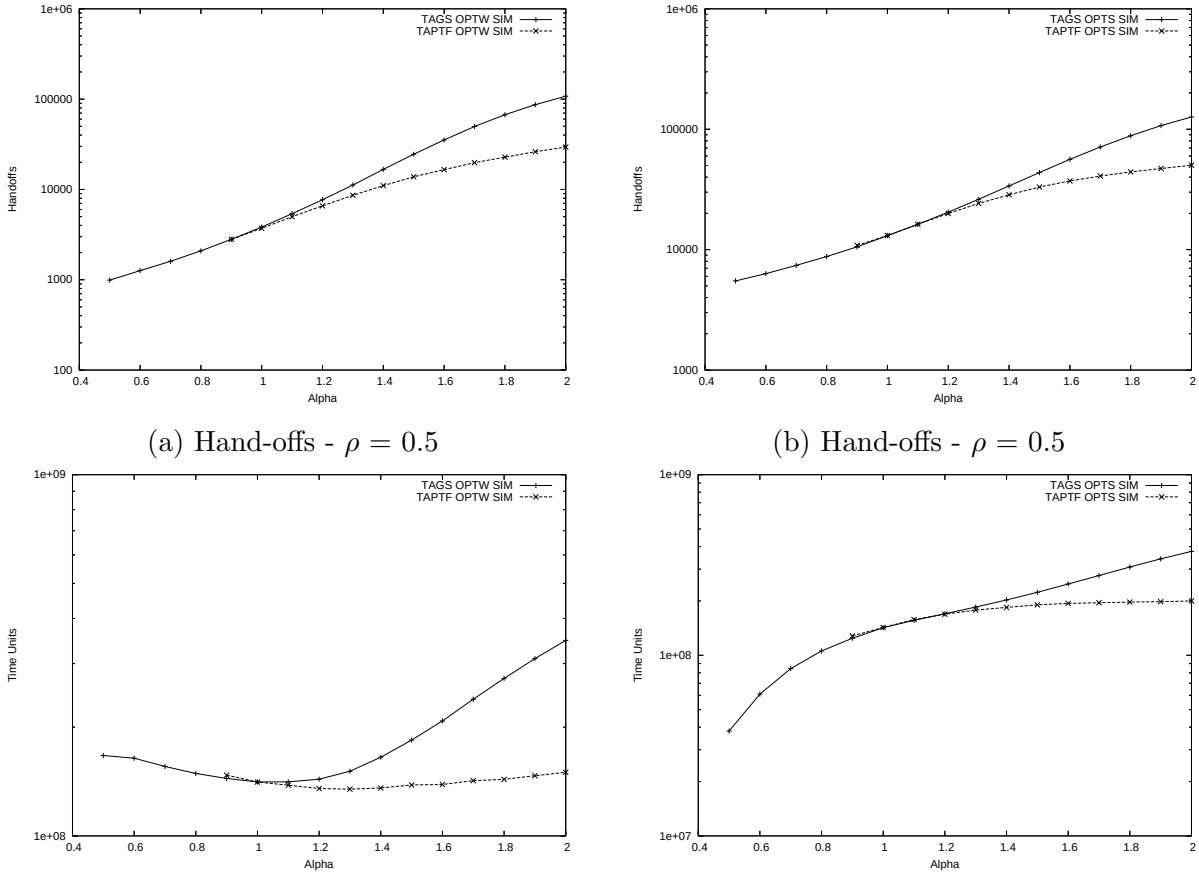


(g) E(W) per queue - $\rho = 0.5$



(h) E(S) per queue - $\rho = 0.5$

Figure 7.6: Performance of a two host distributed system with system load of 0.5.



(a) Hand-offs - $\rho = 0.5$

(b) Hand-offs - $\rho = 0.5$

(c) Waste - $\rho = 0.5$

(d) Waste - $\rho = 0.5$

Figure 7.7: The number of hand-offs for systems optimised for waiting time or slowdown are shown in (a) and (b) respectively. The corresponding waste is shown in (c) and (d).

has the lowest expected waiting time and slowdown. This queue services the smallest *range* of tasks, keeping the performance measures low. Conversely, the Restart queue at Host 2 tends to service the largest tasks, which has a negative impact on key performance measures. However, when optimised for slowdown, the Ordinary queue has worse expected slowdown than the restart queue at the second host, due to the fact that it services the widest range of task sizes. As such, there is increased probability of small tasks sharing that queue with larger tasks, which can lead to higher slowdown.

The average number of hand-offs that occur in systems optimised for waiting time and slowdown is shown in Figures 7.5(a) and (b). TAPTF shows a significant reduction in hand-offs for both cases as the workload becomes less variable. By the time α reaches 2,

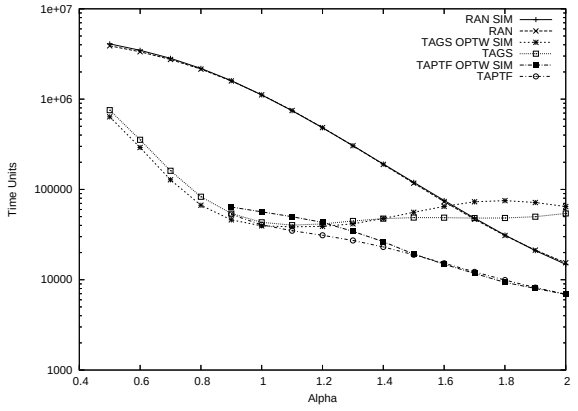
TAPTF has half as many hand-offs as TAGS. The corresponding wasted processing is shown in Figures 7.5(c) and (d). The corresponding reduction in wasted processing is not as large as you would think, but clearly is significant enough to result in improved performance of TAPTF over TAGS, as highlighted by Figure 7.4

The waiting time and slowdown metrics shown in 7.6(a) and (b) for a system load of 0.5 are consistent with the analytical results found previously. In keeping with these results, we see a clear improvement for TAPTF over TAGS for a wider range of α values on the horizontal axis.

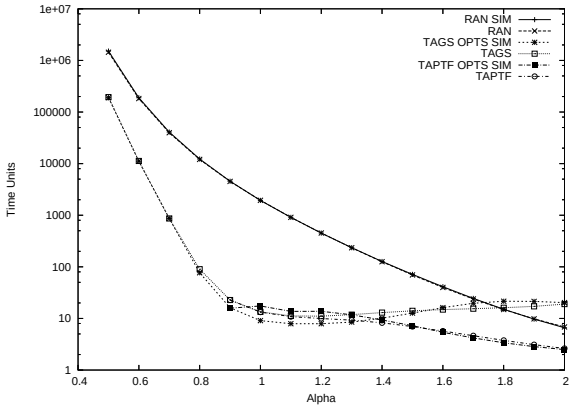
The variance in the waiting time and slowdown is depicted in Figures 7.6(c) and (d) respectively. Regarding the variance in waiting time, we can see that the breadth and magnitude of the improvement of TAPTF over TAGS has increased as the system load was raised to 0.5. Under a key area of high variance ($0.9 \leq \alpha < 1.4$) we can see that while TAGS has slightly better mean waiting time than TAPTF, the opposite is true regarding the variance in waiting time. We also note reduced variance in slowdown where $\alpha > 1.5$, corresponding with an improvement in mean slowdown over the same region.

On a per host basis (7.6(e) and (f)), we again observe that for expected waiting time, TAGS shows both the best (Host 1) and the worst (Host 2) performance results. Again we note that this is not as troublesome as it seems under a moderate utilisation of 0.5, as the overwhelming majority of tasks are processed by the first host. This only becomes detrimental as the workload tends toward being less variable. TAPTF has a much less variable spread of expected waiting times for its respective hosts. TAGS does much better when optimised for expected slowdown in this case, being only marginally bested by TAPTF as α decreases.

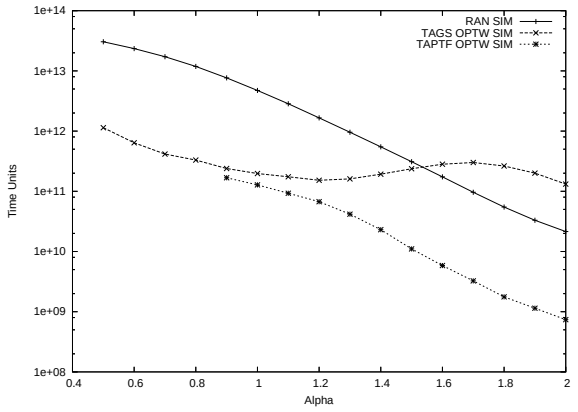
The per queue metrics measured for expected waiting time and slowdown under a system load of 0.5, depicted in 7.6(g) and (h), are consistent with the trends observed earlier under lower load. When optimised for expected waiting time, the metrics increase as we consider Host 1's Ordinary queue, Host 2's Ordinary queue, and finally Host 2's restart queue - which has the largest expected waiting time. As we saw when the system load was 0.3, when optimised for slowdown the Ordinary queue at Host 2 has a larger expected slowdown than the Restart queue at the same host. On reflection this is not that surprising, given what we know about the size ranges of tasks these two queues service. Tasks in the Restart queue are much more likely to be queueing behind other tasks of a similar size. Tasks in the Ordinary queue could potentially range from the smallest task, k, to the largest task, p. As the slowdown metric considers the waiting time proportional to a task's size, these findings make sense.



