Imperial College London
Department of Computing

Queueing network models of Zoned RAID system performance

Abigail Lebrecht

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy and the Diploma of Imperial College in Computing
under Imperial College London, October 2009

Abstract

RAID systems are widely deployed, both as standalone storage solutions and as the building blocks of modern virtualised storage platforms. An accurate model of RAID system performance is therefore critical towards fulfilling quality of service constraints for fast, reliable storage.

This thesis presents techniques and tools that model response times in zoned RAID systems. The inputs to this analysis are a specified I/O request arrival rate, an I/O request access profile, a given RAID configuration and physical disk parameters. The primary output of this analysis is an approximation to the cumulative distribution function of I/O request response time. From this, it is straightforward to calculate response time quantiles, as well as the mean, variance and higher moments of I/O request response time. The model supports RAID levels 0, 01, 10 and 5 and a variety of workload types.

Our RAID model is developed in a bottom-up hierarchical fashion. We begin by modelling each zoned disk drive in the array as a single M/G/1 queue. The service time is modelled as the sum of the random variables of seek time, rotational latency and data transfer time. In doing so, we take into account the properties of zoned disks. We then abstract a RAID system as a fork-join queueing network. This comprises several queues, each of which represents one disk drive in the array. We tailor our basic fork-join approximation to account for the I/O request patterns associated with particular request types and request sizes under different RAID levels. We extend the RAID and disk models to support bulk arrivals, requests of different sizes and scheduling algorithms that reorder queueing requests to minimise disk head positioning time. Finally, we develop a corresponding simulation to improve and validate the model. To test the accuracy of all our models, we validate them against disk drive and RAID device measurements throughout.

Acknowledgements

I would like to thank the following people, without whom this thesis would never have been possible:

- My supervisor, Dr William Knottenbelt, for his patience, inspiration, guidance and constant enthusiasm throughout my research.
- My second supervisor, Professor Peter Harrison, for always making time to share his expertise, encouraging me to explore new ideas and for introducing me to the wonders of Queueing Theory.
- The members of the Analysis, Engineering, Simulation and Optimisation of Performance (AESOP) research group for interesting and constructive discussions and debates on many research topics including but not exclusively those discussed in this thesis. Particular thanks to Dr Nicholas Dingle who I enjoyed collaborating with on much of this work.
- The Engineering and Physical Sciences Research Council (EPSRC) for providing me with the funding to do my PhD and the Department of Computing at Imperial College London for assigning this funding to me.
- My parents, for their constant love and support and for always encouraging me to follow the path to intellectual and professional fulfilment, wherever it may take me.
- My husband Bernard, for making the time spent writing this thesis the best time of my life.

‘Mathematics is the majestic structure conceived by man to grant him comprehension of the universe.’ *Le Corbusier*

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Despite the current economic downturn, demand for disk storage continues its unrelenting rise. Indeed, the IDC forecasts that shipped disk storage capacity will increase at a compound annual growth rate of over 38% for the next three years [102]. The efficient operation of public and private enterprises worldwide remains critically dependent on reliable, high performance storage. RAID[1] has revolutionised data storage because of its ability to synthesise a set of low-cost commodity storage devices into a single logical unit that can deliver high reliability with high performance. However, RAID system performance varies heavily in practice, depending on chosen configuration and operating context. Given a budget and an expected workload, it is therefore a major challenge for system

[1]Redundant Array of Inexpensive Disks [91]; RAID levels describe various ways of spreading data across multiple storage devices using striping, mirroring, and/or parity

designers and engineers to select RAID components and corresponding configurations capable of delivering a required level of quality of service. Performance models provide a low-cost means to evaluate the suitability of candidate system designs ahead of implementation.

RAID systems consist of a controller and member hard disk drives of which the disks drives represent the greatest performance bottleneck. An accurate hard disk drive performance model provides the foundations of an effective performance model of any RAID system. A significant recent development in disk drive technology is zoning[2] which enables greater space efficiency on each disk. A performance model must reflect the time and capacity benefits that this technology introduces over its unzoned counterpart. No prior work exists that produces an analytical response time distribution performance model of a RAID system consisting of zoned disk drives.

In the context of modern Service Level Agreements, effective performance prediction must provide the ability to reason not only about mean response times, but also higher moments and percentiles of response time. Therefore, our target in this work is the full cumulative distribution function of I/O request response time, from which all of the previous measures can be easily derived.

In this thesis we provide means to calculate response time distributions of zoned RAID systems for varying RAID levels and types of workload. This improves on the state-of-the-art in RAID performance models, which provide only the mean

[2]On modern hard drives there are more blocks on cylinders on the outside of the platter than those closer to the centre. Cylinders with the same number of blocks are grouped together in zones. Disks rotate with a constant angular velocity and therefore data throughput is higher for outer zones than for inner ones.

response time and no support for zoned disk drives [26, 54, 79, 124, 128]. RAID systems are most commonly and effectively modelled by fork-join queueing networks. This thesis provides a study and discussion on the benefits and drawbacks of response time approximations for this type of queue, for implementation in our model.

Most queueing network analytical models and specifically most analytical RAID models are only validated against a simulation model. We aim to always validate our RAID performance models against device measurements as well as simulation results, providing additional confidence in our models and their applications to commonly used storage systems.

1.2 Aims and Objectives

The aim of this thesis is to create a response time performance model of zoned RAID systems using analytical queueing network models.

In order to fulfil this, the following objectives must be achieved:

- Develop an analytical queueing model of I/O request response time in a zoned hard disk drive. This involves defining the service time distribution based on the mechanical behaviour of a disk during a read or write request.
- Choose an approximation of the fork-join queue that will best suit the needs of our RAID model.

- Tailor the fork-join queue approximation for the specific requirements of RAID levels 0, 01, 10 and 5 for both read and write requests of any size.
- Consider likely workload variations to a RAID system and show that the model can be adapted to accept these workloads.
- Create a simulation of a hard disk drive and RAID 0, 01 and 5 systems, both to compare to and improve the analytical model and to be used as a stand-alone simulation.
- For confidence in the models, validate both analytical and simulation models against device measurements from real disk drives and RAID systems.

1.3 Contributions

This thesis presents queueing network modelling techniques that enable the development of a response time performance model of zoned RAID systems. We use some existing techniques and extend some existing models to create first a model of a disk drive, then extend it through the abstraction of a fork-join queue to a RAID model. We then look at modelling different types of workloads that could be expected on a modern zoned disk array. The specific contributions of this thesis are described below:

1.3.1 Response Time Distribution Model of Zoned RAID

We develop a model for full distributions of I/O request response time of a single hard disk drive by extending Zertal and Harrison's work on service time distributions on a zoned disk [139]. We choose to model a RAID system with a fork-join queue and since no exact response time solutions exist for this queue, we use the maximum order statistic method to approximate its response time distribution. We introduce extensions to this approximation to enable features of RAID systems that differ from standard fork-join queue behaviour to be modelled. We consider different types of workloads that a disk and RAID system could expect, such as mixed arrival streams of read and write requests of varying size, bursty arrival streams and scheduling algorithms with request reordering. We incorporate support for these workloads into the disk drive and RAID system models. All these models are extensively validated against device measurements.

1.3.2 Developments in Queueing Theory

Often, the creation of a queueing network model of a hard disk drive or RAID system presents a problem that demands the derivation of new results in queueing theory. This thesis provides a number of contributions to queueing theory. Firstly, we provide a discussion on the relative benefits of existing fork-join queue response time approximations. Secondly, we develop analytical methods for calculating response times in $M/G/1$ queues with bulk arrivals, as well as for approximating response times in fork-join networks of such queues. Finally, we introduce a novel approximation for the response time of $M/G/1$ queues with state-dependent service times distributions for application to disk drives that employ scheduling algorithms with request reordering.

1.3.3 Queueing Simulation of Zoned RAID

In parallel to our analytical queueing network model, we have developed a queueing based discrete-event simulator of zoned hard disk drives and RAID systems based on the JINQS queueing simulation library [38]. This simulation supports similar workloads to the analytical model, so that it can be directly compared to the analytical model which will aid its development. The simulator can also be used as a standalone zoned RAID simulation. The simulator requires a small number of parameters that can be obtained either from the disk specification or measurements taken from the disk making the simulator simple and transparent to use. The simulation is extensively validated against device measurements.

1.4 Outline

The remainder of this thesis is organised as follows:

Chapter 2 describes the background theory required by the research presented in this thesis. Some elementary probability theory is introduced that is needed for the study of stochastic processes and Markov chains. With these details it is possible to introduce queueing theory with specific focus on the $M/G/1$ queue. A literature survey is then presented on work to do with all aspects of creating performance models of zoned and unzoned disk drives and RAID systems.

Chapter 3 presents an analytical queueing network model for I/O request response time in a hard disk drive. Probability distributions are derived for the contributing mechanical factors that combine to form the service time distribution. In addition a corresponding simulation is presented for a disk drive. Both these models are validated against device measurements.

Chapter 4 extends this disk drive model into a RAID system model. A RAID system is abstracted as a fork-join queue. There is an initial discussion on the best response time approximation for fork-join queueing networks. The fork-join response time approximation is tailored to reflect the specific needs of read and write requests of any size on RAID 0, 01, 10 and 5. Again, a corresponding simulation is introduced and compared to the analytical model and all models are validated against device measurements.

Chapter 5 explores a variety of workload types that a disk drive and RAID system could expect to experience and creates analytical models to support them. Specifically, this chapter deals with workloads consisting of mixtures of read and write requests, workloads that contain requests whose size is decided by a specified probability distribution and bursty arrival streams. Modern disk drives employ a scheduling algorithm that services the request in the disk queue with the shortest disk arm positioning time. This has a particularly pronounced effect on response times for bursty or heavily loaded arrival streams. A novel model for a disk drive with this scheduling algorithm implemented is presented here. We also look at using multiclass and priority queues to model the parity update operations in a partial stripe write request to RAID 5. Throughout, the analytical models are compared with device measurements and simulation results where possible.

Chapter 6 concludes the thesis by summarising and evaluating the achievements presented and highlighting opportunities for future work.

1.5 Publications and Statement of Originality

I declare that this thesis was composed by myself, and that the work that it presents is my own, except where otherwise stated.

The following publications arose from work conducted during the course of this PhD:

- **UK Performance Evaluation Workshop 2007** (UKPEW) [78] discusses different approximations for the response time of a fork-join queue highlighting the benefits of using the maximum order statistic approximation. The work on fork-join queues in Chapter 4 is based on this paper.
- **International Conference on Analytical and Stochastic Modelling Techniques and Applications 2008** (ASMTA) [73] presents the analytical zoned disk model, applies the approximation of response times in fork-join queues to model RAID systems and introduces modifications to the fork-join model to account for the specific needs of RAID 01 and 5. The disk drive model

is in Chapter 3 and the RAID model is in Chapter 4. This is joint work with Nicholas Dingle.

- **IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems 2008** (MASCOTS) [72] studies a wider range of workloads to RAID 01 and RAID 5 systems. It considers workloads that consist of both read and write requests and models RAID 5 partial stripe write requests as two subrequests using mulitclass and priority queues. This work is presented in Chapter 5. This is joint work with Nicholas Dingle.

- **UK Performance Evaluation Workshop 2008** (UKPEW) [74] continues to validate the RAID model introduced in the previous two papers against device measurements on a large disk array and makes some modifications to the RAID 5 write request model. These validations and modifications are presented in Chapters 4 and 5. This is joint work with Nicholas Dingle.

- **European Performance Engineering Workshop 2009** (EPEW) [75] and **European Simulation and Modelling Conference 2008** (ESM) [131] introduce queueing simulation models of a single disk, RAID 0, 01 and 5 and for a variety of workloads. These simulations are validated against device measurements. The disk drive simulation is discussed in Chapter 3, the fork-join queue and RAID simulation is described in Chapter 4 and other workloads are simulated in Chapter 5. This is joint work with Nicholas Dingle and Francis Wan.

- **International Conference on Quantitative Evaluation of Systems 2009**

(QEST) [76] presents an approximate response time model of the shortest positioning time first scheduling algorithm for disk drives, which has a significant effect on response time for increasing load. This work is described in Chapter 5. This is joint work with Nicholas Dingle.

- **International Conference on Performance Evaluation Methodologies and Tools 2009** (VALUETOOLS) [77] considers disks and RAID systems with bursty arrival streams and different sized requests arriving in the queue using the theory of queues with bulk arrivals. This work is described in Chapter 5. This is joint work with Nicholas Dingle, Peter Harrison and Soraya Zertal.

Chapter 2

Background Theory

This chapter introduces the background theory relevant to the research that is presented in this thesis. This background theory consists of two parts. In the first part, the mathematical techniques required in this thesis are introduced. This includes the use of random variables, stochastic processes, renewal theory and Markov chains and an introduction to certain aspects of queueing theory. In addition, there is a brief introduction to Laplace transforms and their numerical inversion and a discussion on the Generalised Lambda Distribution which can be used to approximate distributions given their first four moments. The second part provides a survey of related work in the area of performance models of disk drives and RAID systems.

2.1 Random Variables

A *random variable* is a function that reflects the result of a random experiment by mapping the sample space of all possible outcomes to some real number [16, 51]. If the set of all values that the random variable can take is finite, or countably infinite, it is discrete. Otherwise the random variable is continuous. The probability of the random variable taking a particular value is calculated using the probability mass function (pmf) for discrete random variables or the probability density function (pdf) for continuous random variables. The pmf, $f_X(x)$, for a discrete random variable X with any real number x is defined as $f_X(x) = \mathbb{P}(X = x)$. The probability that the value of the random variable will be below some specified value can be calculated using the cumulative distribution function (cdf) of X, $F_X(x)$:

$$F_X(x) = \mathbb{P}(X \leq x) = \begin{cases} \sum_{\forall i \leq x} f_X(i) & X \text{ is discrete} \\ \int_{-\infty}^{x} f_X(u)du & X \text{ is continuous} \end{cases}$$

The properties of a random variable can be described by defining moments of X, where $E[X^n]$ is the nth moment of X[51]. These moments can provide concise summary information about a random variable. Specifically we use the mean, variance, skewness and kurtosis to describe the properties of a random variable. We define $E[X^n]$ as

$$E[X^n] = \begin{cases} \sum_i x_i^n f_X(x_i) & X \text{ is discrete} \\ \int_{-\infty}^{\infty} x^n f_X(x)dx & X \text{ is continuous} \end{cases}$$

The first moment of X, $E[X]$, is called the *mean* or *expectation* and provides the weighted average of the possible values that X can take. The *variance* of X utilises the first and second moments and represents the spread of the density with respect to the mean [59]:

$$\mathrm{Var}[X] = \sigma^2 = E[X^2] - E[X]^2$$

The skewness indicates the asymmetry of the density around its mean which affects the shape of the distribution. Whether the skewness is positive or negative indicates that the density is skewed towards values greater or less than the mean [59]. It is defined as

$$\alpha_3 = \frac{1}{\sigma^3}\left(E[X^3] - 3E[X^2]E[X] + 2E[X]^3\right)$$

The kurtosis indicates the flatness of the distribution with respect to the normal distribution [59]. It is defined as

$$\alpha_4 = \frac{1}{\sigma^4}\left(E[X^4] - 4E[X^3]E[X] + 6E[X^2]E[X]^2 - 3E[X]^4\right)$$

For discrete random variables X with non-negative integer values, the probability generating function (pgf), $G_X(z)$ is defined as [51]

$$G_X(z) = E[z^X] = \sum_{i=0}^{\infty} f_X(i) z^i$$

The nth derivative of $G_X(z)$ with $z = 1$ is called the nth factorial moment, $E[X(X-1)\ldots(X-n+1)]$.

We define the probability that an event X occurs given that an event Y has already occurred as the *conditional probability*, $\mathbb{P}(X|Y) = \frac{\mathbb{P}(X \cap Y)}{\mathbb{P}(Y)}$. We can then define the conditional pdf as

$$f_{X|Y}(x|y) = \frac{f_{X,Y}(x,y)}{f_Y(y)}$$

where $f_{X,Y}$ is the joint pdf of random variables X and Y [119]. The conditional expectation of X given that $Y = y$ is

$$E[X|Y=y] = \begin{cases} \sum_x x f_{X|Y}(x|y) & X \text{ and } Y \text{are discrete and } f_Y(y) > 0 \\ \int_{-\infty}^{\infty} x f_{X|Y}(x|y) dx & X \text{ and } Y \text{are continuous and } f_Y(y) > 0 \end{cases}$$

If the sample space, S, can be partitioned into sets $\{Y_1, Y_2, \ldots, Y_n\}$ then any event X can be written as $X = \cup_{i=1}^{n} X \cap Y_i$. Consequently, the law of total probability can be derived:

$$\mathbb{P}(X) = \sum_{i=1}^{n} \mathbb{P}(X \cap Y_i) = \sum_{i=1}^{n} \mathbb{P}(X|Y_i)\mathbb{P}(Y_i) \tag{2.1}$$

The law of total probability can be used in tandem with conditional expectations by redefining $E[X]$ as $E[X] = E[E[X|Y_1, Y_2, \ldots, Y_n]]$ This implies that [51],

$$E[X] = \begin{cases} \sum_y E[X|Y=y] f_Y(y) & Y \text{ is discrete} \\ \int_{-\infty}^{\infty} E[X|Y=y] f_Y(y) dy & Y \text{ is continuous} \end{cases}$$

There are some well known discrete and continuous random variables whose properties are utilised in this thesis. We describe these below [99, 51].

The Bernoulli Random Variable X is an experiment that can have only two possible outcomes, 0 or 1. If $\mathbb{P}(X = 1) = p$, the pmf is

$$f_X(x) = \begin{cases} 1-p & x = 0 \\ p & x = 1 \end{cases}$$

The Binomial Random Variable If X represents the number of successful outcomes of n independent Bernoulli trials, the pmf with parameters (n, p) becomes

$$f_X(x) = \binom{n}{x} p^x (1-p)^{n-x} \quad x = 0, 1, \ldots, n$$

The pgf is $G_X(z) = (pz + 1 - p)^n$ and $E[X] = np$.

The Geometric Random Variable Let X represent the number of Bernoulli trials required for one successful outcome to occur. Then the pmf is

$$f_X(x) = (1-p)^{x-1} p \quad x = 1, 2, \ldots$$

The pgf is $G_X(z) = \frac{pz}{1-(1-p)z}$ and $E[X] = \frac{1}{p}$.

The Poisson Random Variable For large n and small p, the Binomial random variable can be approximated by the Poisson random variable which has parameter $\lambda = np$ and pmf

$$f_X(x) = e^{-\lambda} \frac{\lambda^x}{x!} \quad x = 0, 1, \ldots$$

The pgf is $G_X(z) = e^{-\lambda(1-z)}$ and $E[X] = \lambda$.

The Uniform Random Variable If the probability of an arbitrary value is constant over a specified period, the Uniform random variable is used. A random variable is distributed uniformly over the interval (a, b) with pdf

$$f_X(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

Here $E[X] = \frac{a+b}{2}$.

The Exponential Random Variable The (negative) exponential random variable has parameter λ and pdf

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where $E[X] = \frac{1}{\lambda}$.

2.1.1 Laplace Transforms

The Laplace transform is an integral transform that has many useful applications in mathematics, physics and engineering. It can often be applied to convert a hard-to-solve problem in the real-valued t-domain into an easier problem in the complex valued s-domain. The solution can then be inverted to provide the solution in the t-domain [8]. The Laplace Transform, $f_X^*(s)$ is defined as follows [51]:

Definition For a continuous function $f_X(t)$, where $t \geq 0$, the Laplace transform can be calculated as

$$f_X^*(s) = \int_0^\infty e^{-st} f_X(t) dt$$

In the context of probability theory, $f_X(t)$ is the probability density function of a random variable X. If only the cumulative distribution function of X is available, the Laplace-Steiltjes transform can be used instead [51]:

$$f_X^*(s) = \int_0^\infty e^{-st} dF_X(t)$$

The Laplace transform can also be expressed in terms of expectation for non-negative random variable X with density $f_X(t)$ as [51]

$$f_X^*(s) = E[e^{-sX}]$$

Properties

A benefit of studying functions in the Laplace domain is that certain computationally intensive complicated procedures become fast and straightforward when translated into the Laplace transform domain. We summarise here some of the properties that are especially relevant to this thesis.

Moments The Laplace Transform can be used to generate moments of a random variable by differentiating it n times and evaluating it at the point $s = 0$:

$$\left. \frac{d^n f_X^*(s)}{ds^n} \right|_{s=0} = (-1)^n E[X^n]$$

Integration To obtain a distribution function from a given density function in the t-domain, the density function must be integrated. However, in the Laplace domain, the Laplace Transform of a cumulative distribution function $F_X^*(s)$ can be calculated from the Laplace Transform of the density function $f_X^*(s)$ as follows [35]:

$$F_X^*(s) = \frac{f_X^*(s)}{s}$$

This can be proved using integration by parts from the definition of Laplace Transforms.

Convolution An important property of the Laplace Transform is the convolution property. If we have two independent random variables, X and Y, then the pdf of $X + Y$ is the convolution of the two individual pdfs:

$$f_{X+Y}(t) = \int_{-\infty}^{\infty} f_X(t-x) f_Y(x) dx$$

However, the Laplace transform of the convoluted pdfs can be shown to be merely the product of the individual Laplace Transforms, a far simpler procedure [69]:

$$f_{X+Y}^*(s) = f_X^*(s) f_Y^*(s)$$

Uniqueness If $f(t)$ and $g(t)$ are functions with corresponding Laplace Transforms $f^*(s)$ and $g^*(s)$, then $f^*(s) = g^*(s) \Leftrightarrow f(t) = g(t)$ [8].

Laplace Transform Inversion

Since the Laplace Transform of a function is unique, it is possible to return a Laplace Transform of a function, $f^*(s)$, to the t-domain, $f(t)$, by finding the inverse of the Laplace transform. An integral formula for the inverse Laplace Transform, known as the *Bromwich contour inversion integral*, is given by:

$$f(t) = \frac{1}{2\pi i} \int_{a-i\infty}^{a+i\infty} e^{st} f^*(s) ds \tag{2.2}$$

where a is a real number which lies to the right of all the singularities of $f^*(s)$ [35].

In practice it is difficult to use this equation to find the inverse of most functions analytically. Therefore many numerical inversion algorithms have been developed including the Euler method [2, 3]. Other inversion methods include the Laguerre method [60], Talbot's technique [112] and Durbin's method [37]. In this work we only use the Euler method for Laplace transform inversion, which we summarise here.

Equation (2.2) can be modified to create a more palatable problem of integrating a real-valued function of a real variable avoiding the use of complex variables and a contour integral. First, the substitution $s = a + iu$ allows Equation (2.2) to be rewritten as

$$f(t) = \frac{1}{2\pi i} \int_{-\infty}^{\infty} e^{(a+iu)t} f^*(a+iu) du$$

By noting that, $e^{(a+iu)t} = e^{at}(\cos ut + i \sin ut)$ and substituting into the above equation, it can be shown that [1]:

$$f(t) = \frac{2e^{at}}{\pi} \int_{0}^{\infty} \mathrm{Re}(f^*(a+iu)\cos(ut))du \tag{2.3}$$

This integral in the real domain can be calculated numerically using the trapezoidal rule of numerical integration. This is a numerical approximation of the integral of a function over the interval $[a, b]$ as

$$\int_a^b f(t)dt \approx h\left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f(a+kh)\right)$$

where $h = \frac{b-a}{n}$. In the case of Equation (2.3), we set the step size $h = \frac{\pi}{2t}$. We define a constant A that controls the discretisation error (set to 19.1 in [3]) and let $a = \frac{A}{2t}$. Then $f(t)$ can be approximated as the following alternating series [35]:

$$f(t) \approx \frac{e^{\frac{A}{2}}}{2t}\mathrm{Re}\left(f^*\left(\frac{A}{2t}\right)\right) + \frac{e^{\frac{A}{2}}}{2t}\sum_{k=1}^{\infty}(-1)^k\mathrm{Re}\left(f^*\left(\frac{A+2k\pi i}{2t}\right)\right)$$

Euler summation can be implemented to accelerate the convergence of this alternating series. The sum of the first n terms are calculated explicitly and Euler summation is used to calculate the next m terms. The mth term after the first n is given by [35]

$$E(t, m, n) = \sum_{k=0}^{m}\binom{m}{k}2^{-m}s_{n+k}(t)$$

where

$$s_n(t) = \sum_{k=0}^{n}(-1)^k\mathrm{Re}\left(f^*\left(\frac{A+2k\pi i}{2t}\right)\right)$$

The truncation error that results from using this Euler summation can be estimated by comparing the magnitudes of the nth and $n+1$th terms, $|E(t, m, n) - E(t, m, n+1)|$. If $n = 20$ and $m = 12$, there will be a truncation error of the order of 10^{-8} [35].

2.2 Stochastic Processes

A *stochastic process* is a set of random variables, $\{X_t\}$, indexed by a time parameter t. X_t represents the state of a system at time t [86]. In this section we summarise some relevant classes of stochastic processes.

2.2.1 Markov Processes

A *Markov process* is a class of stochastic process in which the set of random variables, $\{X_t\}$, must have an additional property known as the *memoryless* or *Markov Property* [69]. The Markov property can be written as follows:

$$\begin{aligned}\mathbb{P}(X_{t_{n+1}} = x_{n+1} | X_{t_n} = x_n, X_{t_{n-1}} = x_{n-1}, \ldots, X_{t_1} = x_1) \\ = \mathbb{P}(X_{t_{n+1}} = x_{n+1} | X_{t_n} = x_n)\end{aligned}$$

That is, the future evolution of the system depends only on the current state of the system.

2.2.2 Poisson Processes

The *Poisson process* is a counting process associated with the Markov property which counts the number of randomly occurring events observed in a time period t. It is defined by a set of random variables $\{N(t) | t > 0\}$, where $N(t)$ counts the number of events that have occurred up to time t. Each random variable, $N(t)$, has a Poisson distribution. It can be shown that, since the counting process has a Poisson distribution, the interarrival time between any two consecutive events has an exponential distribution and consequently exhibits the memoryless property [51].

2.2.3 Markov Chains

A *Markov chain* is a discrete time Markov process $\{X_n \in S \mid n = 0, 1, 2, \ldots\}$ with countable state space, $S = \{0, 1, 2, \ldots |S|\}$ [51]. The probability of moving to a future state j from a current state i are defined by the *one-step transition probabilities*,

$$\begin{aligned} p_{i,j}(n) &= \mathbb{P}(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots, X_1 = i_1) \\ &= \mathbb{P}(X_{n+1} = j | X_n = i) \quad i, j = 0, 1, 2, \ldots \end{aligned}$$

If the one-step transition probabilities do not depend on the time instant ($p_{i,j}(n) = p_{i,j}$), the Markov chain is *time homogeneous*. All time homogeneous one-step transition probabilities can be presented in the transition probability matrix P, with members $p_{i,j}$ and indices ranging over the state space [86]. All row sums in P are 1.

A Markov chain is *irreducible* if every state is reachable from every other state. A state is periodic with period m if state j is returned to at some multiple of m steps. If there does not exist an integer $m > 1$ that fulfils this for a state, then the state is *aperiodic*. In an irreducible Markov chain, either all states are periodic or all states are aperiodic.

It is possible that over a long period of time (n steps) the probability of a Markov chain being in a particular state will be independent of the initial step:

$$\lim_{n\to\infty} \mathbb{P}(X_n = j \mid X_0 = i) = \lim_{n\to\infty} p_{i,j}^{(n)} = \pi_j \quad j = 0, 1, \ldots, |S|$$

If these limiting probabilities π_j exist and sum to 1, then a steady-state exists for the Markov chain and they are referred to as the *steady-state distribution* or *equilibrium distribution*. The steady-state theorem states:

Theorem 2.1. *Consider an irreducible and aperiodic Markov chain, X, with one-step transition probability matrix $P = (p_{i,j}), i, j = 0, 1, \ldots, |S|$. Then, if the set of equations*

$$\pi_j = \sum_{i=0}^{|S|} \pi_i p_{i,j} \quad j = 0, 1, \ldots \tag{2.4}$$

$$\sum_{j=0}^{|S|} \pi_j = 1 \tag{2.5}$$

has a positive solution, the solution is unique and is the steady-state distribution of X.

These equations are referred to as the *balance equations* of the Markov chain. By defining the row vector $\pi = (\pi_0, \pi_1, \ldots)$, Equation (2.4) can be rewritten as $\pi = \pi P$.

If the state space is discrete but the time parameter is continuous, so $X = \{X_t \mid t > 0\}$ and X has the Markov property, it is referred to as a continuous time Markov process. Each continuous time Markov process has an embedded discrete time

Markov chain which defines state transition probabilities at state transition instants [86].

The Markov property defines that the amount of time spent in any state must be *memoryless*. For continuous time Markov chains, it can be proved that this is only the case if there are exponentially distributed state sojourn times [69].

2.2.4 Renewal Theory

We consider an event that happens recurrently, first at time 0 and then at further random intervals forever. Let each instance of occurrence be represented by the random variable T_n, $n = 0, 1, \ldots$ as a renewal or arrival point. The set $\{T_n \mid n = 0, 1, \ldots\}$ is called a *renewal process*. In addition the intervals between these random occurrences are defined by the random variables $S_n = T_n - T_{n-1}$, $n = 1, 2, \ldots$. The random variables S_n are assumed to be independent and identically distributed and are called renewal or interarrival intervals. The time of the arrival of the nth renewal point is equivalent to the sum of all the inter-arrival times prior to it [86]:

$$T_n = \sum_{i=1}^{n} S_i \quad n = 1, 2, \ldots$$

N_t is defined as the number of renewals in time interval $(0, t]$ or $N_t = \max\{n \,|\, T_n \leq t\}$.

Continuous Recurrence Times

Another interesting random variable is the time from a randomly chosen time point to the next or previous renewal point. The time to the next renewal from this random time instant is called the *forward recurrence time* or *residual life* and is denoted by the random variable U_t. The *backward recurrence time* is the time that has passed since the most recent renewal, V_t. In addition the random variable $W_t = U_t + V_t$ gives the length of a renewal period. This is different from the interval between random occurrences, S_n as it is dependent on the renewal interval containing time point t.

It is useful in this work to find the joint probability distribution of forward and backward recurrence times, U_t and V_t. Denoting the density time of a renewal period, S_n, as $f_{S_n}(t)$ and mean length of a renewal period $E[S_n]$ it is shown in [51] that:

$$f_{U_t,V_t}(u,v) \to \frac{1}{E[S_n]} f_{S_n}(u+v) \quad \text{as } t \to \infty \tag{2.6}$$

Discrete Backward Recurrences

Renewal theory is applicable not just in the continuous domain but also in the discrete domain. An example of this would be a stream of batches of objects all containing a different number of objects. The size of the batch is equivalent to the renewal period. Let us assume a random object is picked in some batch. We provide here the derivation of the size distribution of discrete backward recurrences (the number of objects counted from the start of the batch containing the specified object up to that object) [29].

Let L be a discrete random variable representing the batch size for a randomly selected object with associated mass function $f_L(x) = \mathbb{P}$(A randomly selected object will have batch size x). Note this is not the same as $f_B(x)$, the batch size pdf, as L is more likely to be a larger batch size, as there are more objects in larger batches. L can be defined with a size-biased density:

$$f_L(x) = \frac{x f_B(x)}{E[B]}$$

The discrete random variable Y is the number of objects in a batch in front of the randomly chosen object. Then,

$$\mathbb{P}(Y = x \mid L = x_0) = \begin{cases} \frac{1}{x_0} & 0 \leq x < x_0 \\ 0 & x \geq x_0 \end{cases}$$

By the law of total probability,

$$\begin{aligned} \mathbb{P}(Y = x) &= \sum_{x_0=x+1}^{\infty} \frac{1}{x_0} f_L(x_0) \\ &= \sum_{x_0=x+1}^{\infty} \frac{f_B(x_0)}{E[B]} \qquad x = 0, 1, 2, \ldots \end{aligned}$$

The density function of a discrete backward recurrence, $f_Y(x)$, can be derived from the result above and the definition of density functions as

$$f_Y(x) = \frac{1 - F_B(x)}{E[B]}$$

Similarly, the probability generating function is defined as

$$\begin{aligned} G_Y(z) &= \frac{1}{E[B]}\sum_{i=0}^{\infty}(1-F_B(i))z^i \\ &= \frac{1-G_B(z)}{E[B](1-z)} \end{aligned}$$

2.3 Queueing Theory

Queueing Theory is the mathematical study of real-world phenomena that can be abstracted as queues and service stations. It was invented by A. K. Erlang in 1909 to avoid telephone traffic congestion [45]. Queueing problems arise throughout daily life today in the retail world, transport systems and health services as well as more technical domains such as manufacturing, computer networking and modern communication systems [122]. The application of queueing models to a real-life system enables system operators to implement routing and scheduling strategies that can be shown to improve overall system performance. A single queue generally consists of a line which is populated by customers that arrive randomly with interarrival times specified by a probability distribution. The queue has a number of servers, which serve customers at a rate defined by another probability distribution. Kendall defined a notation for identifying different queue types, specifying a queue with the notation $A/S/m$, where A is nature of the arrival process, S similarly describes the service process and m are the number of servers available to the single queue [51, 65]. An extended form of Kendall's notation defines a queue as $A/S/m/c/p/d$, where c is the capacity of the queue, p is the available population of customers and d is the queueing discipline. If the notation $A/S/m$ is used, it is assumed that both c and p are infinite and d is First Come First Served (FCFS). Some other possible queueing disciplines are Last Come First Served (LCFS) and priority queueing.

The simplest queue is the $M/M/1$ queue. This queue has *Markovian* arrivals and service times and one server. This implies that the interarrival times and service times are exponentially distributed with arrival and service rates λ and μ respectively [45]. An $M/M/1$ queue is a continuous time Markov chain, where the state is defined by the number of tasks queueing or in service. Markovian arrivals and service times ensure that the Markov property is fulfilled. The $M/G/1$ queue has a generalised service time distribution, which could be any continuous probability distribution.

For any type of queue in the steady-state, Little [81] proved that in all cases the mean number of tasks in the system (queueing or in service), L, is related to the response time of a task, W, by $L = \lambda W$.

Systems with multiple resources can be modelled as networks of queues. Queueing networks can be classified as one of three types: open, closed or mixed. An open network has at least one incoming source of customers and at least one exit from the network for customers. A closed network has neither entrance nor exit and a mixed network is a multi-class network in which customers of certain classes see an open network and other classes see a closed network. Results exist for steady state distributions of both open and closed networks [33, 42, 61].

In this thesis, we exclusively study $M/G/1$ queues and queueing networks of or queues derived from $M/G/1$ queues. We therefore summarise results relating to the $M/G/1$ queue here.

2.3.1 The M/G/1 queue

By exploiting the Markovian nature of an $M/M/1$ queue it is not difficult to calculate its response time distribution explicitly using the properties of Markov chains. This is not possible with an $M/G/1$ queue where the service times do not have the Markov property. However, the arrival process is still Markovian and we can exploit that fact.

An $M/G/1$ queue has a Poisson arrival process with rate λ and service time distribution $F_X(x)$ and service rate μ. The queue utilisation is $\rho = \frac{\lambda}{\mu}$. $N(t)$ is defined as the number of customers queueing and in service at time t. Since $N(t)$ is dependent on the non-Markovian service times as well as Markovian arrival times, $\{N(t) \mid t \geq 0\}$ is not in general a Markov chain. L_n is defined to be the number of customers queueing immediately after the completion of service and departure of the nth customer at time t_n [119].

It is important to note here the PASTA property (Poisson Arrivals See Time Averages) [134] or the random observer property which states that as $t \to \infty$, the state of the system seen by an arrival from a Poisson process has the same distribution as the state of the system observed at a randomly chosen time [86].

By defining Z_n as the number of new arrivals in the queue during the service of the $n+1$th customer then

$$L_{n+1} = \begin{cases} L_n - 1 + Z_n & L_n > 0 \\ Z_n & L_n = 0 \end{cases} \tag{2.7}$$

If the queue was not empty then the new queue length will be the previous queue length minus the departing customer plus all arriving customers. If the queue was empty at the previous departure point then the next departure point must wait for a customer to arrive before service begins and another departure can take place. The Z_n are independent and identically distributed random variables. In addition, they are independent of $L_1, L_2, \ldots, L_n$. Therefore, to determine the value of L_{n+1} it is only necessary to know L_n and Z_n and not any of the previous queue lengths $L_1, L_2, \ldots, L_{n-1}$. Thus $\{L_n \mid n = 1, 2, \ldots\}$ is a discrete time Markov chain called the *embedded Markov chain* of the stochastic process $\{N(t) \mid t \geq 0\}$ for an $M/G/1$ queue. Defining the pmf of Z_n, $n \geq 0$ as $p_j = \mathbb{P}(Z_n = j)$, $j \geq 0$ and pgf, $G_{Z_n}(z) = \sum_{j=0}^{\infty} p_j z^j$, we can start to calculate the transition probabilities for the embedded Markov chain. If X_n is the service time random variable for the service of the nth customer and there is a Poisson arrival process with rate λ, it follows that

$$\mathbb{P}(Z_n = j \mid X_{n+1} = x) = \frac{(\lambda x)^j}{j!} e^{-\lambda x}$$

Hence, by the law of total probability,

$$p_j = \int_0^{\infty} \frac{(\lambda x)^j}{j!} e^{-\lambda x} dF_X(x)$$

and therefore,

$$G_{Z_n}(z) = \int_0^\infty e^{\lambda x z} e^{-\lambda x} dF_X(x) = X^*[\lambda(1-z)] \tag{2.8}$$

where $X^*(s)$ is the Laplace-Stieltjes transform of the distribution function $F_X(x)$.

The embedded Markov chain has transition matrix $Q = (q_{ij} \mid i, j \geq 0)$ where transition probabilities are derived from Equation (2.7).

$$q_{ij} = \mathbb{P}(L_{n+1} = j \mid L_n = i) = \begin{cases} \mathbb{P}(Z_n = j - i + 1) & i \neq 0, j \geq i-1 \\ \mathbb{P}(Z_n = j) & i = 0, j \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

Hence,

$$Q = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 & p_4 & \cdots \\ p_0 & p_1 & p_2 & p_3 & p_4 & \cdots \\ 0 & p_0 & p_1 & p_2 & p_3 & \cdots \\ 0 & 0 & p_0 & p_1 & p_2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The steady state equations for this Markov chain, $\pi = \pi Q$ are

$$\pi_i = \pi_0 p_i + \sum_{j=0}^{i} \pi_{j+1} p_{i-j} \tag{2.10}$$

The generating function $\Pi(z) = \sum_{i=0}^{\infty} \pi_i z^i$, if it exists, will become,

$$\begin{aligned}\Pi(z) &= \pi_0 G_{Z_n}(z) + \sum_{i=0}^{\infty}\sum_{j=0}^{i} \pi_{j+1} p_{i-j} z^i \\ &= \pi_0 G_{Z_n}(z) + \sum_{j=0}^{\infty}\sum_{i=j}^{\infty} \pi_{j+1} p_{i-j} z^i\end{aligned}$$

since $0 \leq j \leq i \leq \infty$.

Hence,

$$\begin{aligned}\Pi(z) &= \pi_0 G_{Z_n}(z) + \sum_{j=0}^{\infty} \pi_{j+1} z^j \sum_{i=0}^{\infty} p_i z^i \\ &= \pi_0 G_{Z_n}(z) + \frac{(\Pi(z) - \pi_0) G_{Z_n}(z)}{z}\end{aligned}$$

Solving this equation results in:

$$\Pi(z) = \frac{\pi_0 (1-z) G_{Z_n}(z)}{G_{Z_n}(z) - z} \tag{2.11}$$

This is dependent on the Markov chain being stationary, i.e. $\Pi(1) = 1$. Since in Equation (2.11), both numerator and denominator tend to 0 as $z \to 1$, we apply L'Hôpital's rule to obtain:

$$\Pi(1) = \frac{\pi_0}{1 - G'(1)} = 1$$

Where $G'(1) = \frac{dG(z)}{dz}|_{z=1}$. Therefore, for the stationary condition to be fulfilled, $G'(1) < 1$ and $1 - G'(1) = \pi_0$. Differentiating Equation (2.8) and letting $z = 1$ (using the properties of moments in Laplace transforms described in Section 2.1.1), it follows that $G'(1) = \frac{\lambda}{\mu} = \rho$. In a steady state, the server utilisation $\rho < 1$. Therefore, using this result and Equation (2.8), Equation (2.11) can be rewritten as

$$\Pi(z) = \frac{(1-\rho)(1-z)X^*(\lambda(1-z))}{X^*(\lambda(1-z)) - z} \tag{2.12}$$

This equation which is the equilibrium probability generating function of the queue length is called the *Pollaczek-Khintchine transform equation*. The response time distribution, $F_W(x)$, can be calculated from this result since the length of the queue L_n that exists on the departure of the nth customer from service is precisely the number of customers that arrived during the waiting time and service of the departing customer. Therefore the response time of a single customer is equivalent to the combined interarrival times of all customers that arrived between a chosen customer arriving and leaving the system. Hence, the generating function of queue length can be expressed as [50, 51]

$$\Pi(z) = E[E[z^L \mid W]] = E[e^{-\lambda W(1-z)}] = W^*[\lambda(1-z)]$$

Then the Laplace-Stieltjes transform of response time for an $M/G/1$ queue is

$$W^*(\theta) = \Pi\left(\frac{\lambda-\theta}{\lambda}\right) = \frac{(1-\rho)\theta X^*(\theta)}{\theta - \lambda(1-X^*(\theta))} \tag{2.13}$$

This equation can be derived differently by using conditional expectation and considering an $M/G/1$ queue in two cases: firstly when a request arrives to an empty queue and hence has a response time equivalent to service time alone, and secondly when a request arrives to a non-empty queue and queueing time must be factored into the response time calculation [49]. In this case, new random variables are defined to describe the state of the queue at the arrival of a random tagged customer. A is defined as the number of customers in the queue (not in service) at the start of service of the customer that is being serviced when a tagged customer arrives. Y is the number of customers that arrive between the start of service of this customer and the arrival of the tagged customer. U and V are the backwards and forwards recurrence times for the service time of this customer – the proportion of the service time before and after the tagged customer arrives. It can be noted that Y is the number of customers that arrive during time U. We can define the total queueing time for the tagged customer as the time for the request in service to complete service (V) and for all requests ahead of the tagged customer in the queue to complete service. Therefore, assuming that the queueing time $Q > 0$ and conditioning on A, Y, U and V, we obtain

$$E[e^{-\theta Q}|U,V,A,Y] = E[e^{-\theta(V+X_1+\ldots+X_{A+Y})}|U,V,A,Y]$$

Deconditioning on A and Y and noting that Y is the number of Poisson distributed arriving customers in time period U gives

$$E[e^{-\theta Q} \mid U,V] = G_A(X^*(\theta))E[e^{-\lambda(1-X^*(\theta))U}e^{-\theta V} \mid U,V]$$

Using the result for the joint density of forward and backward recurrence times in

Equation (2.6), deconditioning further yields

$$\begin{aligned}
E[e^{-\theta Q} \mid Q > 0] &= G_A(X^*(\theta))\mu \int_0^\infty \int_0^\infty e^{-\lambda(1-X^*(\theta))u} e^{-\theta v} f_X(u+v)\,du\,dv \\
&= G_A(X^*(\theta))\mu \int_0^\infty \int_0^w e^{-\theta w} f_X(w) e^{(\theta-\lambda(1-X^*(\theta)))u}\,dw\,du \\
&= \frac{\mu G_A(X^*(\theta))(X^*(\lambda(1-X^*(\theta))) - X^*(\theta))}{\theta - \lambda(1-X^*(\theta))} \qquad (2.14)
\end{aligned}$$

By the random observer property, the number of customers queueing at the beginning of a service is the same as the number of customers queueing immediately after the start of service of a customer. Equation (2.12) provides the queue length generating function for an $M/G/1$ queue that counts the customer currently in service. Discounting this customer, it can easily be observed that the generating function of the number of queueing customers is

$$G_A(z) = \frac{(1-\rho)(1-z)}{X^*(\lambda(1-z)) - z}$$

and substituting this into Equation (2.14) yields

$$E[e^{-\theta Q} \mid Q > 0] = \frac{\mu(1-\rho)(1-X^*(\theta))}{\theta - \lambda(1-X^*(\theta))} \qquad (2.15)$$

Then the Laplace transform of queueing time can be obtained by

$$\begin{aligned}
Q^*(\theta) &= \mathbb{P}(Q > 0)E[e^{-\theta Q} \mid Q > 0] + \mathbb{P}(Q = 0)E[e^{-\theta 0}] \\
&= \rho E[e^{-\theta Q} \mid Q > 0] + (1-\rho)
\end{aligned}$$

By the convolution principle of Laplace transforms, the response time Laplace transform is

$$W^*(\theta) = Q^*(\theta)X^*(\theta)$$

which reduces simply to Equation (2.13).