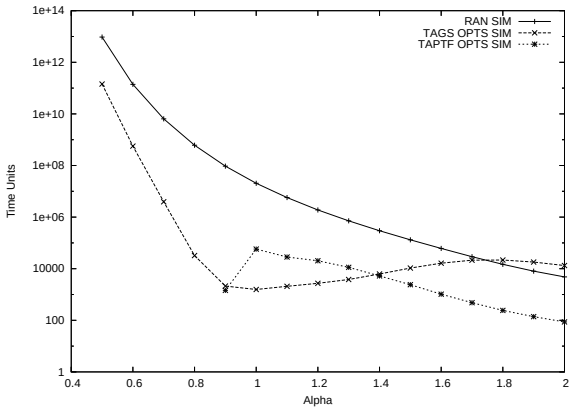
(a) E(W) - $\rho = 0.7$



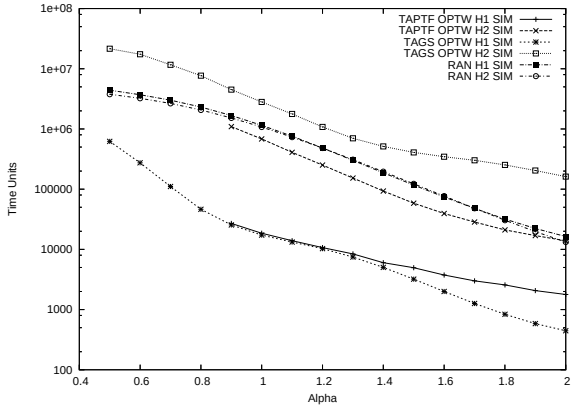
(b) E(S) - $\rho = 0.7$



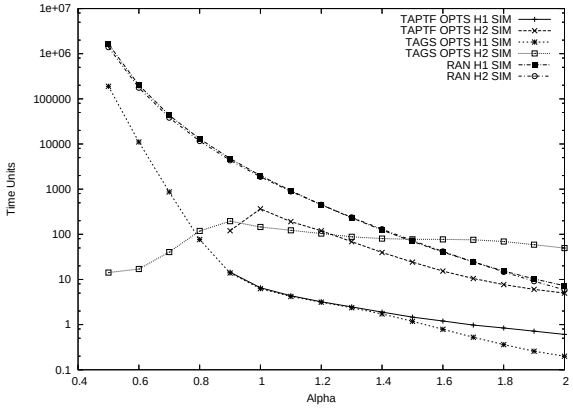
(c) Var(W) - $\rho = 0.7$



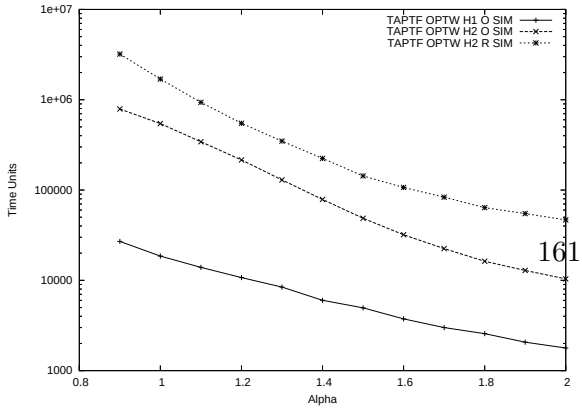
(d) Var(S) - $\rho = 0.7$



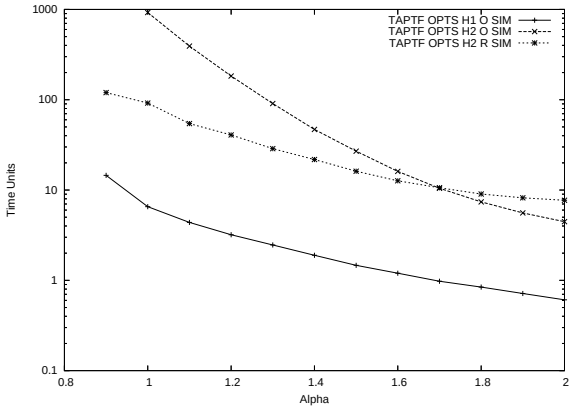
(e) E(W) per host - $\rho = 0.7$



(f) E(S) per host - $\rho = 0.7$

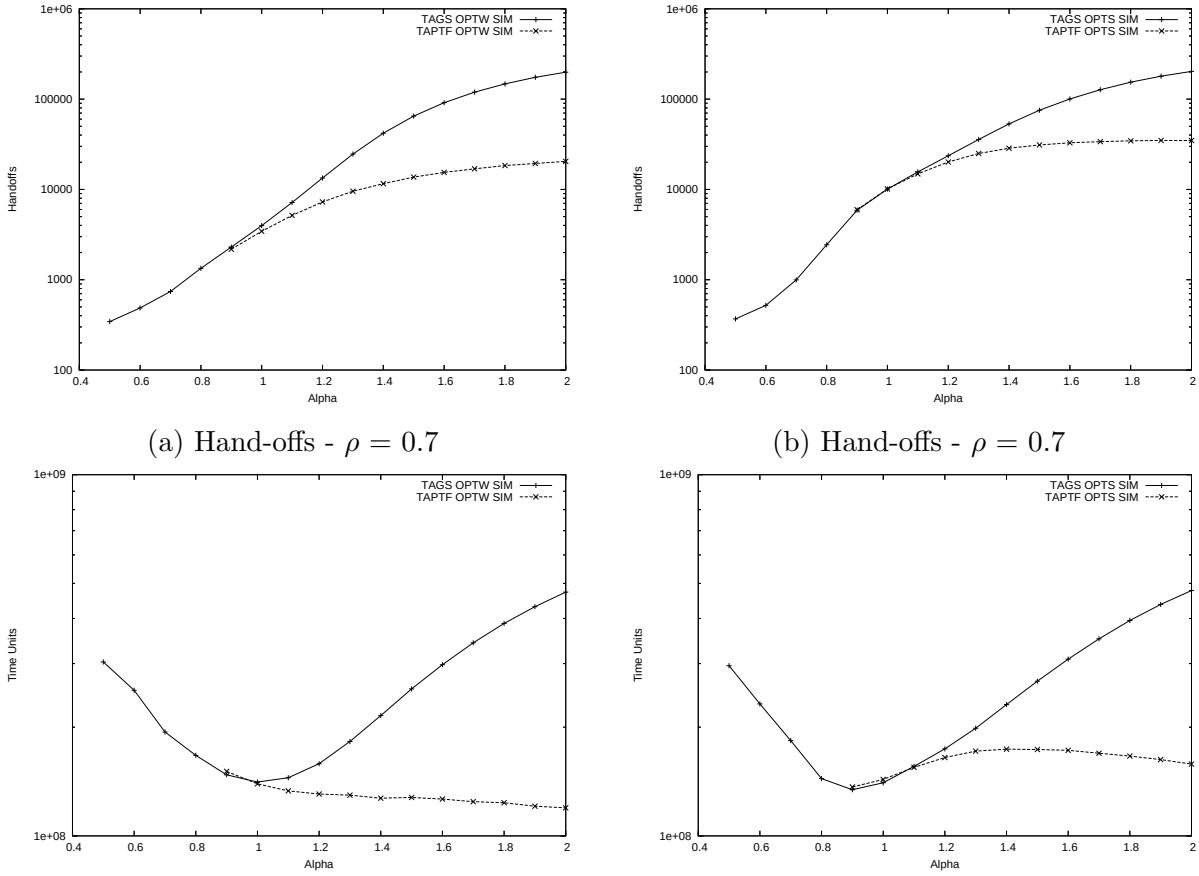


(g) E(W) per queue - $\rho = 0.7$



(h) E(S) per queue - $\rho = 0.7$

Figure 7.8: Performance of a two host distributed system with system load of 0.7



(a) Hand-offs - $\rho = 0.7$

(b) Hand-offs - $\rho = 0.7$

(c) Waste - $\rho = 0.7$

(d) Waste - $\rho = 0.7$

Figure 7.9: The number of hand-offs for systems optimised for waiting time or slowdown are shown in (a) and (b) respectively. The corresponding waste is shown in (c) and (d).

Figures 7.7(a) and (b) illustrates the number of hand-offs for policies optimised for waiting time and slowdown, where the system load is 0.5. We can clearly see the reduction in the number of hand-offs (in both instances) has increased for TAPTF over TAGS. The TAPTF policy has nearly three times less hand-offs in the example illustrated in Figure 7.7(a). Correspondingly, the reduction in wasted processing is also increasing, as highlighted by Figures 7.7(c) and (d).

Figures 7.8(a) and (b) show the expected waiting time and slowdown under a high system load of 0.7. We can see from the figures that the trends identified via analytical measurement are followed very closely by the simulation results. As expected, the range of improvement for TAPTF over TAGS on the horizontal axis has expanded for both the waiting time and

slowdown as the system load has increased.

The variance in waiting time and slowdown for this scenario are shown in Figures 7.8(c) and (d) respectively. We can see a significant improvement in the variance in waiting time that is largely consistent with that exhibited in Figure 7.8(a), for the expected waiting time. TAPTF not only improved on the expected waiting time, but there is also significantly less variance in the waiting time values that were measured, giving us great confidence in the mean. The variance in slowdown is also reduced where $\alpha \geq 1.4$. This corresponds with the improvement shown in the mean slowdown depicted in Figure 7.8(b).

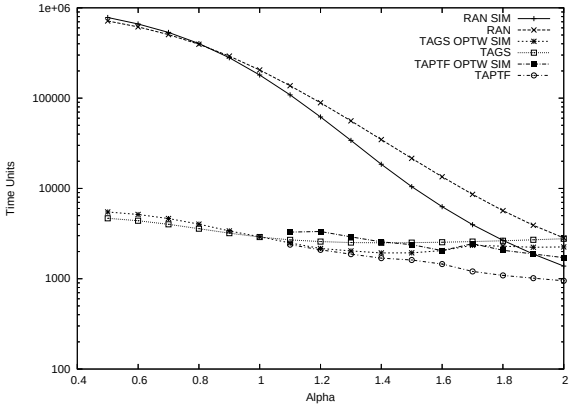
Similar trends are becoming evident when we consider the measurements on a per host basis, shown in Figures 7.8(e) and (f). As expected, the waiting time and slow down for the Random policy are indistinguishable for both hosts. The TAGS policy, optimised for waiting time, again has both the best (Host 1) and worst (Host 2) expected waiting time. In the case of TAPTF (optimised for waiting time), the difference in expected waiting time for the two hosts is not so pronounced. For the policies optimised for slowdown, TAPTF shows much better expected slowdown for its second Host than TAGS. This becomes important as the variance reduces (i.e. α approaches 2), as more hosts need to be processed on the second host for both TAGS and TAPTF.

Figures 7.8(g) and (h) continue to exhibit similar trends in performance for the TAPTF policy, for the expected waiting time and slowdown on a per queue basis. The only change occurs for mean slowdown, where $\alpha > ge1.7$ we can observe at Host 2 that the Ordinary queue has better slowdown than the Restart queue.

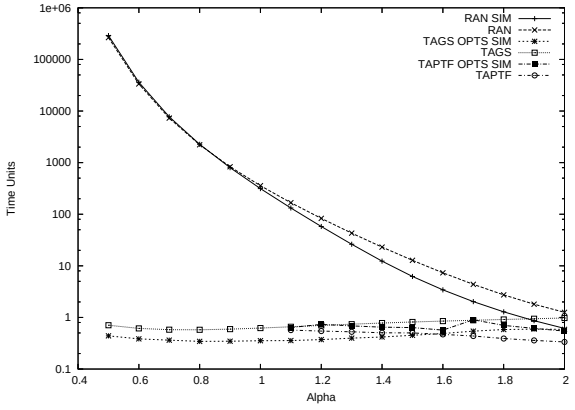
We can begin to see a trend develop when examining Figures 7.9(a) and (b). As the system load increases, the performance of TAGS is obviously suffering. It is easy to identify why. All tasks are dispatched initially to the first host. As the variance begins to decrease (coupled with high system loads), the amount of hand-offs generated is expanding rapidly. From Figures 7.9(c) and (d) we can see that, consequently, the amount of wasted processing created by the TAGS policy has increased significantly also. The amount of hand-offs and waste generated by the TAPTF policy remains relatively steady in comparison.

7.3.2 Three Hosts

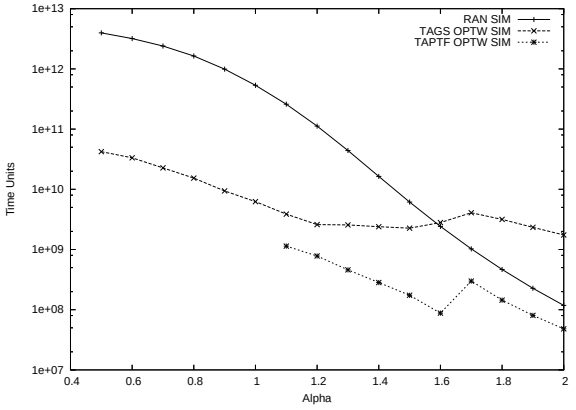
Figures 7.10(a) and (b) depict the expected waiting time for a three host distributed system under a low system load of 0.3. We can see some divergence in the metrics, especially as the variance decreases (and α increases). However the general performance trends are



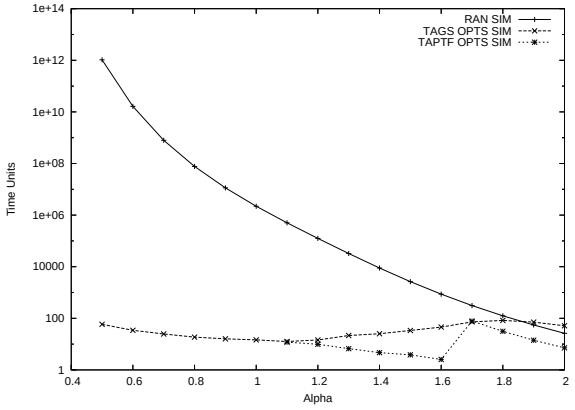
(a) E(W) - $\rho = 0.3$



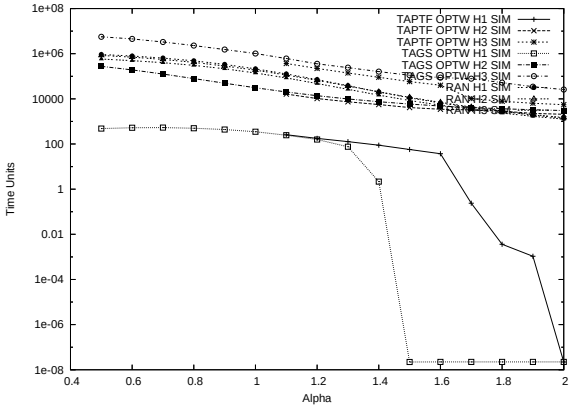
(b) E(S) - $\rho = 0.3$



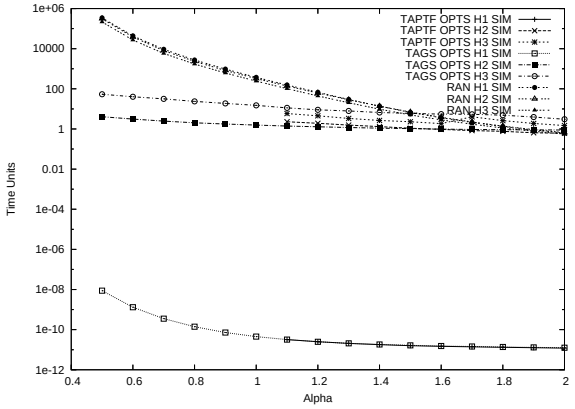
(c) Var(W) - $\rho = 0.3$



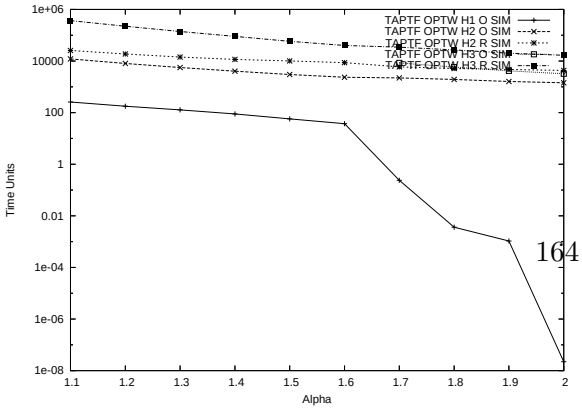
(d) Var(S) - $\rho = 0.3$



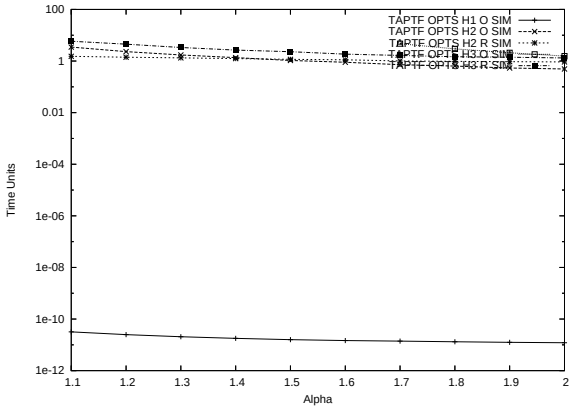
(e) E(W) per host - $\rho = 0.3$



(f) E(S) per host - $\rho = 0.3$

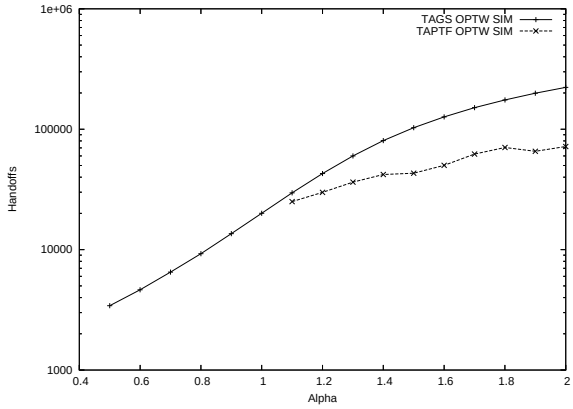


(g) E(W) per queue - $\rho = 0.3$

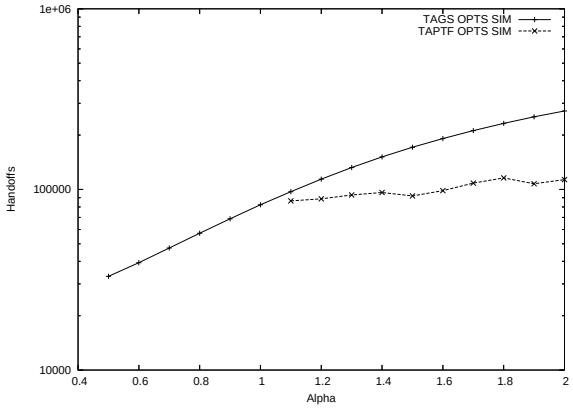


(h) E(S) per queue - $\rho = 0.3$

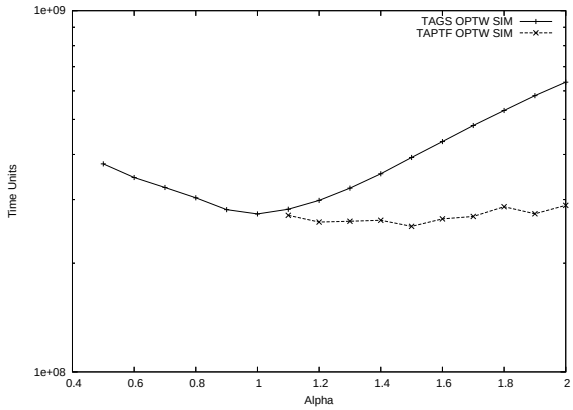
Figure 7.10: Performance of a three host distributed system with system load of 0.3



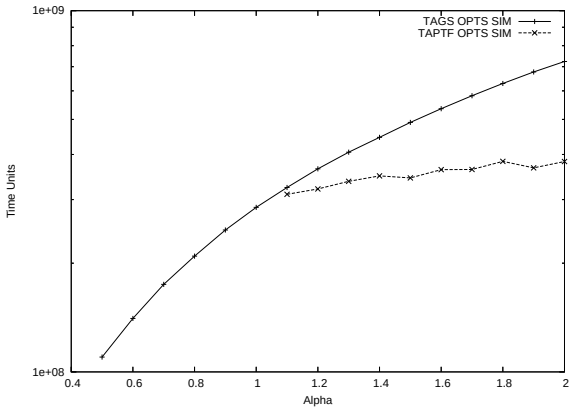
(a) Hand-offs - $\rho = 0.3$



(b) Hand-offs - $\rho = 0.3$



(c) Waste - $\rho = 0.3$



(d) Waste - $\rho = 0.3$

Figure 7.11: The number of hand-offs for systems optimised for waiting time or slowdown are shown in (a) and (b) respectively. The corresponding waste is shown in (c) and (d).

similar. We also note that for each three host scenario, some of the TAPTF parameters (q_1, q_2 and q_3) were tuned by hand rather than found via a solver, as described in Chapter 4. As such, the performance could be improved further for all TAPTF three host cases. The simulated performance of TAGS and TAPTF are largely indistinguishable here, despite a small improvement being evident when comparing the analytical results.

The variance in waiting time and slowdown for this scenario are show in Figures 7.10(c) and (d) respectively. Despite TAPTF only showing a small improvement in expected waiting time over TAGS (Figure 7.10(a)), we can see significantly less variance in the waiting time in Figure 7.10(c). There is also a respectable reduction in the variability of slowdown measured, highlighted in Figure 7.10(d).

Figures 7.10(e) and (f) show the expected waiting time and slowdown on a per host basis. For both expected waiting time and slowdown, the results for each host in the random system are indistinguishable, as anticipated. Again, TAGS is shown to have both the best (Host 1) and the worst (Host 3) expected waiting time. This has little material effect on the performance of TAGS however. With three hosts available (and thus more flexibility in choosing the cutoffs), as well as low system load, means that TAGS performs exceptionally well under these circumstances. Due to this very low utilisation and flexibility in choosing cutoffs, both TAGS and TAPTF have exceptionally low expected waiting time at the first host as the task size variation decreases.

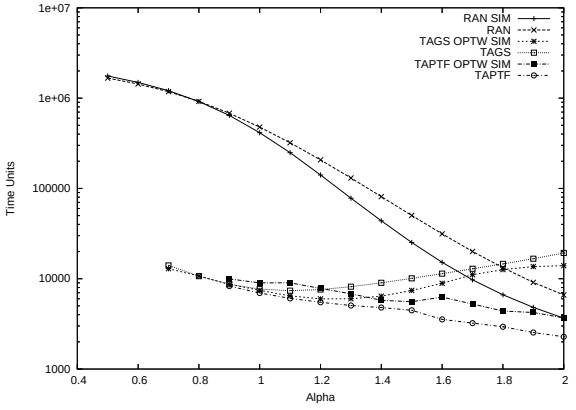
Figure 7.10(g) shows similar trends that were exhibited in the two host scenarios for per queue waiting time. As we move along the prospective queues in a TAPTF system, we note that the expected waiting time increases. In Figure 7.10(h) we observe that while the Ordinary queue at Host 1 has extremely low slowdown due to the tight range of tasks it services and the low overall system load. When $\alpha > 1.5$ we observe that the Restart queue at Host 2 has better slowdown than the Ordinary queue. At all other times, the converse is true. The slowdown for both the Ordinary and Restart queues at Host 3 are indistinguishable.

Despite almost indistinguishable performs gains for the three host scenario (with a system load of 0.3), Figures 7.11(a) and (b) show a healthy reduction in the number of hand-offs demonstrated by a TAPTF policy over a TAGS policy. For policies optimised for both waiting time and slowdown, we can see a significant reduction in hand-offs as the task size variation decreases. Correspondingly, we can also see a reasonable reduction in the amount of wasted processing, depicted in Figures 7.11(c) and (d).

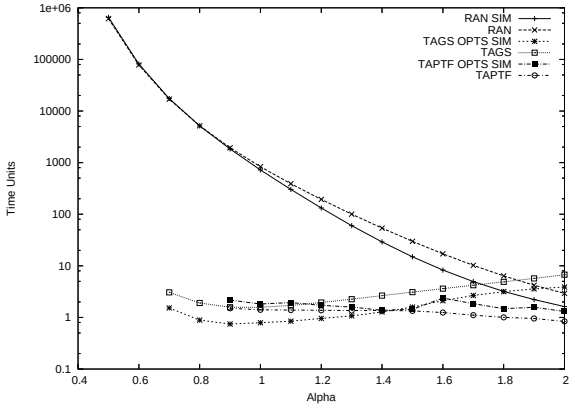
Figures 7.12(a) and (b) show the expected waiting time and slowdown respectively, for a three host system under a moderate load of 0.5. A reasonable improvement in waiting time for TAPTF over TAGS is measured via the simulation, despite the improvement being more pronounced in the analytical comparison. TAPTF's improvement in slowdown is also evident and consistent with the analytical trends found in Chapter 4.