Busy Periods

In a queueing system, busy periods are defined as the intervals between idle periods on the server. It can be observed that the distribution of the busy period will be the same for all queueing disciplines that are work-conserving and for which the server is never idle when the queue is empty [50]. We denote the length of a busy period by the random variable M. N customers arrive during the service of the first customer in the busy period. For each of these customers, $i = 1, \ldots, N$, we define their pseudo-busy period as the time to service them and all customers that arrive during their service as M_i. The service time of the first customer is quantified by the random variable X, as are all subsequent service times. Then, since $\{M_i \mid i = 1 \ldots N\}$ are independent and identically distributed,

$$E[e^{-\theta M} \mid X = x, N = n, M_1 = m_1, M_2 = m_2, \ldots M_n = m_n] \\ = e^{-\theta(x+m_1+m_2+\ldots+m_n)}$$

Deconditioning on the M_is we get

$$E[e^{-\theta M} \mid X = x, N = n] = e^{-\theta x}(M^*(\theta))^n$$

Since arrivals are Markovian, N has a Poisson distribution with mean λx and we can decondition further to obtain

$$E[e^{-\theta M} \mid X = x] = e^{-\theta x} e^{-\lambda x} \sum_{n=0}^{\infty} \frac{(\lambda x M^*(\theta))^n}{n!} = e^{-x(\theta + \lambda - \lambda M^*(\theta))}$$

Finally we decondition on X to calculate the LST of the busy period M:

$$M^*(\theta) = E[e^{-\theta M}] = \int_0^{\infty} e^{-x(\theta+\lambda-\lambda M^*(\theta))} dF_X(x) = X^*(\theta + \lambda - \lambda M^*(\theta)) \tag{2.16}$$

thus creating a recursive equation for the LST of the busy period [28]. This can also be derived by assuming a last-come first-served queueing discipline for the busy period without loss of generality (as the length of the busy period does not change between scheduling strategies) and deriving the busy period distribution using conditional expectation [50]. This technique, in which a time delay is defined in terms of independent, identically distributed time delays is called *delay cycle analysis*.

M/G/1 Queues with Non-preemptive Priority

In a priority queue, customers arrive with an assigned priority class which defines a relative priority for order of service. There are two types of priority, preemptive and non-preemptive. For preemptive priority, if a customer arrives in the queue with a higher priority class than the customer currently servicing, the servicing customer will cease service and the arriving customer will replace it at the server. In non-preemptive priority a servicing customer cannot be interrupted and upon completion the server will next choose the highest priority customer waiting in the queue. Much work has been done on both these cases for different types of queues [28, 49, 63, 111]. In this thesis, we are specifically interested in $M/G/1$ queues with non-preemptive priority and servers with a different type of service time distribution for each class. The derivation for the response time distribution of this type of class can be found in Conway, Maxwell and Miller [28] based on the theory of busy periods.

They consider an $M/G/1$ queue in which customers have a class $i = 1, 2, \ldots, r$ with an attached priority. Each priority class has a Markovian arrival rate of λ_i and service time random variable X_i so the utilisation is $\rho_i = \lambda_i E[X_i]$. They aim to find the response time distribution of type i customers. In order to do this they group customers that are not in class i into two composite classes: class a contains customers with a higher priority than class i and class b contains customers with a lower priority than class i. Then the arrival rates for these new classes will be $\lambda_a = \sum_{j=1}^{i-1} \lambda_j$ and $\lambda_b = \sum_{j=i+1}^{r} \lambda_j$. The distributions of service time become $F_{X_a}(t) = \frac{1}{\lambda_a} \sum_{j=1}^{i-1} \lambda_j F_{X_j}(t)$ and $F_{X_b}(t) = \frac{1}{\lambda_b} \sum_{j=i+1}^{r} \lambda_j F_{X_j}(t)$.

Then the Laplace-Stieltjes transform of queueing time for a priority class i customer will be

$$Q_i^*(\theta) = \frac{(1-\rho)(\theta + \lambda_a(1 - M_a^*(\theta))) + \lambda_b(1 - X_b^*(\theta + \lambda_a(1 - M_a^*(\theta))))}{\lambda_i X_i^*(\theta + \lambda_a(1 - M_a^*(\theta))) - \lambda_i + \theta} \tag{2.17}$$

The response time distribution LST for a priority class i customer will be

$$W_i^*(\theta) = Q_i^*(\theta) X_i^*(\theta) \tag{2.18}$$

2.4 Generalised Lambda Distribution

Sometimes it is not possible to derive an equation for the response time distribution function in the t-domain exactly. Since all queues in this thesis are $M/G/1$ or derived from $M/G/1$ queues, the response time distribution is always initially derived in the Laplace domain. Usually we numerically invert the resulting Laplace transform but occasionally in this thesis the complexities of these derivations make the Laplace transform too complicated to numerically invert quickly. In these cases, we consider distribution fitting approximations to calculate the response time distribution and density.

One such distribution fitting approximation is the Generalised Lambda Distribution (GLD) [71]. The GLD takes as parameters the first four moments of response time which are easily obtainable from any Laplace transform (see Section 2.1.1). The GLD is parameterised using the mean (μ), variance (σ^2), skewness (α_3) and kurtosis (α_4), and approximates the related distribution from these values. If the cdf of response time is defined as $F_W(t)$, then its inverse, $Q(u)$ is approximated as:

$$Q(u) \approx \lambda_1 + \frac{1}{\lambda_2}\left(\frac{u^{\lambda_3} - 1}{\lambda_3} - \frac{(1-u)^{\lambda_4} - 1}{\lambda_4}\right) \tag{2.19}$$

The parameters λ_3 and λ_4 are calculated by solving the following simultaneous equations numerically:

$$\alpha_3 = \frac{v_3 - 3v_1v_2 + 2v_1^3}{(v_2 - v_1^2)^{\frac{3}{2}}}$$
$$\alpha_4 = \frac{v_4 - 4v_1v_3 + 6v_1^2v_2 - 3v_1^4}{(v_2 - v_1^2)^2}$$

where

$$\begin{aligned} v_1 &= \frac{1}{\lambda_3(\lambda_3+1)} - \frac{1}{\lambda_4(\lambda_4+1)} \\ v_2 &= \frac{1}{\lambda_3^2(2\lambda_3+1)} + \frac{1}{\lambda_4^2(2\lambda_4+1)} - \frac{2}{\lambda_3\lambda_4}\beta(\lambda_3+1,\lambda_4+1) \end{aligned}$$

$$\begin{aligned} v_3 &= \frac{1}{\lambda_3^3(3\lambda_3+1)} + \frac{1}{\lambda_4^3(3\lambda_4+1)} - \frac{3}{\lambda_3^2\lambda_4}\beta(2\lambda_3+1,\lambda_4+1) \\ &\quad + \frac{3}{\lambda_3\lambda_4^2}\beta(\lambda_3+1,2\lambda_4+1) \\ v_4 &= \frac{1}{\lambda_3^4(4\lambda_3+1)} + \frac{1}{\lambda_4^4(4\lambda_4+1)} + \frac{6}{\lambda_3^2\lambda_4^2}\beta(2\lambda_3+1,2\lambda_4+10) \\ &\quad - \frac{4}{\lambda_3^3\lambda_4}\beta(3\lambda_3+1,\lambda_4+1) - \frac{4}{\lambda_3\lambda_4^3}\beta(\lambda_3+1,3\lambda_4+1) \end{aligned}$$

and $\beta(x, y)$ is the Euler Beta Function [135]. Then λ_1 and λ_2 can be calculated using:

$$\lambda_2 = \frac{\sqrt{v_2 - v_1^2}}{\sigma}$$
$$\lambda_1 = \mu + \frac{1}{\lambda_2}\left(\frac{1}{\lambda_3+1} - \frac{1}{\lambda_4+1}\right)$$

An approximation of the response time distribution, $F_W(t)$, can consequently be obtained by setting $t = Q(u)$ and $F_W(t) = u$. To obtain the response time density, again $t = Q(u)$ and $f_W(t) = f_W(Q(u))$ which is defined in [9] as

$$f_W(Q(u)) = \frac{\lambda_2}{u^{\lambda_3-1} + (1-u)^{\lambda_4-1}} \tag{2.20}$$

2.5 Disk Drive and Disk Array Modelling

We summarise here past work in the field of performance models of hard disk drives and RAID systems. We first describe the physical structure of disk drives and arrays and then introduce some of the different methods used to model these features effectively. Those models that are more relevant to the research in this thesis are discussed in detail; other less relevant models are summarised. We also discuss the effect of zoning and caching on response times and summarise research in these areas.

2.5.1 Disk Drives

Disk drives consist of a mechanism and a controller. The mechanism contains recording and positioning components and the controller manages the storage and retrieval of data [101]. Figure 2.1 is a diagram of the mechanical components of a single disk drive. A single disk drive is comprised of at least one and as many as twelve platters. These platters rotate around a spindle. As disk rotation speed increases, transfer rates are improved and rotational latency shortens. The disk head senses the magnetic flux variations on the disk's surface in order to read data [5].

Data can be written to each platter on the disk. Tracks in the same position on each platter are grouped together and referred to as cylinders. Any location on the disk is uniquely identified by the cylinder number, platter number and sector number [4].

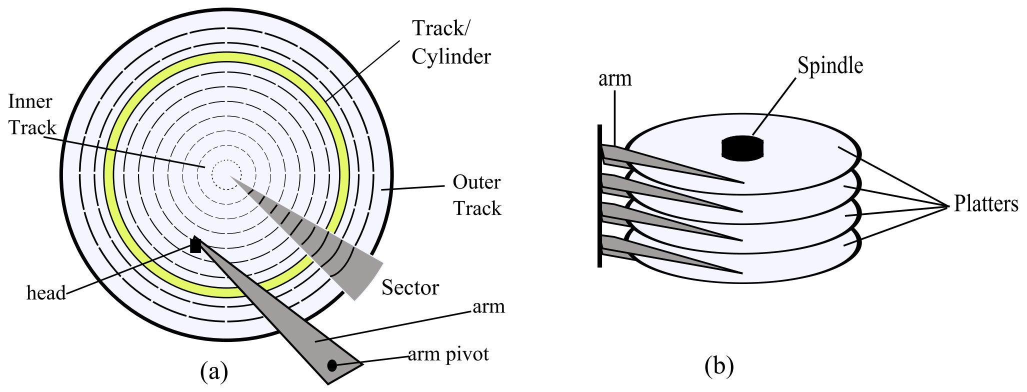


Figure 2.1: The mechanical components of a disk drive, (a) top view, (b) side view [101].

The execution of a request by a disk is dependent on seek time, rotational latency and data transfer time. These factors combine to make up the disk service time. Seek time is the time it takes the disk arm to move the head to the cylinder of choice from its current position. Rotational latency is the time for the required sector to rotate under the disk head after the seek completes. The performance impact of seek time depends upon the diameter of the disk and rotational latency on the angular distance of the chosen sector from the original position of the disk head. The seek time and rotational latency together are referred to as the *disk positioning time*.

The data transfer time is dependent on the rate at which data can be read and written onto a platter's surface. This is a function of the platter's rotation rate, the density of the magnetic media and the distance of the head from the centre of the platter. Transfer time can vary across the disk due to disk zoning [101]. The further the disk head is from the centre of the platter, the faster its disk transfer time. This is because there are more sectors on the outer than inner tracks and these additional sectors can be read from or written to in the same time as the few sectors per track at the centre of the disk. Zoning utilises this effect, by grouping tracks into zones. Each track in a single zone contains the same number of sectors. By grouping these tracks, data can be written to the correct zone according to its performance requirements. On a disk without zoning, each track has the same number of sectors and the same amount of data is written to each track. Hence, there will be an increasing distance between data blocks on outer tracks. On a zoned disk, the number of sectors on a track increases as the track gets closer to the disk circumference, and the distance between blocks on a track remains constant [92]. There will be further discussion on modelling zoned disks in Section 2.5.5.

Other factors need to be considered when modelling the service time of a disk:

Sparing All disks contain some sectors that are flawed and cannot be used. The flaws are found during manufacturing and hence the controller knows where the damaged sectors are and not to use them. References to these sectors are remapped to other parts of the disk. This is called sparing [101].

Caching Caching occurs for both reads and writes [101] and improves the response time and throughput. Policies are needed to define what is stored in the cache at any time. We discuss the issues that need to be addressed when modelling caching in Section 2.5.6.

Rotational Positioning Ordering On many modern disk drives, requests in the disk queue are re-ordered and rescheduled in order to minimise disk head positioning time [5]. This reduces the time needed to service each request which inevitably reduces overall request response times [57]. There exist many possible scheduling algorithms to choose the order in which requests are serviced.

The simplest and most often modelled is First Come First Served (FCFS) in which disk arm positioning time is not minimised. The Shortest Seek Time First (SSTF) algorithm minimises track-to-track seek time only. SSTF can be implemented using the SCAN algorithm [32] in which requests are serviced in order of the disk cylinder number in a particular direction. The main drawback of SSTF is that it does not consider rotational latency; the latter makes up an increasing proportion of disk head positioning time as recent advances in disk technology have shortened seek times significantly, while rotational speeds have increased only slightly [62]. To address this, Jacobson and Wilkes [62] and Seltzer et al. [107] introduce Shortest Access Time First (SATF), where access time is disk head positioning time. This strategy introduces the possibility that certain requests can suffer from starvation. The Aged Shortest Access Time First (ASATF) algorithm avoids this by basing ordering on a metric that takes into account the amount of time that a request has been queueing. These adjusted access times will decrease the longer they remain in the queue. Andrews et al. [6] and Bachmat [11] study other, more optimal scheduling algorithms for example cases of the Asymmetric Travelling Salesman Problem.

Worthington et al. [138] carry out a simulation study of FCFS, SSTF and SATF and resolve that SATF provides the fastest mean response times, and that FCFS can yield particularly poor performance metrics. Thomasian and Fu [113] also look at a new scheduling algorithm that minimises seek time. Burkhard and

Palmer [20] present an SATF-like scheduling algorithm for optimising positioning time that takes into account the fact that an aggressive head movement may fail to settle in time to read from the target sector. In this case, the disk must complete a full rotation before data transfer can begin. The probability of this occurring is known as the miss probability, and is drive-dependent. Seagate disks implement Rotational Positioning Ordering (RPO) using Native Command Queueing (NCQ) which aims to optimally re-order commands to maximise performance [58].

2.5.2 Disk Drive Model

The disks are the slowest part of a disk array. It is therefore fundamentally important to model disk service time parameters accurately to ensure precise performance predictions for a disk array or storage system.

The parameters that combine to make up disk service time are seek time, rotational latency and data transfer time. Ruemmler and Wilkes [101] suggest that analytical models are unlikely to model disk drive response time accurately because of the disk's non-linear state-dependent behaviour. However, there exist many analytical results modelling the response time of disk drives which have compared favourably to their simulated counterparts.

A disk drive can be modelled as a single server queue, usually $M/G/1$ since service time is unlikely to be Markovian. The service time of each request is the sum of queueing time, seek time, rotational latency and data transfer time. It is assumed that requests are independent.

Lee [79] defines seek time, S, in terms of seek distance D as

$$S = \begin{cases} 0 & \text{if } D = 0 \\ a\sqrt{D-1} + b(D-1) + c & \text{if } D > 0 \end{cases}$$

where the square root term models the acceleration and deceleration of the disk head in each seek, and the linear term models the period after the maximum velocity is reached. a, b and c are chosen according to the disk to satisfy the average seek time on a disk ($AvgSeek$), the time to seek from one cylinder to an adjacent cylinder ($MinSeek$) and the time to seek from the outermost cylinder to the innermost cylinder ($MaxSeek$). If there are approximately 200 or more cylinders ($Cyls$) per disk, a, b and c can be approximated as [79]

$$\begin{aligned} a &= (-10MinSeek + 15AvgSeek - 5MaxSeek)/(3\sqrt{Cyls}) \\ b &= (7MinSeek - 15AvgSeek + 8MaxSeek)/(3Cyls) \\ c &= MinSeek \end{aligned}$$

A simplified version defines seek time as only a non-linear function of seek distance [15, 104]. If the seek is sequential, it follows that the seek distance and time are zero, otherwise,

$$S = a + b\sqrt{D} \quad D > 0 \tag{2.21}$$

where a is the arm acceleration time and b is the mechanical seek factor [26]. Bitton and Gray [15] elaborate that b is a constant determined by the disk speed and the track density on the magnetic media. These values are usually calculated experimentally.

Kuratti and Sanders [70] approximate the service time distribution by modelling it as an Erlang-k probability distribution based on the moments of rotational latency, seek time and data transfer time.

Park and Shin [90] present a disk model that incorporates bad sectors on the disk, which cannot be read from or written to. They suggest that ignoring the bad sectors leads to less accurate disk service time models.

Models of Disk Head Positioning Optimisation

Modelling response times for disks with minimised disk head positioning time is analytically difficult, and hence there do not exist many analytical models of this and none for zoned disk drives. Chen et al. [24] present a model for a scheduling algorithm that only minimises seek time. Shriver et al. [108] define the distance (in terms of number of bytes) between two random requests with minimised positioning time as

$$\frac{\textit{no\_of\_Cylinders} \times \textit{Bytes\_per\_Cylinder}}{E[\textit{Queue\_length}] + 2}$$

However, this is not applicable in the context of zoned disks since *Bytes_per_Cylinder* is not constant.

Varki et al. [128] approach their model in a similar manner, defining disk positioning time as a function of queue length. They argue that seek distance can be approximated as the distance from one end of the disk to the other, called $full\_stroke\_distance$, divided by the number of tasks being serviced or awaiting service, $1 + Q$. Thus seek time is defined as,

$$S = a + b\sqrt{\mathit{full\_stroke\_distance}/(\mathit{1} + Q)}$$

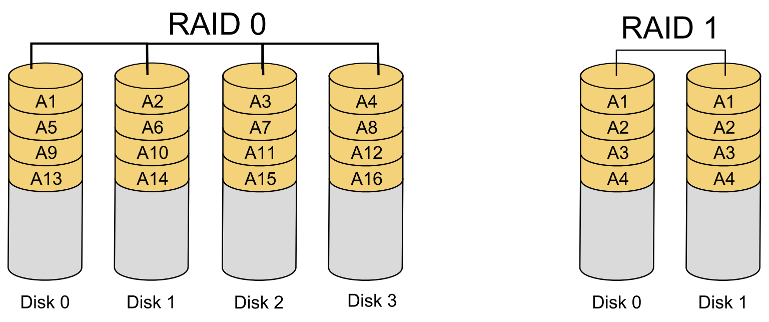
The most comprehensive existing analytical performance model including queue re-ordering is that of Gotlieb and MacEwen [43]. However, this only models SSTF, not SATF. They use the theory of state-dependent queues in their model, whereby the service time distribution can depend on the queue length at the start of a service. This work is primarily based on the research of Harris [48].

There are a number of studies of $M/M/1$ queues with state-dependent service times which could be used to loosely model a disk with RPO, including those by Harris [48] and Morrison [87]. A number of other studies consider the simpler case of two service time states [18, 27, 44]. Brill and Posner [18] allow for different service rates depending on whether or not there are customers queueing behind a request at the start of service. Gray and Wang [44] study the case in which the service rate changes when the queue length exceeds a given number (N) and then changes back when the queue length is less than K ($K \leq N$).

However, no general result exists for response time in $M/G/1$ queues with state-dependent service times.

2.5.3 RAID

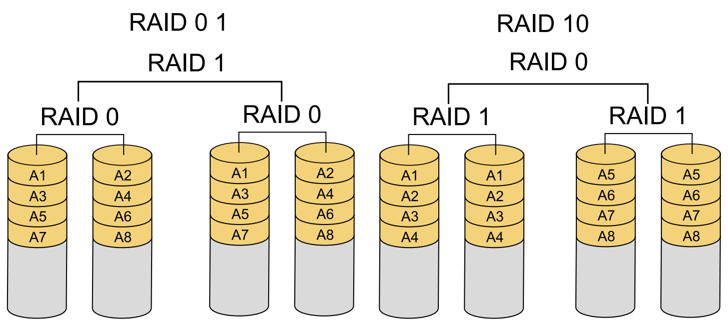
Disks arrays were introduced in the 1980s as a way to utilise parallelism between multiple disks to improve aggregate I/O performance [23]. Disk arrays organise multiple independent disks into a large logical disk unit. By striping data across multiple disks and accessing the disks in parallel, higher data transfer rates are achieved, especially with larger I/O requests. Data striping also ensures that data is balanced across the disks, avoiding data hot spots, where a few disks are constantly accessed while most are not. However, the larger the disk array, the more likely it is that a disk in it will fail. In order to avoid data loss as a result of failures, redundancy can be employed. However, redundancy causes a decrease in performance dependent upon the choice of redundancy scheme. Disk arrays with redundancy are known as *Redundant Arrays of Inexpensive Disks* (RAID). There are numerous different schemes of RAID employing different types of redundancy that are defined as levels in [23, 91].



(a) RAID 0 (b) RAID 1

Figure 2.2: RAID levels 0 and 1 [133].

RAID 0 This array is not redundant, so it is given level zero. It is sometimes called JBOD (Just a Bunch of Disks) [56]. The array uses striping and this, combined with non-redundancy, results in providing the benefits of low cost and high performance but has the disadvantage of low reliability. Disk striping involves



(a) RAID 01 (b) RAID 10

Figure 2.3: RAID levels 01 and 10 [133].

writing data blocks to successive disk array members in a cyclical pattern and is demonstrated in the RAID 0 diagram (Figure 2.2(a)). It has the best write performance as no redundant information is updated. It is widely used in supercomputing environments where performance and space efficiency take precedence over reliability [23].

RAID 1 Redundancy is implemented by storing an exact copy of the contents of each disk (a mirrored disk) in the array. Therefore, RAID 1 uses twice as many disks as RAID 0. RAID 1 offers excellent reliability, but with the worst space efficiency of all RAID levels and less impressive performance for write requests than read requests. RAID 1 does not implement any disk striping [23].

RAID 01 and RAID 10 RAID 01 and 10 combine the mirroring in RAID 1 and striping in RAID 0. RAID 01 is a mirror of stripes and RAID 10 is a stripe of mirrors.

RAID 2, 3 and 4 RAID 2, 3 and 4 do not present any reliability, performance or space efficiency improvements on either RAID 0, 1, 5 or 6 and are therefore not relevant to our research.

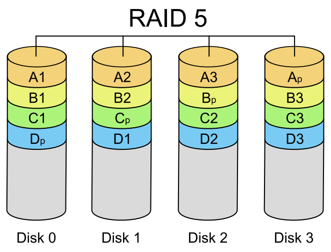
RAID 5 Under this scheme, each stripe is given a parity block that can be referred to if a disk fails. The parity is block-interleaved and distributed across all disks (illustrated in Figure 2.4(a)). Consequently, each disk will contain both parity and data blocks, in contrast to RAID 3 and RAID 4, in which one disk contains only parity blocks and the remaining disks only contain data. This alleviates the performance bottleneck created by all requests accessing the parity disk for write requests in the preceding two RAID levels. Furthermore, because parity is distributed, all disks are able to participate in servicing read requests, not all but one [79]. Redundancy in RAID 5 is achieved at a lower cost than mirroring in RAID 1.

A parity block is the exclusive or (XOR) of all the other data blocks in the stripe. If there are n data blocks, $D_0 \dots D_{n-1}$, then

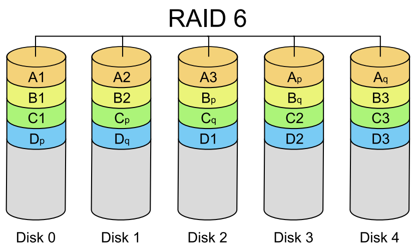
$$P = D_0 \oplus D_1 \oplus \dots \oplus D_{n-1} \tag{2.22}$$

If a request writes to a small proportion of the stripe, the parity block can be updated by pre-reading the data and parity that will be rewritten and using the following formula:

$$new\_parity = new\_data \oplus old\_data \oplus old\_parity \tag{2.23}$$



(a) RAID 5



(b) RAID 6

Figure 2.4: RAID levels 5 and 6 [133].

RAID 6 As larger disk arrays are used, multiple disk failures are possible. RAID 5 only allows for one disk failure at a time, because there is only one parity block per stripe. RAID 6 extends RAID 5, by using two parity blocks, P and Q, so that data can be recovered if two disks fail at once. Anvin [7] explains the procedure of calculating the two parities and regaining data from failed disks based on Plank's work on Reed-Solomon codes [94, 95]. P is calculated in the same way as the parity in RAID 5 by XOR-ing all the data blocks. Q is a Reed-Solomon code that uses the properties of Galois fields.

$$Q = g^0 D_0 \oplus g^1 D_1 \oplus \ldots \oplus g^{n-1} D_{n-1}$$

where g is a generator of the Galois field.

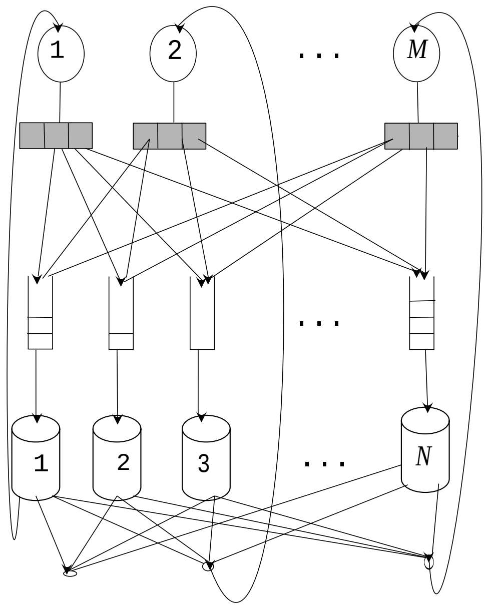
Using these two different parities, any two of the data blocks or parities can be recalculated if they fail consecutively.

Parity Q is computationally more expensive than parity P. RAID 6 suffers from the same performance issues as RAID 5, as well as the added performance issues of calculating the extra parity. It is also slightly less space efficient than RAID 5, but is much more reliable.

Block Sizes Files to be written to disk are split into blocks of a specified constant size (the *stripe width*). These blocks are striped across the disk array according to the chosen RAID level. The block size is chosen dependent on data needs. Bigger block sizes require fewer I/O operations to read and less seeks for non-contiguous blocks. The performance advantages of the bigger block is traded-off with a decrease in space efficiency. If the block size is large, some blocks, particularly when dealing with small files, will be partially empty [121].

2.5.4 Disk Array Model

A disk array receives requests that must be queued for service. Each request is split into a number of tasks equivalent to the number of blocks that need to be read or written in the job. These tasks are queued across the disks, then joined together to fulfil the disk array request. Lee [79] illustrates this with a diagram abstracting some of the features of disk arrays as queues, adapted in Figure 2.5.



M = Number of processes issuing requests
N = Number of disks
Each request split into N tasks

Figure 2.5: Lee's model of a disk array.

Ideally, this scenario should be modelled as a closed queueing network with an N-server fork-join queue, each server modelling a disk in the array. In a fork-join queueing system (Figure 2.6), each incoming job is split into N tasks at the fork point. Each of these tasks queues for service at a parallel service node before joining a queue for the join point. When all N tasks in the job are at the front of their respective queues, they rejoin (synchronise) at the join point. The model also needs to take into account the synchronous nature of tasks. The disk scheduler may re-order the queue to minimise access time, resulting in an element of dependence between a task and the preceding task so a significant amount of synchronicity occurs. In addition, the behaviour of the cache must be modelled in this network. Any analytical result, whether it is exact or an approximation, must be capable of modelling of the order of 50 disk drives in a disk array [54].

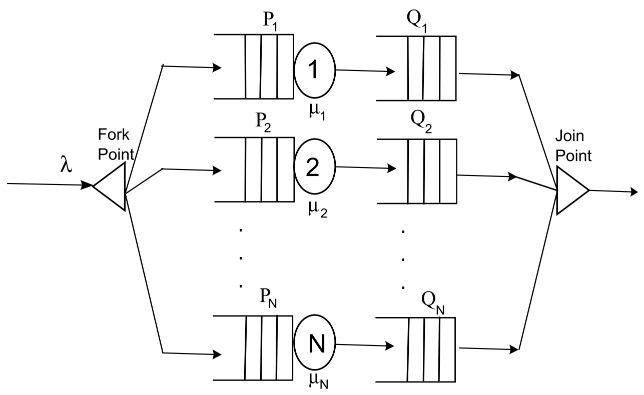


Figure 2.6: Fork-join queueing model.

Lee [79] stated in 1993 that to date, "*a definitive analytic model for block-interleaved redundant disk arrays does not exist*". He attributes this to the difficulties involved in deriving analytical results for the response time of a fork-join queueing system. We have found no indication that this statement no longer holds today. All disk models have had to make compromises or approximations on some of these properties to create an analytical model.

Some of the earlier analytical models ignored queueing completely. These are usually used for calculating expected service time and throughput values. Kim and Tantawi [67] present an approximation for service time distributions striped over N disks, ignoring queueing, redundancy and synchronisation.

An improvement on this are models that involve queueing but not the fork-join synchronisation. Chen and Towsley [26] create analytical models for mirrored clustering and RAID-5 for small and large I/O requests. Mean response times are derived from the disk parameters needed to predict service time on a single disk, the specifications of the RAID level and whether the request is a read or a write. This model is used as an opportunity to discuss scheduling options for these two

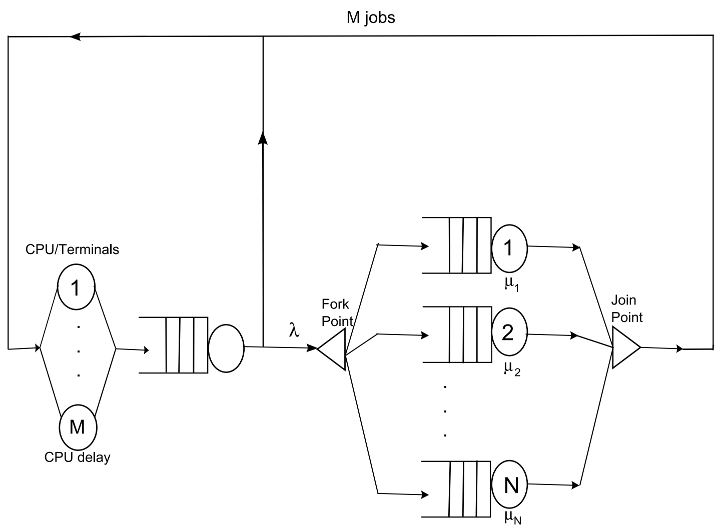


Figure 2.7: Varki et al.'s model of a disk array.

RAID levels. The response time is calculated for each disk drive separately. In RAID 5, each queue is modelled with two classes, one for parity and one for data, giving the parity class non-preemptive priority.

Most recent models include queueing and fork-join synchronisation but make other approximations. Lee and Katz [80] were the first to model disk arrays as a closed queueing network with fork-join synchronisation and a variety of request sizes, presenting an approximation for utilisation. They do not, however, model redundancy or synchronised arrivals or present any results for response times.

Although exact results are only available for a 2-queue system [17], there exist many approximations for response time in a fork-join queue. We focus here on those which were developed for the specific application of modelling a disk array. Therefore, we will primarily focus on the fork-join and disk array modelling work of Varki [123, 124, 125, 126, 127, 128, 129] and various collaborators, and Harrison and Zertal [52, 53, 54].

Varki's work develops results for the fork-join queueing model before applying it to disk arrays. The closed queueing network model dealt with in these works is displayed in Figure 2.7. Jobs accessing the disk array are synchronous, so each I/O request can only be issued when the previous one has completed.

Harrison and Zertal [52] propose an approximation to fork-join synchronisation by deriving the maxima of the service time random variables for each fork. Then each fork-join queue is reduced to only the slowest performing queue in the fork and the mean response times can be easily derived. This is one of the only models that allows disk arrays made up of different RAID levels, a multi-RAID system. They define a method for finding the maximum of multiple random variables [52]. From this they derive an expression for the moments of this maximum random variable.

Let $f_n(\boldsymbol{\alpha}, t)$ be a probability density function that describes the maximum of n independent, negative exponential random variables, with parameters $\boldsymbol{\alpha} = (\alpha_1, \dots \alpha_n)$. The following recurrence relation can be obtained for the Laplace transform of $f_n(\boldsymbol{\alpha}, t)$, $L_n(\boldsymbol{\alpha}, s)$.

$$(s + \sum_{j=1}^{m} \alpha_j) L_m(\boldsymbol{\alpha}, s) = \sum_{j=1}^{m} \alpha_j L_{m-1}(\boldsymbol{\alpha}_{\setminus j}, s) \quad s \geq 0 \tag{2.24}$$

for $1 \leq m \leq n$, where $\boldsymbol{\alpha}_{\setminus j} = (\alpha_1, \dots \alpha_{j-1}, \alpha_{j+1}, \dots \alpha_m)$, $L_0(\boldsymbol{\epsilon}, s) = 1$ and $\boldsymbol{\epsilon}$ is the zero vector.

The proof for this is presented in [52]. The kth moment, $M_n(\boldsymbol{\alpha}, k)$ for $f_n(\boldsymbol{\alpha}, t)$, can be derived by differentiating Equation (2.24) k times using Leibniz's theorem and setting s to 0.

$$M_n(\boldsymbol{\alpha}, k) = \frac{k}{\sum_{j=1}^{n} \alpha_j} M_n(\boldsymbol{\alpha}, k-1) + \frac{\sum_{j=1}^{n} \alpha_j M_{n-1}(\boldsymbol{\alpha}_{\setminus j}, k)}{\sum_{j=1}^{n} \alpha_j} \qquad (2.25)$$

where $n \geq 1$ and $M_0(\boldsymbol{\epsilon}, k) = 0, \forall k \geq 1$; $M_n(\boldsymbol{\alpha}, 0) = 0, \forall n \geq 0$.

These results can be combined into a recurrence relation for approximating the mean value of the maximum of n independent, non-negative random variables with means $\mathbf{m} = (m_1, \ldots, m_n)$. $I(n, \boldsymbol{\alpha}, \mathbf{M})$ is the approximation function, which uses a recurrence relation to generate the mean of the maxima. Then, $I(n, \boldsymbol{\alpha}, \mathbf{M})$ is

$$\begin{aligned} I(k, \boldsymbol{\alpha}, \mathbf{M}) &= \frac{1}{k} \sum_{i=1}^{k} I(k-1, \boldsymbol{\alpha}_{\setminus i}, \mathbf{M}_{\setminus i}) + \alpha_i M_i L_{k-1}(\boldsymbol{\alpha}_{\setminus i}, \alpha_i)/2 \qquad (2.26) \\ & \qquad k = 2 \ldots n \\ I(1, \alpha_1, M_1) &= 1/\alpha_1 \end{aligned}$$

where, $\boldsymbol{\alpha} = (m_1^{-1}, \ldots, m_n^{-1})$, and the second moments are $\mathbf{M} = (M_1, \ldots, M_n)$. The result is exact if the n random variables are exponentially distributed. These results are applied specifically to RAID 01 and 5 in the following ways [139]:

RAID 01

Reads Since each disk is mirrored, the data can be read from either disk. This is decided by a scheduling policy. Possible scheduling policies are:

Random join the read request is assigned to a disk with a defined probability.

Shortest queue the read request is assigned to the disk with the shortest queue.

Minimum seek the read request is directed to the disk with the minimum seek distance.

Reading from both a disk and its mirror produces better performance results for medium and large sized requests. However, for small read requests, the seek and rotation time overheads on the additional disks result in higher response times than searching over half the disks exclusively.

If the mirrored disks are physically located far away from each other, then random join is the best policy as no prior information needs to be gathered from the disks before routing the requests. Harrison and Zertal [52, 53, 54] use random join.

For a one-block read request on disk i, the mean read response time $Z_r(i)$ is defined as

$$Z_r(i) = Q_i + Y_i + T + t$$

where Q_i is the queueing time, Y_i is the disk positioning time of seek time and rotational latency, T is the constant data transfer time and t is the bus transfer time for one data block. Disks are assumed to be unzoned. There are no parallel disk services in a one block read and hence no fork-join synchronisation.

The response time for a multiple block read is the maximum response time of all the disks accessed in the request. The maximum joint queueing and positioning time, which varies according to disk is found. This is then added to the mean transfer time which is assumed to be constant per block and dependent only on the number of blocks being transferred. The recurrence relation defined in Equation (2.26) for finding the maximum of multiple random variables, in this case the random variables $Y_i + Q_i$, $i = 1, \ldots, k$, can then be applied. The mean response time of a request of size B blocks can be estimated as

$$Z_r = \max_{i=1}^{k}(Q_i + Y_i) + \frac{B}{k}(T + t)$$

where $k = \min(B, N)$. The mean read response time can be approximated as

$$\begin{aligned} E[Z_r] &= I(k, \boldsymbol{\alpha}, \mathbf{M}) + \left\lceil \frac{B}{k} \right\rceil (T + t) \\ \alpha_i &= \frac{1}{E[Q_i] + E[Y_i]} \\ M_i &= E[Q_i^2] + E[Y_i^2] + 2E[Q_i]E[Y_i] \quad i = 1, \ldots, k \end{aligned}$$

Writes In mirrored RAID, every write request must write to both the disk and its mirror. The mean response time will be the maximum response time across all these disks. Therefore, if the write request consists of B blocks, it is treated in a similar manner to a $2B$ block read request:

$$E[Z_w] = I(k, \boldsymbol{\alpha}, \mathbf{M}) + \left\lceil \frac{2B}{N} \right\rceil (T + t)$$

and $k = \min(2B, N)$.

RAID 5

Reads Reads are calculated in a similar manner to RAID 01, except that there is no choice of disks and one disk in each stripe is not considered as it contains the parity value, so the disk array is treated as if it only contains $N-1$ disks.

$$E[Z_r] = I(k, \boldsymbol{\alpha}, \mathbf{M}) + \left\lceil \frac{B}{N-1} \right\rceil (T+t)$$

and $k = \min(B, N)$.

Writes The response time of write requests are more complicated to calculate because the time taken to calculate the new parity has to be added to the standard write considerations. The complication derives from the fact that if a partial stripe write takes place, the new parity is calculated by pre-reading the old parity value and existing data first. A full stripe write does not need the old parity to calculate the new value, as all data on the stripe is replaced. To cope with these intricacies, the problem is split up into different cases: full stripes, small partial stripes and large partial stripes.

A differentiation is made between small or large stripes that follow a number of full stripes and small and large stripes that make up the entire request. A small or large stripe is defined by whether the number of blocks to be written, B, is less than, or greater than or equal to $\frac{N-1}{2}$ mod $(N-1)$. Small partial stripes calculate the new parity by pre-reading the old parity and the old data (a read-modify-write, see Equation (2.23)). A large partial stripe reads the data blocks from the disks which will not be rewritten and then calculates the new parity by calculating the exclusive or of this data with the new data (a read-reconstruct-write, see Equation (2.22)). Thus the response times can be calculated as

$$Z_w = \delta_{full}(Z_{full} + Z_+) + (1 - \delta_{full})(\delta_{large} Z_{large} + \delta_{small} Z_{small})$$

where δ is the delta function, 1 if the subscripted statement is true and 0 otherwise. Z_+ refers to a partial stripe following a full stripe and Z_{small} and Z_{large} are exclusive partial stripe writes.

Hence, in order to calculate $E[Z_w]$, mean values need to be found for Z_{full}, Z_{small}, Z_{large} and Z_+.

Z_{full} All the disks are accessed and parity is calculated without any prior information. The mean response time is calculated similarly to a read request:

$$E[Z_{full}] = I(N, \boldsymbol{\alpha}, \mathbf{M}) + \left\lceil \frac{B}{N-1} \right\rceil (T + t)$$

Z_{large} All the disks are accessed once. Either they are accessed to pre-read to update the parity or to write the new data blocks and parity.

The mean response time consists of the time to pre-read the unused blocks for calculating the parity; a full disk rotation to return to the correct rotational position to write, and write transfer time T. The mean response time is

$$E[Z_{large}] = \overline{pre\_read} + R_{MAX} + T$$

$\overline{pre\_read}$ is the mean of the maximum time to read each of the $N - k$ blocks pre-read to calculate parity, where $k = B \; mod \; (N-1)$,

$$\overline{pre\_read} = I(N-k, \boldsymbol{\alpha}, \mathbf{M}) + T + t$$

Z_{small} Each of the $k+1$ disks utilised is accessed twice. The mean response time is then,

$$E[Z_{small}] = \overline{pre\_read} + R_{MAX} + T$$

with

$$\overline{pre\_read} = I(1+k, \boldsymbol{\alpha}, \mathbf{M}) + T + t$$

Z_{+} If a partial stripe write follows a full stripe write, then there is no need to seek for the pre-read as the head is in the correct position. Hence, there is just an additional block read, for the pre-read to re-calculate the parity. Then, as with Z_{small} and Z_{large}, there is a complete disk revolution so that the new data and parity can be written in the same place. Therefore, the additional response time for a partial stripe write following a full stripe write is,

$$E[Z_{+}] = 2T + t + R_{MAX}$$

The scheduling and synchronisation problems of the RAID 5 partial stripe writes are addressed in [25, 70]. Chen and Towsley [25] present a solution by modelling the disk array as a fork-join queue in which each server maintains two queues. One queue contains a read or write request; the other contains parity requests. When a job services, the disk services the read requests required to calculate the new parity and sends the parity write request to the parity queue of the required disk. The parity queue is a priority non-preemptive queue; hence these write request will be immediately serviced as soon as the server finishes with the customer it is servicing and assuming there are no customers ahead of it in the priority queue. The authors define two policies for when the jobs should be added to the parity queue. The *Before Service* policy issues the parity write request as the pre-read requests start servicing. The *After Read-Out* policy only issues the request when all the read requests to calculate the new parity complete. *After Read-Out* will result in a slower response time for each partial stripe write job; however, the *Before Service* policy risks the possibility that the new parity will not have been re-calculated to be written when the write parity request is ready to be serviced, slowing down the disk array. The sophistication of this RAID 5 model is possible because fork-join queueing is not accounted for. Instead, the mean response time of each queue is derived, and then averaged for the mean response time of the disk array.