Figures 7.12(c) and (d) illustrate the variance in waiting time and slowdown for this three host scenario. With respect to the variance in waiting time, TAPTF shows a clear and distinct improvement over TAGS over a wide range of α parameters, where $\alpha \geq 0.9$. A commendable improvement in the variance in slowdown measured is demonstrated by TAPTF over TAGS where $\alpha > 1$, despite only recording a relatively modest performance increase in the expected slowdown.

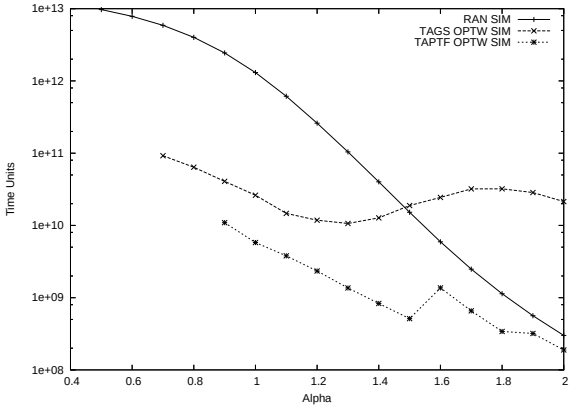
Per host metrics for expected waiting time and slowdown are shown in Figures 7.12(e)



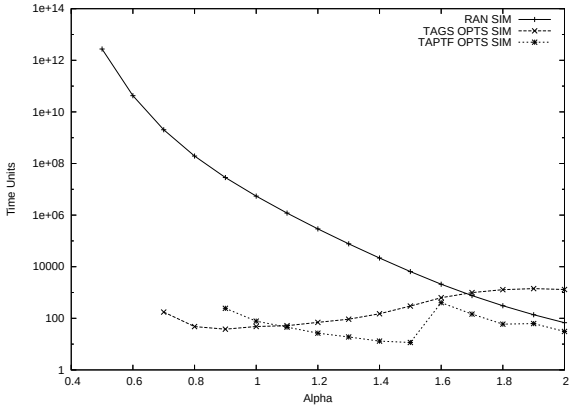
(a) E(W) - $\rho = 0.5$



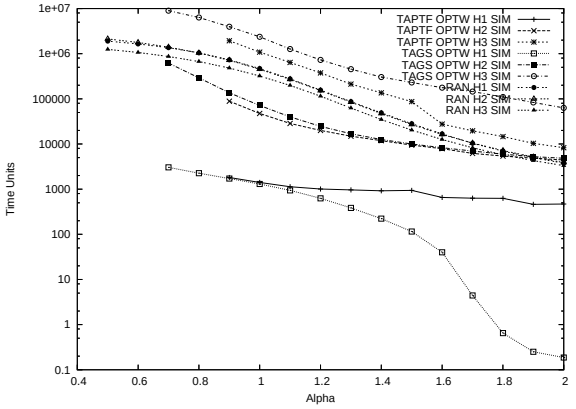
(b) E(S) - $\rho = 0.5$



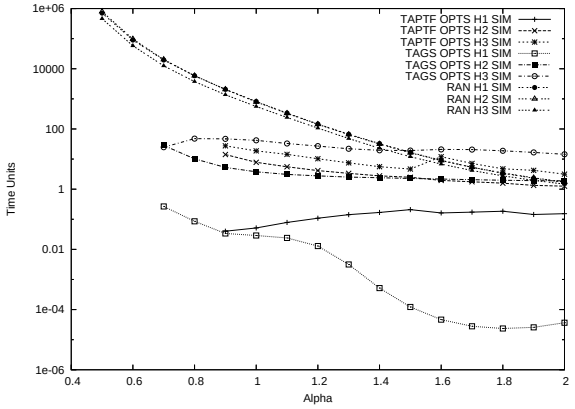
(c) Var(W) - $\rho = 0.5$



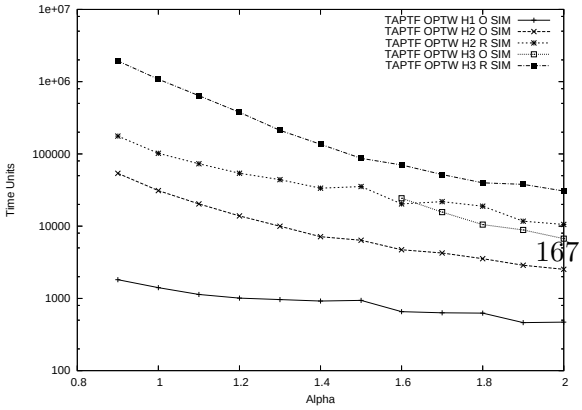
(d) Var(S) - $\rho = 0.5$



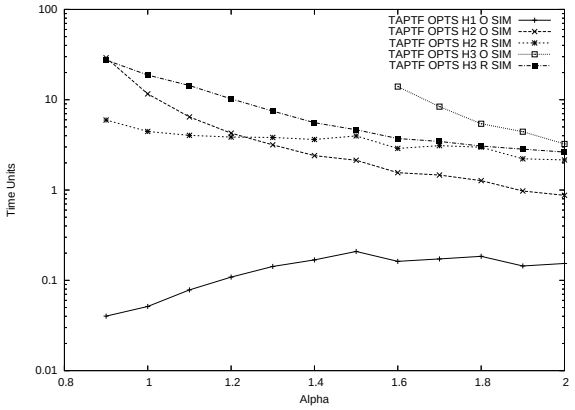
(e) E(W) per host - $\rho = 0.5$



(f) E(S) per host - $\rho = 0.5$

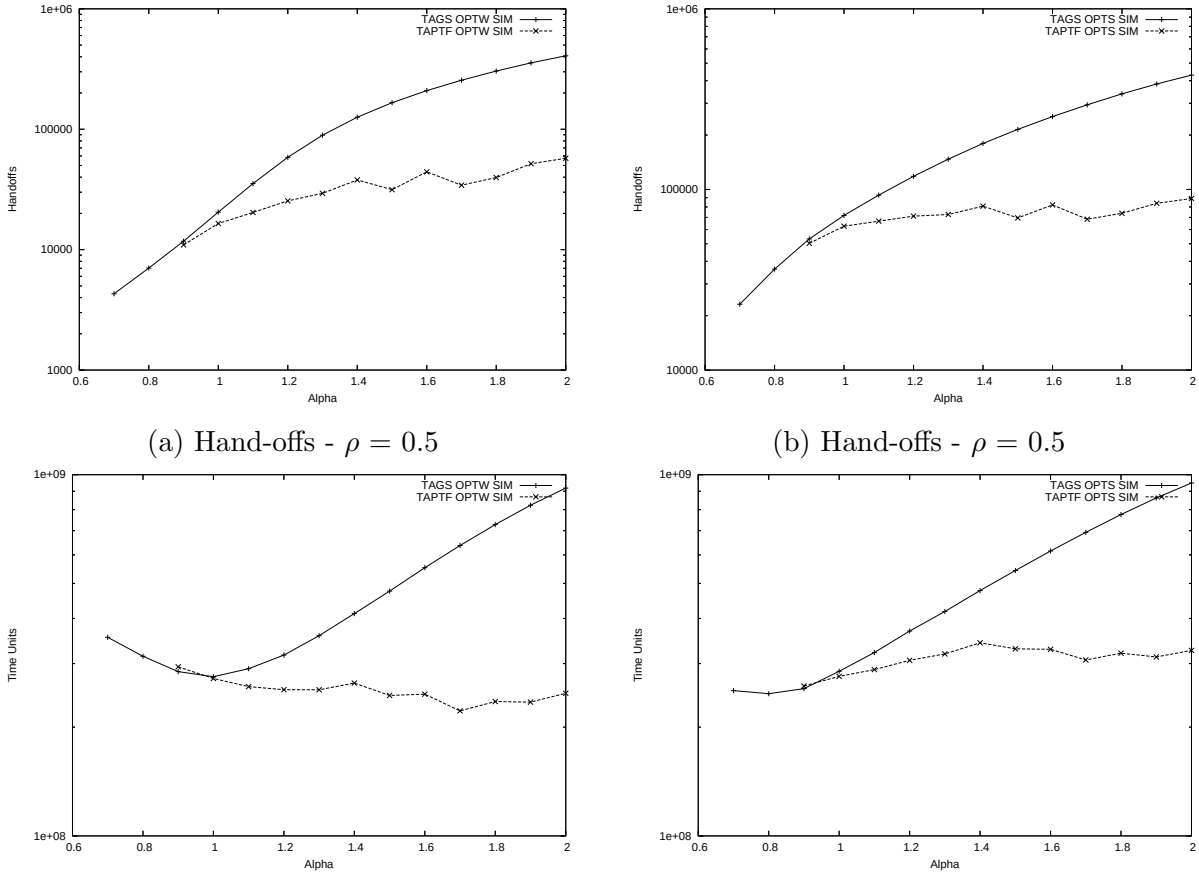


(g) E(W) per queue - $\rho = 0.5$



(h) E(S) per queue - $\rho = 0.5$

Figure 7.12: Performance of a three host distributed system with system load of 0.5



(a) Hand-offs - $\rho = 0.5$

(b) Hand-offs - $\rho = 0.5$

(c) Waste - $\rho = 0.5$

(d) Waste - $\rho = 0.5$

Figure 7.13: The number of hand-offs for systems optimised for waiting time or slowdown are shown in (a) and (b) respectively. The corresponding waste is shown in (c) and (d).

and (f) respectively. As has been the case in all previous observations of per host metrics, TAGS has the best and worst expected waiting times (for Host's 1 and 3 respectively). The expected slowdown for TAGS at the second host is more favourable, especially as the variation increases (and α decreases). Also at the second host, TAPTF exhibits improved expected slowdown especially as variance decreases. As more and more tasks are assigned to the second TAPTF host, the overall expected slowdown improves, as illustrated by Figure 7.12(b). As expected, the mean waiting time and slowdown for the Random policy at each host are almost identical.

There are not too many surprises in the per queue metrics depicted in Figure 7.12(g) and (h). The expected waiting time increases as we progress through the queues at each host, up

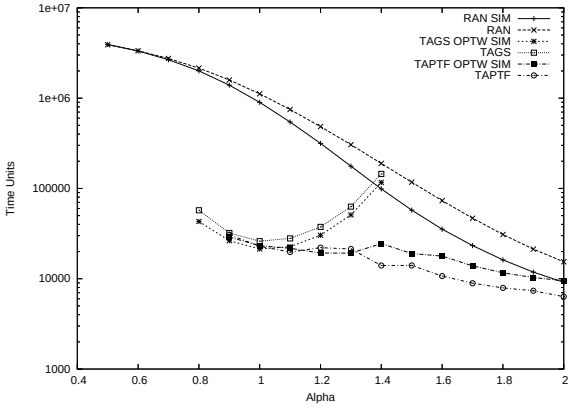
until the final host. For performance reasons, the Ordinary queue at Host 3 does not have any tasks assigned to it until $\alpha = 1.6$ as it is too risky due to the variability in the task size distribution. For mean slowdown, the Ordinary queue at the first host has the lowest slowdown (due to its tight range of task sizes that it services). From $\alpha > 1.2$ we observe that the Ordinary queue at Host 2 has the next best slowdown, followed by the Restart queue at the same host. Following those queues, the Restart queue at Host 3 has larger slowdown still, with the Ordinary queue having the worst slowdown when used.