Kuratti and Sanders [70] suggest a simpler strategy, although it does result in larger response times. When all the read requests have completed, the data and parity write requests are then issued to the back of the required disk queues. The disk is assumed to have an Erlang-distributed service time distribution.

Workload

The arrival rate assumes that each request is split into as many subtasks as there are disks. However this is not the case, as a request splits into a set number of subtasks according to how many blocks there are in the job. Additionally the RAID level and type of I/O operation will affect the number of blocks transferred. Therefore, the arrival rate at a disk is adjusted to account for this variation of arrival rate from the rate that arrives at the array. Harrison and Zertal [54] calculate the arrival rate at each disk by considering the proportion of jobs that are RAID 01 (p_{R1}) or RAID 5 (p_{R5}), the proportion of read (p_r) and write (p_w) requests, and the number of blocks, B that each request consists of. Therefore, the arrival rate to each disk for RAID 01 given an arrival rate to the array of λ is

$$\lambda_{R1} = p_{R1}\lambda(p_r \min(B, N) + p_w \min(2B, N))/N$$

Similarly the disk arrival rate for RAID 5 is

$$\lambda_{R5} = p_{R5}\lambda(p_r \min(B, N) + p_w\kappa(B))(1 + p_w)/N$$

where

$$\kappa(B) = \begin{cases} N & \text{if } 2B \geq N - 1 \\ B + 1 & \text{otherwise} \end{cases}$$

The RAID 5 models described in [25, 70] involve requests being routed back to write new data in partial stripe writes. The combined arrival rates of new and feedback data would not be Poisson. However, it is assumed that partial stripe writes make up a small enough percentage of total requests that the overall arrival process remains approximately Poisson.

Fork-Join Queues

There are other ways to approximate the fork-join queue's response time than Harrison and Zertal's method. Nelson and Tantawi [88] define bounds for the mean response time of a fork-join queue in a system of M/M/1 queues. A scaling approximation is introduced, based on the observation that both the lower and upper bounds of the mean response time grow at the same rate as a function of the number of servers. By running simulations of the fork-join network with different values of N, the mean response time approximation is calibrated to the following result:

$$R_N \approx \left[\frac{H_N}{H_2} + \frac{4}{11}\left(1 - \frac{H_N}{H_2}\right)\rho\right]\left(\frac{12-\rho}{8}\right)\frac{1}{\mu(1-\rho)} \qquad N \geq 2 \tag{2.27}$$

where H_N is the harmonic series, $\sum_{i=1}^{N} \frac{1}{i}$ and $\rho = \lambda/\mu$ here and in all further approximations. Varma and Makowski [130] use interpolation between light and heavy traffic to approximate the mean response time for the same M/M/1 fork-join situation:

$$R_N \approx \left[H_N + \left(\left(\sum_{i=1}^{N}\binom{N}{i}(-1)^{i-1}\sum_{m=1}^{i}\binom{i}{m}\frac{(m-1)!}{i^{m+1}}\right) - H_N\right)\frac{\lambda}{\mu}\right]\frac{1}{\mu-\lambda}$$
$$0 \leq \lambda < \mu \qquad N \geq 2 \tag{2.28}$$

This result can be extended to non-exponential service and arrival times, but in all cases, is only applicable for homogeneous servers.

Varki et al. [126] present another approximation for the same conditions as Equations (2.27) and (2.28),

$$R_N \approx \frac{1}{\mu}\left(H_N + \frac{\rho}{2(1-\rho)}\left(\sum_{i=1}^{N}\frac{1}{i-\rho} + (1-2\rho)\sum_{i=1}^{N}\frac{1}{i(i-\rho)}\right)\right) \tag{2.29}$$

Hartley and David [31, 55] and Gumbel [46] present an upper bound for the mean of the maximum of a set of n independent, identically distributed random variables, X_i as

$$E[X_{(n)}] \leq \mu + \frac{\sigma(n-1)}{\sqrt{2n-1}} \tag{2.30}$$

where μ is the mean of X and σ is the standard deviation. This use of *Order Statistics* to approximate fork-join queues is discussed further in Section 4.2.1.

Thomasian and Tantawi [115] adapt Equation (2.30) to create their own fork-join response time approximation. Using Nelson and Tantawi's method of observing simulation results they present an approximation to the mean response time of a fork-join queue consisting of $M/G/1$ queues in parallel service. Their approximation proposes:

$$R_N(\rho) \approx R_1(\rho) + \sigma_1(\rho)F_N\alpha_N(\rho) \tag{2.31}$$

$R_1(\rho)$ and $\sigma_1(\rho)$ are the mean response time and standard deviation respectively for one $M/G/1$ queue with no fork-join properties. F_N is a constant dependent on the service time distribution of the parallel servers. $\alpha_N(\rho)$ is a linear function generated by simulating the fork-join synchronisation with the specified service time distribution. $\alpha_N(\rho)$ will have to be recalculated and hence resimulated for any change of service time distribution.

Kim and Agrawala [66] present an analysis of the fork-join queue for $M/G/1$ queues with heterogeneous servers. They study the joint distributions of the virtual waiting times of each queue, the time to serve all customers waiting in a queue at a specified time. However the method is significantly more computationally intensive than other results.

Varki [123] extends the Mean Value Analysis approximation technique for closed queueing networks [97] to closed fork-join networks. The Mean Value Analysis iteratively calculates the mean response time, throughput and queue length for each disk in the network. This cannot be directly applied to closed fork join networks, as the Mean Value Analysis only applies to product form networks and fork-join queues are not product form. However, a similar iterative approach can be developed to approximate the fork-join case.

First, in [124] the underlying Markov chain of the fork-join queue is documented. If there are N parallel servers, then the mean response time of the system is the mean response time for the quickest queue added to the response times of the slower queues. From these results, the following bound is calculated for response time:

$$R_N \leq \frac{1}{\mu}[H_N + A_N] \tag{2.32}$$

where R_N is the mean response time for N servers and A_N is the mean number of jobs seen by an arriving job. This can be simplified if the fork-join queue is the only case of queueing in the network. Since this is a closed network, the mean number of jobs seen by an arriving job will be all but itself, namely $m-1$, if there are m jobs in the system. The bound can then be simplified to,

$$R_N \leq \frac{1}{\mu}[H_N + m - 1]$$

Varki's mean value approach describes a situation in which a closed queueing network consists of a number of fork-join queues constructed either in series or parallel with each other. If there are K sub-systems in the network, then using Equation (2.32) iterative mean value equations are derived that calculate the mean response time for each subsystem, $i = 0, \ldots, K$ as

$$R_i(m) \approx \frac{1}{\mu_i}[H_{N_i} + Q_i(m-1)]$$

Q_i is calculated iteratively in a similar manner to mean value analysis, by defining:

$$\begin{aligned} X_i(m) &= \frac{m}{\sum_{n=1}^{K}[R_n(m)V_n/V_i]} \\ Q_i(m) &= X_i(m)R_i(m) \end{aligned}$$

where V_i is the average number of visits per job to subsystem i in the closed queueing network. The iteration is initialised by the result $Q_i(0) = 0$ for all subsystems. Each subsystem must have homogeneous servers.

Generally when modelling a disk array, as seen in Figure 2.7, there are two sub-systems to iterate between: the fork-join synchronisation of the disk array, and a single server queue from which the I/O requests are generated. The service rate of the single server queue will be λ, which is the arrival rate to the fork-join queue.

Varki and Wang [129] apply this method for RAID 5 read requests. The mean value technique is extended in [127] to include multi-class requests in the fork-join queue.

2.5.5 Zoned Disk Drives and Disk Arrays

Zoned Disk Drives

Disk zones enable faster transfer times for data that requires better performance. Since tracks get larger closer to the circumference of the disk, more sectors can be read or written from outer tracks than inner tracks in the same period of time. In the models described so far, block transfer time was defined as a constant. More accurately, block transfer time is a variable, decreasing as the disk cylinder's radius increases.

Ghandeharizadeh et al. [40] describe a strategy for optimal data layout on a zoned disk. They suggest keeping the *hottest* files in the outer zones, defining hot as the files most frequently accessed. This is done by lining up the files in order of heat, with the hottest first, and then laying them out on the disk contiguously, starting with the outermost part of the disk and working inwards.

Once there exists an initial optimal data placement, a plan is needed for dynamic reorganisation when the heat of a file changes. A formula is presented that recalculates the heat of a file, f, after a number of file accesses to f using timestamping [40]. The timestamps, t_i, are recorded in a queue of maximum length K and when the queue is full the new heat is recalculated, taking into account the times of the last accesses:

$$heat_{new}(f) = \frac{1-c}{\frac{1}{K}\sum_{i=1}^{K-1} t_{i+1} - ti} + c \times heat_{old}(f) \tag{2.33}$$

where c is a constant between 0 and 1.

Once it is decided that a file should be promoted to a faster zone, a strategy needs to be defined for migrating the file. In order to move a file into another zone, a file or files may need to be moved out of the destination zone, and since files are of differing size, file fragmentation could occur. Fragmentation gives poor performance when reading a file as the file will not be laid out sequentially. The approach defined to deal with fragmentation is to ensure that the file being promoted remains contiguous, even if it is not contiguous in its original location. The data blocks that it replaces are directly swapped into the original location of the hotter file and may become fragmented. However, these files are accessed less often so their poor performance is less important. If they were to be accessed more often, they would be dynamically migrated to a faster zone and no longer fragmented. To ensure that the performance improvement of the hotter file moving to a faster zone outweighs the bad performance of moving the data in the faster zone to slower zones and possibly fragmenting it, a threshold is defined. A swap occurs only if the improvement in expected service time is larger than this threshold.

Calculations for optimal placement on zoned disks is confronted in [118]. To calculate the optimal placement of a file on a disk, a cost metric is presented in terms of both the best zone to place the data in, and the seek time from the last request on the disk. The cost function is represented by an expression that is ten pages long. To find the optimal placement, the minimum value for the cost function is found using mathematical software.

Zoned Disk Arrays

Applying disk zoning to RAID (Z-RAID), is introduced in [30, 68]. For Z-RAID level 1, the primary copies are all put in the faster zones and the mirrors are placed in the inner zones, with the disks split into an inner and outer zone of equal capacity. Then all read data can be accessed from the outer zones improving performance. Write performance would not improve as each write procedure would involve writing to both inner and outer zones.

Z-RAID level 5 places the parity block on the inner zone and all other data on an outer zone. Hence, if there are n disks in the Z-RAID 5 disk array, then $\frac{1}{n}$th of the total disk space is given to the inner zone for parity, and the remaining $\frac{n-1}{n}$th of space constitutes the outer zone for the data blocks. Similarly, RAID 6, which has two parity blocks per stripe, will have an inner zone consisting of $\frac{2}{n}$th of the disk and an outer zone of $\frac{n-2}{n}$th of the disk. This may be detrimental for write performance as partial stripe write requests involve two accesses to the parity block which will always be on the inner zone.

Thomasian and Han [114] have a similar method to Z-RAID 1 to allow load balancing with mirrored disks. They calculate a pivot point on a disk that will effectively split the disk into two zones, outer and inner. If a block has been written to the outer zone, then it will be written to the inner zone on its mirror. This ensures that all tracks are used on a disk while ensuring that all data can take advantage of the read performance benefits of being on an outer zone, either on the disk or on its mirror. However, it also provides poor write performance as data must always be written to an inner zone.

2.5.6 Caching

Caching can occur on both a disk drive and disk array. Wong and Wilkes [136] state that there is no communication between the disk drive cache and the disk array cache and hence there is often data overlap between the two. There may be performance overheads on the disk array cache from the array controller. [117, 120, 128] study the disk array cache and [14, 47, 108, 136] look at the disk drive cache.

There are two separate caches for reads and writes [14]. A read request will check both caches for a hit. The read cache needs to be updated if data stored in it is rewritten in the write cache. The cache sizes needed to achieve readahead and writebehind are typically tiny compared to the capacity on the disks [136]. Treiber and Menon [117] describe a simulation study of the effects of cache size on performance.

Readahead cache

Figure 2.8 displays a queueing model of the possible routes through the disk array cache of a read request (a model of a disk drive cache would replace the fork-join queue in the diagram with a single M/G/1 queue). If there is a cache miss,

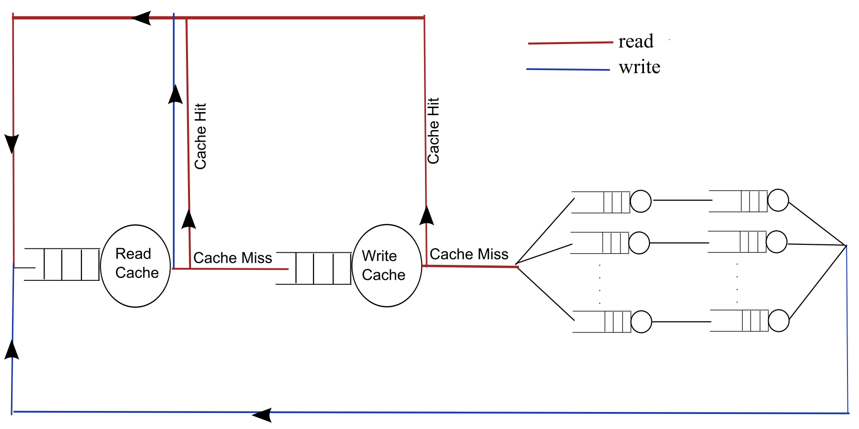


Figure 2.8: Readahead Disk Array cache.

then the cache service time is 0, but the total request response time is slower than a cache hit. If there is a cache hit, the cache has a service time of $t = $ *request_size/cache_transfer_rate* [128]. There is no zoning in the cache, so transfer time for one block is constant. However, the size of request can vary; therefore t varies depending on the distribution of request sizes.

Shriver et al. [108] treat the combined cache and disk as a single server queue. A cache service is followed by a disk service, but the disk gets a reduced workload to account for the cache hits. If there is a cache miss followed by a sequential request, then the sequential request will service in the cache immediately afterwards, during the time that the previous request is still reading from the disk. Therefore the sequential request will also count as a cache miss and have to read the same data from the disk to the cache. This is called a partial cache hit [108]. Therefore, we can not assume that all sequential requests will be cache hits, but must allow a proportion of sequential requests to be cache misses. This proportion is defined as the probability that a sequential arrival occurs before the read request that reads the required data into the cache has finished servicing on the disks.

The probability of a cache hit is dependent on the size of the cache, the amount of spatial locality multiplied by the proportion of read requests and the amount of temporal locality. Shriver et al. [108] ignore temporal locality, focusing on sequential locality. They define the probability of a cache miss as:

$$1/(\min(request\_size + readahead, seq\_requests\_length))$$

If the queue is in equilibrium, the miss rate becomes:

$$\rho/(\min(request\_size + readahead, seq\_requests\_length))$$

Varki et al. [128] include temporal locality in their miss probability metric, but do not define how to calculate it.

$$cache\_miss\_prob = readahead\_miss\_prob \times rereference\_miss\_prob$$

Other works look at calculating cache miss rates in terms of temporal locality, by analysing the Least Recently Used (LRU) scheduling algorithm [10, 137].

Writethrough Cache

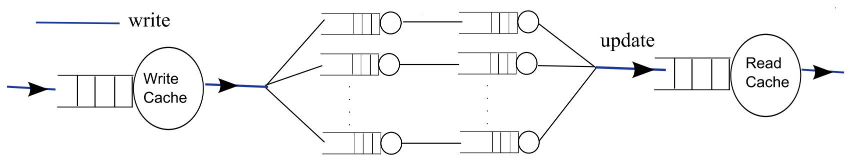


Figure 2.9: Writethrough Disk Array cache.

The writethrough cache is the most simple to model, and is often considered not really caching as it does nothing to improve write performance, just read. Figure 2.9 displays the behaviour of the writethrough cache. Each request is written to both the cache and disk array. If the writethrough cache or the read cache are full and a new request needs to be written to either of them, a scheduling algorithm must be applied to decide which data should be written over in the cache [109].

Writeback Cache

This is also called a writebehind cache [14, 47, 136]. In a writeback cache, a cache hit is defined as the case when there is space in the cache to write an arriving job [85]. The request response time is the time to complete writing to the cache only. When there is a cache miss, a request is blocked from writing to the cache until there is space to write in the cache. There is space to write in the cache if the new data is writing over data already in the cache or there are ‘clean’ blocks in the cache. ‘Clean’ blocks are data that has already been transferred to the disk or disk array. All arriving data in the cache is initially ‘dirty’. Pure writeback caching waits until the cache is full of ‘dirty’ blocks and then starts writing ‘dirty’ blocks to disk. The advantage of this is that requests in the cache can be reordered to be written to disk in the most efficient manner, with the additional benefit that due to the high likelihood of temporal locality, a read request can find a recent write in the cache. However, this results in a lot of blocking in the queue, so more popular is writeback with thresholds, in which a certain threshold is set in the cache. Once a proportion of the cache the size of the threshold is populated by ‘dirty’ blocks, ‘dirty’ blocks start being written to disk. Implementation of thresholds is a trade-off between decreasing blocking and increasingly efficient writes to disk.

Varki et al. [128] model a writeback cache as an $M/M/1/K$ queue, where K is defined as the difference between the cache capacity and the threshold, i.e. the maximum number of blocks that can be written to disk at any time.

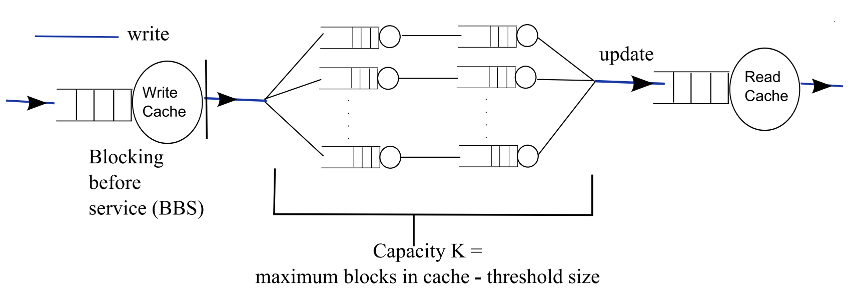

write
Write Cache
Blocking before service (BBS)
update
Read Cache
Capacity K = maximum blocks in cache - threshold size

Figure 2.10: Writeback Disk Array cache.

The cache is modelled as a queue with blocking before service (BBS) [93]. If there are K blocks servicing in the disk array, the cache will block incoming requests until there is space to move more data to disk. However, this model assumes that the population of the cache is always in excess of its threshold.

There are a number of scheduling algorithms used to choose which data to write to the disk when the threshold is passed [47]. The three most popular are Least Recently Used (LRU), Shortest Access Time First (SATF) and Largest Segment per Track (LST). There is another strategy called Piggybacking [14], in which a write request is written after a read cache miss has occurred and the read required is on the same track of the disk as the write request. There are also different strategies for updating the read cache after a write replaces the data [14]. The data could be updated, or just removed from the read cache, or the read cache could be updated after every write (with the newly written data added to the read cache), irrespective of whether it was originally in the read cache or not. The scheduling algorithm chosen will significantly affect the resulting model.

Chapter 3

Disk Drive Model

3.1 Introduction

Over the past two decades disk drive performance improvements have significantly lagged behind all other system component performance enhancements. Consequently, disk system performance is the increasingly dominant factor in overall system behaviour [25, 101, 106]. In turn, improving overall I/O performance depends upon effective predictive models of hard disk drive I/O request response times.

It is important to consider the physical construction of the drive when creating a predictive disk drive performance model. Disk drives consist of a mechanism and a controller. The mechanism contains recording and positioning components and the controller manages the storage and retrieval of data [101]. The disk is split into tracks and each track is populated with sectors. Section 2.5.1 provides a more extensive discussion of the key mechanical components of a hard disk drive.

Hard disk drive technology has advanced enormously in its fifty-three year history, consistently providing lower cost units with higher capacity and better performance. It is fundamental for disk drive performance models to be constantly updated to reflect these technological advances. A significant recent phenomenon in disk drives impacting upon I/O request response time was the introduction of zoned disks (also known as zoned bit recording or zoned constant angular velocity [84]). Zoned bit recording can be found on most hard disk drives manufactured since the introduction of disk zoning in the early 1990s [34, 64]. Prior to the zoned disk it was assumed that there were the same number of sectors on each track throughout the disk. However the circular nature of disks makes it feasible that the longer tracks on the outside of the disk could store up to 50% more sectors than a track near the disk's centre. Figure 3.1 illustrates the difference in sector layout on a disk with and without zoning.

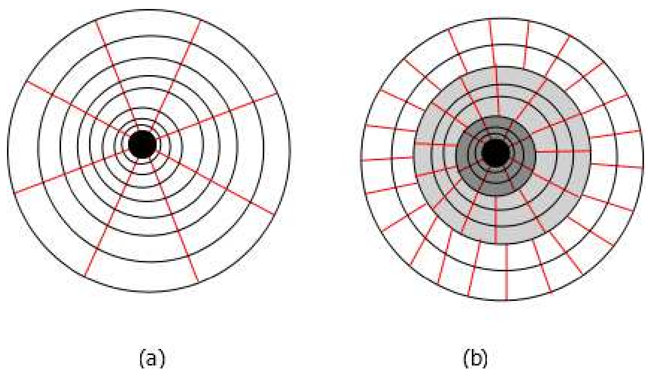


Figure 3.1: A diagram of sector layout on a disk with (a) no zoning and (b) zoning.

Many existing analytical disk drive performance models do not support disk zoning [25, 54, 70, 101, 128]. One of the few analytical zoned disk drive performance models is Zertal and Harrison's [139]. The model we present here is inspired by their original work. In our model, a queueing model abstracts a disk drive to be an $M/G/1$ queue. The queue's service time represents the time to complete operations involved in reading or writing a request from or to a disk. This service time is defined to be the sum of track to track seek time, rotational latency and data transfer time. In order to incorporate these into the queueing model, pdfs must be derived for each of these quantities and convolved to create a service time density function. Service time is described by the random variable X and seek time, rotational latency and block transfer time are described by independent random variables S, R and T. Hence, $X = S + R + T$. If I/O requests are sequential, then seek and rotation time are both zero and each request's service time consists of transfer time only. This is a more straightforward case than random access in which any combination of seek, rotation and transfer times must be considered. Throughout this thesis we assume all requests are independent and random access.

In the context of modern Service Level Agreements, effective performance prediction must provide the ability to reason not only about mean response times, but also higher moments and percentiles of response time. Therefore, our target throughout this work is the full cdf of I/O request response time, from which all of the previous measures can be easily derived. Analytical queueing network models of disk drive and RAID performance [26, 54, 79, 124, 128] developed prior to our work approximate only the mean response time of the system.

In this chapter we present the derivation of an analytical queueing model for response times on a single hard disk drive. In Sections 3.2, 3.3 and 3.4, the density functions are derived for seek time, rotational latency and data transfer time whose convolution forms the service time density of a disk drive. Using these, an analytical queueing model of a single disk drive is described in Section 3.5. Additionally, we develop a corresponding disk drive simulation to compare with the analytical model. Section 3.6 introduces this disk drive simulation based on the queueing based simulator *JINQS* [38].

Each stage of deriving both analytical and simulation models is validated against device measurements to ensure accuracy throughout. Unless otherwise stated, we use a Seagate ST3500630NS disk to validate the models. Each disk has 60801 cylinders. A sector is 512 bytes and we have approximated, based on measurements from the disk drive, that the time to write a single physical sector on the innermost and outermost tracks are 0.012064ms (t_{max}) and 0.005976ms (t_{min}) respectively. The time for a full disk revolution is 8.33ms. A track to track seek takes 0.8ms and a full-stroke seek requires 17ms for a read; the same measurements are 1ms and 18ms respectively for a write [105].

Eventually we will extend the disk model for use in a RAID system. On a RAID system blocks are defined to be a particular size and striped across all the disks in the array. In preparation for this, in the disk model we refer to transfer sizes in terms of blocks rather than sectors. We define the block size as 128KB which is the stripe width on an Infortrend A16F-G2430 RAID system. Therefore there are 256 sectors per block.

To obtain response time measurements from this system, we implemented a benchmarking program that issues read and write requests using a master process and multiple child processes. These child processes are responsible for issuing and timing I/O requests, leaving the master free to spawn further child processes without the need for it to wait for previously-issued operations to complete. In order to validate the analytical model effectively, it was necessary to minimise the effects of buffering and caching as these are not currently represented in the model. We therefore disabled the system's write-back cache, set the read-ahead buffer to 0 and opened the device with the `O_DIRECT` flag set. We also disabled the operating system's I/O scheduler. For each of the experiments presented here, 100 000 requests were issued and the resulting means, variances and cumulative distribution and density functions of the response times were calculated using the statistical package `R` [96].

3.2 Seek Time

A seek, S, is the time taken for the disk head to move from the cylinder where it is currently located, C_1, to the cylinder containing a target sector, C_2. We define a random variable, $D = |C_1 - C_2|$, as the seek distance. Seek time can then be defined in terms of seek distance. Specifically [25],

$$S(D) = \begin{cases} 0 & \text{if } D = 0 \\ a + b\sqrt{D} & \text{otherwise} \end{cases} \tag{3.1}$$

where a is the arm acceleration time and b is the mechanical seek factor [25]. Here we define them in terms of minimum and maximum seek times which are constants that are provided by the disk manufacturer or could be measured from the disk. Using Equation (3.1) and setting $D = 1$ and then $D = Cyls - 1$ (the maximum and minimum seek distances), a and b can be defined as follows:

$$\begin{aligned} a &= \frac{minseek\ \sqrt{Cyls-1} - maxseek}{\sqrt{Cyls-1}-1} \\ b &= \frac{maxseek - minseek}{\sqrt{Cyls-1}-1} \end{aligned}$$

where $Cyls$ is the total number of cylinders on the disk, $minseek$ is the track-to-track seek time and $maxseek$ is the full-stroke seek time.

The disk model must reflect the layout of a zoned disk accurately. As cylinders get closer to the disk edge, their circumference increases and the number of sectors per cylinder increases. Therefore, a random request has an increased probability of being directed to a sector on an outer cylinder. Let C be a random variable representing the cylinder number of a randomly selected disk sector. The pdf of C can be approximated by assuming that the number of sectors per track increases linearly and approximating the discrete C as a continuous random variable [139]. That is,

$$f_C(x) = \frac{\alpha + \beta x}{\gamma} \qquad x = 0, 1, \ldots, Cyls - 1 \tag{3.2}$$

with constants α, β and γ defined as:

$$\begin{aligned} \alpha &= \frac{SEC[0]}{spb} \\ \beta &= \frac{SEC[Cyls-1] - SEC[0]}{(Cyls-1)\ spb} \\ \gamma &= \alpha(Cyls-1) + \frac{\beta}{2}(Cyls-1)^2 \end{aligned}$$

where $SEC[0]$ and $SEC[Cyls - 1]$ are the number of sectors on the innermost and outermost tracks respectively and spb is the number of physical sectors per logical block. α represents the number of logical blocks on the innermost track and β charts the rate of increase in blocks per cylinder.

Often disk specifications supplied by manufacturers do not provide information on the number of sectors on the innermost and outermost tracks. However, it is possible to take measurements from the disk drive to ascertain the mean transfer time to a single sector on the innermost (t_{max}) and outermost (t_{min}) tracks. α and β can then also be calculated from the transfer time equation detailed later in Equation (3.4) using the transfer time parameters which allows $SEC[0]$ and $SEC[Cyls - 1]$ to become obsolete.

This model assumes cylinder capacity increases linearly. In reality it increases in steps as there are collections of tracks that all have the same number of sectors. An example of the effectiveness of this assumption in one particular case can be seen in Figure 3.2. We compare the modelled density function of which cylinder a randomly selected sector on the disk will be (Equation (3.2)) with a density function calculated from zoning information provided by the manufacturer for a Fujitsu MAN3367FC disk drive [39]. We observe adequate agreement between the linear model and reality. Furthermore, the use of a linear approximation in the analytical model avoids future complications in the derivation of the seek time density.

The pdf of seek distance is calculated by assuming the two random variables C_1 and C_2 as two distinct cylinder numbers, and calculating the seek distance between all possible cylinder numbers. This is split into two terms, one for the case when

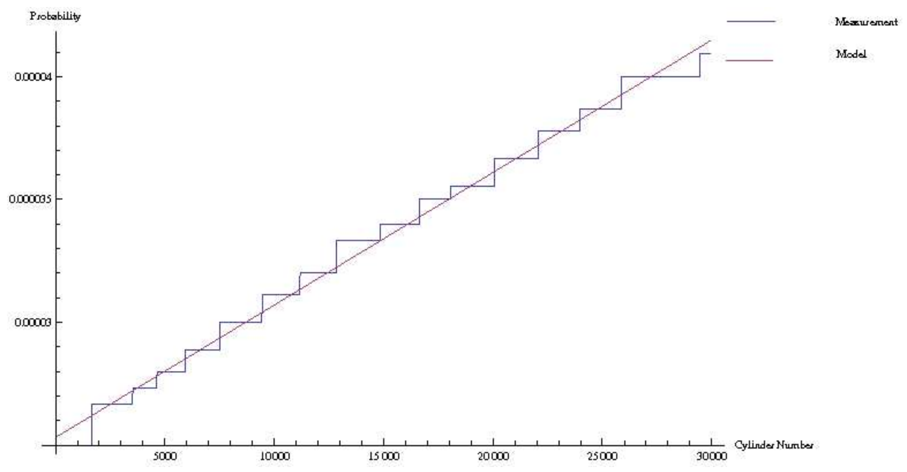


Figure 3.2: Comparison of density functions for cylinder layout for both model and measurement on a Fujitsu MAN3367FC disk drive.

$C_1 \leq C_2$ and one for the case where $C_1 > C_2$ [139]:

$$f_D(x) = \int_0^{Cyls-1-x} f_C(y) f_C(x+y) dy + \int_x^{Cyls-1} f_C(y) f_C(y-x) dy$$

This can be shown to equate to

$$f_D(x) = A + Gx + Ex^3 \qquad 0 \leq x \leq Cyls - 1$$

where,

$$\begin{aligned} A &= \frac{V(Cyls-1)}{3\gamma^2} \\ G &= -\frac{V + \beta^2(Cyls-1)^2}{3\gamma^2} \\ E &= \frac{\beta^2}{3\gamma^2} \\ V &= 6\alpha^2 + 6\alpha\beta(Cyls-1) + 2\beta^2(Cyls-1)^2 \end{aligned}$$

The cdf of seek time, $F_S(t)$, can be defined in terms of the cdf of D, $F_D(x)$, as [25]:

$$F_S(t) = \begin{cases} F_D(0) & 0 \leq t < a + b \\ F_D\left(\left(\frac{t-a}{b}\right)^2\right) & \text{otherwise} \end{cases}$$

3.3 Rotational Latency

Rotational latency, R, is the time to rotate to the angle of a target sector. R has a uniform distribution with a range between 0 and the time for a full disk revolution, R_{max} [26]; thus,

$$f_R(x) = 1/R_{max} \qquad 0 \leq x \leq R_{max} \tag{3.3}$$

3.4 Data Transfer Time

The time to transfer k logical blocks on cylinder x of a zoned disk can be approximated as [139]

$$t(x) = \frac{k\ spb\ R_{max}}{\alpha + \beta x} \tag{3.4}$$

$\alpha + \beta x$ approximates the number of sectors on a cylinder; therefore $\frac{R_{max}}{\alpha+\beta x}$ provides the time to write to each sector on cylinder x.

As mentioned in Section 3.2, often the number of sectors in each zone are not provided in the manufacturer's disk specification. If this is the case, α and β can be calculated using Equation (3.4). Disk specifications often provide the minimum and maximum logical block transfer times and if they are not provided these parameters can be easily calculated from device measurements. We define t_{min} as the minimum data transfer time and t_{max} as the maximum data transfer time for one 512 Byte sector. Subsequently, t_{min} is the time to transfer to the outermost track and t_{max} is the time to transfer to the innermost track. Thus substituting into Equation (3.4),

$$\begin{aligned} t_{max} &= \frac{R_{max}}{\alpha} \\ t_{min} &= \frac{R_{max}}{\alpha + \beta(Cyls - 1)} \end{aligned}$$

α and β can subsequently be redefined as

$$\alpha = \frac{R_{max}}{t_{max}} \qquad \beta = \frac{R_{max}}{Cyls - 1}\left(\frac{1}{t_{min}} - \frac{1}{t_{max}}\right)$$

We assume that T_k is independent of seek time and seek distance. Denoting T_k as the random variable of the time to transfer k blocks of data, its cdf is

$$\begin{aligned} F_{T_k}(t) &= \int \mathbb{P}(T_k \leq t \mid C = x) f_C(x)dx \\ &= \int \mathbb{P}(x \geq \frac{k\ spb\ R_{max}}{\beta t} - \frac{\alpha}{\beta}) f_C(x)dx \\ &= \int_{\max(\phi_k(t),0)}^{Cyls-1} f_C(x)dx \end{aligned} \tag{3.5}$$

where

$$\phi_k(t) = \frac{k\ spb\ R_{max}}{\beta t} - \frac{\alpha}{\beta}$$

calculates the minimum cylinder number it is possible to transfer k logical blocks of data to or from in less than t ms. The solution of the integral in Equation (3.5) is a function of t with a domain bounded between the minimum and maximum possible k-block transfer times.

Equation (3.5) expands to

$$F_{T_k}(t) = \begin{cases} 0 & t < k\ spb\ t_{min} \\ \frac{\alpha}{\gamma}(Cyls - 1) + \frac{\alpha^2}{2\beta\gamma} + \frac{\beta(Cyls-1)^2}{2\gamma} - \frac{k^2 R_{max}^2 spb^2}{2t^2\beta\gamma} & t < k\ spb\ t_{max} \\ 1 & \text{otherwise} \end{cases} \quad (3.6)$$

This model can be validated by setting the number of sectors to transfer to at a large enough number that the seek and rotation time will become insignificant in comparison to the transfer time. We thus write 100MB in each request to random locations on a single ST3500630NS disk connected directly to a test machine. We also ensure that no queueing occurs by waiting until a request completes before issuing another. Figure 3.3 compares cdfs produced by the analytical data transfer time model and device measurements. The effects of disk zoning are clearly evident in the measurements, which are a close match to the analytical model. It can be observed that the measurement cdf increases by steps reflecting the disk zones, whereas the model curve is smooth because of our linear approximation.

3.5 Disk Drive Model

A disk drive is modelled as an $M/G/1$ queue in which the arrival process corresponds to I/O requests arriving at a disk and the service process corresponds to the time to position the disk head and transfer the data. In order to find the density or distribution of the response time, W, we must use the Pollaczek-Khintchine trans-

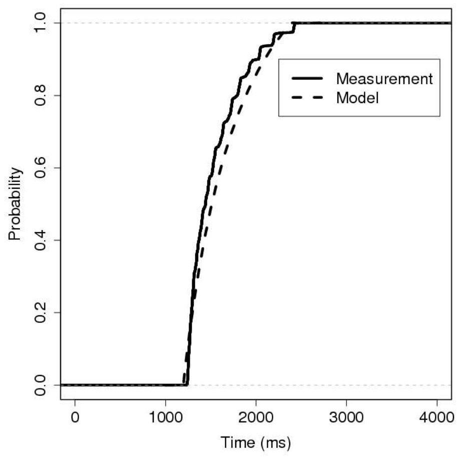


Figure 3.3: Comparison of modelled and measured cdfs of zoned data transfer time for 100MB requests on a single disk.

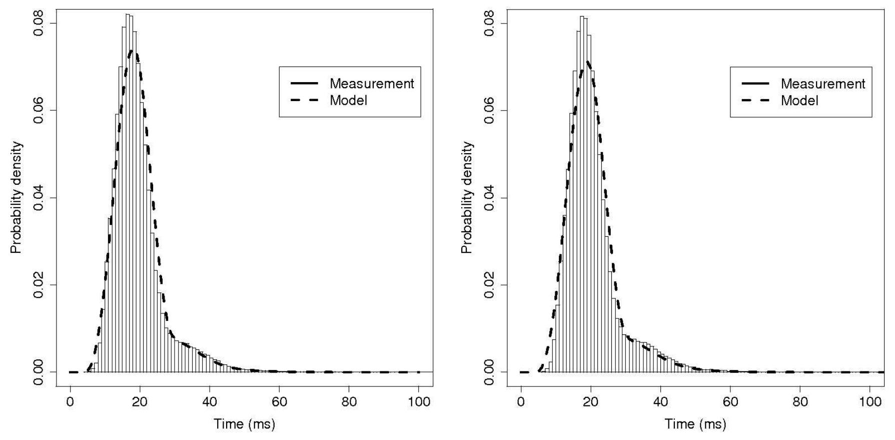
form equation for the Laplace transform of response time of $M/G/1$ queues [51]:

$$W^*(\theta) = \frac{(1-\rho)\theta X^*(\theta)}{\lambda X^*(\theta) - \lambda + \theta} \tag{3.7}$$

$X^*(\theta)$ is the Laplace transform of the service time pdf, which is the product of the Laplace transforms of the pdfs of S, R and T_k, i.e. $S^*(\theta)R^*(\theta)T_k^*(\theta)$. We assume S, R and T_k are independent of each other. Also, $\rho = \frac{\lambda}{\mu}$, where λ is the I/O request arrival rate to the disk and μ is the mean service rate, which in our case is given by $\frac{1}{E[R]+E[S]+E[T_k]}$. As $W^*(\theta)$ is unlikely to have an analytical inversion, we invert it numerically using the Euler method [2] to obtain the response time pdf $f_W(t)$. The cdf $W(t)$ is also easily obtained by inverting $W^*(\theta)/\theta$.

The mean response time, variance and further moments of response time can be calculated in two ways. Either Equation (3.7) can be differentiated n times at the point $\theta = 0$ to give a recursive formula for the response time in terms of moments of the service time, or the moments can be calculated numerically from the inverted density function. Our experience in comparing these two methods is that the difference between results from both methods is marginal and they are equally computationally intensive.

In Figure 3.4 we compare the response time density functions of this model with measurements from the disk drive. It is fundamental to have good agreement between model and measurement in this most basic case in order to be able to extend the model for more sophisticated workloads and RAID systems. In this case we use the same request size (256KB) and a small arrival rate (0.01 requests per ms) for read and write requests. For this disk we observe excellent agreement for both read requests (Figure 3.4(a)) and write requests (Figure 3.4(b)). Table 3.1 provides means and variances for these two cases.



(a) read requests (b) write requests

Figure 3.4: Measured and modelled densities for workloads of constant 256KB size on a single disk with arrival rate $\lambda = 0.01$ requests/ms.

	Mean measured (ms)	Mean model (ms)	Variance measured (ms^2)	Variance model (ms^2)
read	19.34	19.55	52.25	49.19
write	20.54	20.32	59.19	54.19

Table 3.1: Response time mean and variance comparison for measurement and model of 256KB read and write requests on a single disk with arrival rate 0.01 requests/ms.

3.6 Disk Drive Simulation

Simulations are often used to validate analytical models. Additionally, they provide the ability to replicate the details of the scheduling algorithms and mechanical behaviour of real disk systems while analytical models can only abstract these details. Consequently simulations can aid the development of more realistic analytical models. Also simulations, although slower than analytical models, are faster to run than taking response time measurements from a disk drive. It is also expensive to invest in different types of disk drives to validate the disk model against device measurements whereas a validated simulation can replicate different disk drives with a simple change of parameters.

To improve and validate our analytical model we have developed a disk drive simulation. Our simulation aims to be an elegant high-level framework that avoids very detailed low-level device simulation (e.g. as performed by the DiskSim [19] and RaidSim [22] simulators) and which can be simply parameterised from disk drive technical specifications. Our simulation takes as input identical parameters to the analytical model to ensure that the two are easily comparable. The simulation generates as its primary output metric the cumulative distribution function of I/O request response time.

Our queueing based simulator models a single disk drive as an $M/G/1$ queue and uses the *JINQS* Java-based simulator [38] to simulate this. A queueing network in *JINQS* is defined in terms of *Node*, *Link* and *Customer* classes. *JINQS* defines a queueing network as a collection of *Node*s that are connected to each other by *Link*s. The network is populated by *Customer*s. A *Source* node injects *Customer*s into a network with the inter-arrival time between customers sampled from a specified probability distribution. In the case of the $M/G/1$ queue, this will be the exponential distribution. The *Source* node is *Link*ed to a *QueueingNode*. *QueueingNode*s allow customers to queue for service with a first-come first-served queueing discipline. The service time is decided by sampling from another specified probability distribution. For an $M/G/1$ queue, the *QueueingNode* is *Link*ed to a *Sink* node where customers are absorbed and customer response times are measured. Figure 3.5 shows a UML class diagram displaying the relationships between the key classes in *JINQS*.

JINQS contains inbuilt distribution samplers for many well-known probability distributions. However, to sample the disk service time, it was necessary to create distribution samplers for seek time, rotational latency and data transfer time based on the analytical probability distributions presented in Sections 3.2, 3.3 and 3.4 using the inverse transform and acceptance-rejection random-variate generation techniques [13]. The simulation generates service times by summing samples from these three distribution samplers.

Similarly to the measurements, each simulation run involves $100\,000$ requests being issued by the simulator and give 99% confidence intervals with a half width of the order 0.01. Figure 3.6 compares cdfs of device measurements, the analytical

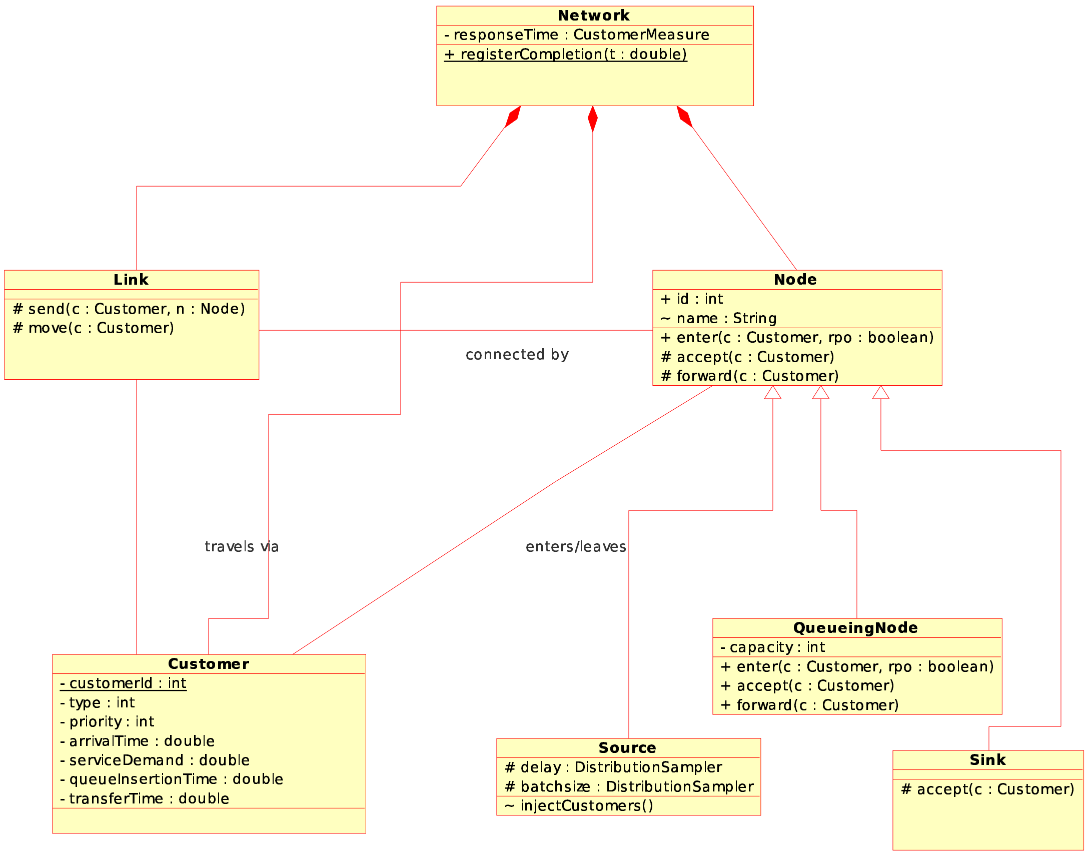
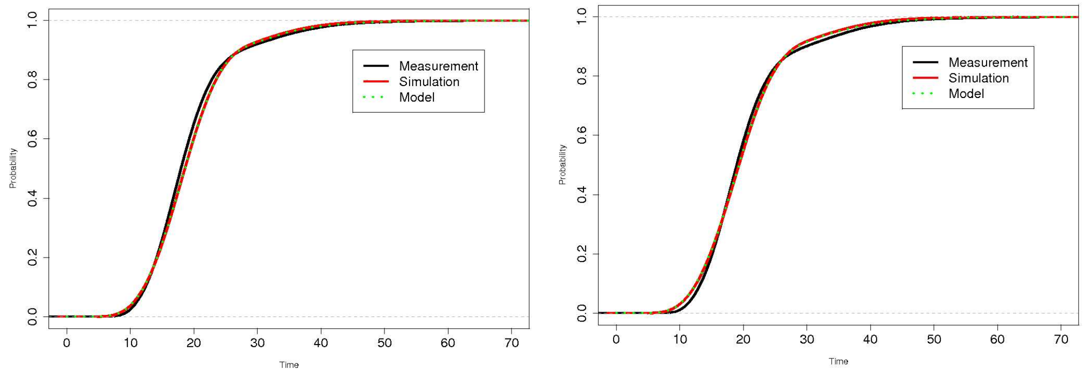


Figure 3.5: *JINQS* M/G/1 queue simulator class diagram.



(a) read request

(b) write request

Figure 3.6: Response time distributions of measurement, simulation and model for workloads of constant 256KB size on a single disk with arrival rate $\lambda = 0.01$ requests/ms.

model and simulation for read and write requests with the same parameters as the pdfs in Figure 3.4. The simulation and analytical models produce almost identical cdfs which are very close to the measurement cdf, particularly for read requests.

Chapter 4

RAID Model

4.1 Introduction

RAID systems are fundamental components of almost all modern data storage systems due to their ability to increase storage infrastructure performance and reliability in a cost-effective manner. As a result they are now widely deployed at every level from personal home storage devices to enterprise-scale storage area networks. Choice of RAID level can critically affect the performance delivered by a storage system. It is therefore important to be able to predict performance of a given RAID configuration for various I/O workloads.

RAID systems consist of a controller managing multiple disks. In the previous chapter a performance model was presented for a single disk. This must be extended to reflect the requirements of a RAID system. The presence of fork-join synchronisation in RAID system behaviour indicates that the most appropriate queueing network model of a RAID system is the fork-join queueing network [80]. There do not exist any exact analytical results for a fork-join queue consisting of more than two queues [79]. However, there exist numerous approximations and bounds of varying accuracy and computational intensity [12, 25, 26, 54, 66, 80, 82, 88, 115, 116, 123, 124, 125, 126, 127, 128, 129]. In Section 4.2 we present an analytical response time approximation of the fork-join queue and compare it to some existing fork-join response time approximations [52, 88, 126, 130].

Combining the fork-join approximation and disk drive model we can develop an analytical RAID model. The various RAID levels provide either performance, redundancy or space efficiency advantages or a combination of these over single disks. We specifically focus on modelling RAID levels 0, 01 and 5. RAID 0 is disk striping without any redundancy. RAID 01 is a mirror of stripes implementing properties of both RAID 0 and RAID 1. The presence of striping gives RAID 01 a performance advantage over RAID 1. The RAID 01 model can easily be modified to model RAID 10 (stripe of mirrors) which we describe in this chapter. We do not model RAID levels 2, 3 or 4 as RAID 01 and 5 present performance, reliability and cost-effective advantages over all these levels. RAID 5 is distributed single parity block. RAID 6 is distributed double parity blocks and hence has a reliability advantage over RAID 5. However, there is no uniform RAID 6 configuration and they tend to be system and manufacturer specific. Consequently, it is not possible at present to create a general performance model for RAID 6 as we do for the lower RAID levels.

Simply abstracting each I/O request as a customer in a fork-join queue is not sufficient to model the intricacies of RAID systems. Although the fork-join queue mimics striping, modifications must be made to the model to represent requests that split into any number of subtasks, mirroring and parity calculations. There is a small body of work that addresses this [25, 26, 54] but no conclusive analytical method for modelling response time distributions of RAID based on fork-join queueing for RAID levels 0, 01 and 5 for any size of request exists. This chapter presents such a model. We introduce the extensions and modifications to the combined fork-join and disk models that are necessary to create an effective RAID response time model in Section 4.3. Finally, we introduce a RAID simulation to compare to our analytical model in Section 4.4.

To provide confidence in our simulation and analytical models, we validate them against device measurements. Our experimental platform consists of an Infortrend A16F-G2430 RAID system containing Seagate ST3500630NS disks. The specifications for these disks are listed at the start of Chapter 3. As mentioned in the previous chapter, the stripe width on this array is set at 128KB which we define as a single block size.

4.2 The Fork-Join Queue

As described in Section 2.5.4 and illustrated in Figure 2.6, in a fork-join queueing system, each incoming job is split into N tasks at the fork point. Each of these tasks queues for service at a parallel service node before joining a queue for the join point and rejoining when all tasks have completed service.

As discussed in Chapter 2, it is difficult to model moments of job response time in a fork-join synchronisation analytically. For more than two parallel servers, approximations exist for the mean response time of homogeneous servers. Ideally a universal solution or accurate approximation is needed to solve for moments of job response time in generic fork-join networks. The closest to this is Varki's modification of mean value analysis applied to closed fork-join networks [123], which approximates mean values only.

Here, fork-join queues are being studied specifically for the purpose of modelling disk arrays. Therefore certain constraints on the fork-join model are preferable, for an accurate disk array model. Each disk in the array is modelled as one of the parallel servers of the fork-join queue. The service time of a disk drive is dependent on the disk cylinder seek time and rotational latency and is unlikely to be distributed exponentially. Hence, the disk array requires a fork-join model with $M/G/1$ parallel queues. The service time distributions and mean service times on each disk may not be identical; hence any analytical approximation should allow for heterogeneous parallel servers. Finally, a disk array could consist of tens of disk drives so the analytical approximation needs to be capable of generating results quickly for a large number of disks.

In order to solve fork-join queueing networks analytically, most results assume that the response times of parallel queues are independent and identically distributed (iid). The arrival rate to the fork is λ and mean service rate for each queue is μ.

One way of approximating fork-join synchronisation, is to model a similar network called the split-merge queue exactly (see Figure 4.1) [16, 36]. In the split-merge queue, a job splits into N tasks which are serviced in parallel. Only when all the tasks finish servicing and rejoin can the next job split into tasks and start servicing. This will lead to slower mean response times than its fork-join equivalent.

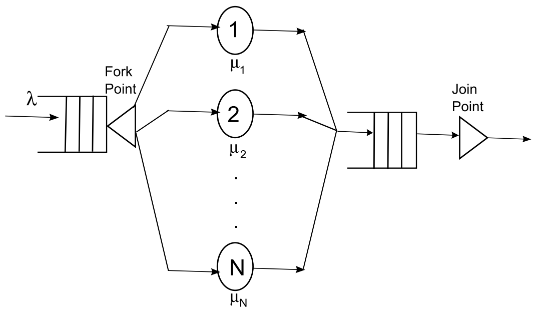


Figure 4.1: Split-merge queueing model.

4.2.1 The Maximum Order Statistic

Here, we present an alternative to Harrison and Zertal's method [52], by finding the mean of the maximum of a set of random variables by utilising the properties of Order Statistics [31, 103]. This will give an exact solution for the response time of a split-merge queue which is a conservative approximation of the response time of a fork-join queue.

Definition It is possible to reorder any random variables, $X_1, X_2, \ldots, X_n$ as $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$, where $X_{(1)} \leq X_{(2)} \leq \ldots \leq X_{(n)}$. Then $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$ are the order statistics of $X_1, X_2, \ldots, X_n$.

The maximum of n random variables using order statistics is $X_{(n)}$, the maximum order statistic. The mean value of this maximum and further moments can be found if the cdf of $X_{(n)}$ is calculated.

$$F_{X_{(n)}}(x) = \mathbb{P}(X_{(n)} \leq x) = \forall i \mathbb{P}(X_{(i)} \leq x)$$

Thus, if $X_1, X_2, \ldots, X_n$ are independent and identically distributed with cdf $F(x)$, $F_{X_{(n)}}(x) = (F(x))^n$.

If the random variables are independent but not identically distributed, and X_i has cdf $F_i(x)$,

$$F_{X_{(n)}}(x) = \prod_{i=1}^{n} F_i(x)$$

The mean of the maximum of n independent random variables with pdf $f_i(x)$, is then

$$E[X_{(n)}] = \int_{-\infty}^{\infty} x \left(\sum_{i=1}^{n} \frac{f_i(x)}{F_i(x)} \right) \prod_{i=1}^{n} F_i(x) dx \tag{4.1}$$

If the random variables are iid, equation (4.1) simplifies to

$$E[X_{(n)}] = n \int_{-\infty}^{\infty} x f(x)(F(x))^{(n-1)} dx \tag{4.2}$$

Further moments, M_k, can be calculated,

$$M_k = E[X_{(n)}^k] = n \int_{-\infty}^{\infty} x^k f(x)(F(x))^{(n-1)} dx$$

These results always give exact solutions to the mean of the maximum random variable, irrespective of the distribution of the random variables.

4.2.2 Validating the Fork-Join Queue Approximation

To validate this approximation, it is compared to simulation and analytical results from Harrison and Zertal [52] (Equation (2.26)) who present an approximation to the mean response time of a split-merge queue. Each simulation run involves $100\,000$ requests being issued by the simulator, giving 98% confidence intervals with a half width of the order 0.01. The analytical results are obtained using Mathematica [135].

Before looking at queueing response times, we start with the simpler case of finding the maximum of some well known random variables. Harrison and Zertal's method and the maximum order statistic produce identical results for exponential random variables, since Equation (2.26) is exact for exponential random variables. Table 4.1 compares the two models and simulation results for an Erlang-k distribution, with parameter, $\lambda = k$. The column HZ contains the results from the approximation in Equation (2.26) and OS contains the exact method in Equation (4.2). The approximation suffers with low variance as $N \to \infty$, with a constantly increasing percentage error for larger N. Equation (4.2) consistently delivers better percentage errors with no clear performance deficits.

For a high variance situation, a Pareto distribution is used. Table 4.2 compares the two models and simulation results for a heavy-tailed Pareto-β distribution. This has cdf $F_P(x) = 1 - \alpha(x+\gamma)^{-\beta}$, with $\alpha = \gamma^{\beta}$ and $\gamma = \beta - 1$. The maximum order statistic consistently outperforms Harrison and Zertal's approximation.

N	Exp-1	Erlang-2				
		Sim	HZ	% err	**OS**	% err
1	1.000	1.003	1.000	-0.334	**1.000**	-0.334
2	1.500	1.373	1.375	0.135	**1.375**	0.135
4	2.083	1.772	1.813	2.265	**1.774**	0.089
8	2.718	2.182	2.288	4.881	**2.180**	-0.078
16	3.381	2.588	2.786	7.648	**2.587**	-0.035

N	Erlang-3					Erlang-4				
	Sim	HZ	% err	**OS**	% err	Sim	HZ	% err	**OS**	% err
1	0.999	1.000	0.062	**1.000**	0.062	0.999	1.000	0.060	**1.000**	0.060
2	1.271	1.313	3.281	**1.313**	3.281	1.195	1.281	7.207	**1.273**	6.127
4	1.546	1.677	8.448	**1.630**	5.153	1.380	1.609	16.64	**1.544**	10.6230
8	1.806	2.074	14.84	**1.945**	7.147	1.555	1.966	26.43	**1.808**	13.993
16	2.061	2.488	20.74	**2.254**	8.562	1.716	2.339	36.30	**2.063**	16.820

Table 4.1: Comparison of simulation and models for means of Erlang-N random variables (low-variance).

N	Exp-1	Pareto-4					Pareto-5				
		Sim	HZ	% err	**OS**	% err	Sim	HZ	% err	**OS**	% err
1	1.000	1.004	1.000	-0.381	**1.000**	-0.381	0.994	1.000	0.614	**1.000**	0.614
2	1.500	1.579	1.750	10.82	**1.571**	-0.509	1.567	1.667	6.350	**1.556**	-0.707
4	2.083	2.327	2.625	12.81	**2.319**	-0.345	2.269	2.444	7.744	**2.266**	-0.132
8	2.718	3.261	3.577	9.698	**3.255**	-0.184	3.129	3.290	5.173	**3.129**	-0.001
16	3.381	4.394	4.571	4.027	**4.395**	0.023	4.153	4.174	0.512	**4.149**	-0.096

Table 4.2: Comparison of simulation and models for means of N Pareto random variables (high-variance).

$M/M/1$ **Queues**

To further validate the use of the maximum order statistic of response time as an approximation for fork-join queue response time, we compare it to the fork-join queue response time approximations described in Section 2.5.4. All these approximations only apply to the mean response time of an $M/M/1$ fork-join queue. Since $M/M/1$ queues have both exponential arrival and service time distributions, the approximation in Equation (2.26) (Harrison and Zertal's approximation [52]) is identical to the maximum order statistic. Figure 4.2 compares analytical approximations for the mean response time for a fork-join queueing network of $M/M/1$ queues with a fork-join queueing simulation. The results were calculated with an arrival rate $\lambda = 1$ request per time unit and a service rate $\mu = 1.1$ requests per time unit for each server and the number of servers (N) varying from 1 to 25. The analytical methods are the maximum order statistic (OS), Nelson and Tantawi's approximation [88] in Equation (2.27) (NT), Varma and Makowski's approximation [130] in Equation (2.28) (VM) and Varki et al.'s approximation [126] in Equation (2.29) (VMC). All these results are compared to a simulation of the network, with the line labelled SIM. Each simulation run involves $100\,000$ requests being issued by the simulator and then each run is replicated 30 times.

The mean of the maximum order statistic, which gives exact results for a split-merge queue but only approximates the fork-join model, performs worst out of all the approximations for the $M/M/1$ queue. This could be expected as the split-merge model waits for all parallel servers to finish servicing before a new job begins service and will hence be slower than the fork-join model. However, it must be noted that all these approximations except OS and VM are limited to $M/M/1$

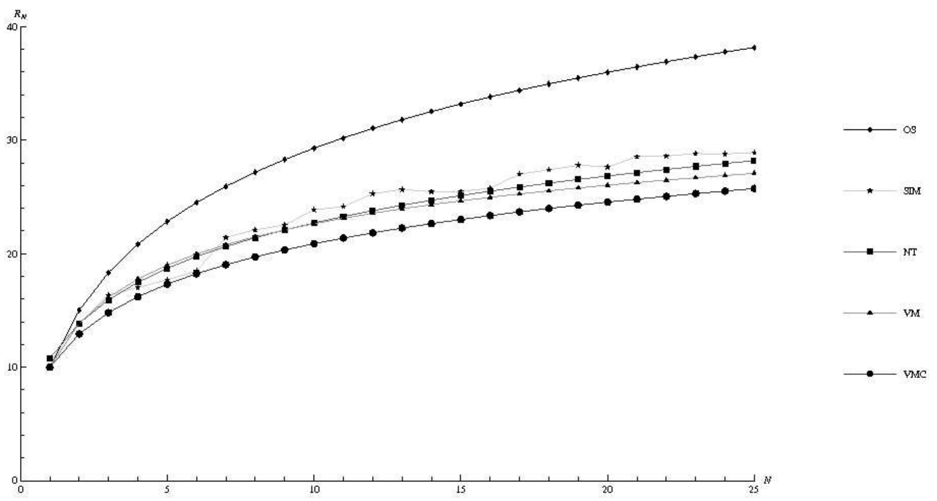


Figure 4.2: Mean response time R_N for $M/M/1$ fork join-queue with N queues, $\lambda = 1$, $\mu = 1.1$.

queues and cannot be extended for $M/G/1$ queues which is a requirement of a RAID model.

$M/G/1$ **queues**

The benefits of the maximum order statistic become more apparent with an $M/G/1$ queue. The approximations defined for $M/M/1$ queues (equations (2.27), (2.28) and (2.29)) only apply for $M/M/1$ queues and most approximations that exist for $M/G/1$ queues are computationally intensive [130], or reliant on simulation results to provide parameters (Equation (2.31) [115]). Varki presents an approximation for mean response time only using a modified mean value analysis in [123]. Harrison and Zertal's method is an approximation of the split-merge queue, whereas the maximum order statistic is exact for the split-merge queue.

Figure 4.3 plots mean response time for an N-server fork-join queue. The ser-

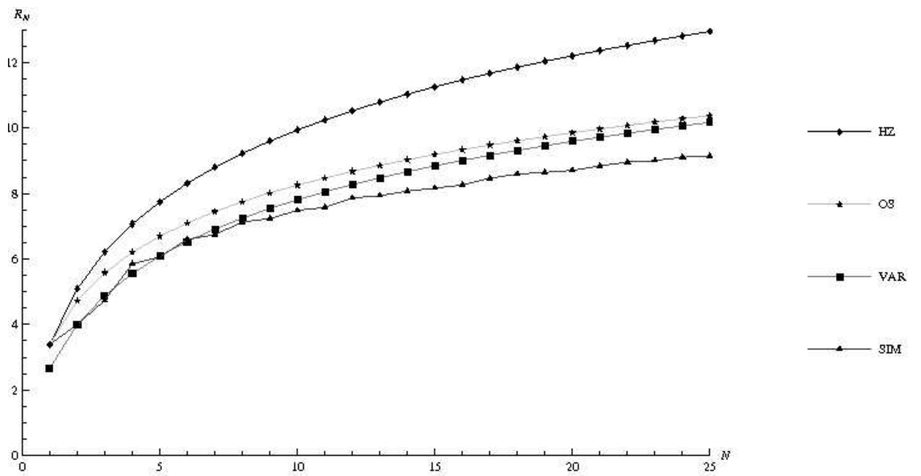


Figure 4.3: Mean response time R_N for Erlang-2 $M/G/1$ fork-join queue with N queues, $\lambda = 0.1$, $\mu = 0.375$.

vice time distribution has an Erlang-2 distribution with mean service rate at each server of 0.375 requests per time unit and Markovian arrival rate 0.1 requests per time unit. The graph compares Harrison and Zertal (HZ), the mean of the maximum order statistic (OS), Varki's mean value analysis method [123] (VAR) and a simulation (SIM). The mean of the maximum order statistic can be seen to be a good approximation of response time for a fork-join queue of $M/G/1$ queues and has the additional benefit of being capable of providing the full probability distribution of response time.

A Large Number of Parallel Queues

Disk arrays often consist of tens of individual disk drives. Any analytical approximation of a disk array needs to quickly and accurately calculate the mean response time as the number of parallel queues increase. Finding the mean of the maximum order statistic is computationally fast for a large number of parallel queues. However, simulating large fork-join queues is very slow. Therefore, Figures 4.2 and 4.3 only show results up to 25 disks. To show how these results compare as the number of queues gets very large, tables are presented for the cases when there are 40 and 50 parallel queues. Table 4.3 shows mean response times for the $M/M/1$ fork-join queue described above with arrival rate 1 request per time unit and service rate 1.1 requests per time unit . Table 4.4 displays mean response times for the $M/G/1$ fork-join queue with Erlang-2 distributed service times, arrival rate 0.1 requests per time unit and service rate 0.375 requests per time unit. Both tables are labelled with the same keys that are used in Figures 4.2 and 4.3. We observe similar trends for the accuracy of the approximations as were visible for a smaller number of queues. Significantly, a larger number of queues make some of the approximations highly computationally intensive (e.g. VM), but this is not the case for the maximum order statistic.

N	Simulation	98% Confi dence Interval half width	OS	NT	VM	VMC
40	32.195	1.201	42.785	31.055	29.196	28.263
50	32.450	0.684	44.992	32.42	30.171	29.469

Table 4.3: Comparison of simulated and modelled mean response times for $M/M/1$ parallel queues with many servers.

N	Simulation	98% Confi dence Interval half width	OS	HZ
40	10.0126	0.0160	11.481	14.521
50	10.406	0.0178	12.0054	15.27

Table 4.4: Comparison of simulated and modelled mean response times for Erlang-2 $M/G/1$ parallel queues with many servers.

Heterogeneous Servers

Heterogeneous parallel servers arranged in a fork-join queueing network is another situation in which approximating the response time with the maximum order statistic is an improvement upon other analytical approximations. The fork-join approximations discussed in Section 2.5.4 are only applicable for homogeneous servers. Figure 4.4 studies the mean response time for an $M/M/1$ fork-join queue with heterogeneous servers. It plots the mean response time for an N branch fork-join queue in which each server has a mean service rate of $1.1 + 0.2i$, where $i = 0, 1, \ldots, N - 1$ and arrival rate 1 request per ms. The line SIM represents a queueing simulation of response time for $N = 1, \ldots, 16$ and the line OS is an approximation using the mean of the maximum order statistic. To show that the approximations for response time in fork-join queues with homogeneous servers cannot approximate the heterogeneous result, two lines are plotted assuming homogeneous servers, using Nelson and Tantawi's approximation (see Equation (2.27)), which was shown in Figure 4.2 to be the most accurate analytical approximation of $M/M/1$ fork-join synchronisation. Firstly, we approximate the heterogeneous servers by assuming homogeneous servers with the minimum, and hence the slowest service rate, 1.1. This is displayed in the line NT $\mu = 1.1$. Secondly in line NT, we define the service rate of the homogeneous servers as the mean of all the service rates on the heterogeneous servers.

The maximum order statistic approximation results stay consistently closer to the fork-join simulation than the homogeneous cases. Furthermore, both attempts at approximating parallel systems with heterogeneous servers by assuming homogeneous servers have increasingly large percentage errors as N increases.

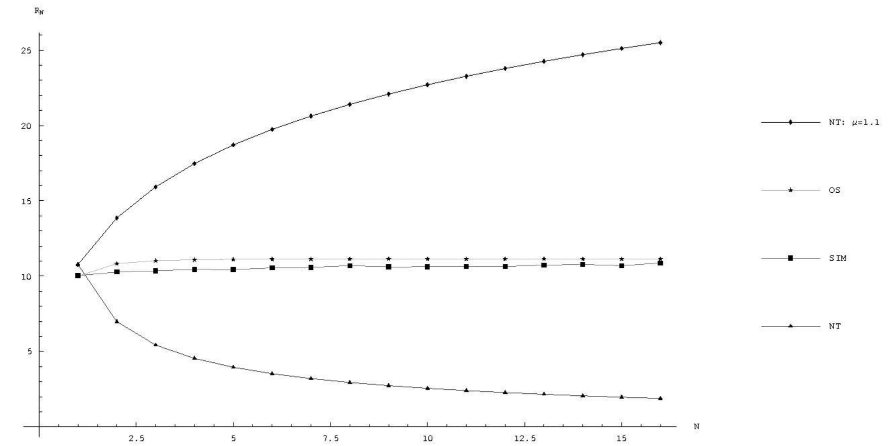


Figure 4.4: Mean response time R_N for an heterogeneous $M/M/1$ fork-join queue with N queues.

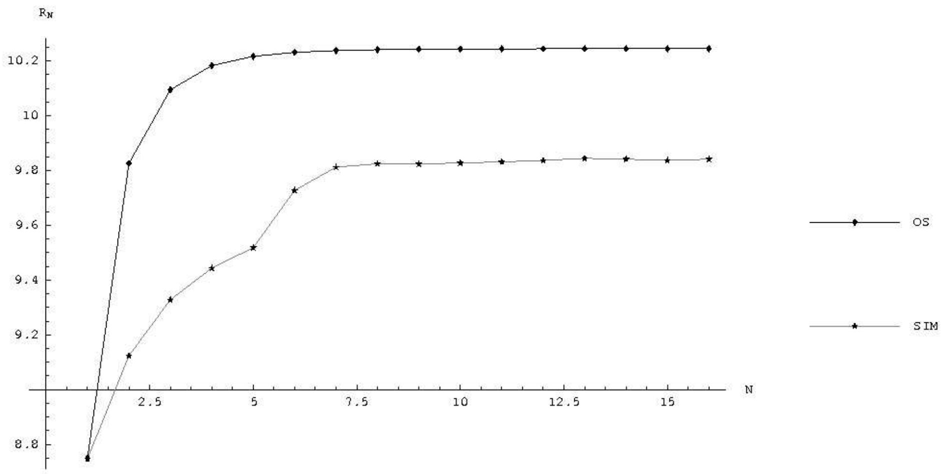


Figure 4.5: Mean response time R_N for an heterogeneous $M/G/1$ fork-join queue with N queues and an Erlang-2 service time distribution, with mean $0.2 + 0.1N$.

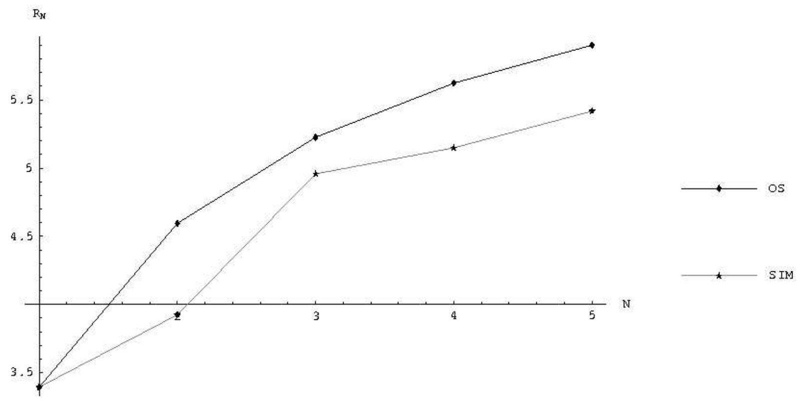


Figure 4.6: Mean response time R_N for an heterogeneous $M/G/1$ fork-join queue with N queues and a service time distribution of Erlang-$(N+1)$,with mean 0.375.

Figures 4.5 and 4.6 compare simulation results with only the mean of the maximum order statistic for response times in $M/G/1$ heterogeneous fork-join queues. Figure 4.5 charts the mean response time for N $M/G/1$ queues with an Erlang-2 distribution, but with a mean service rate that varies according to N ($\mu = 0.2 + 0.1N$). Figure 4.6 keeps the mean service rate constant at 0.375, but varies the service time distribution according to N. The service time distribution is Erlang-$(N + 1)$. In both cases the arrival rate is 0.1 requests per ms.

In Figures 4.4 and 4.5, the mean response time tends to a constant value as N increases. This is because queues are added to the network with increasingly fast mean response times. The slow response times of the queues initially added to the network have a larger effect on the overall mean response time of the fork-join network.

4.2.3 Choosing a Fork-Join Approximation

The benefits of the maximum order statistic as an approximation to predict the response time of a fork-join queue have been documented in this section. To summarise:

- Although not the most accurate approximation of fork-join queue response time, the mean of the maximum order statistic adequately represents fork-join queue response time with $M/M/1$ queues.
- To model a disk array, the fork-join queue must consist of $M/G/1$ queues. In this case, the maximum order statistic is one of few available approximations, fast and not reliant on prior simulation results.
- The maximum order statistic of response time continues to model the response time of a fork-join queue well for heterogeneous servers and a large number of servers.
- Most fork-join queue approximations only calculate the mean response time. The maximum order statistic derives the full response time distribution from which further moments and quantiles of response time can be obtained.

4.2.4 Fork-Join Simulation

The simulation used throughout this section is an extension of the *JINQS* queueing software. In Section 3.6 we described how *JINQS* single queue simulation is specified in terms of *QueueingNode*, *Link* and *Customer* classes. *QueueingNode*s are connected by *Link*s to create a network of queues. Response times measurements are obtained by recording the time each *Customer* spends in the network.

To extend this simulation for fork-join queueing, we introduce *ForkLink* and *JoinLink* classes to extend the *Link* class and a *ForkedCustomer* class to extend the *Customer* class. Figure 4.7 is a UML diagram displaying the relationship of these new classes with the original *JINQS* simulation. The *Source* node now links to the *QueueingNode*s with a *ForkLink*. At the *ForkLink* a new *ForkedCustomer* is created for each subtask in an arriving *Customer*, each with a reference to that original *Customer*. These *ForkedCustomer*s are sent to one of the n single queues. These queues could be either $M/M/1$ or $M/G/1$. When a *ForkedCustomer* leaves a queue, it is sent to the *JoinLink*, which collects all *ForkedCustomer*s. When all the *ForkedCustomer*s that reference a particular *Customer* have arrived, the original *Customer* is sent on its way to the *Sink* node and all of its *ForkedCustomer*s are destroyed.

4.3 RAID Model

Modelling RAID systems as a fork-join queue suffices to calculate the response time cdf for read or write requests to an n-disk RAID 0 system in which each request consists of a multiple of n blocks. However, not every I/O request leads to an access to all disks, being influenced by I/O request size and type, and also by RAID level. Here, we introduce models of RAID levels 0, 01 and 5, for both read and write requests.

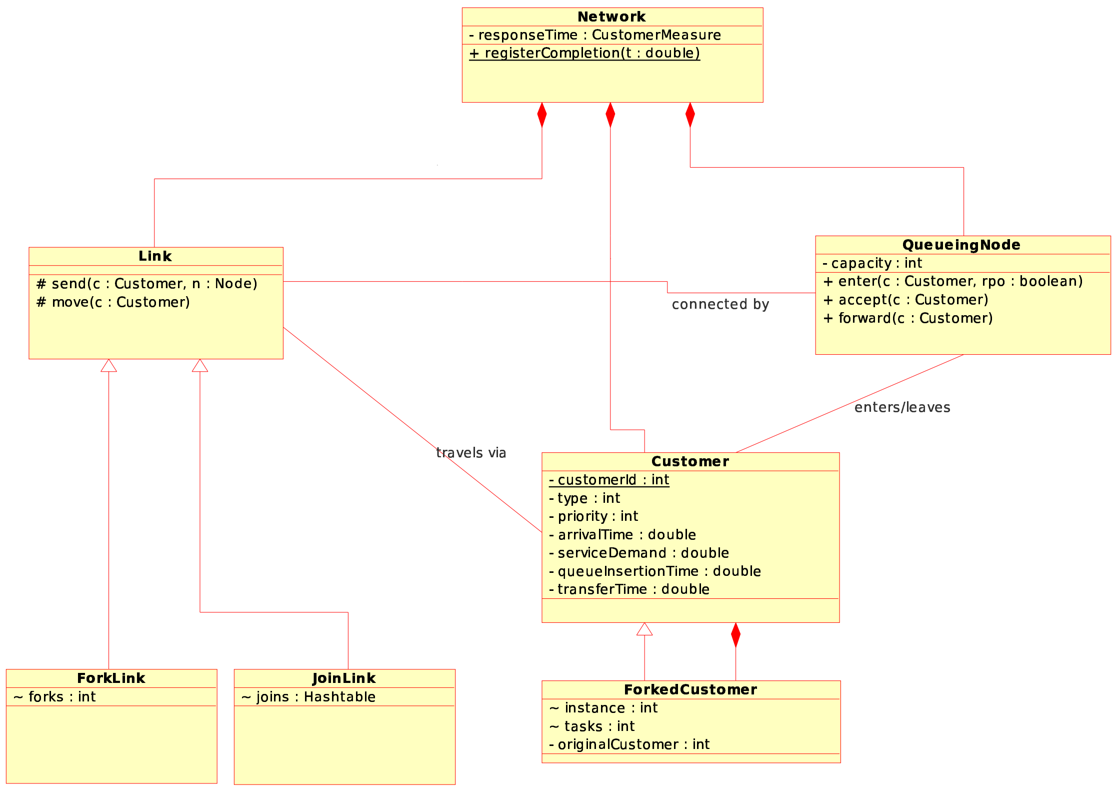


Figure 4.7: Fork-Join simulator class diagram.

Our model is initially designed to accept a homogeneous stream of I/O requests of a given size and type. We further assume that all the service time distributions on all disks are identically distributed. For the sake of notational simplicity, let $W_d(t, \gamma, \mu)$ define the cdf of the response time distribution of a single $M/G/1$ queue (disk), γ the arrival rate at an individual disk and μ the mean service rate. We assume there are n disks in the array and that the arrival rate of logical I/O requests to the disk array as a whole is λ. The service time parameters (seek time, rotational latency and data transfer time) are defined in the disk model description in Chapter 3.

4.3.1 RAID 0

RAID 0 is striping with no redundancy. In terms of its performance model a RAID 0 read or write operation is modelled in an identical way to a RAID 01 read operation (see Equation (4.4)) despite the physical differences between these three operations. The only difference between a RAID 0 read and write performance model are the read or write specific parameters such as track-to-track seek time and full stroke seek times, but the RAID striping operations are identical. For a b-block request, if $b < n$ only b disks are utilised at any time. To account for this, we view the system as a b-queue fork-join queue in these cases. The arrival rate at the disks would then need to be modified since each request would only arrive at b of the n disks. If $b \geq n$, all n disks of the array are utilised, but the service time must account for the number of blocks transferred to each disks on average during a request.

Therefore the cdf of the response time distribution for a b-block read or write request on a RAID 0 system using the maximum order statistic approximation of the fork-join queue is:

$$W_{R0}(t) = \begin{cases} \left(W_d\left(t, \frac{\lambda b}{n}, \frac{1}{E[R]+E[S]+E[T_1]}\right)\right)^b & \text{if } b < n \\ \left(W_d\left(t, \lambda, \frac{1}{E[R]+E[S]+E[T_{\frac{b}{n}}]}\right)\right)^n & \text{otherwise} \end{cases} \tag{4.3}$$

Validation

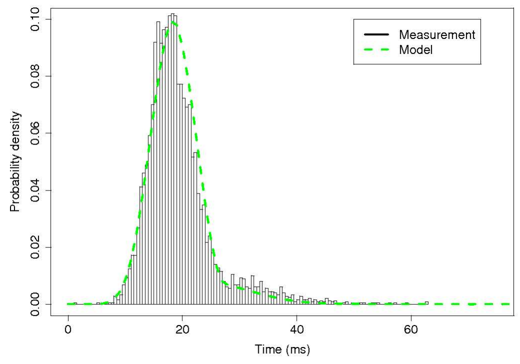
Figure 4.8 compares pdfs for measurement and model of response time on 4-disk RAID 0 with different arrival rates and request sizes. We observe excellent agreement between model and measurement for read requests and adequate agreement for write requests. This slight lag is possibly due to a RAID system overhead on write requests which is not attributed for in the model.

4.3.2 RAID 01

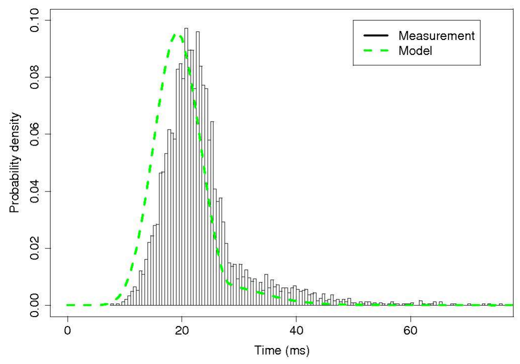
Read Requests

Assuming an efficient RAID controller, a b-block read on RAID 01 can read data from either primary or mirror disks. With $b \geq n$, we thus utilise all n disks of the array (and not $\frac{n}{2}$ disks) to give better performance results for medium and large sized requests. The array controller chooses whether to read from a disk or its mirror. In our model we assume this choice is made randomly.

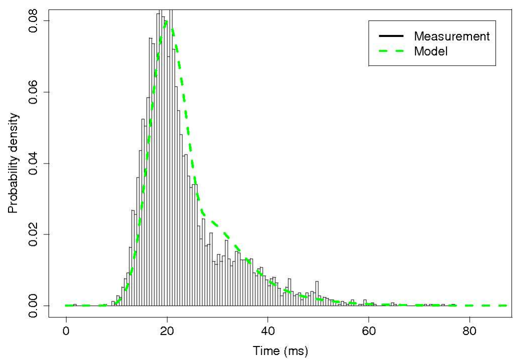
Therefore, similar to a RAID 0 request, the cdf of the response time distribution



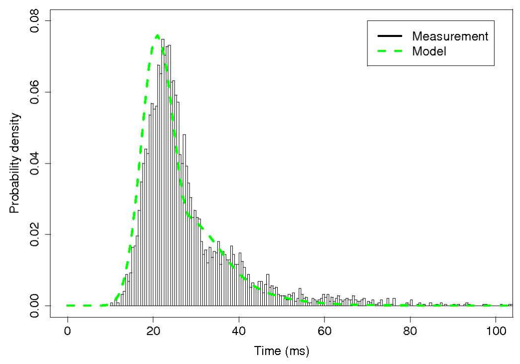
(a) read requests, $b = 2$, $\lambda = 0.01$



(b) write requests, $b = 2$, $\lambda = 0.01$



(c) read requests, $b = 3$, $\lambda = 0.02$



(d) write requests, $b = 3$, $\lambda = 0.02$

Figure 4.8: 4-disk RAID 0 b-block request response time pdfs for arrival streams of reads or writes with arrival rate λ requests/ms.

for a read on a RAID 01 system using the maximum order statistic approximation of the fork-join queue is:

$$W_{R01_r}(t) = \begin{cases} \left(W_d\left(t, \frac{\lambda b}{n}, \frac{1}{E[R]+E[S]+E[T_1]}\right)\right)^b & \text{if } b < n \\ \left(W_d\left(t, \lambda, \frac{1}{E[R]+E[S]+E[T_{\frac{b}{n}}]}\right)\right)^n & \text{otherwise} \end{cases} \tag{4.4}$$

Write Requests

A b-block write must account for each request being written on both primary and mirrored disks, therefore $2b$ blocks are written in each request. The corresponding response time cdf is defined as:

$$W_{R01_w}(t) = \begin{cases} \left(W_d\left(t, \frac{2\lambda b}{n}, \frac{1}{E[R]+E[S]+E[T_1]}\right)\right)^{2b} & \text{if } 2b < n \\ \left(W_d\left(t, \lambda, \frac{1}{E[R]+E[S]+E[T_{\frac{2b}{n}}]}\right)\right)^n & \text{otherwise} \end{cases}$$

In a RAID 10 system data is striped across half the disks and mirrored on the other half of the disks (i.e. the odd numbered disks are the original and the even numbered disks are mirrors as opposed to RAID 01 where the first $\frac{n}{2}$ disks are the original and the remaining $\frac{n}{2}$ disks are mirrored). However, an analytical model of RAID 01 would focus only on the performance implications of this arrangement which are identical to those of RAID 01. Therefore RAID 10 could be modelled using the same analytical models as RAID 01.

Validation

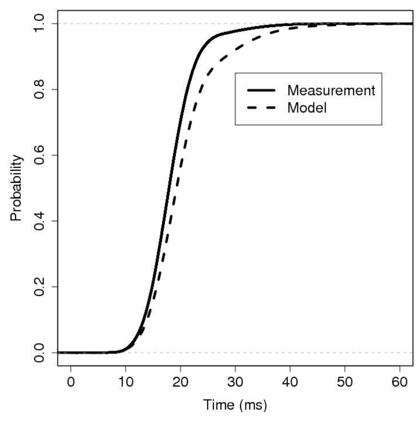
To support these analytical models for RAID 01, we validate them against device measurements. We do not have access to a RAID 10 system to validate our model in that case. Figure 4.9(a) displays the measured and modelled cdfs for the response time of 256kB read requests on a four disk RAID 01 system, and Figure 4.9(b) shows the corresponding cdf for write requests. We observe good agreement between model and measurement. Table 4.5 further illustrates the accuracy of the model, comparing mean and variance for the model and measured results for the cases illustrated in Figure 4.9 and showing excellent agreement in all cases.

Figure 4.10 shows measured and modelled mean response times of reads and writes for both 4-disk and 8 disk RAID 01 for a light load of $\lambda = 0.01$ requests/ms and a variety of request sizes. For write requests agreement between model and measurement is excellent, even for large block sizes.

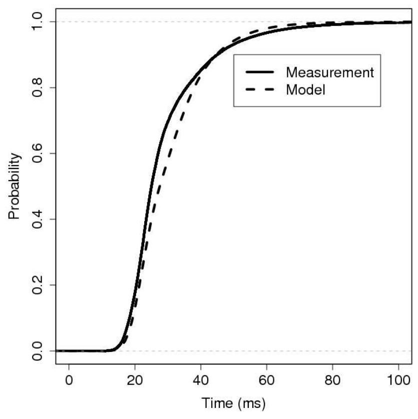
	Blocks	Mean measured (ms)	Mean model (ms)	Variance measured (ms^2)	Variance model (ms^2)
read	2	18.3	20.4	20.3	40.1
write	2	29.0	30.3	164.9	119.8

Table 4.5: Response time mean and variance comparison for measurement and model of read and write requests on 4-disk RAID 01 with an arrival rate of 0.02 requests/ms.