From Figures 7.13(a) and (b) we can see a significant increase for TAGS in the number of hand-offs (for both waiting time and slowdown optimised cases). This increase begins from where the workload is still extremely variable ($\alpha = 0.9$) and expands as α approaches 2. The amount of wasted processing generated by TAGS has increased also, as highlighted by Figures 7.13(c) and (d). Corresponding with the hand-offs, this increase in waste starts from a level of high variation ($\alpha = 0.9$), and continues to expand as the task size variation decreases. TAPTF remains relatively steady with respect to the number of hand-offs, and the corresponding waste generated, over the same area.

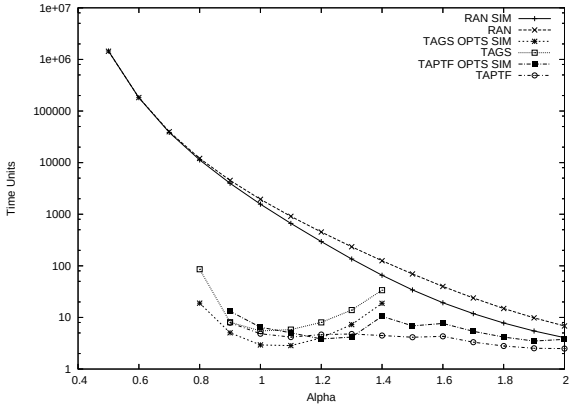
Figure 7.14(a) and (b) depict a three host scenario under a high system load of 0.7. This scenario is of particular interest to us, as it represents the point where the TAGS policy reaches its limits and cannot adequately service the incoming workload in many instances. From Figure 7.14(a) and (b) we can see that the simulation and analytical results follow a consistent performance trend. We can see for both the expected waiting time and slowdown, when α is less than 0.8, and when α is greater than 1.4, there is no results shown for TAGS. This is simply because at those regions, no cutoff parameters exist that can keep the load at each host below 1.0. It is not feasible to run a system at continuous overload, as the queue lengths will increase unbounded. We can observe from the figures than TAPTF maintains a relatively steady expected waiting time and slowdown from $0.9 \leq \alpha \leq 2.0$.

Figure 7.14(c) and (d) show the variance in waiting time and slowdown for this high load scenario. The variance exhibited by TAPTF and TAGS is largely consistent with the corresponding expected waiting time and slowdown depicted in Figure 7.14(a) and (b), as are the areas where TAGS cannot operate due to the reasons outlined above. TAPTF can operate successfully in areas where TAGS cannot, whilst still maintaining acceptable variance in waiting time and slowdown, clearly besting TAGS and the base-line Random policy.

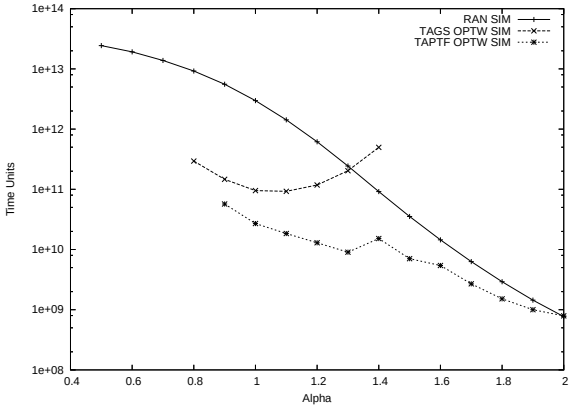
The per host metrics are shown in Figures 7.14(e) and (f) for the expected waiting time and slowdown respectively. TAPTF maintains good mean waiting time at the first and second host, where the vast majority of tasks are serviced (and run-to-completion). TAPTF



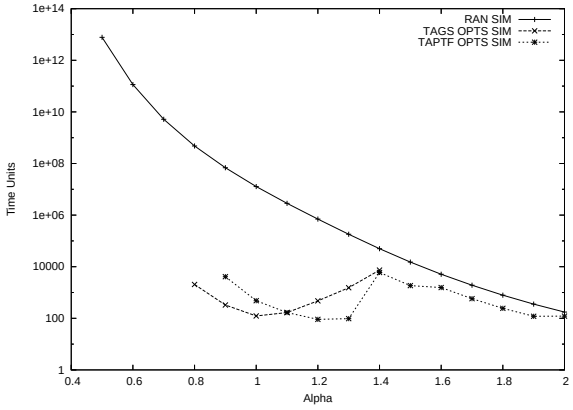
(a) E(W) - $\rho = 0.7$



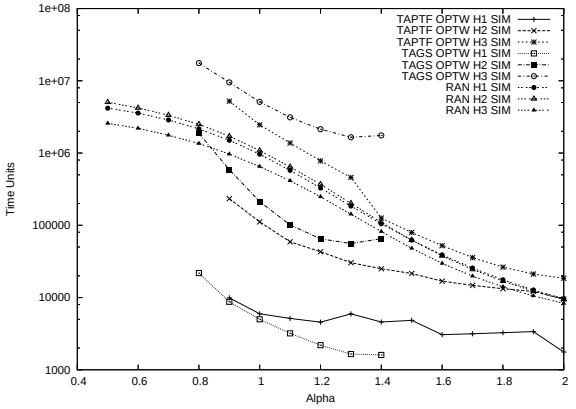
(b) E(S) - $\rho = 0.7$



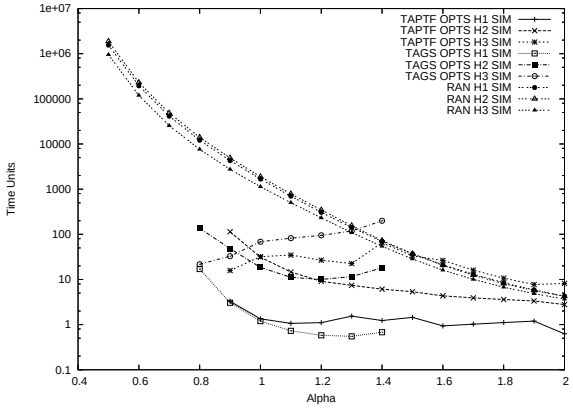
(a) Var(W) - $\rho = 0.7$



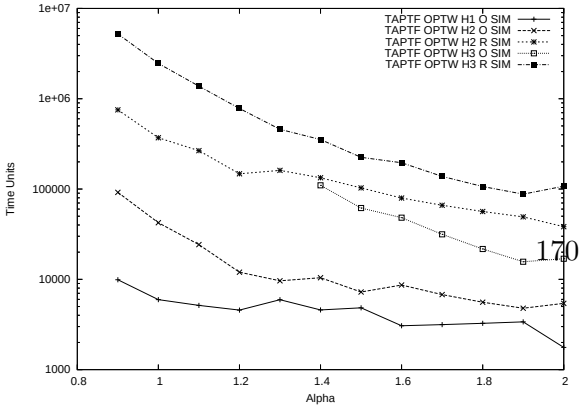
(b) Var(S) - $\rho = 0.7$



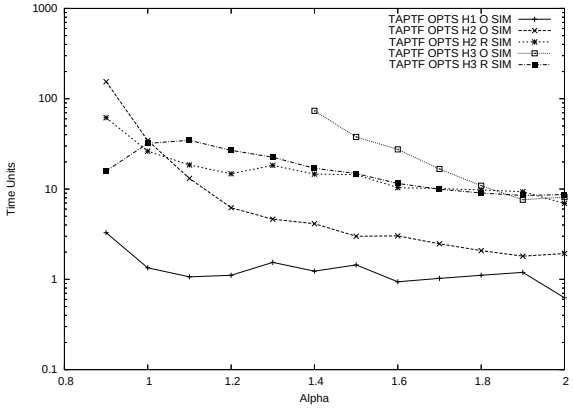
(c) E(W) per host - $\rho = 0.7$



(d) E(S) per host - $\rho = 0.7$

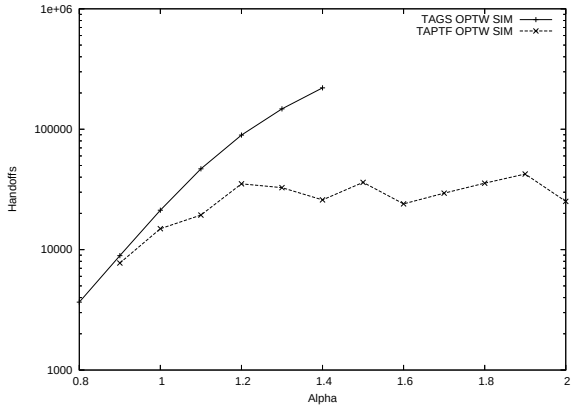


(e) E(W) per queue - $\rho = 0.7$

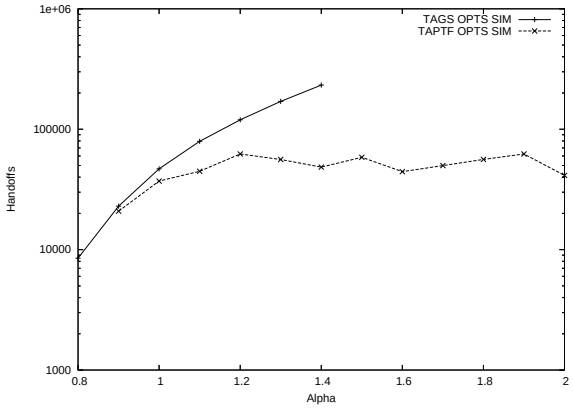


(f) E(S) per queue - $\rho = 0.7$

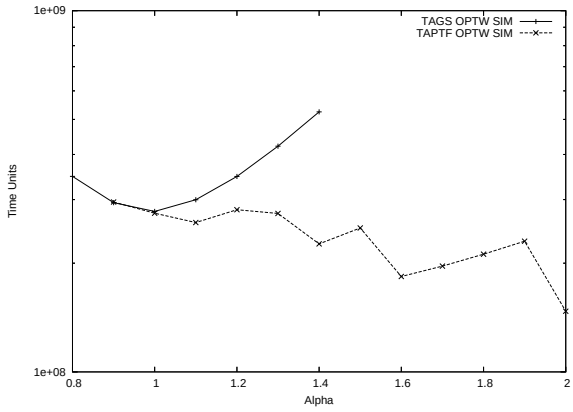
Figure 7.14: Performance of a three host distributed system with system load of 0.7



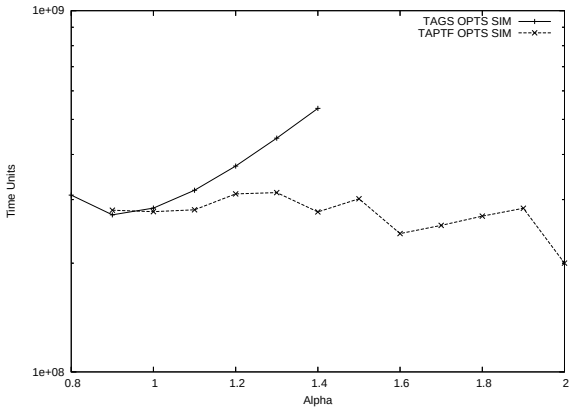
(a) Hand-offs - $\rho = 0.7$



(b) Hand-offs - $\rho = 0.7$



(c) Waste - $\rho = 0.7$



(d) Waste - $\rho = 0.7$

Figure 7.15: The number of hand-offs for systems optimised for waiting time or slowdown are shown in (a) and (b) respectively. The corresponding waste is shown in (c) and (d).

has consistently better expected waiting time at both Host 2 and Host 3 than TAGS does (at the same hosts). The same can also be said for the expected slowdown, with the measured values at Host 2 and 3 for TAPTF being lower than TAGS the vast majority of the time. This would have a significant material effect on the overall performance metrics.