For read requests we observe good agreement, with a slight tendency for the model to overestimate for small block sizes. Table 4.6 contains means and variances for the cases presented in Figure 4.10(a). For larger block sizes, the model tends to increasingly underestimate the measurements. This behaviour is interesting be-



(a) 2-block read request

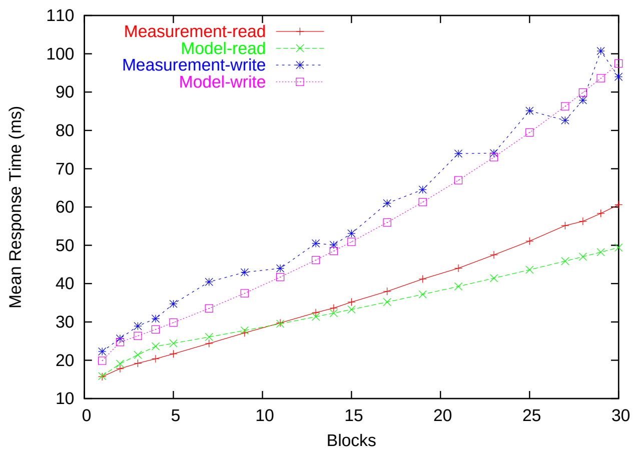


(b) 2-block write request

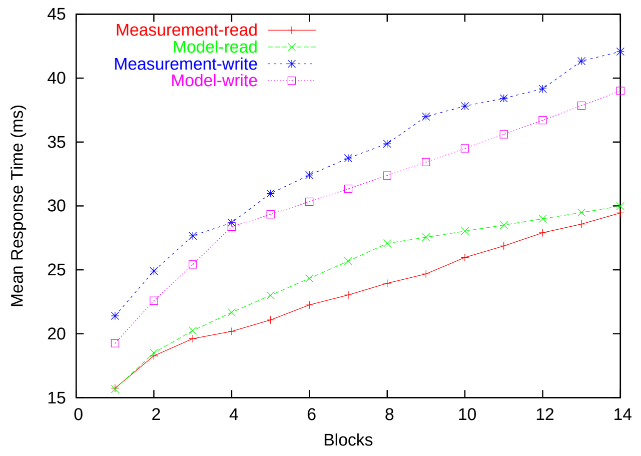
Figure 4.9: I/O request response time distributions on 4-disk RAID 01 with arrival rate 0.02 requests/ms.

cause it does not occur with RAID 01 writes or RAID 5 reads (see Figure 4.12); we speculate that this is possibly because of the drive selection policy (which controls whether to read from a primary disk or its mirror) implemented by the RAID controller or controller overhead. This may be disk drive or RAID system specific and ideally should be investigated further by comparing the model with device measurements from disk drives and RAID systems produced by other manufacturers.

Figure 4.11 compares pdfs and cdfs for some randomly chosen parameters. We generally see excellent agreement between model and measurement. Interestingly, even if the measured and modelled cdfs do not have excellent agreement, their pdfs show some similar trends. For example, in Figure 4.11(c) the model exhibits the bimodal nature of the measurement although it does not share its peaks.



(a) 4-disk RAID 01

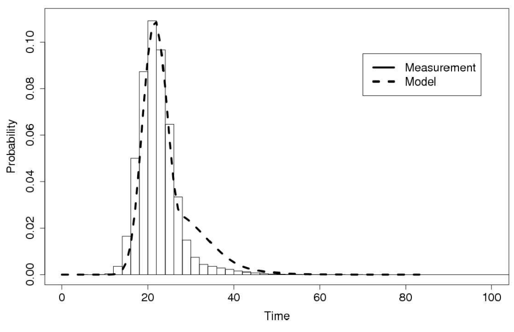


(b) 8-disk RAID 01

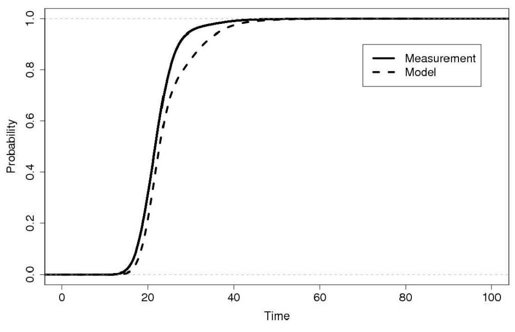
Figure 4.10: Comparison of measured and modelled mean response time against block size for RAID 01 with arrival rate 0.01 requests/ms.

# Blks	Reads				Writes			
	Measured		Modelled		Measured		Modelled	
	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)
1	15.7	15.2	15.9	22.9	22.3	38.9	19.9	26.8
2	17.8	14.8	19.1	24.4	25.7	65.8	24.7	50.4
3	19.2	16.8	21.4	32.8	28.9	88.8	26.4	59.4
4	20.4	19.3	23.6	44.9	30.9	100.5	28.1	69.8
5	21.7	21.0	24.4	48.9	34.7	135.6	29.8	81.6
7	24.4	35.1	26.1	57.8	40.5	317.4	33.5	110.3
9	27.1	52.3	27.8	68.0	43.0	218.5	37.5	146.4
11	29.7	68.9	29.5	79.8	44.0	267.9	41.7	191.3
13	32.4	93.3	31.4	93.1	50.5	426.2	46.2	246.5
14	33.6	112.5	32.3	100.4	50.1	430.1	48.5	278.4
15	35.2	126.1	33.2	108.1	53.1	561.6	50.9	313.7
17	38.0	169.6	35.2	125.0	61.0	756.7	55.9	395.1
19	41.2	230.0	37.2	143.8	64.6	1179.1	61.3	493.1
21	44.0	299.1	39.3	164.9	73.9	1656.0	67.0	611.0
23	47.5	372.6	41.4	188.3	74.1	1735.4	73.0	752.2
25	51.1	524.0	43.6	214.2	85.1	2357.6	79.4	921.1
27	55.1	743.5	45.9	242.9	82.6	2647.4	86.3	1122.9
28	56.3	627.7	47.0	258.3	87.9	2726.3	89.9	1238.2
29	58.3	731.0	48.2	274.5	100.7	3883.3	93.6	1364.4
30	60.6	839.4	49.4	291.5	94.0	3439.3	97.5	1501.9

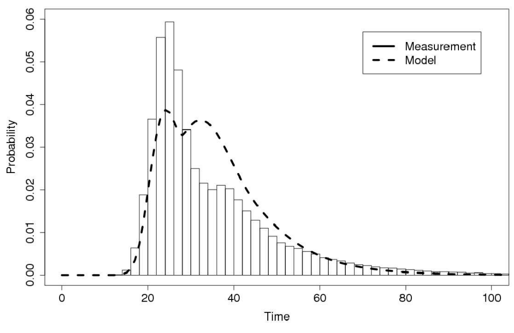
Table 4.6: Response time mean and variance comparison for measurement and model of 4-disk RAID 01 read and write requests with arrival rate 0.01 requests per ms.



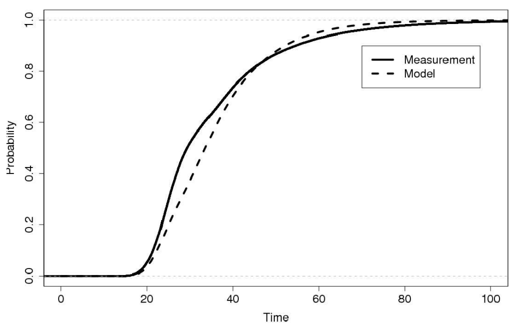
(a) read requests pdf, $b = 6$, $\lambda = 0.01$



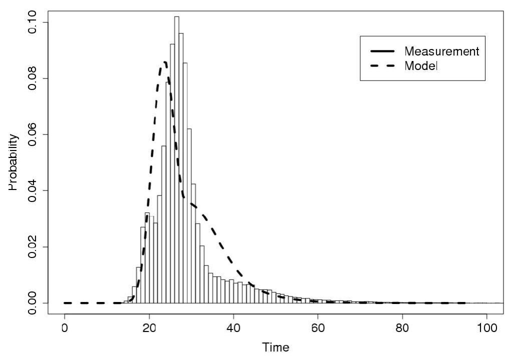
(b) read requests cdf, $b = 6$, $\lambda = 0.01$



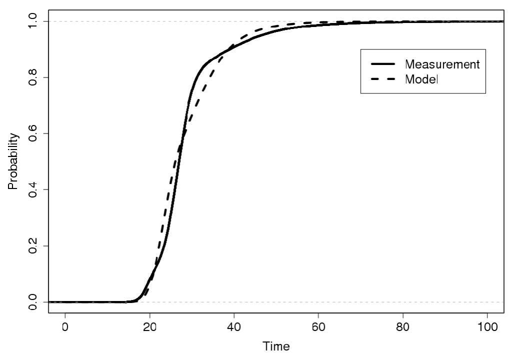
(c) write requests pdf, $b = 3$, $\lambda = 0.03$



(d) write requests cdf, $b = 3$, $\lambda = 0.03$



(e) write requests pdf, $b = 4$, $\lambda = 0.01$



(f) write requests cdf, $b = 4$, $\lambda = 0.01$

Figure 4.11: 8-disk RAID 01 b-block request response time pdfs and cdfs for arrival streams of reads or writes with rate λ requests/ms.

4.3.3 RAID 5

Read Requests

A read request under RAID 5 is modelled in the same way as the equivalent read request under RAID 01. Note that, since RAID 5 distributes data (and parity) across all disks, a $b \geq n$ read request will access all n disks, despite the stripe size of $n-1$ disks. The response time cdf is:

$$W_{R5_r}(t) = \begin{cases} \left(W_d\left(t, \frac{\lambda b}{n}, \frac{1}{E[R]+E[S]+E[T_1]}\right)\right)^b & \text{if } b < n \\ \left(W_d\left(t, \lambda, \frac{1}{E[R]+E[S]+E[T_{\frac{b}{n}}]}\right)\right)^n & \text{otherwise} \end{cases}$$

Write Requests

In our RAID 5 model we assume that all write requests start striping from the first disk in the array. The behaviour of a RAID 5 write depends on the size of the request, with different methods used to update the parity. Write requests that include partial stripe writes will need to pre-read data to calculate the new parity before writing the partial stripe. Therefore these types of writes must consist of two subrequests, one to write any preceding full stripes and pre-read data and the second to write the partial stripe. We assume that the time to complete a single pre-read and a single partial stripe write is equivalent to the weighted average of completing two pre-reads or two partial stripe writes. We note that these two subrequests are not independent. Indeed, we assume that they are highly dependent, and if the mean response time of each type of subrequest was represented by random variable W, the total request response time for both subrequests would be:

$$W_{R5_w}(t) = \mathbb{P}(2W \leq t) = \mathbb{P}\left(W \leq \left(\frac{t}{2}\right)\right) \tag{4.5}$$

We consider RAID 5 write requests in the following-size dependent categories:

Small Partial Stripe Write If a request consists of $b < \frac{n-1}{2}$ blocks, then parity is calculated using [91]:

$$new\_parity = new\_data \oplus old\_data \oplus old\_parity$$

where $\oplus$ is the exclusive-or (XOR) operator. This is a *read-modify-write* operation. Each of the b blocks and the single parity block must be transferred twice, first to read the old data and parity, then to write the new data and parity. When the old data and parity have been read from all disks, a new subrequest will be issued to write the new data and parity to the same disks. This request is given priority in the queue, so at least one disk (the last to complete the pre-read) will just have completed reading a data or parity block that now needs to be re-written. Therefore we add a full disk rotation into the service time distribution for one disk only. However, it is likely that by the time the last disk has completed its pre-read, the remaining disks accessed will have started servicing the next I/O request in their queues. These disks will need to re-seek back to the original disk sector which was pre-read from to write the new data and parity. Therefore, we assume that b disks seek again on the second subrequest, while one disk needs a complete rotation only.

The request to pre-read will have a mean service time of $E[R] + E[S] + E[T_1]$ on each disk. Therefore the service rate for both subrequests is the mean over all the disks for both the pre-read and partial stripe write subrequests. In the pre-read, $b+1$ disks are accessed each with mean service time $E[R]+E[S]+E[T_1]$. In the partial stripe write, b disks are accessed with mean service time $E[R] + E[S] + E[T_1]$ and one disk is accessed with mean service time $R_{max} + E[T_1]$. The arrival rate at each of the $b+1$ disks for both subrequests is $\lambda(b+1)/n$. Combining both arrival streams, we approximate the cdf of the response time as:

$$W_{R5_w}(t) = \left(W_d \left(\frac{t}{2}, \frac{2\lambda(b+1)}{n}, \frac{1}{\frac{(2b+1)(E[R]+E[S])+R_{max}}{2(b+1)} + E[T_1]} \right) \right)^{b+1}$$

The mean service time in the above is calculated by averaging the mean service times of the pre-read and partial stripe write operations. Thus, the pdfs of seek time and rotational latency are altered to:

$$f'(t) = \begin{cases} \frac{1}{2(b+1)} & \text{if } t = 0 \\ \frac{2b+1}{2(b+1)} f(t) & \text{otherwise} \end{cases}$$

where $f(t)$ represents the pdf of seek time or rotational latency.

Large Partial Stripe Write If $\frac{n-1}{2} \leq b < n-1$, then to minimise disk accesses the parity is calculated by pre-reading only from the disks that are not being written to in the partial stripe write operation. The new parity is calculated by XOR-ing the data that will be written with the data from the disks that will remain unchanged. This is a *read-reconstruct-write* operation. The first subrequest pre-reads $n-1-b$ blocks of data for the calculation of the new parity. When all $n-1-b$ disks complete their respective pre-reads, a new subrequest is sent to the other $b+1$ disks to write the new data and parity. The arrival rate for the pre-read will be $\lambda(n-1-b)/n$, and we compute the time to complete this phase as the slowest of the $n-1-b$ queues. The arrival rate of the partial stripe write subrequest will be $\lambda(b+1)/n$ to $b+1$ queues. Both pre-read and partial stripe write subrequests will have the same mean service time of $E[R]+E[S]+E[T_1]$, as the disks accessed for the partial stripe write are not the same as those accessed for the pre-read. Averaging the number of queues we are finding the maximum response time of ($\frac{(n-1-b)+(b+1)}{2}$) queues; we thus approximate the response time cdf of the two subrequests required as:

$$W_{R5_w}(t) = \left(W_d\left(\frac{t}{2}, \lambda, \frac{1}{E[R]+E[S]+E[T_1]}\right)\right)^{n/2}$$

Full Stripe Write If a request consists of a number of complete stripes (i.e. $b \bmod (n-1) = 0$), no pre-reads are needed to calculate the parity. All the disks are utilised, with either the new data block or the new parity block written to each disk. The response time cdf is:

$$W_{R5_w}(t) = \left(W_d\left(t, \lambda, \frac{1}{E[R]+E[S]+E[T_{\frac{b}{n-1}}]}\right)\right)^{n}$$

Full Stripe followed by Small Partial Stripe Write If $0 < b \bmod (n-1) < \frac{n-1}{2}$ and $b > n-1$, at least one full stripe write will occur followed by a small partial stripe write. Let $k = \lfloor \frac{b}{n-1} \rfloor$ and $b_{mod} = b \bmod (n-1)$. We assume that there are again two subrequests to be averaged. The first subrequest involves

k full stripe writes, followed by a parity pre-read to $b_{mod}+1$ disks. The second subrequest writes the new data and parity to $b_{mod}+1$ disks. The mean service time is calculated similarly to the small partial stripe write mean service time. The main difference between these two operations in terms of disk head positioning time is that the first subrequest will access all the disks while the partial stripe write will only access $b_{mod}+1$ disks. In the first subrequest, there will be k data blocks written to the disks not being pre-read from and $k+1$ data blocks written to and read from the $b_{mod}+1$ disks that are pre-read from for parity calculation. In the second subrequest, one data block is written to $b_{mod}+1$ disks. Hence the total number of data transfers over all disks and the two subrequests is $nk+2(b_{mod}+1)$ and the mean number of transfers per subrequest, per disks is $\frac{k}{2}+\frac{b_{mod}+1}{n}$. The response time cdf is then approximated as:

$$\begin{aligned} W_{R5_w}(t) &= \Bigg(W\Bigg(\frac{t}{2}, \frac{\lambda(n+b_{mod}+1)}{n}, \\ &\qquad \frac{1}{\frac{(n+b_{mod})(E[R]+E[S])+R_{max}}{n+b_{mod}+1}+E[T_{\frac{k}{2}+\frac{b_{mod}+1}{n}}]}\Bigg)\Bigg)^{\frac{n+b_{mod}+1}{2}} \end{aligned}$$

Full Stripe followed by Large Partial Stripe Write If $\frac{n-1}{2} \leq b_{mod} < n-1$ and $b > n-1$, at least one full stripe write will occur followed by a large partial stripe write. The initial subrequest will be to write k blocks to all disks and then pre-read an additional block on $n-b_{mod}-1$ disks. The second subrequest, issued upon the completion of the first, writes the new data and parity to the remaining $b_{mod}+1$ disks.

A large partial stripe write that follows a full stripe write is less straightforward to model than the other cases. Specifically the amount of seeking each disk must do between the time that a partial stripe parity pre-read completes and the partial stripe write begins varies dependent on the size of the request. The fewer disks that are pre-read ($n - b_{mod} - 1$), the more likely that the pre-read will complete before the remaining $b_{mod} + 1$ disks complete their respective full stripe writes. If any of the $b_{mod} + 1$ disks complete the full stripe write operations before the pre-read has completed servicing then that disk must wait to write the new data or parity. In this time, that disk may start servicing the next request in its queue, or just rotate away from the desired position. Henceforth, when the pre-read eventually completes, those disks will have to re-seek back causing additional seek and rotational latency. However, if the pre-read completes first then, when one of the $b_{mod} + 1$ disks completes their full stripe write, they can immediately write the new data or parity for the large partial stripe write without any additional seeking. We accordingly approximate the probability of the $b_{mod} + 1$ disks having to seek as $\frac{n-b_{mod}-1}{n}$, as the less disks there are to pre-read, the quicker the pre-read operation will complete. Then it is more likely that the pre-reading on the $n - b_{mod} - 1$ disks will complete prior to the completion of the full stripe write operations on the other $b_{mod} + 1$ disks.

However, as the number of full stripes written increases this relationship becomes less relevant. This is because each disk will take different amounts of time to write the (larger amount of) full stripe data and the additional pre-read time on some disks will be insignificant in comparison. The effect of zoning amplifies these differences. As the number of full stripes (k) increases, the disk that finishes first is less likely to depend on whether there was an additional pre-read on that disk, and it is more likely that all the disks will need to re-seek. Therefore, we define the probability of seeking as $1 - \frac{b_{mod}-1}{nk}$. Since all disks have to seek initially for the start of the full-stripe write, the mean seek time becomes $\left(1 - \frac{b_{mod}-1}{2nk}\right)(E[R] + E[S])$. There will be $nk+n-b_{mod}-1$ data transfers across the array in the pre-read and $b_{mod}+1$ data transfers in the write which averages to $\frac{k+1}{2}$ data transfers per subrequest per disk. The cdf or request response time is:

$$\begin{aligned} W_{R5_w}(t) &= \left(W_d\left(\frac{t}{2}, \frac{\lambda(n+b_{mod}+1)}{n},\right.\right. \\ &\qquad \left.\left.\frac{1}{\left(1-\frac{b_{mod}-1}{2nk}\right)(E[R]+E[S]) + E[T_{\frac{k+1}{2}}]}\right)\right)^{\frac{n+b_{mod}+1}{2}} \end{aligned}$$

Validation

We validate our RAID 5 read model by comparing the measured and modelled mean response times on a 4-disk array in Figure 4.12. Results are presented for two values of λ (0.01 and 0.02 requests/ms) and for block sizes from 1 to 15. We generally see good agreement between model and measurement. Table 4.7 contains means and variances for this case. The model variances agree more closely with measured variances than any of our other RAID operation models.

Figure 4.13(a) shows measured and modelled mean response times for 8-disk RAID 5 reads under light and heavy loads. Agreement is excellent for light load, but under heavier load for larger block sizes, the model increasingly overestimates the measurements (this problem will be dealt with in Chapter 5).

Figure 4.14 compares pdfs and cdfs for some RAID 5 read requests. The modelled

# Blks	$\lambda = 0.01$ ms^{-1}				$\lambda = 0.02$ ms^{-1}			
	Measured		Modelled		Measured		Modelled	
	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)
1	16.5	24.3	16.2	27.6	16.8	29.4	16.6	33.0
2	20.4	42.5	20.4	40.1	21.9	63.5	21.9	61.1
3	22.5	58.6	24.2	66.3	24.9	99.0	27.9	119.1
4	24.0	77.9	28.6	104.6	27.6	157.6	36.3	222.1
5	24.9	94.0	29.9	116.0	29.0	200.0	38.4	253.3
6	24.5	56.2	25.2	53.2	28.1	123.9	31.2	128.5
7	28.6	137.2	32.5	142.3	34.4	303.2	42.9	329.6
8	30.9	161.0	33.9	157.3	37.5	367.1	45.4	376.2
9	31.7	177.9	35.3	173.8	39.0	436.6	48.1	429.7
10	33.5	216.1	36.7	191.8	43.3	618.4	50.9	491.3
11	34.8	271.7	38.3	211.4	46.3	838.3	54.0	562.1
12	36.9	309.7	39.8	232.9	50.4	1010.2	57.2	644.1
13	39.7	411.9	41.4	256.4	55.4	1297.1	60.7	739.2
14	42.3	452.1	43.1	282.1	61.4	1617.6	64.5	849.5
15	43.9	523.6	44.9	310.0	66.5	2089.2	68.6	978.6

Table 4.7: Response time mean and variance comparison for measurement and model of 4-disk RAID 5 read requests.

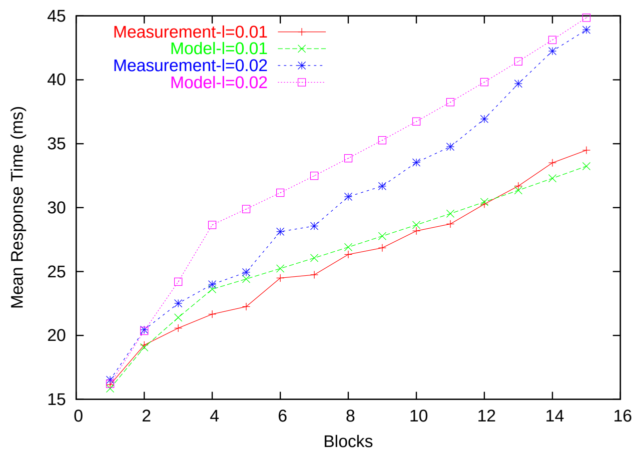
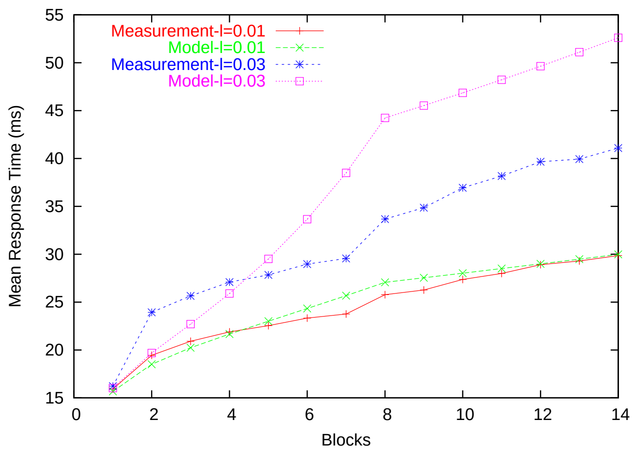
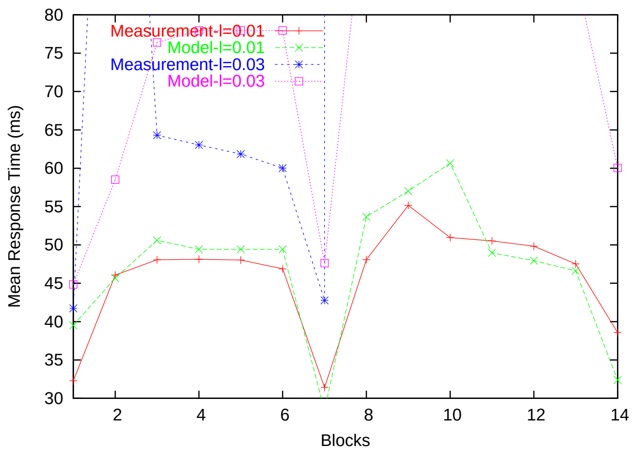


Figure 4.12: Comparison of mean response time against block size for 4-disk RAID 5 reads for different arrival rate values, λ (l).

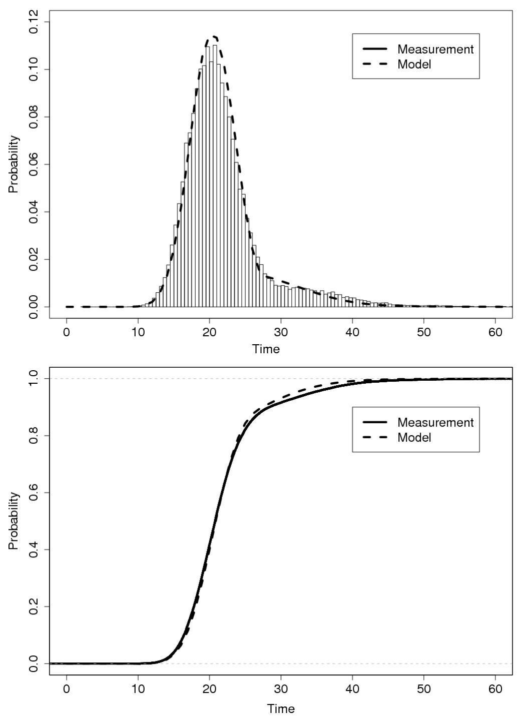


(a) RAID 5 read requests

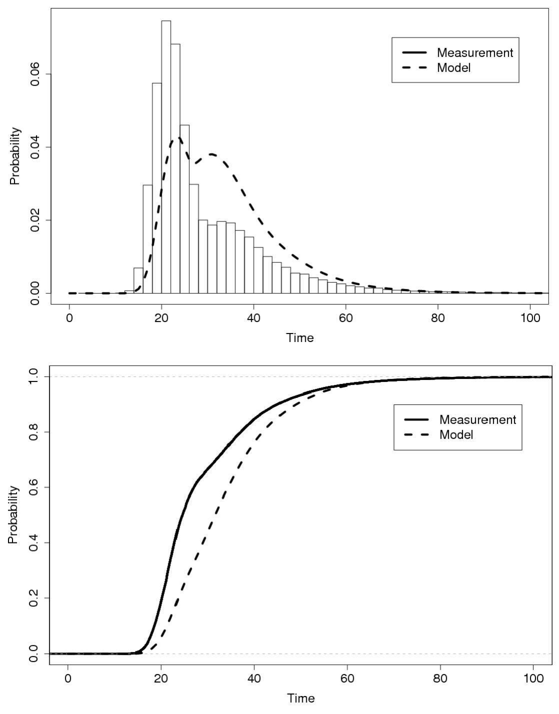


(b) RAID 5 write requests

Figure 4.13: Comparison of mean response time against block size for 8-disk RAID 5 for different arrival rate values, λ (l).



(a) $b = 4$, $\lambda = 0.01$



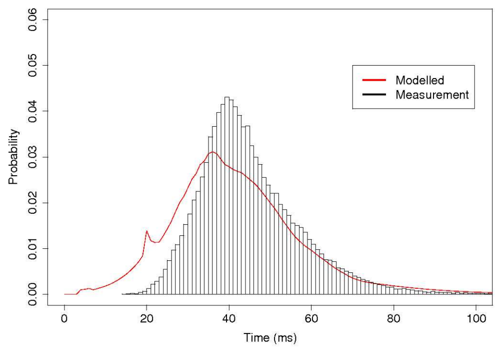
(b) $b = 6$, $\lambda = 0.03$

Figure 4.14: 4-disk RAID 5 b-block read request response time pdfs and cdfs for arrival streams with rate λ requests/ms.

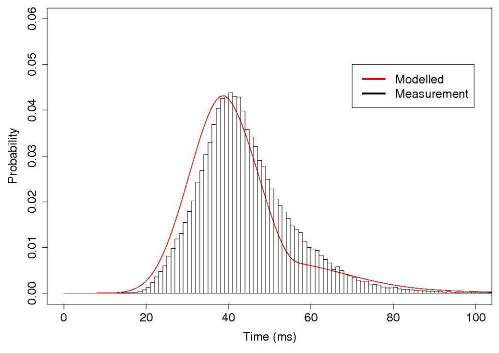
pdf in Figure 4.14(b) displays the bimodal nature of the measured result, but not the peak of the maximum value. Figure 4.14(a) shows close agreement between model and measurement.

Figure 4.13(b) shows measured and modelled results for RAID 5 writes under light and heavy loads. The dips for both measurement and model at 7 and 14 blocks occur because these are full stripe writes with no slow parity pre-reads. For light load there is good agreement between model and measurement. For a heavier load, both measurement and model quickly show signs of saturation.

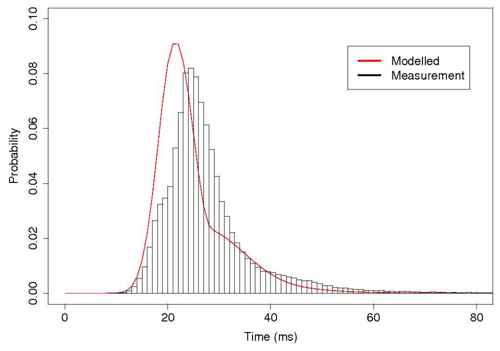
Figures 4.15 and 4.16 present pdfs for RAID 5 write requests on both a 4 and 8-disk array. Figures 4.15(a) and 4.16(a) show small partial stripe write operations. We observe excellent agreement for the 8-disk case, but less good agreement for the 4-disk case. Figures 4.15(b), 4.16(b) and 4.16(c) show large partial stripe write operations. Here we observe excellent agreement in all cases. Figures 4.15(c) and 4.16(d) are full stripe write operations – in both cases the model peaks a little earlier than the measurement. This may be because no parity calculation RAID controller overhead time is taken into account. Figures 4.15(d) and 4.16(e) are small partial stripes that follow full stripe writes and Figures 4.15(e) and 4.16(f) are large partial stripe writes that follow full stripe writes. In both these cases there is poor agreement between measured and modelled pdfs although mean response times are similar. However, these results suggest there is room for improvement in some of these models.



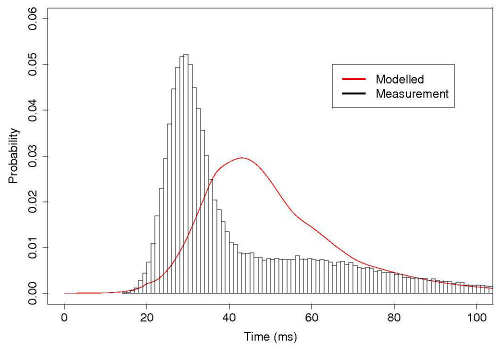
(a) $b = 1$



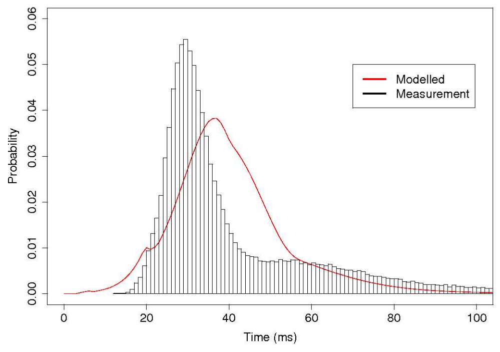
(b) $b = 2$



(c) $b = 3$

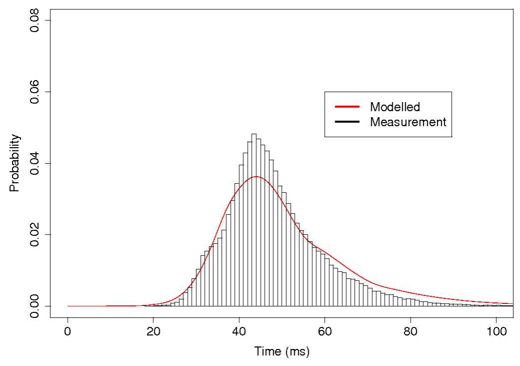


(d) $b = 4$

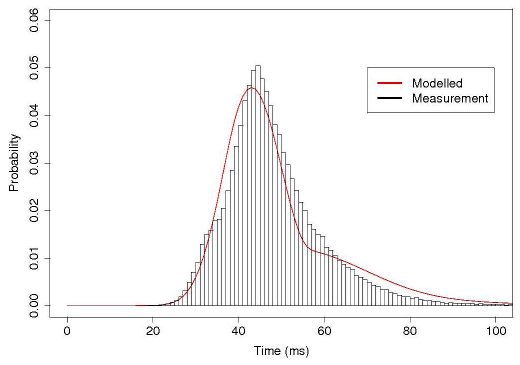


(e) $b = 5$

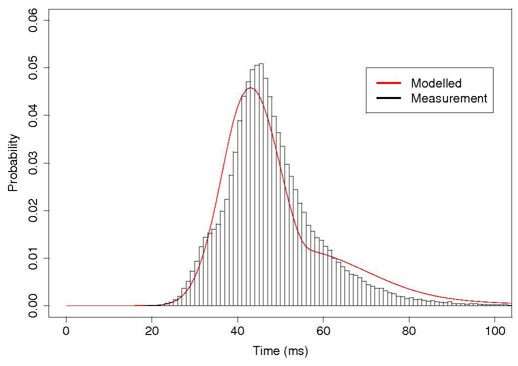
Figure 4.15: 4-disk RAID 5 b-block write request response time pdfs for arrival streams with rate 0.01 requests/ms.



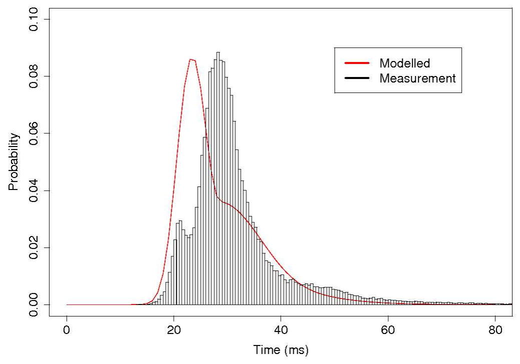
(a) $b = 3$



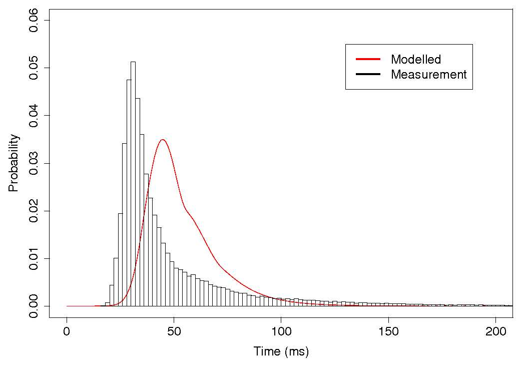
(b) $b = 4$



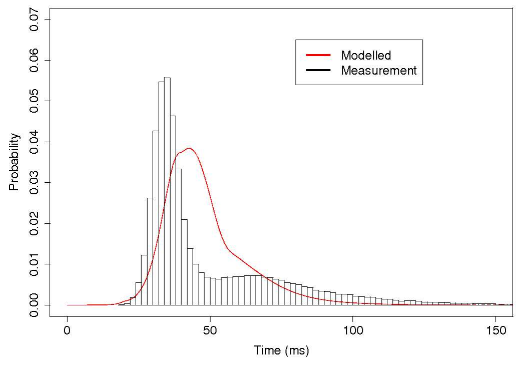
(c) $b = 5$



(d) $b = 7$



(e) $b = 8$



(f) $b = 12$

Figure 4.16: 8-disk RAID 5 b-block write request response time pdfs for arrival streams with rate 0.01 requests/ms.

4.4 RAID Simulation

To assist in improving and validating these models, we have developed a corresponding RAID simulation. Similarly to the analytical model, we must modify the fork-join simulation to model RAID operations more realistically. The advantage of simulation over analytical models when modelling RAID systems is that the simulation is capable of mimicking physical behaviour on each individual disk whereas the analytical model must only assume an average identical behaviour for each disk. This is helpful for improving the analytical model, as it helps us to observe if our assumptions about the disk and RAID operations are likely to be correct. We present simulations for RAID 0, RAID 01 and RAID 5 and compare them with device measurements and the existing analytical models described in this chapter. The UML diagram in Figure 4.17 depicts the classes that must be added to the fork-join simulator to create a RAID 0, 01 and 5 simulation.

4.4.1 RAID 0 and RAID 01 Simulation

There are extensions to the fork-join simulation described in Section 4.2.4 that must be made to model a RAID 0 and 01 system accurately.

In particular, both the fork-join simulation and the RAID 0 simulation of [131] are limited to supporting requests consisting of a number of subtasks that is a multiple of the number of disks. We therefore extend the *ForkLink* class with *RAID01ReadForkLink* and *RAID01WriteForkLink* classes, both of which support striping of variable size subtasks across disks starting from a randomly selected

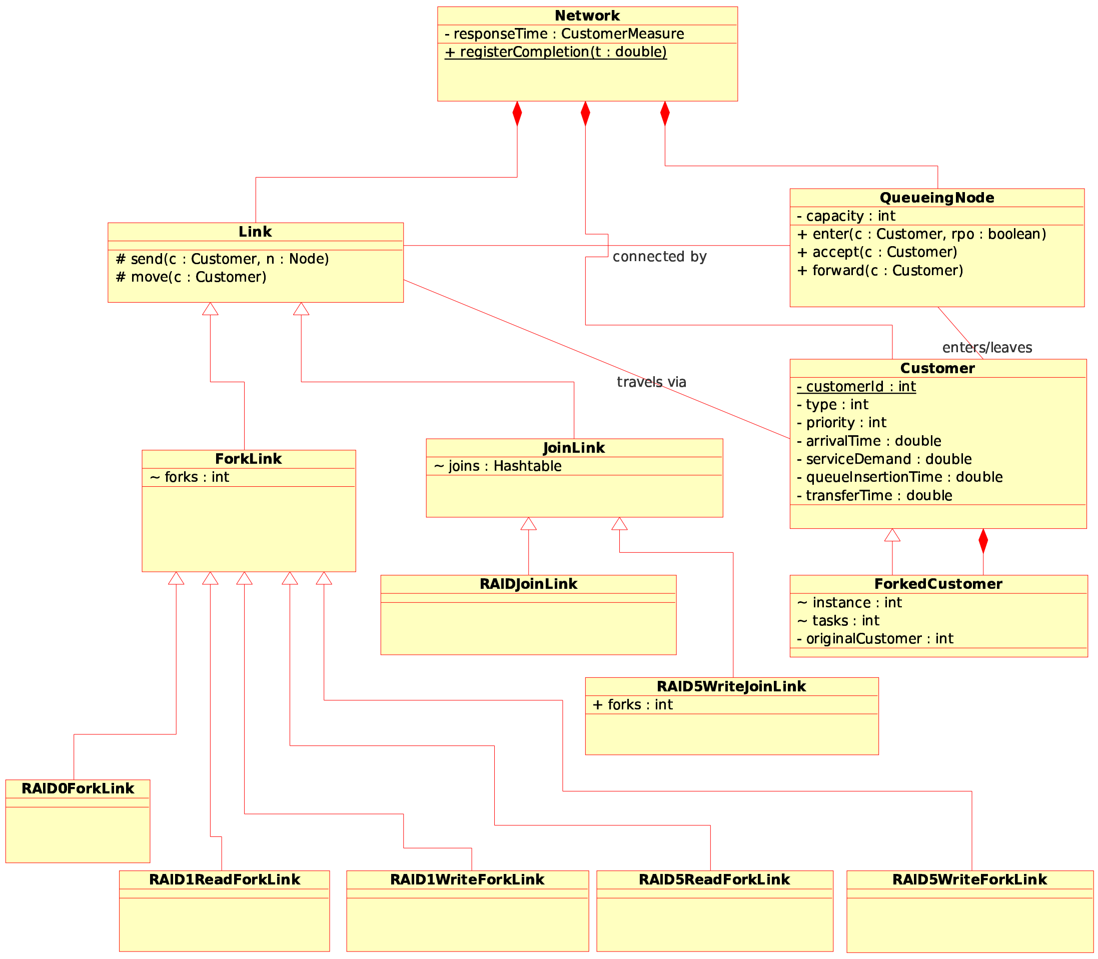


Figure 4.17: RAID simulator class diagram.

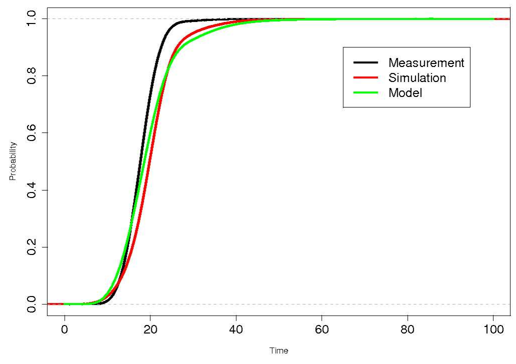
disk. Additionally, we extend the *JoinLink* class with the *RAIDJoinLink* class to support joining of variable sized requests (see Figure 4.17 for UML diagram).

In terms of subtask scheduling for RAID 01 read operations, we assume an efficient RAID controller which reads half the data from the primary disks and half the data from the mirror disks [54]. RAID 01 write operations send each subtask to both the primary and mirror disks and create double the number of *ForkedCustomer*s as for a read request of the same size.

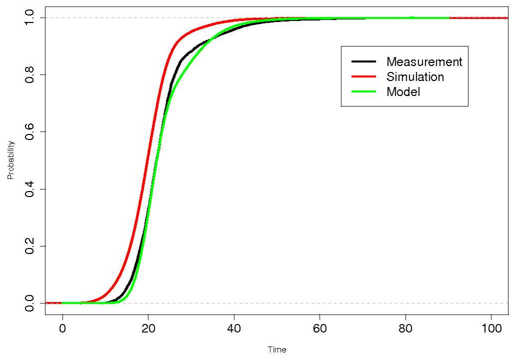
RAID 0 read and write operations are identical in terms of RAID disk behaviour operations. We therefore extend the *ForkLink* class with a *RAID0ForkLink* class which supports striping of variably sized requests across disks starting from a randomly selected disk.

Validation

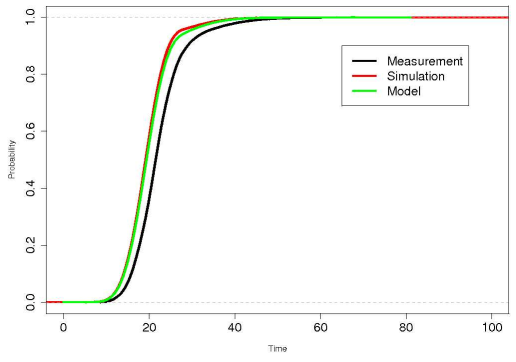
In Figure 4.18 we compare measurement, model and simulation cdfs for RAID 01 systems with Markovian arrivals, arrival rate 0.01 requests/ms and requests with a constant size. We generally observe good agreement between model and measurement, particularly in Figure 4.18(b), in which a full stripe read is taking place. The simulation and analytical model differ most for larger block sizes and write requests (which write double the number of blocks for each request). The analytical model produces slower response times than the simulation model. This is caused by the overhead of using the split-merge queue approximation in the analytical model, rather than the fork-join queue in the simulation. However, particularly in the case of write requests, both simulation and analytical models underestimate measurements slightly. This may be because neither model or simulation take into account any RAID controller overheads.



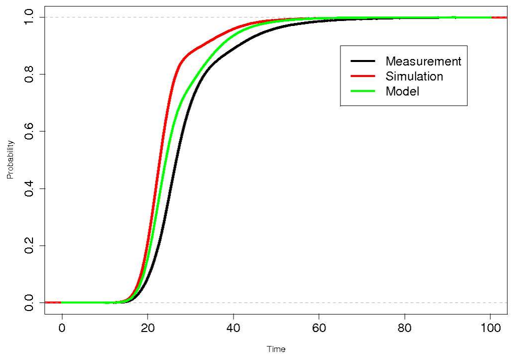
(a) 2-block read request



(b) 4-block read request



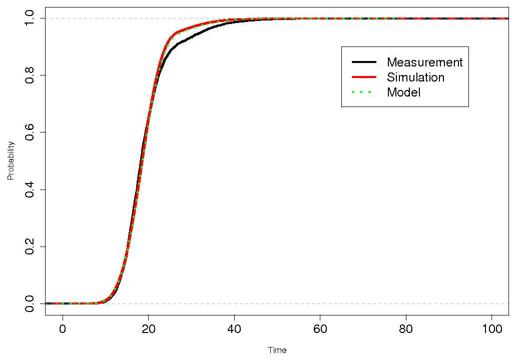
(c) 1-block write request



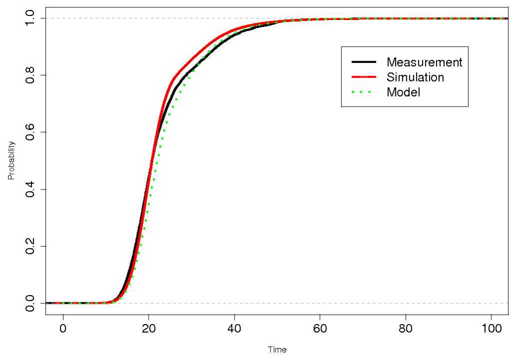
(d) 3-block write request

Figure 4.18: I/O request response time distributions of 4-disk RAID 01 with arrival rate 0.01 requests/ms.

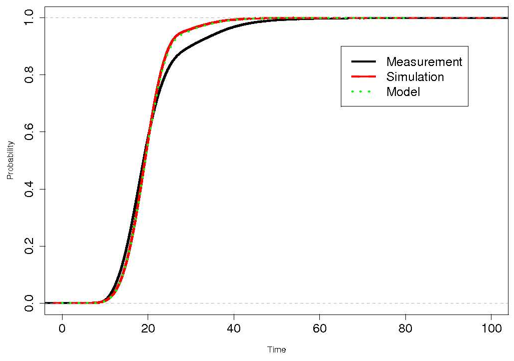
In Figure 4.19 we compare RAID 0 simulation with the analytical model and device measurements. The cdfs presented correspond to the pdfs in Figure 4.8. We observe similar trends to the RAID 01 case but generally better agreement between models and measurement. This is because RAID 0 operations are more transparent then any other RAID level and hence can be modelled very accurately.



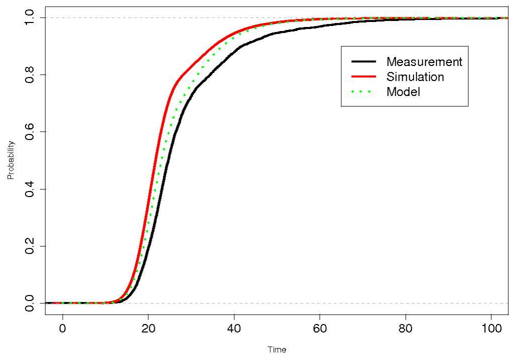
(a) 2-block read request, $\lambda = 0.01$



(b) 3-block read request, $\lambda = 0.02$



(c) 2-block write request, $\lambda = 0.01$



(d) 3-block write request, $\lambda = 0.02$

Figure 4.19: I/O request response time distributions of 4-disk RAID 0 with arrival rate λ requests/ms.

4.4.2 RAID 5 Simulation

Manufacturers of RAID controllers seldom reveal the mechanisms and scheduling strategies involved in their products. In the cases of RAID 0 and 01, the likely disk accesses are relatively straightforward to predict. However in RAID 5, particularly with operations involving pre-reads and parity updates, there are many possibilities for scheduling strategies and disk head positioning times within a request.

In a manner analogous to the RAID 01 case, we extend the *ForkLink* class with *RAID5ReadForkLink* and *RAID5WriteForkLink* classes.

A RAID 5 read request will read only from the disks containing data blocks in a stripe and not the disk with the single parity block in each stripe. To simulate this, when forking each request, the position of the parity disk is randomly chosen as well as the starting disk. If a request accesses more than one stripe, then the position of the parity disk within the array is incremented (modulo the number of disks) at the end of each stripe.

The behaviour of a RAID 5 write is complex, with different parity-update schemes that depend on the size of the request. For simplicity, we assume requests are aligned to start striping from the first disk in the array.

Given a b-block write request on an n-disk RAID 5 system, the possibilities are:

If a request consists of a number of complete stripes (i.e. $b_{mod} = 0$), all the disks are utilised, with either the new data block or the new parity block written to each disk. Full stripe writes can be simulated by sending *ForkedCustomer*s to each disk and joining them at the *RAID5WriteJoinLink* when all subtasks have completed.

If a request consists of $b_{mod} < \frac{n-1}{2}$ blocks the simulation must reflect the following procedures. After transferring the full stripes, each of the b_{mod} blocks and parity must be transferred twice, first to read the old data and parity, then to write the new data and parity. When the old data and parity have been read from all disks, a new request will be issued to write the new data and parity to the same disks. This request is given non-preemptive priority in the queue, so at least one disk (the last to complete the pre-read) will have just completed reading a data or parity block that now needs to be re-written.

If $\frac{n-1}{2} \leq b_{mod} < n-1$ then after transfer of the full stripes, $n-1-b_{mod}$ blocks of data are pre-read for the calculation of the new parity. When all $n-1-b_{mod}$ disks complete their pre-read, a new request is sent to the other $b_{mod}+1$ disks to write the new data and parity.

Simulation of these operations is supported in the *RAID5WriteForkLink* and the *RAID5WriteJoinLink* classes (see Figure 4.17 for UML diagram).

The *RAID5WriteForkLink* subdivides any arriving request into full stripe subtasks followed by pre-read subtasks. These subtasks are then routed to the relevant $M/G/1$ queues. When the pre-read subtasks have completed and are accounted for at the *RAID5WriteJoinLink* then, instead of completing the request, the *RAID5WriteJoinLink* creates a new high priority request to send back to the *RAID5WriteForkLink*, where it splits into $b_{mod}+1$ subtasks (the number of blocks to write plus the parity). In order for the simulation to differentiate between full stripe writes and pre-reads and the following partial stripe write, the *ForkedCus-*

*tomer*s are assigned classes representing the type of request.

The simulation has the most obvious benefit over the analytical model when modelling the service times of each subtask on each disk. In the analytical model, whether the disk must re-seek or not between subtasks is assumed and averaged over all the disk. In the simulation these can be programmed to be specific to each request. The subtasks of the partial stripe write will have different service times dependent on the nature of the previous request serviced by the disk. If $b_{mod} < \frac{n-1}{2}$, there are four possible scenarios that must be considered.

The first scenario is when the disk is busy at the arrival instant of any of the partial stripe write subtasks. Since the partial stripe write is accessing all the disks used for the pre-read, and all the pre-reads must complete before the partial stripe write is issued, it is not possible that the job currently servicing is a *ForkedCustomer* from the same *Customer*. Hence to simulate a return to the required disk position to transfer data, a random sample of seek and rotation time is taken.

If the disk is idle on arrival of a subtask, then there are a further three mutually exclusive scenarios with different positioning times:

- If another request has been in service between the pre-read and partial stripe write subtasks then the simulator needs to sample a new seek and rotation time.

- If the disk was the last to complete the pre-read, then it will be positioned on the correct track, but just past the rotational position. In this case, the simulator returns a positioning time of one full disk rotation.

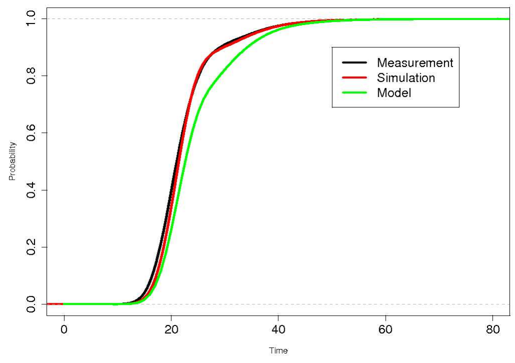
- Otherwise, the disk is still positioned at the correct track and the simulator needs to sample from the rotational latency for positioning time.

If $b_{mod} \geq \frac{n-1}{2}$, there are again a number of scenarios to consider. Since the pre-read involves different disks than the partial stripe write, it is possible that previous full stripe subtasks from the same request could still be servicing on the disks required for the partial stripe write after the pre-read has completed.

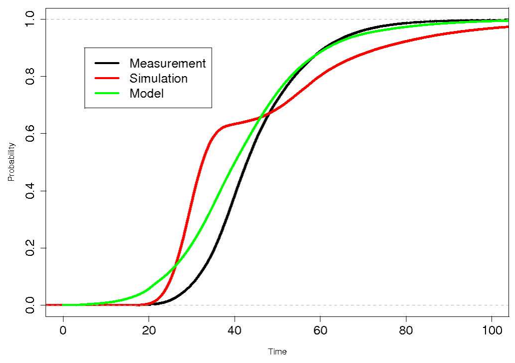
In this context, if a subtask arrives to a busy disk, we consider whether the job currently in service is part of the same request. If it is, the subtask will follow on with no positioning time. If it arrives to an idle disk, the simulator checks if the previous job was part of the same request. If it was then the disk head is pointing to the correct track and the simulator needs to sample rotational latency only. In all other cases the positioning time is obtained by sampling both seek and rotation time.

Since we are simulating zoned disks, we must take into account that the transfer time must be the same both for the full-stripe and pre-read and for the partial stripe write requests, since they are both accessing the same position on the disk. Therefore, the transfer time for each subtask to each disk is recorded in a hash table and referred to when the partial stripe write is serviced.

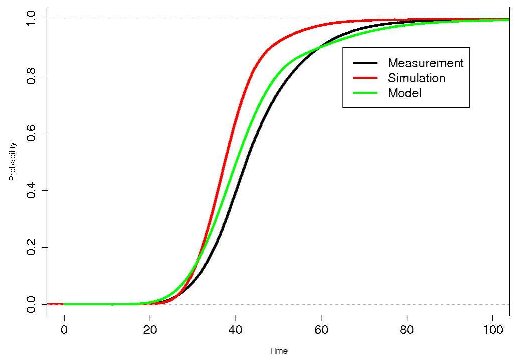
When all the partial stripe write subtasks complete, the *RAID5WriteJoinLink* sends the single request on its way and removes all *ForkedCustomer*s attached to that request.



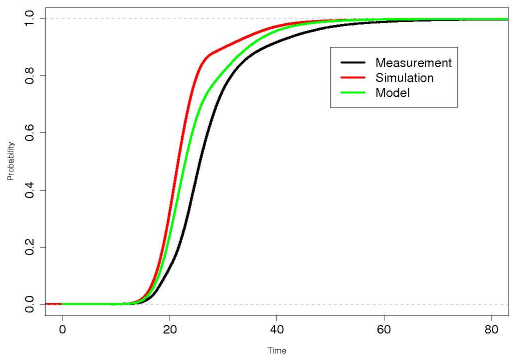
(a) 5-block read request



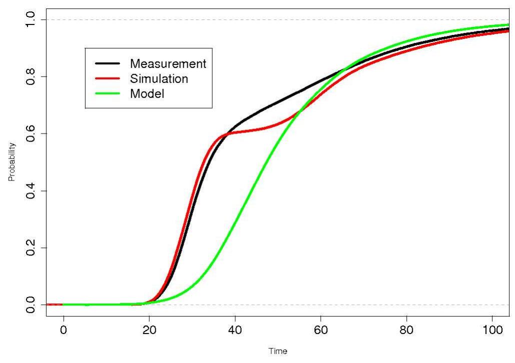
(b) 1-block write request



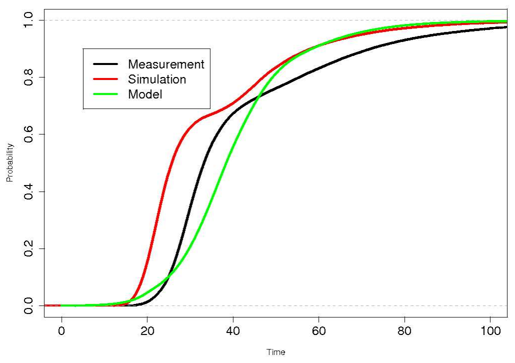
(c) 2-block write request



(d) 3-block write request



(e) 4-block write request



(f) 5-block write request

Figure 4.20: I/O request response time distributions of 4-disk RAID 5 with arrival rate 0.01 requests/ms.

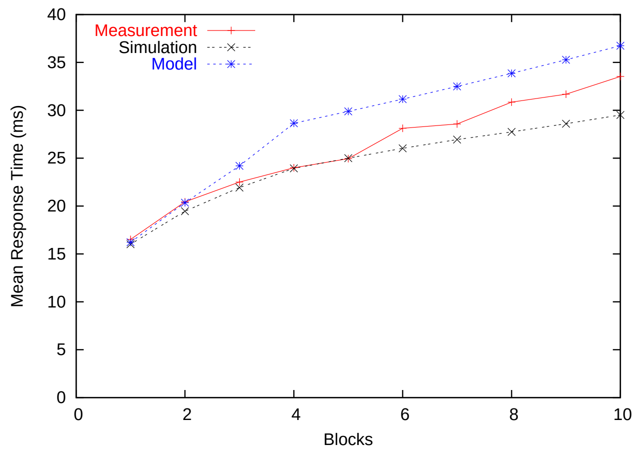
Validation

In Figure 4.20 we compare measurement, analytical model and simulation cdfs for RAID 5 systems with Markovian arrivals with arrival rate 0.01 requests/ms and constant size requests. This enables us to judge the effectiveness of both the simulation and analytical models. We observe in Figure 4.20(a) excellent agreement between read request simulation and measurement. It is possible that the analytical model is less effective in this case because it ignores the parity block while the simulation takes the parity block into account.

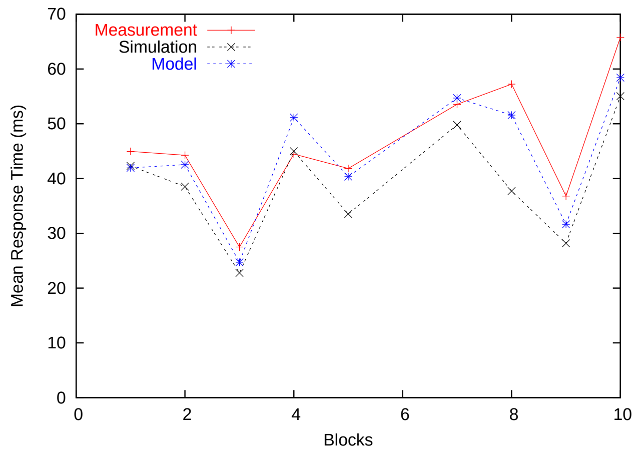
Figure 4.20(b) compares models and measurement for a small partial stripe write. Interestingly the analytical model is much closer to measurement than the simulation. However, the simulation is likely to have a mean response time closer to the measurement than the analytical model. Figures 4.20(c) compares models and measurement for a large partial stripe writes. In this case, both simulation and analytical models show good agreement with the measurements. The analytical model shows slightly better agreement than the simulation model but this is most probably because of the split-merge overhead rather than superior modelling of any specific RAID 5 operation. However both appear to consistently underestimate the measurements, as they again do for the full stripe write request in Figure 4.20(d). It is possible that this underestimation can be attributed to not factoring into the simulation RAID controller overheads including parity computation time. Figure 4.20(e) compares models and measurement for a small partial stripe write that follows a full stripe write. In this case the simulation is close to the measurement and the analytical model is less impressive. We can see here that the shape of the simulation graph is similar to its shape in figure 4.20(b) representing a similar service time procedure, but this time is positioned much closer to the measurement curve. Figure 4.20(f) compares models and measurement for a large partial stripe write that follows a full stripe write. We observe good agreement between the simulation and measurement cdfs. The analytical model curve is not similar to the measurement curve but will yield a mean response time closer to the measurements than the simulation.

Figure 4.21 compares mean response times for simulation, analytical model and measurement for up to ten block jobs. The model predicts effectively the qualitative characteristics of mean RAID 5 response times as block size varies. The simulation provides consistently closer mean response times to measurement than the analytical model for read requests. However for write requests the analytical model generally has closer mean response times to the measurements than the simulation. This is an interesting and unexpected result as one would expect the simulation to be a closer model than the analytical model, since it uses true fork-join queueing instead of split-merge and allows the simulator to mimic actual disk arm behaviour, rather than assuming an average service time.

The transparency of the simulation model can help to analyse the correctness of some of the assumptions we have made about disk head movements in a RAID 5 partial stripe write request in the context of service time. In the simulation the choice of service time is decided based on whether the server is empty when a partial stripe write subtask arrives and whether the last served subtask was part of the same request as the arriving subtask. This can only be loosely estimated in the analytical model. A significant difference is that the analytical model only considers two cases for each partial stripe write, in the small partial stripe case:



(a) mean read request, $\lambda = 0.02$



(b) mean write request, $\lambda = 0.01$

Figure 4.21: Plot of mean response time against request size on a 4-disk RAID 5 system with arrival rate λ requests/ms.

whether the disk must complete a full random seek and rotation or whether the disk will instead make one complete revolution. In the large partial stripe case: whether the disk must complete a full random seek and rotation or whether it immediately transfers data. However, the simulation considers an additional case for both types which is that the head is pointing to the correct track as the last subtask served was from the same request, and need only complete a random rotation. In Tables 4.8, 4.9, 4.10, and 4.11 we present the proportion of subtasks in a partial stripe write operation that fall into the three categories according to the simulation and analytical model.

# Disks	λ requests/ms	Size # blocks	Rotate		R_{max}		Seek	
			Sim	Ana	Sim	Ana	Sim	Ana
4	0.01	1	0.328	N/A	0.328	0.5	0.344	0.5
4	0.02	1	0.166	N/A	0.167	0.5	0.668	0.5
4	0.03	1	0.012	N/A	0.012	0.5	0.975	0.5
8	0.01	1	0.327	N/A	0.328	0.5	0.344	0.5
8	0.01	2	0.424	N/A	0.213	0.333	0.362	0.667

Table 4.8: Proportion of service time operations used by the disks for small partial stripe write requests on different sized arrays.

# Disks	λ requests/ms	Size # blocks	Rotate		R_{max}		Seek	
			Sim	Ana	Sim	Ana	Sim	Ana
4	0.01	4	0.625	N/A	0.364	0.5	0.011	0.5
4	0.02	4	0.284	N/A	0.695	0.5	0.021	0.5
4	0.01	7	0.594	N/A	0.394	0.5	0.012	0.5
8	0.01	8	0.624	N/A	0.364	0.5	0.011	0.5
8	0.01	9	0.607	N/A	0.374	0.333	0.019	0.667
8	0.01	10	0.597	N/A	0.379	0.25	0.025	0.75

Table 4.9: Proportion of service time operations used by the disks for small partial stripe requests that follow full stripe writes on different sized arrays.

Table 4.8 compares these proportions for a small partial stripe write where *Rotate* is a random rotation, R_{max} is a complete disk revolution and *Seek* is a random seek and rotation. The most interesting difference between the analytical and simulation proportions is that the simulation appears to be very clearly load-dependent.

The analytical model does not currently have capability to include load-dependent service times.

Table 4.9 compares proportions for a small partial stripe write that follows at least one full stripe write. We again observe load-dependent service in the simulation. In addition the largest proportion of procedures in almost all cases is the random rotation which is not supported in the analytical model.

# Disks	λ	Size	Rotate		Transfer		Seek	
	requests/ms	# blocks	Sim	Ana	Sim	Ana	Sim	Ana
4	0.01	2	0	N/A	0	0	1	1
4	0.02	2	0	N/A	0	0	1	1
4	0.03	2	0	N/A	0	0	1	1

Table 4.10: Proportion of service time operations used by the disks for large partial stripe write requests.

In Table 4.10 our simulation and analytical models agree completely for large partial stripe write requests providing confidence in both models. In this case *Transfer* refers to operations where the disk head is positioned correctly and no seek or rotational activity is needed. Table 4.11 compares proportions for a small partial stripe write that follows at least one full stripe write. We again observe load dependence in the simulation which is not supported in the analytical model. In general, the analytical model tends to follow the trends of the simulation, showing a decrease or increase in seek and transfer time in tandem with the simulation. However, the proportions are very different and again the rotational time that the analytical model ignores is quite a large factor in most of the simulation results.

It is difficult to know how much importance to give to these results. In our validations of the simulation, we have not observed excellent agreement with device measurements. However, it is definitely clear that the analytical model is flawed

# Disks	λ	Size	Rotate		Transfer		Seek	
	requests/ms	# blocks	Sim	Ana	Sim	Ana	Sim	Ana
4	0.01	5	0.39	N/A	0.0293	0.75	0.317	0.25
4	0.02	5	0.199	N/A	0.244	0.75	0.566	0.25
4	0.01	8	0.369	N/A	0.284	0.375	0.348	0.625
4	0.01	11	0.348	N/A	0.275	0.25	0.377	0.75
4	0.01	14	0.328	N/A	0.266	0.188	0.406	0.813
8	0.01	11	0.505	N/A	0.124	0.625	0.371	0.375
8	0.01	12	0.47	N/A	0.174	0.75	0.355	0.25
8	0.01	13	0.389	N/A	0.294	0.875	0.317	0.125

Table 4.11: Proportion of service time operations used by the disks for large partial stripe write requests that follow full stripe writes on different sized arrays.

by not taking into account the third option of a random rotation without a seek. In all cases the analytical model compensates for this rotation by having a higher proportion of the other two procedures which average to approximately the same service time. In addition the model could probably be improved with load-dependent service times, although we conjecture that this would make the analytical model significantly more complex for a very small improvement in accuracy.

Chapter 5

Workload Modelling

5.1 Introduction

It is imperative that models of storage system performance should be capable of reflecting in their inputs the features found in real I/O workloads. In the previous chapter we only considered homogeneous Poisson arrival streams. By homogeneous we mean that all I/O requests are assumed to be random accesses of the same type (read/write) and size. In this chapter we develop our analytical RAID model to accept more realistic workloads. In order to do this we must incorporate some existing results from queueing theory into our models and also develop new analytical queueing network results for application within the model.

Our initial extension to the model in this chapter is to support heterogeneous I/O request arrival streams with mixtures of read and write requests. This improved realism is possible through applying the theory of multiclass queues in Section 5.2.

Multiclass queueing networks also allow us to reflect the scheduling of subtasks in RAID 5 write requests more authentically in our analytical model. One of the problems with the RAID 5 write model highlighted by the simulation model is that it coarsely models the scheduling of subtasks in a RAID 5 write request. In our original single-class RAID 5 write model we find the average service time and arrival rate to each disk for the two subrequests of a write request that a disk is likely to see: the pre-read and partial stripe write operations. However a multiclass system enables us to treat these two subrequests as different entities – there will be two classes, one for pre-reads and one for the following partial stripe writes. This still does not entirely represent the behaviour of a RAID 5 write request since after the pre-read has completed and a new parity is computed, the partial stripe write is given non-preemptive queueing priority over other subtasks [25]. Therefore we implement theory on priority queueing [28] combined with the theory of multiclass queues to present a new RAID 5 write request model in Section 5.2.2. We compare this new model with the single class model presented in the previous chapter and device measurements to assess its effectiveness.

It is essential for models of RAID 01 and 5 to take as input arrival streams with an arbitrary distribution of request response size. In order to model a workload of different-sized jobs, we define each job as a single bulk arrival, where the size distribution of the bulk arrivals is the same as the job size distribution. We then derive the response time distribution of an entire batch, rather than a random customer within a batch, which is the usual focus of work on bulk arrivals. To the best of our knowledge, only Nelson, Towsley and Tantawi [89] have applied this perspective, in modelling the parallel processing of different sized requests in an

$M^X/M/c$ queue. Here we consider fork-join networks of $M^X/G/1$ queues. We note that the RAID 5 write model presented in this context is more general than existing analytical models including work in the previous chapter which presume RAID 5 write operations consist of a fixed number of subtasks in a job [54, 73] or a variable number that never exceeds a full stripe [25]. Furthermore, all these models assume that all requests are performed without skewing (i.e. assuming that requests are stripe-aligned) which is not in general the case in reality. Our model relaxes both these restrictions. Section 5.3 presents our approach for calculating the response time distribution of an entire batch of arrivals, using $M^X/G/1$ models of single disks and fork-join network models of RAID systems. This gives us the ability to represent variable-sized I/O requests arriving to an $M/G/1$ queue.

We continue to study $M^X/G/1$ queues and utilise their properties by turning our attention to developing methods to model bursty I/O request arrival streams. In our original disk and RAID model we have assumed arrival streams are strictly Markovian. Over the last decade, however, there have been many studies of storage system I/O traces (e.g. [41, 98, 100, 108, 132]) consistently showing that real-life arrivals to storage systems exhibit burstiness. To account for this factor in our model, we derive the response time distribution of a randomly-selected queueing request within a batch. Previous works [21, 51] only calculate the queue length generating function of a single $M^X/G/1$ queue. In Section 5.4 we present our development of bulk arrival theory to permit the analysis of $M^X/G/1$ queues for full response time distributions. We extend this approach to allow the calculation of such distributions in fork-join networks composed of several of these queues, thus enabling us to model single disks and RAID systems with bursty arrivals.

Bursty workloads [100] result in highly-variable queue lengths. As queue length increases, response time suffers. To lessen this effect, many disk drives employ scheduling algorithms to reorder jobs in the queue to minimise head positioning time [20, 32, 62, 107]. This reduces the time needed to service each job, which in turn reduces the waiting time for all jobs [57]. The best way to minimise the total response time of all queueing I/O requests is to dynamically reorder them so that the next request chosen to be serviced has the lowest disk head positioning time of all queueing requests. With this strategy employed, as queue lengths increase response times do not suffer excessively since service times are simultaneously reduced.

Disk drive and RAID models including the models presented in the last two chapters generally model disk queue scheduling discipline as First Come First Served (FCFS). This is an adequate approximation for small workloads and request sizes. However, as these increase the FCFS model will increasingly overestimate the response time of the disk or RAID system. For example, Figure 5.1 compares our RAID 01 model presented in the previous chapter against device measurements for an arrival rate of 0.03 requests/ms. We observe that as request size increases the model increasingly overestimates the device measurements with scheduling strategies employed. It is therefore fundamentally important that these scheduling algorithms must be represented in any disk or RAID model.

In Section 5.5, we model the operation of a disk drive with Shortest Access Time First (SATF) scheduling by using an $M/G/1$ queue with queue-length dependent service time distributions. There does not currently exist a generally-applicable exact result for the response time distribution of this variety of queue. We present

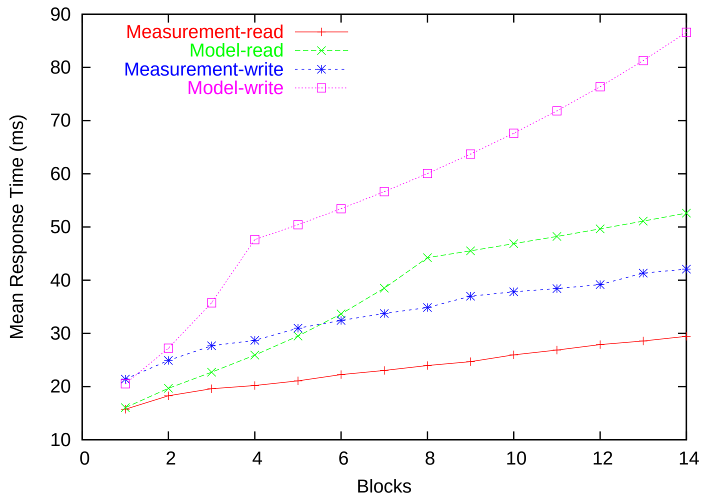


Figure 5.1: Comparison of mean response time against block size for 8-disk RAID 01 with arrival rate 0.03 requests/ms.

a novel approximation for the response time distribution of such a queue. Additionally, it is a non-trivial challenge to derive realistic service time distributions for each queue length such that expected positioning time is minimised. Our model is developed for a single disk and we consider ways of extending it to a RAID system. We demonstrate the accuracy of our model by comparing model predictions with real device measurements.

Throughout this chapter we use the same model terminology defined in the previous two chapters unless otherwise stated. We also validate against measurements from the same disk and RAID system as in previous chapters.

5.2 Multiclass RAID Model

Multiclass queues allow for different customer types and service time distributions [16]. To analyse the response time of these queues we must find an expression for the response time distribution of a queue receiving all class types.

In a system where arrivals have class i, $i = 1, \ldots, m$, let W be a random variable representing a request's response time. The cumulative distribution function of W, $F_W(t)$, is approximated as:

$$\begin{aligned} F_W(t) &= \mathbb{P}(W \leq t) \\ &= \sum_{i=1}^{m} \mathbb{P}(W \leq t \mid \text{class}_i)\mathbb{P}(\text{class}_i) \end{aligned} \tag{5.1}$$

This approximation works best for low arrival rates. It would be exact if the multiclass techniques were used for the service times rather than the response times but this can be computationally restrictive. This equation is used throughout this section, where classes are either read and write request for either RAID 01 or RAID 5 requests (see Section 5.2.1) or pre-read and partial stripe write for a RAID 5 write request (see Section 5.2.2).

5.2.1 Heterogeneous Arrival Streams

Thus far, our RAID models have assumed homogeneous arrival streams. Here we use multiclass queues to generalise these models for heterogeneous streams composed of both read and write requests. This is achieved using Equation (5.1) to calculate the request response time cdf:

$$W(t) = p_{read} W_{\text{read}}(t) + (1 - p_{read}) W_{\text{write}}(t)$$

where p_{read} is the probability that a request is a read. $W_{\text{read}}(t)$ and $W_{\text{write}}(t)$ are defined in the previous chapter for RAID levels 0, 01 and 5. We note that the arrival rate to the disk array used in these RAID models must be modified to take the combined stream into account.

For RAID 01 the arrival rate at each disk is:

$$\frac{\lambda(p_{read} \min(b, n) + (1 - p_{read}) \min(2b, n))}{n}$$

On RAID 5, the arrival rate at each disk is:

$$p_{read} \lambda \frac{\min(b, n)}{n} + (1 - p_{read}) \gamma$$

where γ is the arrival rate at each disk in the array in the case that $p_{read} = 0$, described for each size of RAID 5 write request in Section 4.3.3.

Validation

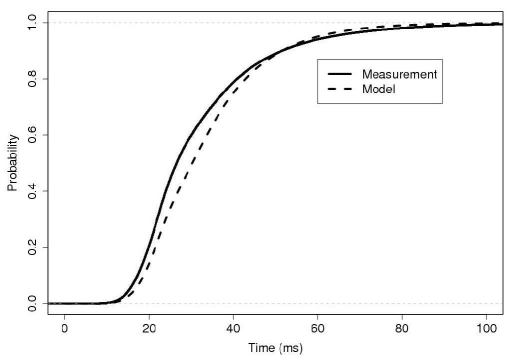
To validate both our RAID 01 and RAID 5 models for mixed reads and writes, we consider arrival streams of 25% reads/75% writes, 50% reads/50% writes, and 75% reads/25% writes.

RAID 01

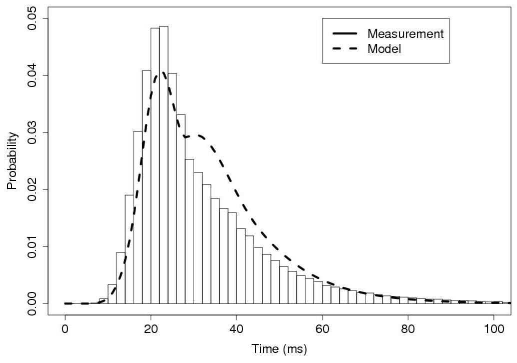
λ	#	25% Reads, 75% Writes				50% Reads, 50% Writes				75% Reads, 25% Writes			
		Measured		Modelled		Measured		Modelled		Measured		Modelled	
(ms^{-1})	Blks	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)
0.01	1	21.0	41.8	18.8	27.7	19.4	39.7	17.8	27.2	17.7	32.7	16.8	25.6
	2	24.4	66.6	23.1	45.3	22.6	62.3	21.7	38.8	20.6	48.3	20.3	31.7
	3	27.3	90.1	25.0	54.6	25.1	81.8	23.8	48.4	22.6	63.4	22.6	41.0
	4	29.2	102.1	27.0	67.3	26.9	98.0	25.8	62.3	24.2	75.8	24.7	54.8
	5	32.5	137.5	28.5	78.9	29.7	131.9	27.1	72.6	26.4	98.0	25.8	62.5
0.03	1	22.9	82.6	21.1	60.4	21.0	72.8	19.5	51.3	18.8	54.5	18.0	42.0
	2	31.5	262.6	33.1	195.4	27.6	180.1	28.5	134.0	23.6	112.3	24.8	95.6
	3	37.3	404.8	38.7	279.4	32.4	283.8	34.6	220.5	27.3	176.2	31.0	166.9
	4	42.5	628.8	45.7	419.0	36.8	441.7	42.6	372.9	30.5	254.2	39.4	307.3
	5	50.4	946.6	50.7	550.5	42.4	596.1	46.6	485.0	34.3	347.7	42.5	385.9

Table 5.1: Response time mean and variance comparison for measurement and model of mixed read and write request streams for 4-disk RAID 01.

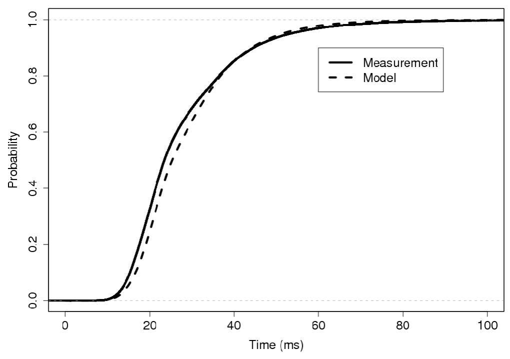
Table 5.1 compares modelled and measured means and variances for a 4-disk RAID 01 array with mixed arrival streams of read and write requests for varying arrival rates and request sizes. Figure 5.2 presents the pdfs and corresponding cdfs for 2 block mixed read and write requests for the three read/write combinations at an arrival rate of $\lambda = 0.03$ requests/ms. We observe excellent agreement between measured and modelled means, variances, pdfs and cdfs. The bimodal nature of both modelled and measured pdfs represent the respective peaks of read and write requests. Figure 5.3 shows measured and modelled mean response times for arrival streams with varying proportions of reads and writes for RAID 01. Figure 5.4 displays a selection of full pdf and cdf results for 8 disk RAID 01 mixed reads and writes. We observe good agreement between measured and modelled results in all these cases.



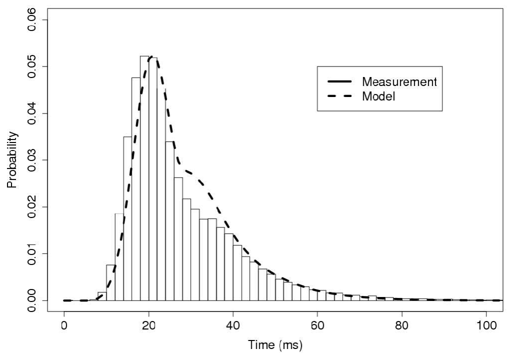
(a) 25% read requests, 75% write requests cdf



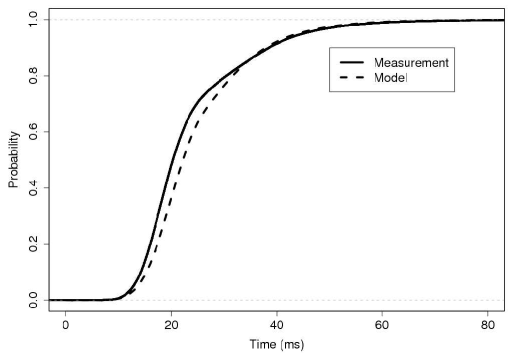
(b) 25% read requests, 75% write requests pdf



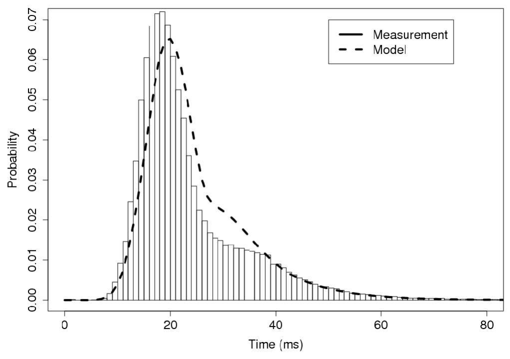
(c) 50% read requests, 50% write requests cdf



(d) 50% read requests, 50% write requests pdf



(e) 75% read requests, 25% write requests cdf



(f) 75% read requests, 25% write requests pdf

Figure 5.2: 4-disk RAID 01 2-block request response time pdfs and cdfs for arrival streams of mixed read and write requests and rate 0.03 requests/ms.

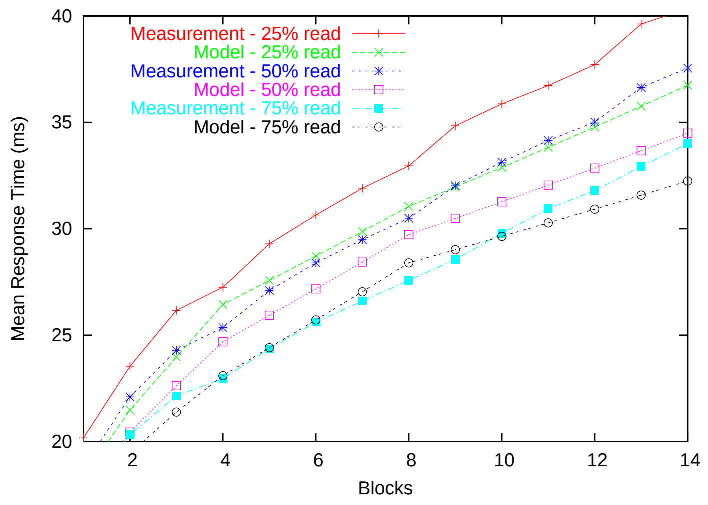


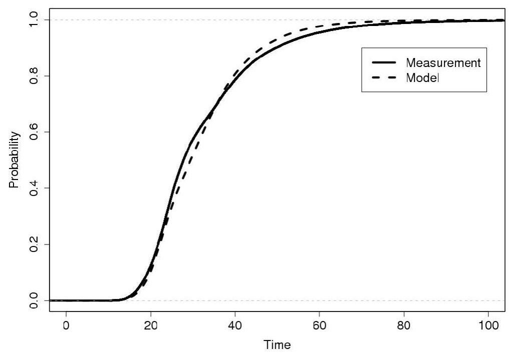
Figure 5.3: Comparison of mean response time against block size for 8-disk RAID 01 with mixed arrival streams of read and write requests and rate 0.01 requests/ms.

RAID 5

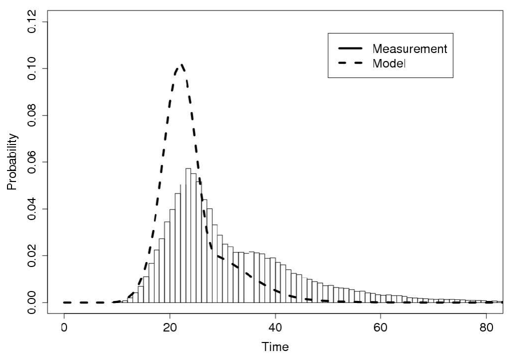
# Blks	25% Reads, 75% Writes				50% Reads, 50% Writes				75% Reads, 25% Writes			
	Measured		Modelled		Measured		Modelled		Measured		Modelled	
	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)	Mean (ms)	σ^2 (ms^2)
1	42.5	544.1	34.9	292.6	34.0	408.8	28.2	258.6	25.8	257.8	21.9	166.1
2	43.3	537.8	36.4	217.2	35.4	401.0	30.4	200.6	28.1	248.8	24.6	134.7
3	27.6	126.9	23.8	46.5	25.8	113.8	23.0	42.2	23.9	89.3	22.2	37.6
4	33.6	244.8	43.9	365.0	30.2	206.9	36.9	312.0	26.7	142.0	30.2	211.5
5	33.7	234.5	36.5	188.4	30.7	198.4	32.5	158.8	27.3	144.4	28.5	112.9

Table 5.2: Response time mean and variance comparison for measurement and model of mixed read and write request streams for 4-disk RAID 5 with arrival rate 0.01 requests/ms.