Figures 7.14(g) and (h) show the per queue simulation results for the TAPTF policy. The waiting time results are largely consistent with what was described for three host scenarios under loads of 0.3 and 0.5. The per queue slowdown results are also mostly consistent with prior observations. The only change of note is the Restart queues at Hosts 2 and 3 showing very similar slowdown figures when $\alpha \geq 1.3$.

Figures 7.15(a) and (b) show the number of hand-offs generated by the TAGS and TAPTF

policies for a three host scenario under a high system load of 0.7. As mentioned previously, the TAGS policy was not able to be utilised when $\alpha > 1.4$, as it could not keep the load at each host below 1. However we can see a rapid increase in the number of hand-offs for both cases (where optimised for waiting time or slowdown) in the areas TAGS is able to operate in. On the other hand, TAPTF maintains a relatively steady number of hand-offs in these cases. The increased number of hand-offs that TAGS generates naturally results in an increased amount of wasted processing, as highlighted by Figures 7.15(c) and (d). Conversely, TAPTF actually has a slight drop in wasted processing as the task size variation decreases.

7.4 Discussion

In this section we discuss the findings presented in Section 7.3 - where a range of two host (Section 7.3.1) and three host (Section 7.3.2) simulation scenarios were investigated.

When considering the mean metrics in all cases for the TAPTF, TAGS and Random policies, the analytical results from Chapter 4 were contrasted against the numerical results obtained from our simulation platform. As discussed in Section 7.2, simulating queueing systems with highly variable workloads (coupled with large events that are 'rare' but crucial to consider) is very difficult. As such, while we did not expect our simulation results to match perfectly, we expected similar performance trends to be present over the ranges of task size variation and system loads tested.

This was certainly the case, and the simulation results assist us in verifying the correctness of the original TAPTF model presented in Chapter 4.

We were also able to measure the variance in waiting time and slowdown for the TAPTF, TAGS and Random queueing systems simulated previously. In the case of TAPTF, we have been unable to compute the variance in waiting time or slowdown due to the difficulty in obtaining transforms for the Bounded Pareto service distribution, coupled with the use of priority queues, making it a very challenging problem. Through simulation we were able to confirm our original hypothesis regarding the design of the TAPTF task assignment policy - that it would be effective in reducing the variance experienced by customers. It achieves this via the use of dual queues, grouping like-sized tasks together. We consistently observed significant reductions by TAPTF over TAGS and Random in the variance in waiting time, and in certain areas of observation, slowdown.

We also measured the number of hand-offs (and the subsequent waste) generated by the TAPTF and TAGS models. This became especially critical as the system load increases,

having a significantly detrimental effect on the performance as the amount of wasted processing increases. If we wished to model application domains where tasks have an initial start-up cost (upon a task starting or restarting service, which is not uncommon) we suspect this problem would be amplified for TAGS (and to a much lesser extend, TAPTF).

The trends observed from the analytical results relating to the performance as the system load increases were confirmed via the simulation presented in this chapter. From the simulation results, we observed that the improvement that TAPTF provides increases in magnitude and breadth (i.e over a wider range of workload variations) as the system load increases. Indeed, when we considered a three host case under a system load of 0.7, we found that TAGS could not operate under certain workloads. TAPTF had no problem handling such workloads, and (whilst not depicted here) TAPTF can perform under system loads of 0.8 and 0.9. This has been verified both analytically and via simulation. TAPTF spreads the load over multiple hosts, and as it does not have single bottleneck (i.e Host 1) it can handle these higher loads easily.

7.5 Conclusion

The OMNeT++ simulation framework provided us with platform to generate a wide range of queueing metrics, many of which we have been unable to compute analytically at this point.

We were able to verify the analytical model found in Chapter 4 via simulation, as well as measure new metrics including the variance of the waiting time and slowdown. We also measured per host and per queue metrics, as well as the number of hand-offs and wasted processing generated by TAGS and TAPTF. This ultimately gives us a greater insight into the operation of the TAPTF policy, providing further understanding regarding how it provides improved performance under specific areas and scenarios. This insight can be used to optimise the TAPTF model further and pursue new methods of task assignment.

Chapter 8

Discussion

In this thesis we have investigated some key issues relating to the problem of effective task assignment strategies for distributed systems under highly variable workloads. We were particularly motivated by the specific problems associated with modern computing workloads. Most past research in task assignment (or scheduling) has been focused on less variable, exponential workloads. Significant recent research has revealed than modern computing workloads are highly variable, and are distributions that characterise them are 'heavy-tailed'. In light of these findings we focused our efforts on the modelling and improvement of task assignment policies under these highly variable workloads.

Our first two research questions were focused on devising more effective task assignment policies under two specific application domains - batch computing, and web serving clusters. Both policies (TAPTF and TAPTF-WC) endeavoured to maximise the performance and utilisation of a distributed system in each respective application domain.

We then considered techniques to simplify the modelling and analysis of queueing systems that incorporate General distributions by approximating such distributions as Hyper-exponential. We also re-computed our TAPTF model to utilise a Hyper-exponential (or Bounded Hyper-exponential) service distribution. This allows the TAPTF model to be used with nearly any General service distribution (e.g. Pareto, Bounded Pareto, Log-normal, etc.), simply by first approximating it as Hyper-exponential or Bounded Hyper-exponential.

Finally, we performed a rigorous simulation of key task assignment policies, measuring a wide range of metrics. A simulation framework to record important queueing metrics was presented, allowing us to measure many metrics that were not possible to compute via analytical means. Several issues regarding simulating highly variable workloads were

identified, and the variance of key metrics such as expected waiting time and slowdown were obtained.

Specifically, the following research questions have been addressed in this thesis:

A) How can we improve task assignment policies for batch computing environments?

B) How can we improve task assignment policies for interactive computing environments?

C) How can we simplify the analysis of distributed systems under highly-variable workloads?

D) How can we obtain advanced performance metrics of task assignment policies?

8.1 Contribution

In response to the research questions mentioned above and originally posed in Section 1.1, the following contributions were made:

8.1.1 Task assignment based on prioritising traffic flows

Task Assignment based on Prioritising Traffic Flows (TAPTF) is a size-based partitioning approach specifically designed to address the problems caused by highly variable workloads in batch and scientific computing systems. TAPTF assumes no prior knowledge of the size of a task. TAPTF address many of the performance issues that existing techniques suffer from, by its highly flexible operational parameters. TAPTF uses dual queues at each host, and groups similar-sized tasks together in an effort to maximise performance and reduce the variance of tasks in a given queue. Tasks that unduly delay other tasks are migrated to the next host in the system and restarted from scratch, ensuring smaller tasks behind it in the queue are not disproportionally delayed. TAPTF also improves utilisation by spreading incoming tasks over more hosts as the workload variation decreases. The TAPTF policy reduces the number of 'hand-offs' as compared to the TAGS policy, and consequently reduces the amount of wasted processing.

8.1.2 Task assignment with work conserving migration

Application domains such as high volume web serving clusters have different operational requirements to that of batch and scientific domains. Specifically, it is not acceptable to migrate a task (e.g. a lengthy web request) and restart it from scratch at another host. Consequently, we introduce the Task Assignment with Work-Conserving Migration (TAPTF-WC) policy, that is specifically designed for this application domain. TAPTF-WC has all the benefits provided by the TAPTF policy, allowing it to mitigate the negative effects of highly variable workloads, whilst allowing for work-conserving migration to occur. This is crucial for this application domain as, like batch and scientific computing domains, the workloads experienced by web servers can be highly variable in nature. As TAPTF-WC is specifically designed to handle these workloads, it exhibits good performance under a wide variety of workload and system load conditions. In addition, we also gained insight by the effects of application domains where the act of migration incurs a fixed or proportional cost placed on either (or both) the source or destination node.

8.1.3 Approximating General service Distributions

The technique utilised to approximate a general distribution (like Pareto or Log-normal) as a sum of exponential distributions (known as a Hyper-exponential) is called Prony's method [Feldmann and Whitt, 1997]. Exponential distributions are especially useful in queueing theory analysis due to their tractability, with their transforms and higher moments trivial to compute. The Bounded Pareto distribution is commonly utilised in queueing analysis of task assignment policies, as they accurately characterise the highly variable workloads that have been measured in many computing environments. Unfortunately, an unbounded Hyper-Exponential approximation of a Bounded Pareto distribution is not going to be suitable. Evidently, we find that a Bounded Hyper-exponential approximation is much better suited when the original distribution itself is bounded. This approach results in a significantly more accurate approximation overall, with certain statistical properties of interest matching perfectly. The accuracy of this technique is verified by integrating the Hyper-exponential and Bounded Hyper-exponential distributions into our original TAPTF model, and comparing queueing metrics. As a useful consequence of this process, the TAPTF model can now be utilised with nearly any general distribution that can be effectively approximated by a Hyper-exponential or Bounded Hyper-exponential distribution. This allows the TAPTF policy to be utilised with a wider range of potential workloads.

8.1.4 Obtaining advanced performance metrics via simulation

At this point, there are still a number of important metrics for our TAPTF model than cannot be computed analytically. Fortunately, we can compute a wide spectrum of metrics via simulation of our proposed task assignment policy, giving us further insight into its behaviour under certain workload conditions. Extensive simulation was performed, comparing one of our proposed approaches, TAPTF, against existing task assignment policies. The simulations provide two valuable purposes. First, we wish to compare the simulations against analytical models of TAPTF and other existing approaches as a means to verify their integrity. Second, we wish to obtain metrics that have thus far proved too difficult to obtain analytically. Specifically, we are interested in the variance of the waiting time and slowdown, for the whole system, for each host and for each queue. Such metrics (among others) provide us with a more detailed picture of the behaviour of tasks in the system, and can ultimately provide us with approximate bounds on performance. This becomes crucial as we ultimately endeavour to provide end-users with specific Quality of Service bounds that can be guaranteed. Consequently, these simulations will provide further insight into the problem of task assignment under highly variable workloads.

8.2 Future Work

There is still critical future work that needs to be achieved, in examining the techniques that assist us in the area of task assignment for distributed systems. Unfortunately there are still many aspects of this area that are difficult to model analytically, making us reliant on modelling via simulation. Simulation models are not without their problems, especially under highly variable workload conditions, as highlighted by Chapter 7. In this section we highlight some of these key areas for future exploration.

In Chapters 4 and 5 we introduced two new task assignment policies, Task Assignment based on Prioritising Traffic Flows (TAPTF) and Task Assignment with Work Conserving Migration (TAPTF-WC). Both policies were supported by a rigorous analytical model based on the fundamentals of queueing theory. Mean metrics for waiting time and slowdown could be obtained trivially thanks to our model. However, characterising the variance in the metrics is a significantly more difficult proposition. Obtaining the higher moments of the expected waiting and slowdown (needed to compute the variance) is difficult due to the challenging nature of computing the Laplace transform of the waiting time and slowdown distributions. Understanding the variance experienced by tasks in our system gives us a greater under-

standing of the behaviour of the majority of tasks, as often mean metrics can be misleading (especially under highly variable workloads). For this very reason we obtained numerical measurements of the variance via simulation, under a variety of scenarios, for both waiting time and slowdown in Chapter 7.