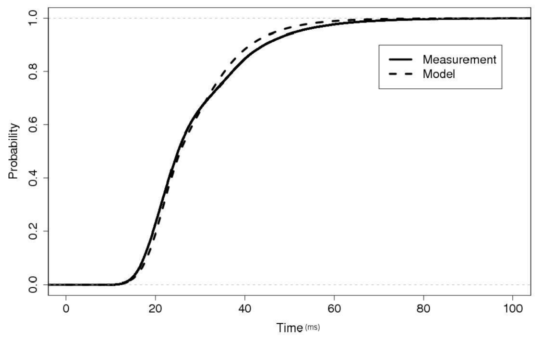
Table 5.2 presents modelled and measured means and variances for RAID 5 with an arrival rate of 0.01 requests per ms. Figure 5.5 shows measured and modelled mean response times for arrival streams with varying proportions of reads and writes for RAID 5. Figure 5.6 displays a selection of full pdf and cdf results for RAID 5 mixed reads and writes. We observe especially good agreement between



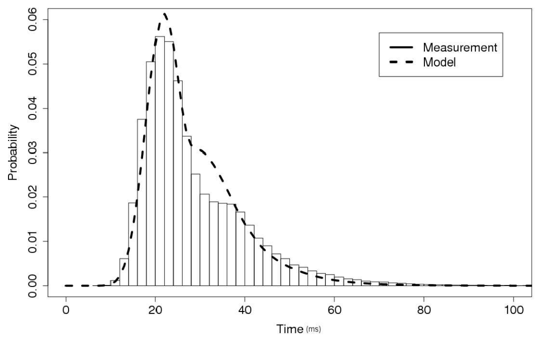
(a) 25% read requests, 75% write requests, $b = 3$, $\lambda = 0.03$, cdf



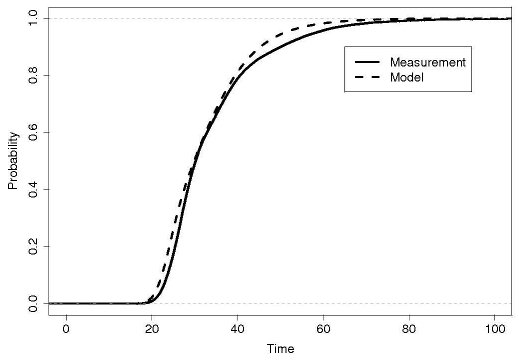
(b) 25% read requests, 75% write requests, $b = 3$, $\lambda = 0.03$, pdf



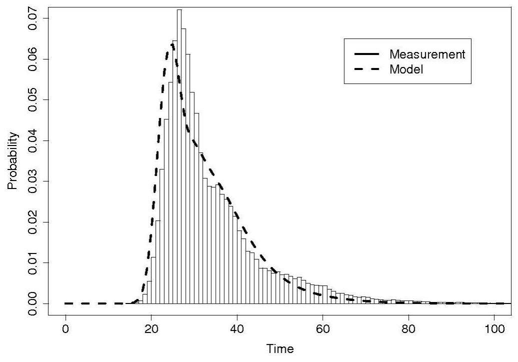
(c) 50% read requests, 50% write requests, $b = 3$, $\lambda = 0.03$, cdf



(d) 50% read requests, 50% write requests, $b = 3$, $\lambda = 0.03$, pdf



(e) 75% read requests, 25% write requests, $b = 14$, $\lambda = 0.01$, cdf



(f) 75% read requests, 25% write requests, $b = 14$, $\lambda = 0.01$, pdf

Figure 5.4: 8-disk RAID 01 b-block request response time pdfs and cdfs for arrival streams of mixed reads and writes with rate λ requests/ms.

measured and modelled results here. Particularly noteworthy is Figure 5.6(b), in which the model accurately captures the bimodal distribution of the measured results.

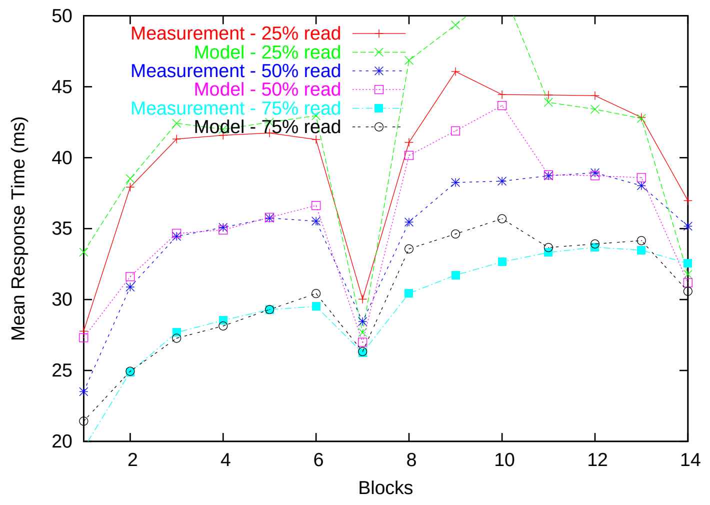
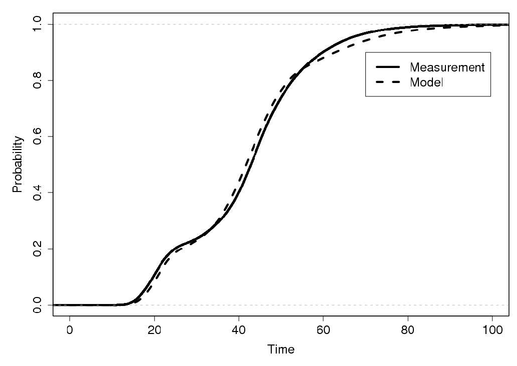


Figure 5.5: Comparison of mean response time against block size for 8-disk RAID 5 with mixed arrival streams of read and write requests and rate 0.01 requests/ms.

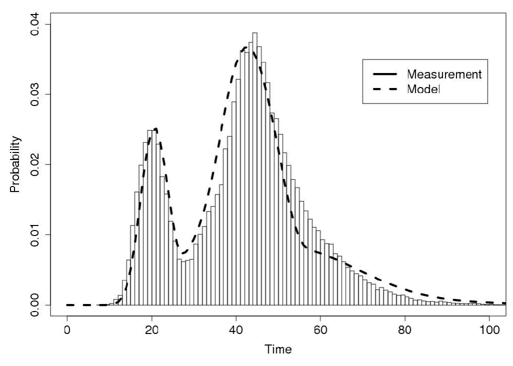
5.2.2 Multiclass RAID 5 Write Model

A RAID 5 partial stripe write is composed of two subrequests: a pre-read followed by writing the partial stripe and new parity block. The array must wait for all the pre-reads to complete and the new parity to be calculated before the partial stripe write and new parity block can be written to disk. The partial stripe write subtasks are then given priority over any other request in the disk queue.

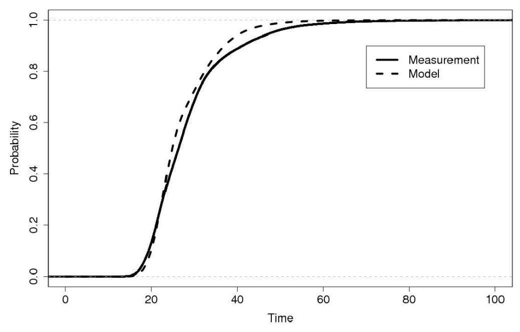
The RAID 5 write model in the previous chapter does not explicitly represent these two subrequests and instead computes the cdf of the overall response time



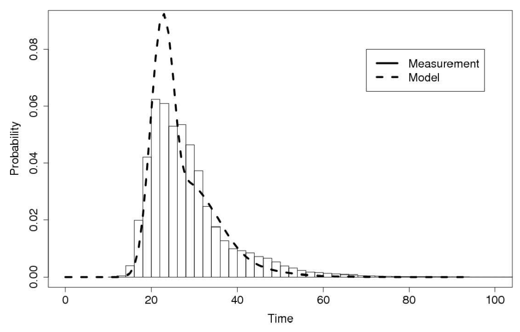
(a) 25% read requests, 75% write requests, $b = 4$, $\lambda = 0.01$, cdf



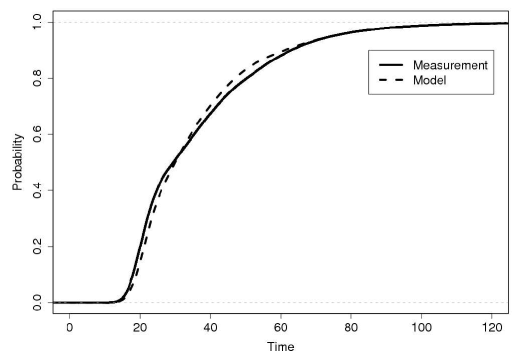
(b) 25% read requests, 75% write requests, $b = 4$, $\lambda = 0.01$, pdf



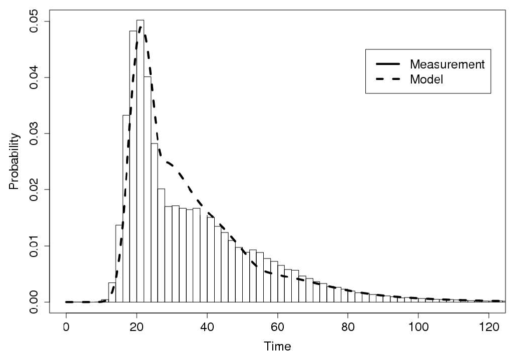
(c) 50% read requests, 50% write requests, $b = 7$, $\lambda = 0.01$, cdf



(d) 50% read requests, 50% write requests, $b = 7$, $\lambda = 0.01$, pdf



(e) 75% read requests, 25% write requests, $b = 4$, $\lambda = 0.03$, cdf



(f) 75% read requests, 25% write requests, $b = 4$, $\lambda = 0.03$, pdf

Figure 5.6: 8-disk RAID 5 b-block request response time pdfs and cdfs for arrival streams of mixed reads and writes with rate λ requests/ms.

based on the average of the service times of the pre-read and the partial stripe write subrequests. Therefore we now present a new multiclass RAID 5 partial stripe write model which employs two classes of request to separately model the pre-read and partial stripe write subrequests.

We assume that the arrival streams to the RAID 5 system are still composed of random access requests of homogeneous sizes and types with b logical blocks in each request. Furthermore there are n homogeneous disks in the array and the arrival rate of logical I/O requests to the array is λ. We first introduce the multiclass RAID 5 write model before extending it to include non-preemptive priority as well.

Multiclass RAID 5 Write Requests

We denote the cdfs of the response time of the pre-read subrequest as $W_1(t)$ and of the partial stripe write subrequest as $W_0(t)$. In a multiclass system, the total arrival rate to a queue (disk), γ, is the sum of the arrival rates to a queue for each class; thus $\gamma = \gamma_1 + \gamma_0$.

Using our assumptions about RAID 5 write requests discussed in Section 4.3.3 and Equation (5.1), the overall response time distribution of a single partial stripe write request $W_{\text{write}}(t)$ (calculated as the weighted average of the time to complete two pre-read subrequests and two partial stripe write subrequests) is:

$$W_{\text{write}}(t) = \frac{\gamma_1}{\gamma} W_1\left(\frac{t}{2}\right) + \frac{\gamma_0}{\gamma} W_0\left(\frac{t}{2}\right) \tag{5.2}$$

In Section 4.3.3 we presented the RAID 5 write model, which averaged the behaviour of the pre-read and partial stripe write subrequests. In that model, the arrival rate to each disk is the sum of the arrival rates to that disk for both subrequests, γ, and the service time distribution is the average of the service time distributions of both subrequests. In the multiclass model, the arrival rate at each disk remains the combined rate, γ, but the service time distributions are no longer averaged but used in their original form as the service time distribution of a pre-read or a partial stripe write. Similarly in the previous model the number of disks accessed in a request was calculated as the average of the number of disks accessed by a pre-read and a partial stripe write subrequest, which is no longer necessary here. The justification for the calculation of service times, number of disks accessed and arrival rate are provided in Section 4.3.3. Here we summarise the resulting multiclass cdfs.

If $b < \frac{n-1}{2}$, the cdfs of the response time of the first subrequest, $W_1(t)$, and of the partial stripe write subrequest, $W_0(t)$, are given by:

$$\begin{aligned} W_1(t) &= \left(W_d\left(t, \tfrac{2\lambda(b+1)}{n}, \tfrac{1}{E[R]+E[S]+E[T_1]}\right)\right)^{b+1} \\ W_0(t) &= \left(W_d\left(t, \tfrac{2\lambda(b+1)}{n}, \tfrac{1}{\frac{b(E[R]+E[S])+R_{max}}{b+1}+E[T_1]}\right)\right)^{b+1} \end{aligned}$$

Since both sets of subtasks access the same number of disks, the arrival rates to each queue are $\gamma_1 = \gamma_0 = \frac{\lambda(b+1)}{n}$.

The cdfs of the response times of the subrequests if $\frac{n-1}{2} \leq b < n-1$ are:

$$\begin{aligned} W_1(t) &= \left(W_d\left(t, \lambda, \tfrac{1}{E[R]+E[S]+E[T_1]}\right)\right)^{n-b-1} \\ W_0(t) &= \left(W_d\left(t, \lambda, \tfrac{1}{E[R]+E[S]+E[T_1]}\right)\right)^{b+1} \end{aligned}$$

The arrival rates to each disk within each class are:

$$\gamma_1 = \frac{\lambda(n-b-1)}{n} \qquad \gamma_0 = \frac{\lambda(b+1)}{n}$$

When a partial stripe write (either large or small) follows at least one full stripe write, the first subrequest includes the full stripe write and so will write to all n disks. The second subrequest is only the partial stripe write and hence will only write to $b_{mod} + 1$ disks. The arrival rates to each disk for each class are:

$$\gamma_1 = \lambda \qquad \gamma_0 = \frac{\lambda(b_{mod}+1)}{n}$$

In the case that a small partial stripe follows at least one full stripe write, the first subrequest is made up of $k = \lfloor \frac{b}{n-1} \rfloor$ block writes to each of the n disks followed by pre-reads to $b_{mod} + 1$ disks. The second subrequest writes the new data and parity to $b_{mod} + 1$ disks. The cdfs of the response times of the subrequests are:

$$\begin{aligned} W_1(t) &= \left(W_d\left(t, \frac{\lambda(n+b_{mod}+1)}{n}, \frac{1}{E[R]+E[S]+E[T_{k+\frac{b_{mod}}{n}}]}\right)\right)^n \\ W_0(t) &= \left(W_d\left(t, \frac{\lambda(n+b_{mod}+1)}{n}, \frac{1}{\frac{b_{mod}(E[R]+E[S])+R_{max}}{b_{mod}+1}+E[T_1]}\right)\right)^{b_{mod}+1} \end{aligned}$$

When a large partial stripe follows at least one full stripe write the first subrequest consists of k block writes to each of the n disks followed by pre-reads to $n - b_{mod} - 1$ disks. The second subrequest writes the new data and parity to the remaining $b_{mod} + 1$ disks. One of these disks will not have to seek again, as it will be the last disk to have finished transferring the full stripe. The cdfs of the response times of the subrequests are then:

$$\begin{aligned} W_1(t) &= \left(W_d\left(t, \frac{\lambda(n+b_{mod}+1)}{n}, \frac{1}{E[R]+E[S]+E[T_{k+\frac{n-b_{mod}-1}{n}}]}\right)\right)^n \\ W_0(t) &= \left(W_d\left(t, \frac{\lambda(n+b_{mod}+1)}{n}, \frac{1}{\frac{b_{mod}(E[R]+E[S])}{b_{mod}+1}+E[T_1]}\right)\right)^{b_{mod}+1} \end{aligned}$$

Priority

In reality, the partial stripe write subrequest is given queueing priority when issued. Therefore, to improve the model, the class 0 jobs are given a high priority and all other jobs (including reads) are given low priority. This requires two priority levels, which we represent by two classes where class 0 has higher, non-preemptive priority than class 1. Each class has an arrival rate γ_i, a service time Laplace-Stieltjes transform (LST) $X_i^*(s)$ and a mean service rate μ_i. The LST of the response time distribution for a job of class i, denoted $W_i^*(s)$, can be derived from Equation (2.17) as:

$$W_0^*(s) = \frac{((1-\rho)s+\gamma_1(1-X_1^*(s)))X_0^*(s)}{\gamma_0(X_0^*(s)-1)+s} \tag{5.3}$$

$$W_1^*(s) = \frac{(1-\rho)(s+\gamma_0(1-M^*(s)))X_1^*(s)}{\gamma_1(X_1^*(s+\gamma_0(1-M^*(s)))-1)+s} \tag{5.4}$$

where $\rho = \frac{\gamma_0}{\mu_0} + \frac{\gamma_1}{\mu_1}$. $M^*(s)$ is the LST representing the sum of the service times of arriving class 0 jobs while a class 1 job is servicing. It is defined self-referentially by:

$$M^*(s) = X_0^*(s+\gamma_0(1-M^*(s)))$$

As soon as the new parity is calculated, the partial stripe write subrequest is prioritised. Thus, we can give class 0 jobs high priority and all other jobs (including reads) low priority. Using Equations 5.3 and 5.4 instead of the Pollaczek-Khintchine transform equation, the response time distribution can be derived for high and low priority jobs on a single disk. The overall write request response time distribution can then be calculated by applying Equation (5.2) using the same method as in the non-priority multiclass case.

Validation

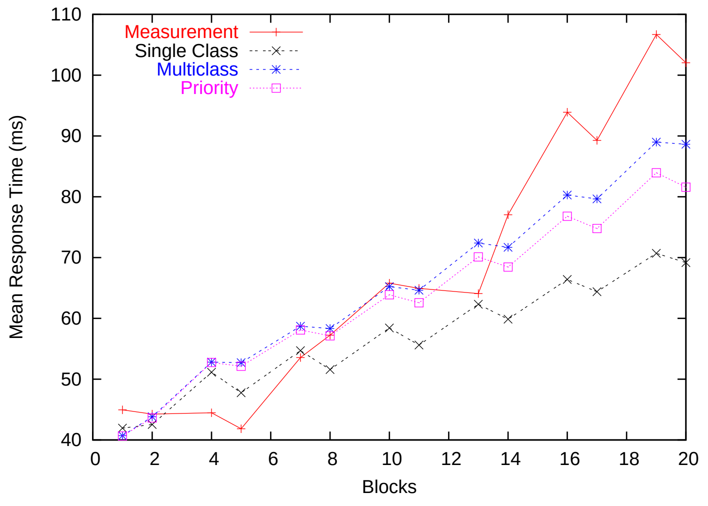
We now validate our three models for RAID 5 partial stripe write requests against device measurements. We refer to the model presented in the previous chapter as the single class model and to the two models presented above as the multiclass and priority models respectively.

In Figure 5.7 mean response times are presented for the three different models against device measurements for increasing block sizes and for arrival rates of 0.01 and 0.02 requests/ms. For small block sizes and loads, the single class model most often predicts means closest to the measured results. As block size increases, the means predicted by the multiclass and priority models are closer to the measured results. For large block sizes, the multiclass model clearly outperforms the other two models. However, the priority model means are reasonably close to the measured results for all block sizes. Table 5.3 contains means and variances for all these cases. It should be noted that the it is not always the case that the model with the closest mean response time has the closest variance to the measurements. In fact, in most cases, the multiclass model consistently has the closest variance to the measured variance.

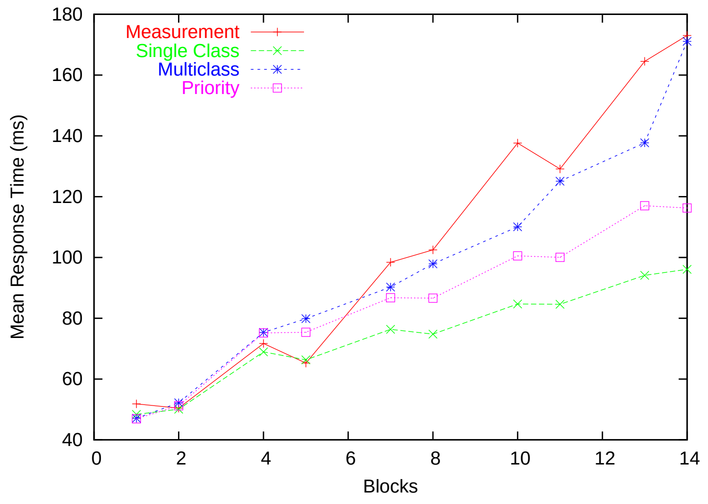
λ	#	Measured		Single Class		Multiclass		Priority	
		Mean	σ^2	Mean	σ^2	Mean	σ^2	Mean	σ^2
(ms^{-1})	Blks	(ms)	(ms^2)	(ms)	(ms^2)	(ms)	(ms^2)	(ms)	(ms^2)
0.01	1	45.0	148.7	41.9	258.5	40.7	227.2	40.7	232.7
	2	44.3	135.2	42.5	179.0	43.8	208.6	43.6	202.4
	4	44.5	595.9	51.1	340.5	52.8	435.0	52.7	468.6
	5	41.8	494.3	47.8	271.7	52.7	403.1	52.1	387.4
	7	53.5	903.4	54.7	394.0	58.7	628.4	58.1	619.7
	8	57.2	1084.4	51.6	326.0	58.3	616.6	57.1	519.8
	10	65.8	1468.0	58.4	456.6	65.2	907.5	63.9	819.5
	11	64.9	1515.5	55.6	391.0	64.6	941.3	62.6	696.9
	13	64.16	1630.6	62.3	530.4	72.4	1295.4	70.1	1076.6
	14	77.0	1992.6	59.9	468.4	71.7	1414.5	68.4	926.1
	16	93.9	3327.9	66.4	616.1	80.3	1822.2	76.8	1400.7
	17	89.3	3216.2	64.4	559.9	79.6	2087.2	74.8	1216.2
	19	106.7	4710.2	70.7	715.5	89.0	2526.2	84.0	1803.2
	20	102.0	4331.6	69.2	667.7	88.6	3029.9	81.6	1578.2
0.02	1	51.9	278.4	48.4	466.8	47.1	429.1	46.9	475.6
	2	50.4	251.9	50.1	411.5	52.2	472.1	51.2	454.8
	4	71.7	2496.7	69.0	975.0	75.4	1430.4	75.2	1726.6
	5	65.3	2139.6	66.3	847.9	79.9	1731.1	75.4	1669.2
	7	98.4	5359.0	76.4	1235.3	90.3	2534.0	86.8	2509.8
	8	102.5	5863.0	74.8	1123.4	97.9	3588.7	86.6	2481.7
	10	137.7	10234.3	84.7	1573.2	110.1	4643.0	100.5	3686.9
	11	129.1	9171.6	84.6	1497.8	125.1	7967.5	100.0	3738.5
	13	164.5	18646.5	94.1	2014.7	137.7	8829.0	117.0	5471.1
	14	173.0	15746.0	96.1	2012.2	171.1	19712.4	116.3	5706.9

Table 5.3: Response time mean and variance comparison for measurement and models of the three 4-disk RAID 5 write models.

Figure 5.8 compares the pdfs and cdfs of the three models with measurements in the cases where each of the models had the closest mean and variance to the measurements. Figures 5.8(a) and 5.8(b) are a 2-block write with an arrival rate of 0.02 requests/ms; the single class model gives the best mean and variance in this case. Figures 5.8(e) and 5.8(f) are a 14-block write with an arrival rate of 0.02 requests/ms; the multiclass model gives the best mean and variance here. Figures 5.8(c) and 5.8(d) are an 8-block write with an arrival rate of 0.01 requests/ms; here the priority model gives the best mean and variance.

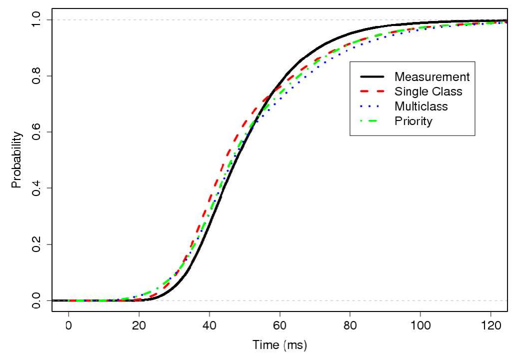


(a) $\lambda = 0.01$ request/ms

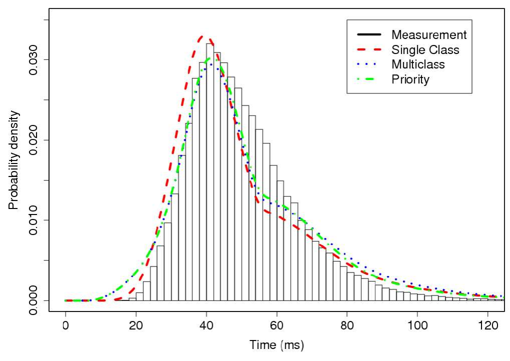


(b) $\lambda = 0.02$ requests/ms

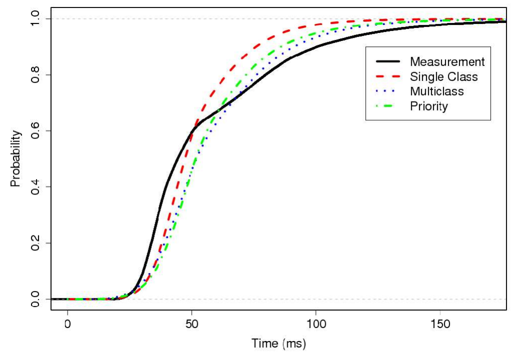
Figure 5.7: Comparison of mean response time for all models against block size for 4-disk RAID 5 partial stripe writes for different values of λ.



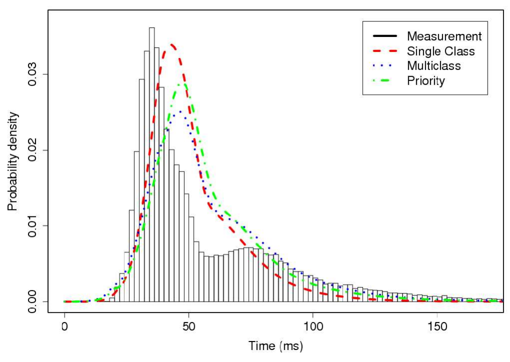
(a) 2-block requests, $\lambda = 0.02$ requests/ms, cdf



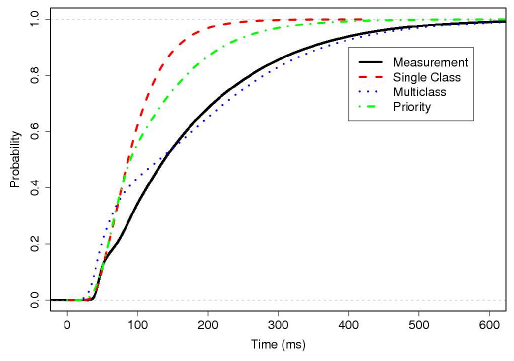
(b) 2-block requests, $\lambda = 0.02$ requests/ms, pdf



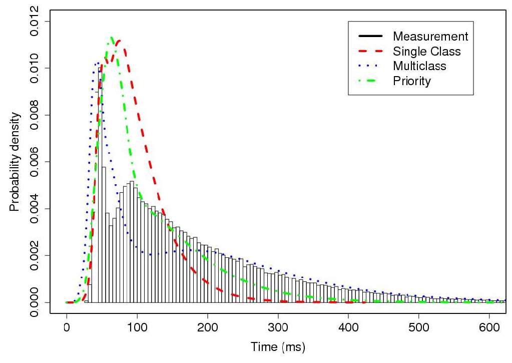
(c) 8-block requests, $\lambda = 0.01$ requests/ms, cdf



(d) 8-block requests, $\lambda = 0.01$ requests/ms, pdf



(e) 14-block requests, $\lambda = 0.02$ requests/ms, cdf



(f) 14-block requests, $\lambda = 0.02$ requests/ms, pdf

Figure 5.8: Selected pdfs and cdfs of 4-disk RAID 5 write request response times for the three models and arrival rate λ requests/ms.

Interestingly, the multiclass and priority models only outperform the single class model for larger workloads and request sizes. This illustrates the difficulty in modelling RAID 5 write requests. We have presented four possible models here for RAID 5 so far including the simulation model and surprisingly, the most consistently accurate model is the single-class model which abstracts RAID 5 behaviour the most.

5.3 Workloads with Different Sized Arrivals

We now consider a different heterogeneous arrival stream to support the modelling of Markovian arrival streams with different size requests. In order to do this, we utilise the properties of queues with bulk arrivals. We define each I/O request to be equivalent to a single bulk arrival, and subtasks making up a request are equivalent to jobs within a batch. This could be an arrival either to a single disk (queue) or a RAID system (fork-join queue). We assume all requests are random disk accesses. Thus, the first subtask within a request will have a service time that includes time to seek and rotate to the desired position on the disk. The remaining subtasks will be sequential. We first present a model for a single disk with different sized requests in Section 5.3.1 and then extend it for RAID 01 and RAID 5 in Sections 5.3.2 and 5.3.3 respectively. We then validate all these models against device measurements.

5.3.1 Single Disk

The number of subtasks in a request, B, can be described by a discrete pdf, $f_B(x)$ with probability generating function $G_B(x)$. The first subtask in the batch will have a service time $X_{RAND} = S + R + T$. The remaining subtasks are sequential and hence have service time $X_{SEQ} = T$. These service time random variables have corresponding LSTs $X^*_{RAND}(\theta)$ and $X^*_{SEQ}(\theta)$. As before, we assume all service times are independent. Then, if we define X_B as the random variable of the service time of all the subtasks in a single request, the corresponding LST can be calculated as follows from the definition of a Laplace-Stieltjes transform, using conditional expectation:

$$X^*_B(\theta) = E[E[e^{-\theta(X_{RAND}+X_{SEQ_1}+\ldots+X_{SEQ_{B-1}})}|B]] \tag{5.5}$$

There is always one fewer sequential subtask than there are subtasks in a batch. The number of sequential subtasks has a probability density function $f_{B-1}(x) = f_B(x+1), x = 0, 1, 2, \ldots$. Since X_{RAND} and X_{SEQ} are independent,

$$\begin{aligned} X^*_B(\theta) &= E[E[e^{-\theta X_{SEQ}}]^{(B-1)}E[e^{-\theta X_{RAND}}]] \\ &= E[(X^*_{SEQ}(\theta))^{(B-1)}X^*_{RAND}(\theta)] \\ &= G_{B-1}(X^*_{SEQ}(\theta))X^*_{RAND}(\theta) \end{aligned}$$

where $G_{B-1}(z) = \sum_{i=0}^{\infty} f_B(i+1)z^i$, a probability generating function. This LST is substituted into the Pollaczek-Khintchine transform equation [51] as the LST of service time; hence the response time LST is:

$$W_d^*(\theta) = \frac{(1-\rho)\theta X_B^*(\theta)}{\lambda X_B^*(\theta) - \lambda + \theta} \tag{5.6}$$

where $\rho = \lambda E[X_B]$. The mean job service time, $E[X_B]$ can be calculated from $X_B^*(\theta)$, yielding:

$$\begin{aligned} E[X_B] &= E[X_{RAND}] + E[X_{SEQ}]E[B-1] \\ &= E[S] + E[R] + E[B]E[T] \end{aligned}$$

As before, $W_d^*(\theta)$ is easily numerically inverted to obtain the distribution of response time.

5.3.2 RAID 01

In RAID 01, subtasks from each request are striped across the disks. A read request stripes all its subtasks across the disks and a corresponding write request stripes double the number of subtasks. As a consequence of the striping, a single disk in a disk array will only see a fraction of the subtasks of a request. If we define B_R as the number of subtasks in a request on a single disk in the array and the mean number of subtasks in a request is $E[B]$, the mean request size per disk, $E[B_R]$, is:

$$E[B_R] = \begin{cases} 1 & E[B] < n \\ \frac{E[B]}{n} & \text{otherwise} \end{cases}$$

If there are less than a mean number of n blocks in a request, there cannot be less than one block written to a disk, but less disks will be written to on the array. Consequently, the number of disks in the array accessed by each request depends on the number of subtasks in a job and is therefore dependent on B. We note all disks will be accessed unless a job has fewer subtasks than there are disks. We define the parameter d_B as the mean number of disks in use:

$$d_B = n - \sum_{i=1}^{n-1} (n-i)\, f_B(i)$$

Similarly the arrival rate, γ, at each disk is dependent on the number of disks accessed.

$$\gamma = \frac{\lambda}{n} \left(p_{read} d_B + p_{write} d_{2B} \right)$$

The response time distribution of a read and write request on RAID 01 can now be defined as:

$$W_{\text{read}}(t) = \left(W_d \left(t, \gamma, \frac{1}{E[R] + E[S] + E[B_R]E[T]} \right) \right)^{d_B}$$
$$W_{\text{write}}(t) = \left(W_d \left(t, \gamma, \frac{1}{E[R] + E[S] + E[2B_R]E[T]} \right) \right)^{d_{2B}}$$

The single disk response time cdf $W_d(t, \gamma, \mu)$ is the numerical inversion of the LST in Equation (5.6) with $X_B^*(\theta)$ replaced by $X_{B_R}^*(\theta)$.

5.3.3 RAID 5

RAID 5 is block-interleaved distributed parity. The parity is defined as the XOR of all the data on a stripe. Owing to the way in which RAID 5 distributes its parity blocks, RAID 5 reads can be modelled in a similar manner to RAID 01 reads. This is because, although a RAID 5 read operation accesses only $n - 1$ disks per stripe (given an n-disk array), the same number of disks can be accessed because the parity is distributed across n disks. RAID 5 write operations are significantly more complex due to the need to update parity blocks. If a full stripe is written, then all the new data is immediately available for parity calculation; however, if less than a full stripe is written, then the new parity can only be calculated with old data already written on the stripe.

Existing analytical queueing models of RAID 5 present RAID 5 write models which have a fixed number of subtasks in a job [54, 73] or a variable number that is fixed to never exceed a full stripe [25]. Furthermore, all these models assume that all data will be written starting from the beginning of a stripe (stripe-aligned). In reality, jobs can be of any size and be written in any place on a stripe.

A randomly-sized RAID 5 write with a skew (i.e. not stripe-aligned) could consist of any or all of the following: a partial stripe write followed by a number of full stripe writes followed by another partial stripe write. We summarise the five possible procedures ($P_1, \ldots, P_5$) for a RAID 5 write in Table 5.4. Here *Rand* is a random seek and transfer, *Seq* is a sequential operation and *NOP* represents no operation. Any initial write or pre-read and all partial stripe write subrequests will demand a random disk access. Writing further full stripes or a pre-read that follows a full-stripe write will have sequential disk accesses.

	P_1	P_2	P_3	P_4	P_5
pre-read	Rand	NOP	NOP	NOP	Rand
partial stripe write	Rand	NOP	NOP	NOP	Rand
full stripe	Seq	Rand	NOP	Rand	NOP
further full stripes	Seq	Seq	NOP	Seq	NOP
pre-read	Seq	Seq	Rand	NOP	Seq
partial stripe write	Rand	Rand	Rand	NOP	Rand

Table 5.4: Possible RAID 5 write procedures.

Table 5.4 shows that there will never be more than three random accesses in a single RAID 5 write request. There will only be one random operation in the case that there are no partial stripes writes, namely that a job begins at the start of a stripe and that the number of subtasks in the job is divisible by $n-1$. However, our RAID model aims to find an average amount of accesses on a single disk and then apply it to all disks. This is more difficult in RAID 5, as the parity pre-reads and partial stripe writes are made only to certain disks in the array, depending on whether the parity calculation demands a read-modify-write or read-reconstruct-write. It is very likely, however, that no single disk will have more than two random accesses directed to it, since a partial stripe does not access all disks. Meanwhile, each disk will only have one random access per request if the request consists of a multiple of $n-1$ subtasks and the request starts at the beginning of a stripe (i.e. a full stripe write). Defining $f_{R5R}(x)$ as the probability of encountering x random subtasks in a request on a single disk, in terms of the job size probability distribution, $f_B(x)$:

$$f_{R5R}(x) = \begin{cases} \frac{1}{n-1}\sum_{i=1}^{\infty} f_B(i(n-1)) & x = 1 \\ 1 - \frac{1}{n-1}\sum_{i=1}^{\infty} f_B(i(n-1)) & x = 2 \\ 0 & \text{otherwise} \end{cases}$$

Similarly, defining $f_{R5S}(x)$ as the probability of encountering x sequential sub-tasks in a request on a single disk:

$$f_{R5S}(x) = \sum_{i=x(n-1)+1}^{(x+1)(n-1)} f_B(i)$$

In a similar way as for Equation (5.5), we derive $X_B^*(\theta)$ as:

$$X_B^*(\theta) = G_{R5R}(X_{RAND}^*(\theta))G_{R5S}(X_{SEQ}^*(\theta))$$

This can be used to calculate $E[X_B]$ straightforwardly.

In terms of the number of disks accessed, a RAID 5 write request accesses all disks unless it is only a partial stripe write with no full stripe writes. A small partial stripe write (read-modify-write), which occurs when $b < \frac{n-1}{2}$, involves accesses to $b+1$ disks (b data disks plus the parity disk). A large partial stripe write (read-reconstruct-write) accesses $n-b-1$ disks in the first subrequest and $b+1$ disks in the second subrequest, on average accessing $\frac{n}{2}$ disks per operation. Hence:

$$d_B = \sum_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} (i+1) f_B(i) + \frac{n}{2} \sum_{i=\lceil \frac{n-1}{2} \rceil}^{n-1} f_B(i) + n\left(1 - \sum_{i=1}^{n} f_B(i)\right)$$

These parameters can be used to calculate the response time distribution of a RAID 5 request in the same way as a RAID 01 request.

Simulation

Our simulation model for both RAID 01 and 5 supports requests consisting of a constant or variable number of subtasks. The number of subtasks is sampled from a specified probability distribution. The simulator does not currently support RAID 5 write requests with a variable number of subtasks. This is because the RAID 5 write simulation described in Chapter 4 is designed to be stripe-aligned (since the analytical model for constant request size is stripe-aligned) and would need to be modified significantly to be compared with the unstripe-aligned analytical model presented in this section. This is scope for future work.

Validation

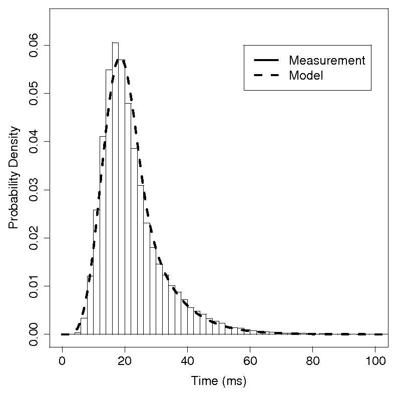
Figure 5.9 compares our analytical model predictions and measurement results for variable-sized arrivals to a single disk where the size of a request is generated using a geometric distribution. Results are presented for two different arrival rates ($\lambda = 0.01, 0.02$ requests per ms) and for arrival streams composed entirely of either read or write requests. In the case of read requests, we observe excellent agreement between model and measurement. In the case of write requests (Figures 5.9(c) and 5.9(f)) the agreement is less good, but is still reasonable and yields similar means and variances of response time. We speculate that the bimodal nature of the measurements may be due to some disk-specific write request handling behaviour that is not accounted for in the model.

Figure 5.10 compares model predictions and measurement results for bulk arrivals to a four-disk RAID 01 system. As with the single disk, the size of request is generated using a geometric distribution and results are presented for two arrival rates. We observe reasonable agreement between model and measurement.

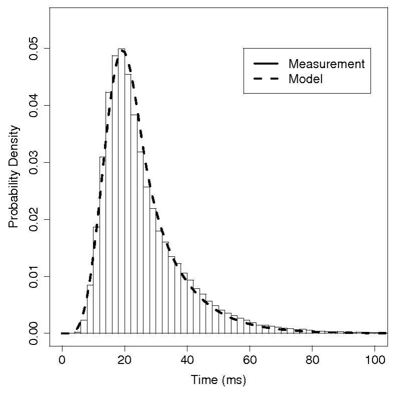
We note that all results thus far have been presented for arrival streams composed entirely of read or write requests. However, our model is capable of calculating results for arrival streams containing a mixture of both reads and writes. Figure 5.11 accordingly compares model predictions and measurement results in this case; we observe good agreement.

Finally, Figures 5.12 and 5.13 compare model predictions and measurement results for arrivals with geometrically-distributed sizes to a four-disk RAID 5 system. We observe excellent agreement for read requests in Figure 5.12. The fit for write requests is a little less exact due to the complicated nature of the pre-read and parity update operations that is approximated in our models. However, we do observe better agreement between model and measurement than we have done in any of our previous RAID 5 write models or simulation.

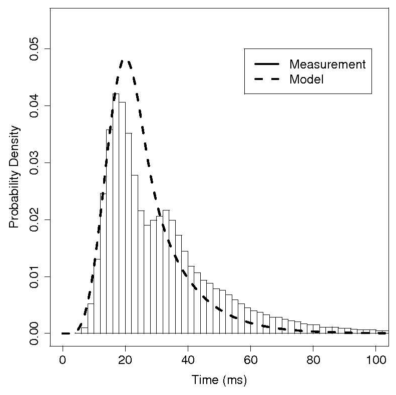
We compare our simulation model to device measurements and the analytical model. Figure 5.14 compares RAID 01 with arrival streams of variable request size sampled from a geometric distribution with a specified mean request size. We observe excellent agreement between simulation model and measurement in these cases. There is very close agreement between the analytical model and simulation model for both read and write requests on RAID 01.



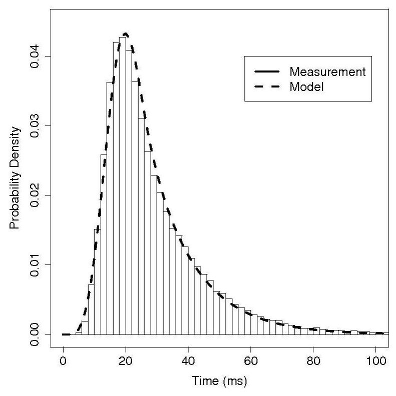
(a) mean job size = 3, reads, λ = 0.01



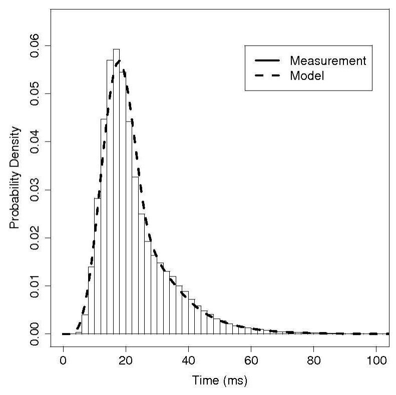
(b) mean job size = 4, reads, λ = 0.01



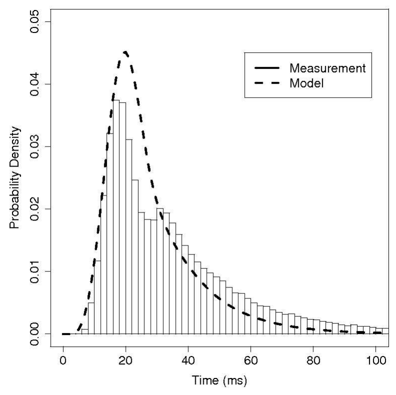
(c) mean job size = 4, writes, λ = 0.01



(d) mean job size = 5, reads, λ = 0.01

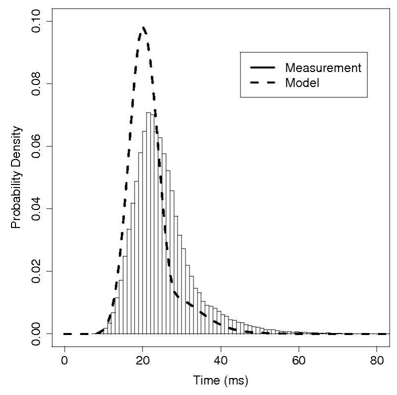


(e) mean job size = 2, reads, λ = 0.02

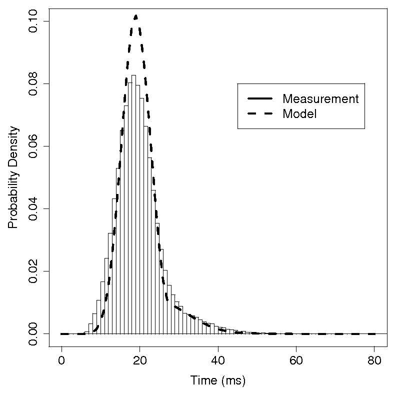


(f) mean job size = 3, writes, λ = 0.02

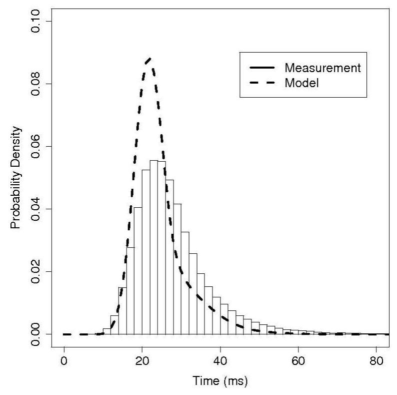
Figure 5.9: I/O request response time pdf of model against measurement on a single disk with different sized arrivals and rate λ requests/ms.