Having said that, computing the variance analytically is a challenging task, but perhaps not impossible. There is a known result (the Laplace transform for waiting time, L_W) that can be used to compute the second moment of waiting time in a $M/G/1$ queue (such as the Ordinary queue in TAPTF and TAPTF-WC) [Kleinrock, 1975b]. There is another result [Kleinrock, 1975b] that describes the Laplace transform of the busy period for a priority queue. This can be used to solve for the Laplace transform of the waiting time for a single server $M/G/1$ priority queue system. However, the question of whether we can successfully adapt this result to suit the modelling of Restart queues in TAPTF and TAPTF-WC appears unclear at this point, for many reasons. A number of analytical challenges exist due to the complexity of the TAPTF and TAPTF-WC models. Not least of these are:

- The difficulty in obtaining transforms of the service distribution (when it is Pareto or Bounded Pareto)
- The size-based partitioning that occurs in both approaches (complicating the service distribution further)
- The non work-conserving nature of TAPTF
- The work-conserving migration that occurs in TAPTF-WC

The techniques shown in Chapter 6 can potentially address (among other things) the first problem, as it is trivial to compute the Laplace transform of a Hyper-exponential or Bounded Hyper-exponential distribution. The other issues will require further consideration. Furthermore, computing the Laplace transform for the slowdown in either TAPTF or TAPTF-WC may be an even more difficult proposition. Nonetheless, obtaining such results for the TAPTF and TAPTF-WC policies would be a significant achievement.

TAPTF and TAPTF-WC are also largely dependent on the respective parameters that governs their operation. In the case of two hosts, we can compute the optimal model parameters (q_1, q_2, s_1), ensuring they are set for the best possible waiting time or slowdown. When dealing with three hosts, we can compute the optimal cutoffs (s_1, s_2), but we must tune the influx of tasks (q_1, q_2, q_3) by hand. As we consider larger systems (i.e. more than

3 hosts) it becomes increasingly difficult to compute optimal parameters. We could likely solve slightly larger problems (e.g. 3, 4 or 5 host) with a more powerful solver package, but that is just an interim solution. Ideally we would like some closed form equations to compute the parameters. These could essentially be based on a large amount of experimentation, and would not be optimal parameters, rather being approximate rules-of-thumb.

Another useful extension would be to consider the situation where the workload changes over time. Currently we solve for fixed workloads, finding optimal or near optimal parameters for a given workload (characterised by a task size distribution and an arrival rate). An 'online' TAPTF or TAPTF-WC policy would be extremely useful, where a record of the workload is kept at the dispatcher, and the model parameters are re-computed as needed. Care would be needed to strike an appropriate balance between frequency of updated and the overhead of re-computing the parameters [Dahlin, 2000; Mitzenmacher, 2000].

Such an automated system could operate in a manner similiar to the EQUILOAD [Ciardo et al., 2001] and ADAPTLOAD [Riska et al., 2002c] web dispatching systems. However, given the assumption of no *a priori* knowledge of the service requirement of incoming tasks, back-end servers would need to report run-times of tasks to the dispatcher once they have run to completion. A dispatcher could maintain a histogram representing a rolling snapshot of the recent workload and choose appropriate size ranges for TAPTF and TAPTF-WC respectively. However, as mentioned above, this would depend on having convenient closed-form solutions or approximations to efficiently compute near-optimal size ranges for arbritrarily large clusters. It would be valuable to integrate such an automated system into web (e.g. Apache [The Apache Software Foundation, 2007]) and batch computing dispatchers (e.g. Portable Batch System [Henderson, 1995], Load Sharing Facility [Platform Computing, 2007]) to improve performance under highly variable workloads.

In Chapter 6 we considered techniques for fitting General distributions to Hyper-exponential or Bounded Hyper-exponential distributions. The resulting Hyper-exponential representations have countless applications in queueing theory. Based on some preliminary experimentation we have since observed that significant improvements in the quality of fit could be achieved by utilising different techniques in choosing the matching points. Specifically, improvements could be made in the critical areas of fit around the base and the tail of the distribution. An algorithm that chooses good matching points depending on the General distribution (e.g. Pareto, Bounded Pareto, Log-normal, Weibull) that is to be approximated would be extremely valuable. By choosing more appropriate matching points we can hopefully improve the quality of fit for a wide range of General distributions.

Chapter 7 presented a wide range of queueing metrics that were measured via simulation. The highly variable nature of the workloads we are interested in makes obtaining good results via simulation challenging. Performing longer running simulations, with a larger sample size may provide results that are closer to the analytical results previously observed, and would allow confidence intervals to be computed, giving us tighter bounds on important metrics such as expected waiting time and slowdown. However increasing the simulation sample size further would require significantly larger resources, and would require the simulation platform to be distributed over many machines. Alternatively, other approaches to speeding up simulation could potentially be adapted to suit our needs [Heidelberger, 1995].

8.3 Conclusion

In this thesis many important advances were made in the understanding, modelling and performance of task assignment strategies for distributed systems. Each contribution was made with close consideration to the difficulties caused by modern, highly variable computing workloads.

A task assignment strategy for batch and super-computing clusters was proposed, called Task Assignment based on Prioritising Traffic Flows (TAPTF). TAPTF exploited the characteristics of 'heavy-tailed' workloads, providing a flexible and high performing task assignment policy, and reduced some of the processing overhead that was problematic with an existing technique, TAGS. TAPTF was particularly effective under conditions of high system load, spreading the load over multiple servers and ensuring the first host does not become a bottleneck.

Task Assignment with Work Conserving Migration (TAPTF-WC) was a task assignment strategy proposed for use in high volume web serving clusters. TAPTF-WC works on similar principles to TAPTF, but most importantly allows work-conserving migration to occur, which is consistent with the operational capabilities of many web cluster architectures. TAPTF has demonstrated good performance over a wide range of workloads and system load scenarios.

Both the TAPTF and TAPTF-WC policies are supported by a rigorous analytical models based on fundamental results in queueing theory.

In an effort to approximate General distributions as Bounded Hyper-exponential distributions, an modified 'Prony's Method' was presented. This modified technique is specifically suited to approximating Bounded distributions, showing significantly increased accuracy over the standard approach prescribed by Prony et al. The TAPTF model was re-computed to

incorporate a Bounded Hyper-exponential service distribution. As a number of different General distributions can be approximated in this fashion, the utility of TAPTF has been significantly increased as it now can handle a wider range of potential workload distributions. The Bounded Hyper-exponential distribution also has other applications, such as modelling highly variable arrival patterns (e.g. $G/G/1$ or $G/M/1$).

Through simulation, additional metrics and bounds were measured for TAPTF and other task assignment policies. Simulation allowed us to measure global, per host and per queue metrics, as well as the variance and bounds (via majority bars), providing greater insight into the operation of the TAPTF policy. Many of these metrics cannot be computed analytically.

There is still significant future work that can be undertaken in this area of research, as outlined in Section 8.2. As we become more dependent on the services that computers provide, the need to effectively service large volumes of customers with highly variable demands will remain. The demand for highly popular services are increasingly unlikely to be satisfied by a single monolithic server. As a consequence, the need for effective task assignment strategies for distributed systems that satisfy a wide range of workloads will only become more critical.

Bibliography

K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-451-0.

S. K. Adhya, S. Das-Purkayastha, and S. Ganguly. Asymmetric splice: optimizing TCP forwarder performance for the HTTP/1.1 protocol. In *ICCC '02: Proceedings of the 15th international conference on Computer communication*, pages 239–251, Washington, DC, USA, 2002. International Council for Computer Communication. ISBN 1-891365-08-8.

Amazon.com, Inc. Amazon Elastic Compute Cloud (Amazon EC2), 2007. Available at http://aws.amazon.com/ec2.

M. Arlitt and T. Jin. Workload characterization of the 1998 World Cup web site. *IEEE Network*, 14:30–37, May/Jun 2000.

M. F. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Transactions on Internet Technology*, 1(1):44–69, 2001. ISSN 1533-5399.

L. Aversa and A. Bestavros. Load balancing a cluster of web servers using distributed packet rewriting. In *Proceedingss of the 19th IEEE International Performance, Computing and Communications Conference (IPCCC)*, pages 30–37, February 2000.

N. Bansal and M. Harchol-Balter. Analysis of SRPT scheduling: investigating unfairness. In *SIGMETRICS '01: Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 279–290, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-334-0.

P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 151–160, New York, NY, USA, 1998. ACM Press. ISBN 0-89791-982-3.

P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-757-5.

A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *Symposium on Networked Systems Design and Implementation*, Mar 2004.

B. L. Bill Devlin, Jim Gray and G. Spix. Scalability terminology, farms, clones, partitions, and packs, racs and raps. Technical Report MS-TR-99-85, Microsoft Research, 1999. URL http://www.microsoft.com.

J. Broberg, Z. Tari, and P. Zeephongsekul. Task assignment based on prioritising traffic flows. In T. Higashino, editor, *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS 2004)*. Springer-Verlag, 2005. ISBN 3-540-27324-7.

J. Broberg, Z. Tari, and P. Zeephongsekul. Task assignment with work-conserving migration. *Journal of Parallel Computing (Special Issue on Performance Evaluation of Networks for Parallel, Cluster and Grid Computing Systems)*, 2006. To appear.

P. J. Brockwell and R. A. Davis. *Time series: theory and methods*. Springer-Verlag New York, Inc., New York, NY, USA, 1986. ISBN 0-387-96406-1.

L.-F. Cabrera. The influence of workload on load balancing strategies. In *Proceedings of the Winter USENIX Conference*, pages 446–458, 1986.

L.-F. Cabrera. The influence of workload on load balancing strategies. *Mobility: processes, computers, and agents*, pages 214–227, 1999.

J. Cao, W. Cleveland, D. Lin, and D. Sun. Internet traffic tends toward Poisson and independent as the load increases. *Nonlinear Estimation and Classification*, Dec 2002.

V. Cardellini, M. Colajanni, and P. S. Yu. Efficient state estimators for load control policies in scalable web server clusters. In *COMPSAC '98: Proceedings of the 22nd International Computer Software and Applications Conference*, pages 449–457, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8585-9.

V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39, 1999. ISSN 1089-7801.

V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):263–311, 2002. ISSN 0360-0300.

V. Cardellini, M. Colajanni, and P. S. Yu. Request redirection algorithms for distributed web systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):355–368, 2003. ISSN 1045-9219.

T. Casavant and J. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, 1988. ISSN 0098-5589.

G. Ciardo, A. Riska, and E. Smirni. Equiload: a load balancing policy for clustered web servers. *Performance Evaluation*, 46(2-3):101–124, 2001. ISSN 0166-5316.

Cisco Systems. Scaling the internet web servers, 1997. URL http://www.cisco.com/warp/public/cc/pd/cxsr/400/tech/scale_wp.pdf.

A. Cobham. Priority assignment in waiting line problems. *Journal of the Operations Research Society*, 2:70–76, 1953.

A. Cohen, S. Rangarajan, and J. H. Slye. On the performance of TCP splicing for URL-aware redirection. In *USENIX Symposium on Internet Technologies and Systems*, 1999. URL citeseer.ist.psu.edu/cohen99performance.html.

M. Colajanni, V. Cardellini, and P. S. Yu. Dynamic load balancing in geographically distributed heterogeneous web servers. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 295, Washington, DC, USA, 1998a. IEEE Computer Society. ISBN 0-8186-8292-2.

M. Colajanni, P. S. Yu, and D. M. Dias. Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(6):585–600, 1998b. ISSN 1045-9219.

R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Courier Dover Publications, 1967. ISBN 0486428176.

M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997. ISSN 1063-6692.

M. E. Crovella, M. Harchol-Balter, and C. D. Murta. Task assignment in a distributed system: Improving performance by unbalancing load. In S. Leutenegger, editor, *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, New York, NY, USA, 1998a. ACM Press. ISBN 0-89791-982-3.

M. E. Crovella, M. S. Taqqu, and A. Bestavros. *Heavy-tailed probability distributions in the World Wide Web*. Birkhauser Boston Inc., Cambridge, MA, USA, 1998b. ISBN 0-8176-3951-9.

D. Ferrari. A study of load indices for load balancing schemes. Technical Report CSD-85-262, UC Berkeley, 1985.

M. Dahlin. Interpreting stale load information. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1033–1047, 2000. ISSN 1045-9219.

J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing*, pages 50–58, September/October 2002.

L. Doolittle and J. Nelson. Boa webserver, 1991. Available at http://www.boa.org.

A. B. Downey. Lognormal and Pareto distributions in the internet. *Computer Communications*, 28(7):790–801, May 2005.

A. B. Downey. Evidence for long-tailed distributions in the internet. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 229–241, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-435-5.

D. L. Eager, E. D. Lazowska, and J. Zahorjan. The limited performance benefits of migrating active processes for load sharing. In *SIGMETRICS '88: Proceedings of the 1988 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 63–72, New York, NY, USA, 1988. ACM Press. ISBN 0-89791-254-3.

A. Feldmann and W. Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. In *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, pages 1096–1104, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7780-5.

D. Ferrari and S. Zhou. A load index for dynamic load balancing. In *ACM '86: Proceedings of 1986 ACM Fall joint computer conference*, pages 684–690, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press. ISBN 0-8186-4743-4.

D. Ferrari and S. Zhou. An empirical investigation of load indices for load balancing applications. In *Performance '87: Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*, pages 515–528. North-Holland, 1988. ISBN 0-444-70347-0.

D. Gross and C. M. Harris. *Fundamemtals of Queueing Theory*. Wiley-Interscience, 1998.

M. Harchol-Balter. Task assignment with unknown duration. *Journal of the ACM*, 49(2): 260–288, 2002. ISSN 0004-5411.

M. Harchol-Balter. The effect of heavy-tailed job size distributions on computer system design. In *Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, June 1999.

M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, 1997. ISSN 0734-2071.

M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2): 204–228, 1999.

M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. S. Squillante. Analysis of task assignment with cycle stealing under central queue. *IEEE International Conference on Distributed Computer Systems*, 00:628, 2003a. ISSN 1063-6927.

M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. S. Squillante. Cycle stealing under immediate dispatch task assignment. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 274–285, New York, NY, USA, 2003b. ACM Press. ISBN 1-58113-661-7.

M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems*, 21(2):207–233, 2003c. ISSN 0734-2071.

L. He, S. A. Jarvis, D. P. Spooner, and G. R. Nudd. Optimising static workload allocation in multiclusters. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'04)*, pages 39–43, Los Alamitos, CA, USA, 2004. IEEE Computer Society. ISBN 0-7695-2132-0.

P. Heidelberger. Fast simulation of rare events in queueing and reliability models. *ACM Transactions on Modelling and Compututer Simulations*, 5(1):43–85, 1995. ISSN 1049-3301.

R. L. Henderson. Job scheduling under the portable batch system. In *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 279–294, London, UK, 1995. Springer-Verlag. ISBN 3-540-60153-8.

G. Irlam. Unix file survey, 1993. Available at http://www.base.com/gordoni/ufs93.html.

A. K. Iyengar, M. S. Squillante, and L. Zhang. Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, 2(1-2):85–100, 1999. ISSN 1386-145X.

K. Kant, R. Iyer, and P. Mohapatra. Architectural impact of secure socket layer on internet servers. In *ICCD '00: Proceedings of the 2000 IEEE International Conference on Computer Design*, page 7, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0801-4.

D. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the embedded markov chain. *Annals of Mathematical Statistics*, pages 338–354, 1953.

L. Kleinrock. *Queueing Systems Volume 1: Theory.* John Wiley and Sons., 1975a.

L. Kleinrock. *Queueing Systems Volume 2: Computer Applications.* John Wiley and Sons., 1975b.

L. Kleinrock. Time-shared systems: a theoretical treatment. *Journal of the ACM*, 14(2): 242–261, 1967. ISSN 0004-5411.

T. Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering*, 17(7):725–730, 1991. ISSN 0098-5589.

D. A. Maltz and P. Bhagwat. TCP splicing for application layer proxy performance. Research Report RC 21139, IBM, Mar. 1998.

S. L. Marple. *Digital spectral analysis: with applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986. ISBN 0-132-14149-3.

M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensional equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

D. A. Menasce, V. A. F. Almeida, R. Fonseca, and M. A. Mendes. A methodology for workload characterization of e-commerce sites. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 119–128, New York, NY, USA, 1999. ACM Press. ISBN 1-58113-176-3.

D. S. Milojicic;, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. *ACM Computing Surveys*, 32(3):241–299, 2000. ISSN 0360-0300.

M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001. ISSN 1045-9219.

M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(3):226–251, 2004.

M. Mitzenmacher. How useful is old information? *IEEE Transactions on Parallel and Distributed Systems*, 11(1):6–20, 2000. ISSN 1045-9219.

M. Mitzenmacher. How useful is old information (extended abstract). In *PODC '97: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 83–91, New York, NY, USA, 1997. ACM Press. ISBN 0-89791-952-1.

M. Nelson, B.-H. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Proceedings of the 2005 USENIX Annual Technical Conference*, pages 391–394, 2005.

R. D. Nelson and T. K. Philips. An approximation to the response time for shortest queue routing. In *SIGMETRICS '89: Proceedings of the 1989 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 181–189, New York, NY, USA, 1989. ACM Press. ISBN 0-89791-315-9.

R. D. Nelson and T. K. Philips. An approximation for the mean response time for shortest queue routing with general interarrival and service times. *Performance Evaluation Review*, 17:123–139, 1993.

H. T. M. Neto, J. M. Almeida, L. C. D. Rocha, W. Meira, P. H. C. Guerra, and V. A. F. Almeida. A characterization of broadband user behavior and their e-business activities. *SIGMETRICS Perform. Eval. Rev.*, 32(3):3–13, 2004. ISSN 0163-5999.

M. Nuttall and M. Sloman. Workload characteristics for process migration and load balancing. *ICDCS '97: Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS'97)*, 00:133–140, 1997. ISSN 1063-6927.

T. Osogami and M. Harchol-Balter. Necessary and sufficient conditions for representing general distributions by coxians. *Computer Performance: Modelling Techniques and Tools (LNCS 2794)*, pages 182–199, September 2003.

V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable Web server. In *Proceedings of the USENIX 1999 Annual Technical Conference*, 1999.

Platform Computing. Load sharing facility, 2007. Available at http://www.platform.com.

I. A. Rai, G. Urvoy-Keller, and E. W. Biersack. Analysis of LAS scheduling for job size distributions with high variance. In *SIGMETRICS '03: Proceedings of the 2003 ACM*

SIGMETRICS international conference on Measurement and modeling of computer systems, pages 218–228, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-664-1.

I. A. Rai, G. Urvoy-Keller, M. K. Vernon, and E. W. Biersack. Performance analysis of LAS-based scheduling disciplines in a packet switched network. In *SIGMETRICS '04/Performance '04: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 106–117, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-873-3.

A. Riska, V. Diev, and E. Smirni. Efficient fitting of long-tailed data sets into phase-type distributions. *SIGMETRICS Performance Evaluation Review*, 30(3):6–8, 2002a. ISSN 0163-5999.

A. Riska, V. Diev, and E. Smirni. Efficient fitting of long-tailed data sets into hyperexponential distributions. In *Proceedings of the IEEE Internet Performance Symposium (GLOBECOM)*, volume 3, pages 2513–2517, November 2002b.

A. Riska, W. Sun, E. Smirni, and G. Ciardo. ADAPTLOAD: Effective balancing in custered web servers under transient load conditions. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 104–111, Washington, DC, USA, 2002c. IEEE Computer Society. ISBN 0-7695-1585-1.

S. M. Ross. *Simulation.* Academic Press, 2001.

S. M. Ross. *Introduction to Probability Models.* Academic Press, 2002.

M.-C. Rosu and D. Rosu. An evaluation of TCP splice benefits in web proxy servers. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 13–24, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-449-5.

L. Schrage. A proof of the optimality of the shortest remaining service time discipline. *Operations Research*, 16:670–690, 1968.

B. Schroeder and M. Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. *Cluster Computing*, 7(2):151–161, 2004. ISSN 1386-7857.

B. Schroeder and M. Harchol-Balter. Web servers under overload: How scheduling can help. *ACM Transactions on Internet Technology*, 6(1):20–52, 2006. ISSN 1533-5399.

A. Silberschatz and P. Galvin. *Operating System Concepts (Fifth Edition).* Addison-Wesley Publishing Company, 1998.

D. R. Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26:197–199, 1978.

S. Sozaki and R. Ross. Approximations in finite capacity multiserver queues with Poisson arrivals. *Journal of Applied Probability*, 13:826–834, 1978.

O. Spatscheck, J. S. Hansen, J. H. Hartman, and L. L. Peterson. Optimizing TCP forwarder performance. *IEEE/ACM Transactions on Networking*, 8(2):146–157, 2000. ISSN 1063-6692.

J. Spirn. Queuing networks with random selection for service. *IEEE Transactions on Software Engineering*, 5(3):287–289, 1980. ISSN 0098-5589.

W. Stallings. *Operating System (Second Edition).* Prentice Hall, 1995.

W. Stallings. *High-Speed Networks and Internets: Performance and Quality of Service.* Prentice-Hall, 2002.

A. Tanenbaum. *Modern Operating Systems.* Prentice Hall, 1995.

X. Tang and S. T. Chanson. Optimizing static job scheduling in a network of heterogeneous computers. In *ICPP '00: Proceedings of the 2000 International Conference on Parallel Processing*, page 373, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0768-9.

Z. Tari, J. Broberg, A. Y. Zomaya, and R. Baldoni. A least flow-time first load sharing approach for distributed server farm. *Journal of Parallel and Distributed Computing*, 65 (7):832–842, 2005. ISSN 0743-7315.

The Apache Software Foundation. Apache http server project, 2007. Available at http://www.apache.org.

A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'01)*, June 2001.

C. A. Waldspurger. Memory resource management in vmware esx server. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 181–194, New York, NY, USA, 2002. ACM Press.

A. Ward and W. Whitt. Predicting response times in processor-sharing queues. *Analysis of Communication Networks: Call Centres, Traffic and Performance*, pages 1–29, 2000.

R. W. Weber. On the optimal assignment of customers to parallel servers. *Journal of Applied Probability*, 15:826–834, 1978.

A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 238–249, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-664-1.

A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to higher moments of conditional response time. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 229–240, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-022-1.

W. Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14: 181–189, 1977.

R. W. Wolff. *Stochastic Modeling and the Theory of Queues.* Prentice-Hall, 1989.

Wolfram Research. Mathematica version 5.0, 2003.