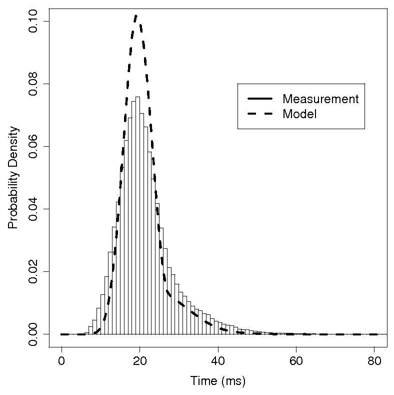
(a) mean job size = 2, writes, λ = 0.01



(b) mean job size = 3, reads, λ = 0.01

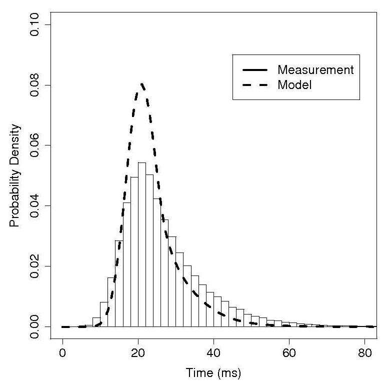


(c) mean job size = 3, writes, λ = 0.01

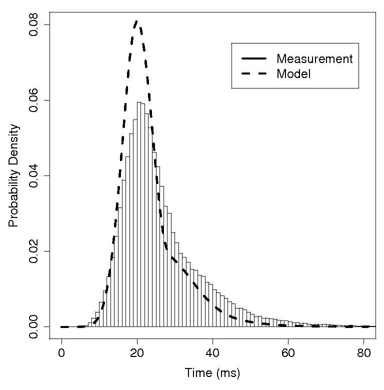


(d) mean job size = 4, reads, λ = 0.01

Figure 5.10: I/O request response time pdf of model against measurement on 4-disk RAID 01 with different sized arrivals and rate λ requests/ms.



(a) mean job size = 4, 50% reads, λ = 0.01



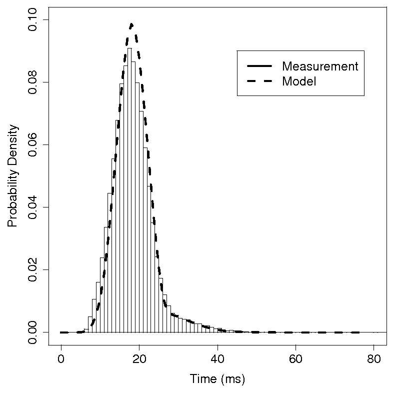
(b) mean job size = 2, 25% reads, λ = 0.02

Figure 5.11: I/O request response time pdf of model against measurement on 4-disk RAID 01 with different sized arrivals and a mix of reads and writes and rate λ requests/ms.

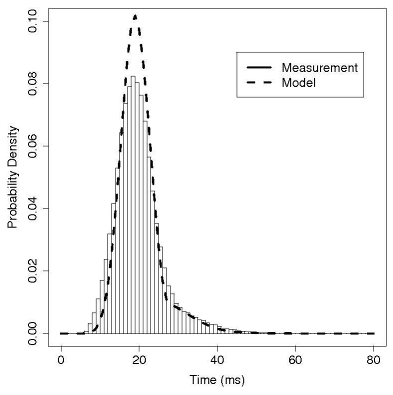
Similarly, Figure 5.15 compares simulation and analytical models and device measurements for RAID 5 read requests with size decided by a geometric distribution. We again observe excellent agreement for read requests.

5.4 Workloads With Bulk Arrivals

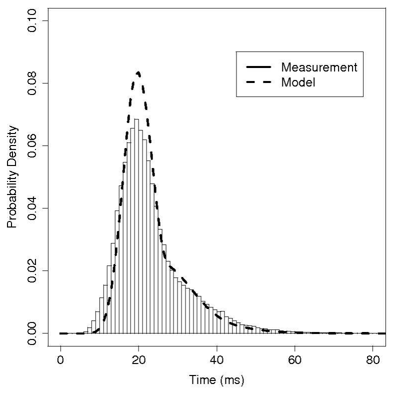
In the previous section we utilised the properties of queues with bulk arrivals to derive a model for arrival streams with different sized requests. In this section, we look to model bursty I/O request arrival streams by modelling each burst as a single bulk arrival. We therefore need to derive results for the response time of a single request in an $M/G/1$ queue with bulk arrivals. We derive the response time distribution for a random customer in an $M^X/G/1$ queue. Our approach is inspired by Harrison's derivation of the Laplace-Stieltjes Transform (LST) of



(a) mean job size = 2, reads, λ = 0.01

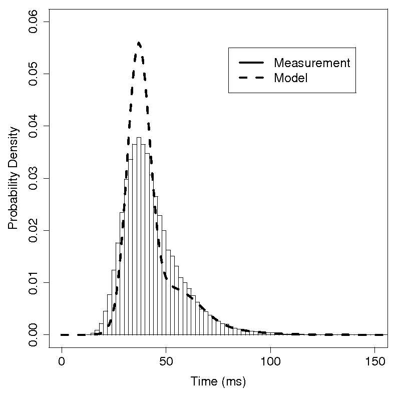


(b) mean job size = 3, reads, λ = 0.01

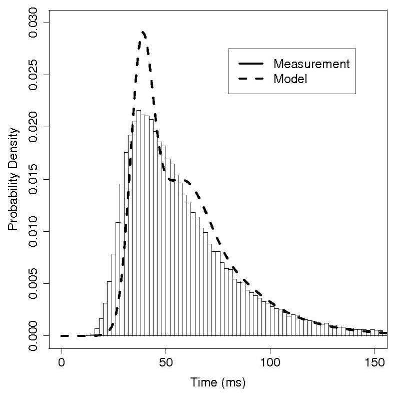


(c) mean job size = 4, reads, λ = 0.02

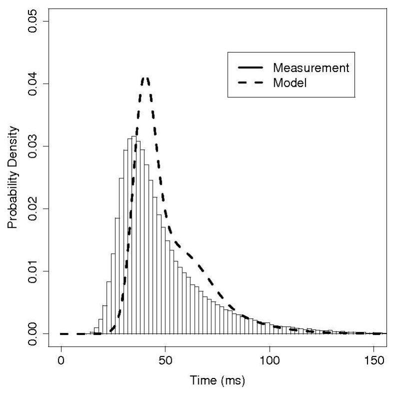
Figure 5.12: I/O request response time pdf of model against measurement for reads on 4-disk RAID 5 with different sized arrivals and rate λ requests/ms.



(a) mean job size = 2, writes, λ = 0.01

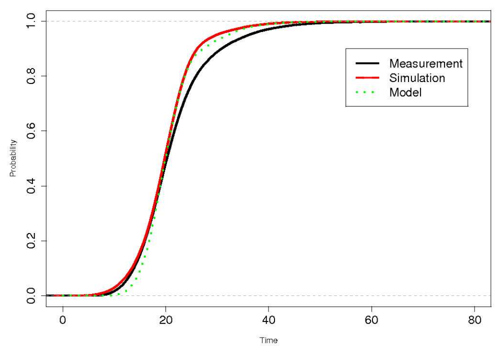


(b) mean job size = 3, writes, λ = 0.02

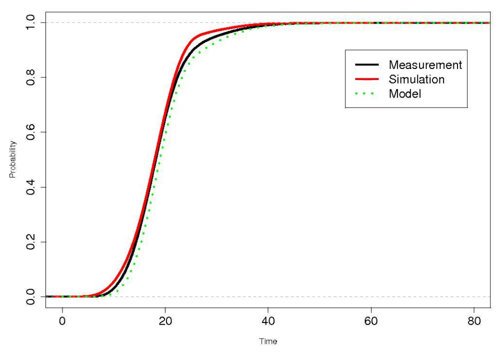


(c) mean job size = 5, writes, λ = 0.01

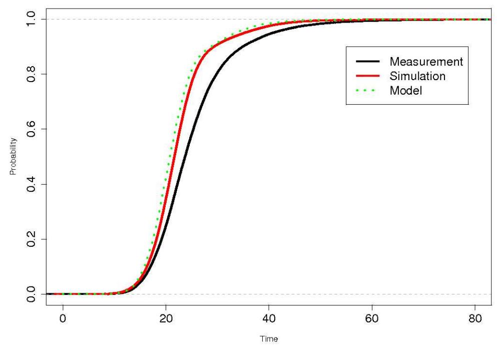
Figure 5.13: I/O request response time pdf of model against measurement for writes on 4-disk RAID 5 with different sized arrivals and rate λ requests/ms.



(a) 4-block mean read request, $\lambda = 0.01$



(b) 2-block mean read request, $\lambda = 0.02$



(c) 2-block mean write request, $\lambda = 0.01$

Figure 5.14: I/O request response time distributions of 4-disk RAID 01 with request sizes chosen from a geometric distribution and arrival rate λ requests/ms.

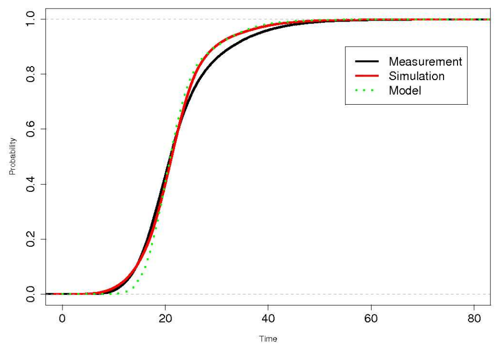


Figure 5.15: I/O read request response time distributions of 4-disk RAID 5 with request sizes chosen from a geometric distribution with mean size 5 blocks and arrival rate 0.01 requests/ms.

customer response time in an $M/G/1$ queue using conditional probability [49]. We can then apply this result to a single disk and RAID 01 before validating these models against device measurements.

5.4.1 Single Disk

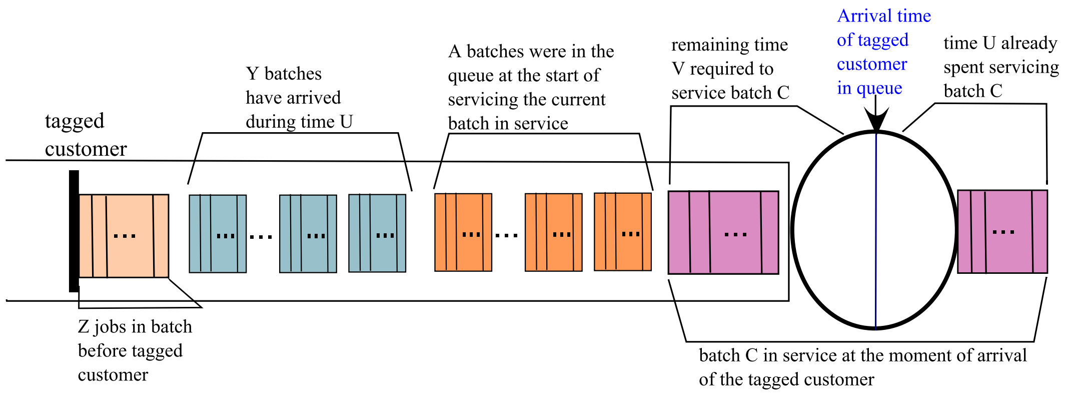

tagged customer
Y batches have arrived during time U
A batches were in the queue at the start of servicing the current batch in service
remaining time V required to service batch C
Arrival time of tagged customer in queue
time U already spent servicing batch C
Z jobs in batch before tagged customer
batch C in service at the moment of arrival of the tagged customer

Figure 5.16: The queue at the arrival instant of a tagged customer.

Here we model a disk drive as an $M^X/G/1$ queue where I/O requests are represented by customers in the queue. Figure 5.16 shows the state of an $M^X/G/1$ queue at the arrival instant of a randomly chosen customer, given that the queue is not empty at the arrival instant. At the arrival instant, a batch C is currently in service. This batch has completed Ums of service (the backward recurrence time [29]) and has Vms of service remaining (the forward recurrence time). When batch C started service, there were A batches queueing behind it. During time U, a further Y batches joined the queue. Within the arriving batch, there are Z customers (the backward recurrence size) ahead of the tagged customer. U and V are continuous random variables, and A, Y and Z are discrete random variables.

If B denotes the random variable describing batch size, and X the service time of a single customer, then the LST of the service time of an entire batch $X_B^*(\theta)$ can be defined as:

$$\begin{aligned} X_B^*(\theta) &= E[E[e^{-\theta(X_1+X_2+\ldots+X_B)}] \\ &= E[(X^*(\theta))^B] \\ &= G_B(X^*(\theta)) \end{aligned}$$

where $G_B(z)$ is the probability generating function of B. It is straightforward to show $E[X_B] = E[X]E[B]$.

The queueing time of the tagged customer, denoted by random variable Q, is the sum of the service times of all customers ahead (including the remaining service time of the customer in service), as follows:

$$\begin{aligned} &E[e^{-\theta Q} \mid Q > 0] = \\ &E[e^{-\theta(V+X_{B_1}+\ldots+X_{B_A}+X_{B_1}+\ldots+X_{B_Y}+X_1+\ldots+X_Z)}|U, V, Y, Z, A] \\ &= X^*(\theta)^Z G_B(X^*(\theta))^A G_B(X^*(\theta))^Y e^{-\theta V} \end{aligned}$$

Deconditioning on A and Z (which are independent of V, Y and each other), and Y (which can be expressed in terms of U given that the number of batches arriving has a Poisson distribution),

$$E[e^{-\theta Q} \mid Q > 0] = G_Z(X^*(\theta))G_A(G_B(X^*(\theta)))E[e^{-\theta V} e^{-\lambda(1-(G_B(X^*(\theta))))U} \mid U, V]$$

Here $G_Z(z)$ is the generating function of the discrete backward recurrence time (see Section 2.2.4). $G_A(z)$ is the generating function of queue length at the beginning of service of the first customer in a batch. By the random observer property [49], at equilibrium A is equivalent to N, the number of customers queueing immediately after the start of a batch service. $G_N(z)$ is a well known result [51]:

$$G_A(z) = G_N(z) = \frac{(1-\rho)(1-z)}{G_B(X^*(\lambda(1-z)) - z}$$

The joint density function of the forward and backward recurrence times U and V at a point (u, v) is $\frac{1}{E[X_B]} f_{X_B}(u+v)$ [51] where $f_{X_B}(t)$ is the pdf of batch service time X_B. Thus, deconditioning further,

$$\begin{aligned}
E[e^{-\theta Q} \mid Q > 0] &= \frac{G_Z(X^*(\theta))G_A(G_B(X^*(\theta)))}{E[X]E[B]} \\
&\quad \int_0^\infty \int_0^\infty e^{-\theta v} e^{-\lambda(1-(G_B(X^*(\theta))))u} f_{X_B}(u+v)\, du\, dv \\
&= \frac{G_Z(X^*(\theta))G_A(G_B(X^*(\theta)))}{E[X]E[B]} \\
&\quad \int_0^\infty \int_0^w e^{-\theta w} f_{X_B}(w) e^{\theta-\lambda(1-(G_B(X^*(\theta))))u}\, dw\, du \\
&= \frac{1}{E[X]E[B]} G_Z(X^*(\theta))G_A(G_B(X^*(\theta))) \\
&\quad (G_B(X^*(\lambda(1-G_B(X^*(\theta)))))) - G_B(X^*(\theta))) \\
&\quad \int_0^\infty e^{(\theta-\lambda(1-(G_B(X^*(\theta)))))u}\, du \\
&= \frac{1}{E[X]E[B](\theta - \lambda(1-(G_B(X^*(\theta)))))} \\
&\quad G_Z(X^*(\theta))G_A(G_B(X^*(\theta))) \\
&\quad (G_B(X^*(\lambda(1-G_B(X^*(\theta)))))) - G_B(X^*(\theta)))
\end{aligned}$$

If a batch arrives to an empty queue, then the queueing time for a randomly selected job in the batch is the time to service all the jobs ahead of it in the batch;

thus

$$E[e^{-\theta Q} \mid Q = 0] = G_Z(X^*(\theta))$$

Considering both cases (empty and non-empty queues), and given queue utilisation ρ, the LST of queueing time is:

$$\begin{aligned} Q^*(\theta) &= (1-\rho)G_Z(X^*(\theta)) + \rho E[e^{-\theta Q}|Q > 0] \\ &= \frac{(1-\rho)\theta G_Z(X^*(\theta))}{\theta - \lambda(1 - (G_B(X^*(\theta))))} \end{aligned}$$

Hence the response time LST for a randomly placed customer in a batch is

$$W^*(\theta) = Q^*(\theta)X^*(\theta). \tag{5.7}$$

The response time distribution is obtained by numerically inverting $W^*(\theta)$ [2].

5.4.2 RAID 01

It is not straightforward to extend the single disk model for batch arrivals to RAID 01. Our fork-join approximation assumes independence of response times for each disk; however, in the case of batch arrivals to RAID 01, although the service time distributions on the different disks can reasonably be assumed to be independent, under the assumption that all operations are full-stripe accesses, each queue will receive the same number of jobs per batch and hence queueing times across disks will have a high level of dependency.

Therefore we use a new method to calculate the response time distribution. The model results in a single queue whose service time distribution is calculated as the distribution of the maximum service time across all the disks in the array. This assumption of dependence of queueing (but not service) times is not exact, but approximates the reality that queueing times of requests to each disk in a RAID system will be highly correlated.

We begin by finding $F_X(t)$, the service time distribution of a single zoned disk, defined as the sum of seek time, rotational latency and data transfer time. The intricacies of zoning mean that this distribution cannot be found analytically, and must instead be calculated numerically by inversion of the service time LST or by convolution of the component parts. It is then possible to find the distribution of maximum service time across n disks using the maximum order statistic, i.e. $(F_X(t))^n$.

Equation (5.7) (from which we will derive our response time distribution) requires the LST of the maximum service time $X^*(\theta)$. Numerical calculation of this LST is theoretically possible, but in practice requires an infeasible amount of computation. As a means to more efficiently and elegantly obtain the LST, we proceed by fitting a logistic function:

$$f(t) = \frac{1}{1 + e^{a-bt}}$$

to the distribution of maximum service time $(F_X(t))^n$. The fitting can be accomplished by using a nonlinear least-squares Marquardt-Levenberg algorithm [83].

The LST of the logistic function is then:

$$X^*(\theta) = \mathrm{Hypergeometric2F1}[1, s/b, (b+s)/b, -e^a]$$

where $\mathrm{Hypergeometric2F1}$ is the hypergeometric function ${}_2F_1(a, b, c, z)$ which is the solution for y of the hypergeometric differential equation [135]:

$$z(1-z)y'' + [c - (a+b+1)z]y' - aby = 0$$

Substitution of $X^*(\theta)$ into Equation (5.7) then gives a readily-invertible expression for the distribution of response time in a RAID 01 system.

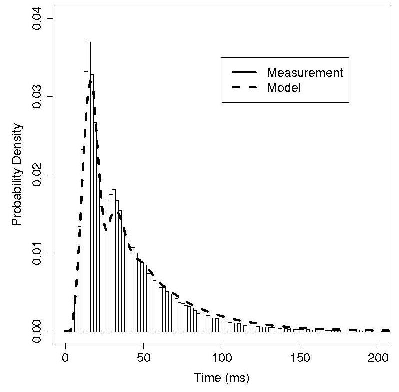
Simulation

The RAID simulator already supports bulk arrivals of I/O requests at the RAID controller, making use of JINQS's in-built support for arrivals that consist of a number of requests defined by a chosen probability distribution. The UML diagram in Figure 3.5 shows that the *Source* class in the disk simulator contains a *batchsize* attribute that will choose the batch size from a specified distribution sampler.

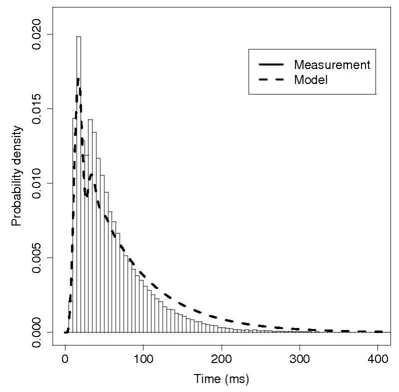
Validation

Figure 5.17 compares model predictions and measurement results for bulk arrivals to a single disk, with the number of requests in a batch generated using a geometric distribution. Results are presented for two different arrival rates ($\lambda = 0.01, 0.02$

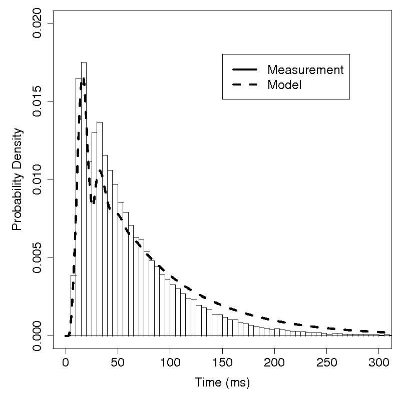
requests per ms) and for arrival streams composed entirely of batches of either read or write requests. All requests within batches consist of one 128KB block and are to random locations. We observe good agreement between model and measurement. The multiple peaks observed in both model and measurement arise from the variation of response time caused by the different possible batch sizes (with the most probable batch sizes yielding the highest peaks).



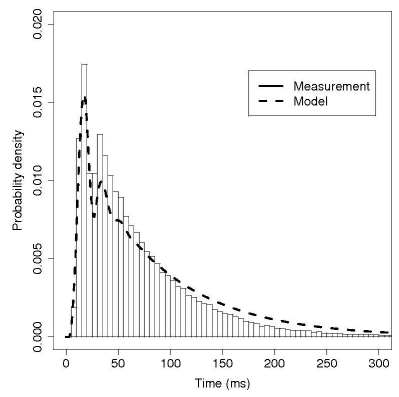
(a) mean batch size = 2, reads, λ = 0.01



(b) mean batch size = 2, writes, λ = 0.02



(c) mean batch size = 3, reads, λ = 0.01



(d) mean batch size = 3, writes, λ = 0.01

Figure 5.17: I/O request response time pdf of model against measurement on a single disk with bulk arrivals with rate λ requests/ms.

As described in Section 5.4.2, we can perform a least-squares fit of a logistic function to the (numerically calculated) distribution of maximum disk service time in order to more efficiently generate response time results for RAID 01 systems with batch arrivals. Figure 5.18 shows the logistic function fit for a 4-block read operation to a 4-disk system. The closeness of fit gives confidence in the accuracy of this approximation step.

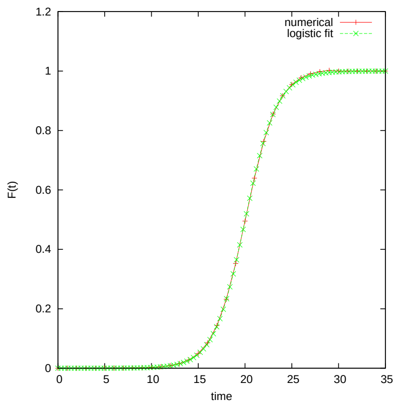
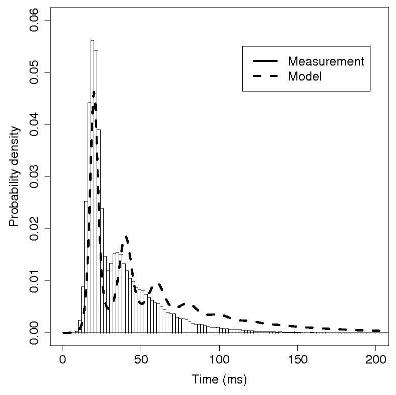
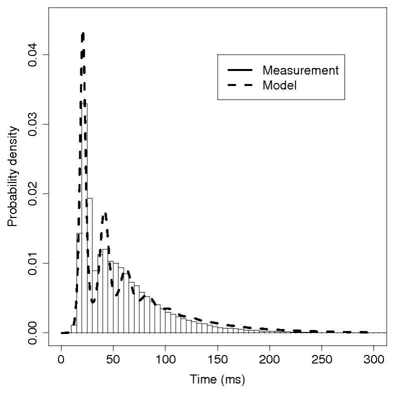


Figure 5.18: Logistic fit to maximum disk service time cdf over four disks.

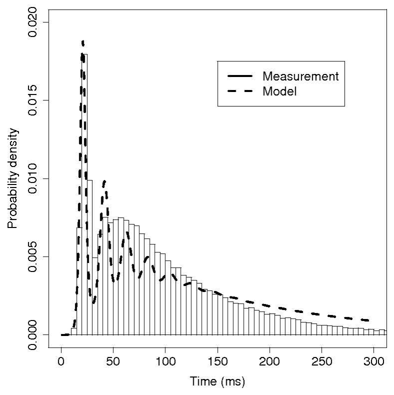
Figure 5.19 compares model predictions and measurement results for full stripe bulk arrivals to a four-disk RAID 01 system. As with the single disk, the number of requests in a batch is generated using a geometric distribution but results are presented for both read and write requests for a single arrival rate of $\lambda = 0.01$ requests per ms. Again, we observe good agreement between model and measurement. It is interesting to note that the model predicts more pronounced peaks than the measurements. We speculate that this is due to the nature of our maximum order statistic approximation for service time in a fork-join queue, which magnifies the sinusoidal behaviour of the single disk model, as well as our assumption of total dependence of queueing times at each disk.



(a) mean batch size = 2, reads



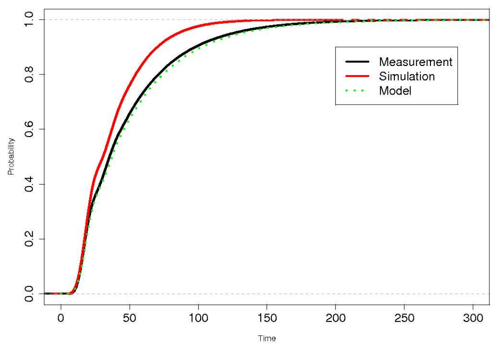
(b) mean batch size = 2, writes



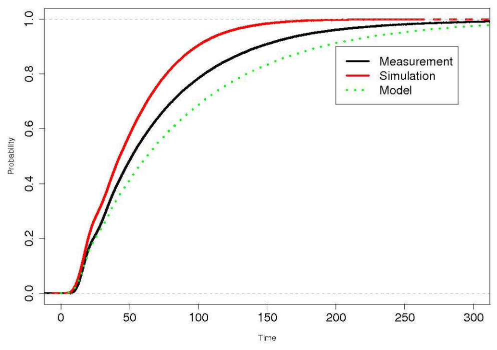
(c) mean batch size = 3, writes

Figure 5.19: I/O request response time pdf of model against measurement for 4-disk RAID 01 with full stripe bulk arrivals and rate 0.01 requests/ms.

In Figure 5.20 we compare the analytical and simulation models against device measurements for both read and write requests with geometrically distributed batch sizes. Both the simulation and analytical models are close to the device measurements, although the analytical model is slightly closer to the measurements. This is probably caused by RAID system overhead which is not taken into account by either model, but can be slightly matched by the lag in the analytical model created from the split-merge queue approximation.



(a) mean batch size = 2, 2-block read requests



(b) mean batch size = 3, 1-block write requests

Figure 5.20: I/O request response time pdf of simulation and analytical models against measurement for requests on a single disk with bulk arrivals and rate 0.01 requests/ms.

The simulation extends easily for the case of batch arrivals on either RAID 01 or RAID 5. The simulation is much more flexible than the analytical model for RAID systems with bulk arrivals. It is capable of predicting response times for any sized request quickly, whilst the RAID 01 analytical model is highly computationally intensive. We compare one case for the simulation, analytical model and device measurements in Figure 5.21 for the case of 4-block read requests with a mean batch size of 2 blocks. Due to the straightforward modelling of bulk arrivals in the simulation as opposed to the complicated and approximate nature of the analytical model, it is not surprising that the simulation produces a much more accurate model than the analytical model.

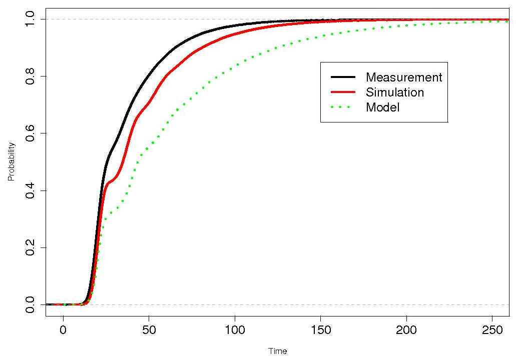


Figure 5.21: I/O request response time distributions of 4-disk RAID 01 with 4-block read requests and geometrically distributed bulk arrivals with mean size 2 and arrival rate 0.01 requests/ms.

5.5 Rotational Position Ordering

Rotational Position Ordering (RPO) is a scheduling strategy that chooses the request with the shortest positioning time to serve next in a disk queue. The analytical model of disk drives with rotational position ordering is developed from the theory of $M/G/1$ queues with queue length or state dependent service times. We first present a new analytical approximation for deriving response time distributions of $M/G/1$ queues with state dependent service times. We then apply it to our existing disk model.

5.5.1 State-Dependent Service Times for an $M/G/1$ queue

In an $M/G/1$ queue with state-dependent service times, we assume that from time $t = 0$, customers $C_0, C_1, \ldots, C_n, \ldots$ arrive at the queue. Let L_n denote the queue length immediately after customer C_n has completed service, and let Z_n denote the number of customers that arrive in the queue during the service of customer C_{n+1}. Then,

$$L_{n+1} = \begin{cases} L_n - 1 + Z_n & L_n > 0 \\ Z_n & L_n = 0 \end{cases}$$

Given state-dependent service times, the number of arrivals during a service period, Z_n, is dependent on the service time, which in turn depends on the queue length at the start of customer C_{n+1}'s service, L_n. Given i requests in the queue at the start of service, we denote the service time of the $M/G/1$ queue by the random variable X_i. Since arrivals are Markovian with arrival rate λ, the probability of j arrivals in a pre-defined service period x is:

$$\mathbb{P}(Z_n = j \mid X_{L_n} = x) = \frac{(\lambda x)^j}{j!} e^{-\lambda x} \quad (j \geq 0)$$

Therefore, by the law of total probability, the probability of j arrivals during a service period given all possible service times for a queue length of i at the start of a service is:

$$p_{j,i} = \mathbb{P}(Z_n = j \mid L_n = i) = \int_0^\infty \frac{(\lambda x)^j}{j!} e^{-\lambda x} dF_{X_i}(x) \tag{5.8}$$

The probability generating function for Z_n given a queue length of i at the start of a service is:

$$G_i(z) = \int_0^\infty e^{\lambda x z} e^{-\lambda x} dF_{X_i}(x) = X_i^*[\lambda(1-z)] \tag{5.9}$$

where X_i^* is the Laplace-Stieltjes Transform (LST) of X_i.

The embedded Markov chain has transition matrix $Q = (q_{ij} \mid i, j \geq 0)$ where:

$$\begin{aligned} q_{0j} &= \mathbb{P}(L_{n+1} = j \mid L_n = 0) &&= p_{j,1} \\ q_{ij} &= \mathbb{P}(L_{n+1} = j \mid L_n = i) &&= \begin{cases} p_{j-i+1,i} & j \geq i-1 \geq 0 \\ 0 & 0 \leq j \leq i-2 \end{cases} \end{aligned}$$

$$Q = \begin{bmatrix} p_{0,1} & p_{1,1} & p_{2,1} & p_{3,1} & \cdots \\ p_{0,1} & p_{1,1} & p_{2,1} & p_{3,1} & \cdots \\ 0 & p_{0,2} & p_{1,2} & p_{2,2} & \cdots \\ 0 & 0 & p_{0,3} & p_{1,3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The steady-state equations for the Markov chain, $\pi = \pi Q$ consequently are:

$$\pi_j = \pi_0 p_{j,1} + \sum_{i=1}^{j+1} \pi_i p_{j-i+1,i} \tag{5.10}$$

where π_i is the steady-state probability of there being i requests in the queue (including the customer currently in service).

Then the queue length generating function $\Pi(z) = \sum_{i=0}^{\infty} \pi_i z^i$, if it exists, is [48]:

$$\begin{aligned}
\Pi(z) &= \pi_0 \sum_{j=0}^{\infty} p_{j,1} z^j + \sum_{j=0}^{\infty} \sum_{i=1}^{j+1} \pi_i p_{j-i+1,i} z^j \\
&= \pi_0 \sum_{j=0}^{\infty} p_{j,1} z^j + \sum_{j=0}^{\infty} \pi_1 p_{j,1} z^j + \sum_{j=1}^{\infty} \pi_2 z p_{j-1,2} z^{j-1} + \\
&\quad \sum_{j=2}^{\infty} \pi_3 z^2 p_{j-2,3} z^{j-2} + \ldots \\
&= \pi_0 G_1(z) + \frac{1}{z} \sum_{i=1}^{\infty} \pi_i z^i G_i(z) \qquad (5.11)
\end{aligned}$$

This is dependent on the chain being stationary, the condition for which is that $\Pi(1) = 1$ [51]. Since the $G_i(z)$ are all probability generating functions, $\forall i\ G_i(1) = 1$ and

$$\Pi(1) = \pi_0 + \sum_{i=1}^{\infty} \pi_i$$

By definition of the steady-state probabilities, $\sum_{i=0}^{\infty} \pi_i = 1$, hence $\Pi(1) = 1$.

Using an approach similar to the derivation of $G_i(z)$ in Equation (5.9), it can be observed that $\Pi(z)$ is related to the response time, W as follows [51]:

$$\begin{aligned}
\Pi(z) &= \int_0^{\infty} e^{\lambda x z - \lambda x} dF_W(x) \\
&= W^*[\lambda(1-z)] \qquad (5.12)
\end{aligned}$$

Hence, by substituting Equation (5.11) into Equation (5.12),

$$W^*(\theta) = \pi_0 X_1^*(\theta) + \frac{\lambda}{\lambda - \theta} \sum_{i=1}^{\infty} \pi_i \left(\frac{\lambda - \theta}{\lambda} \right)^i X_i^*(\theta) \qquad (5.13)$$

In practice we would need to know the service time distribution for all possible queue lengths to be able to apply this equation. An elegant simplification that eradicates this problem assumes that if the queue length is greater than or equal to a specified length n then all corresponding service times are represented by the random variable X_n. This is an increasingly accurate approximation when there is a relatively low probability of high queue lengths or if the service time distributions are similar for higher queue lengths. Then,

$$\begin{aligned}
\Pi(z) &= \pi_0 G_1(z) + \frac{1}{z}\sum_{i=1}^{n-1} \pi_i z^i G_i(z) + \frac{1}{z}\sum_{i=n}^{\infty} \pi_i z^i G_n(z) \\
&= \pi_0 G_1(z) + \frac{1}{z}\sum_{i=1}^{n-1} \pi_i z^i G_i(z) + \\
&\quad \frac{1}{z} G_n(z)(\Pi(z) - \sum_{i=0}^{n-1} \pi_i z^i) \\
&= \frac{z\pi_0 G_1(z) + \sum_{i=1}^{n-1} \pi_i z^i G_i(z) - G_n(z)\sum_{i=0}^{n-1} \pi_i z^i}{z - G_n(z)}
\end{aligned}$$

We need to ensure that $\Pi(1) = 1$ to fulfil the stationary condition. Using L'Hôpital's rule to find the limit as $z \longrightarrow 1$, it becomes apparent that in order for $\Pi(1) \longrightarrow 1$ as $z \longrightarrow 1$, the following equation must hold:

$$\pi_0 = \frac{1 - \lambda E[X_n] - \sum_{i=1}^{n-1} \pi_i(\lambda E[X_i] - \lambda E[X_n])}{1 + \lambda E[X_1] - \lambda E[X_n]} \tag{5.14}$$

Solving the set of linear equations arising from Equations (5.10) and (5.14), the queue length probabilities $\pi_0, \pi_1, \ldots, \pi_n$ can be calculated.

The approximated response time LST can be calculated using Equation (5.12):

$$\begin{aligned} W^*(\theta) &= \frac{1}{\lambda(1-X_n^*(\theta))-\theta}\pi_0((\lambda-\theta)X_1^*(\theta)-\lambda X_n^*(\theta))+ \\ &\quad (\lambda-\theta)\sum_{i=1}^{n-1}(\pi_i\left(\frac{\lambda-\theta}{\lambda}\right)^{i-1}(X_i^*(\theta)-X_n^*(\theta))) \end{aligned}$$

By differentiating this equation m times and evaluating at $\theta = 0$, a recurrence relation for moments of response time can be derived:

$$\begin{aligned} E[W^m] &= \frac{1}{(m+1)(1-\lambda E[X_n])} \\ &\quad \left(\pi_0\left(\lambda E[B_1^{m+1}]+(m+1)E[X_1^m]-\lambda E[X_n^{m+1}]\right)+\right. \\ &\quad \sum_{i=1}^{n-1}\pi_i\lambda\sum_{j=0}^{\min[i,m+1]}\binom{m+1}{j}\binom{i}{j}\frac{j!}{\lambda^j}\left(E[X_i^{m+1-j}]-E[X_n^{m+1-j}]\right) \\ &\quad \left.+\lambda\sum_{j=2}^{m+1}\binom{m+1}{j}E[X_n^j]E[W^{m+1-j}]\right) \end{aligned} \tag{5.15}$$

5.5.2 Application to Zoned Disk Model

In the case of RPO, we define service time as the minimum disk head positioning time of all queueing I/O requests plus any additional rotations needed if the head fails to settle in time to read from target sectors. The probability that the disk head misses the correct rotational position at the end of a seek (termed a *latency miss*) is denoted as p_{miss} [20]. If there are i requests in the queue immediately prior to the start of a service, the service time of a request is thus:

$$X_i = \min_{h=1,\ldots,i}(S_h + R_h) + p_{miss}R_{max} + T_k$$

where R_{max} is the time to complete a complete disk revolution and S, R and T_k are seek time, rotational latency and k-block data transfer time respectively. In order to calculate the probability distribution of X_i we employ order statistics [31]. We find the first order statistic (i.e. minimum) of i convolutions of seek time and rotational latency ($S + R$). If a set of independent and identically distributed random variables, $X_1, X_2, \ldots, X_i$ are ordered in terms of size, the cdf of the smallest, $X_{(1)}$, will be:

$$\begin{aligned} F_{X_{(1)}}(x) &= \mathbb{P}(X_{(1)} \leq x) = 1 - \mathbb{P}(X_{(1)} > x) \\ &= 1 - \forall j \mathbb{P}(X_{(j)} > x) \quad j = 1, 2, \ldots, i \\ &= 1 - \forall j (1 - \mathbb{P}(X_{(j)} \leq x)) \\ &= 1 - (1 - F_X(x))^i \end{aligned}$$

In our case X is $S + R$ which has a convolved cdf of:

$$\begin{aligned} F_{R+S}(x) &= \frac{1}{R_{max}} \int_0^{R_{max}} F_S(x - z) dz \\ &= \frac{1}{R_{max}} \int_{x-R_{max}}^{x} F_S(u) du \end{aligned}$$

The pdf of a random variable M that models the occurrence of a latency miss, based on a single Bernoulli trial, is:

$$f_M(x) = \begin{cases} 1 - p_{miss} & x = 0 \\ p_{miss} & x = R_{max} \\ 0 & \text{otherwise} \end{cases}$$

It should be noted that the latency miss is only noticeable when RPO is switched on due to the more aggressive seeks that RPO entails. Since for the case $n = 1$ there is no queue re-ordering, and hence no RPO, there will be no latency misses. If the convolved minimum positioning time and transfer time have density function $f_{Y_i}(x)$ then convolving $f_{Y_i}(x)$ with $f_M(x)$ yields

$$f_{X_i}(x) = \begin{cases} f_{Y_i}(x) & i = 1 \\ (1 - p_{miss})f_{Y_i}(x) + p_{miss}f_{Y_i}(x - R_{max}) & i > 1 \end{cases} \tag{5.16}$$

Here x is bounded between the minimum transfer time, and the sum of maximum seek time, maximum latency (which is the time to complete two full disk revolutions for positioning and latency miss time) and maximum transfer time, irrespective of how much request reordering occurs.

Using Equation (5.15) the mean, variance and further moments of response time can be calculated. In order to do this it must be noted that the mth moment of service time is

$$E[X_i^m] = \begin{cases} E[Y_i^m] & i = 1 \\ (1 - p_{miss})E[Y_i^m] + & \\ p_{miss} \sum_{j=0}^{m} \binom{m}{j} E[Y_i^j] R_{max}^{j-i} & i > 1 \end{cases}$$

where

$$E[Y_i^m] = \sum_{j=0}^{m} \binom{m}{j} E[((R+S)_i)^j] E[T_k^{m-j}]$$

The service time pdf, $f_{X_i}(x)$, cannot be obtained analytically, and is expensive to evaluate numerically. Hence, it is very difficult to calculate the response time pdf, $f_W(x)$, exactly, either analytically or numerically. However, $f_W(x)$ can be readily approximated from its first four moments (calculated from Equation (5.15) using the Generalised Lambda Distribution (GLD) [71] (see Section 2.4)).

5.5.3 RAID 01 Extension

There are quite a number of difficulties involved in extending this model of a disk drive with RPO to RAID 01. The main difficulty is that, similarly to the case of bulk arrivals, we cannot assume that the queues in the fork-join queue are independent making it feasible to use the maximum order statistic approximation for the response time distribution. However, unlike the case of bulk arrivals, we also cannot assume total dependence between the queues. This is because of the larger service time distributions for smaller queue lengths. If we assume the same queue length and find the maximum service time using the maximum order statistic, this service time will be dominated by the larger service times of the shorter queues suggesting there is never any queueing. This is not possible with such a large service time and highlights the fact that this system is not fully dependent. Whether there is queueing at each queue in the array and how long these queues are becomes very important in this case.

To address this we consider different queueing situations for the component queues in the fork-join queue. First we consider the case when there is no queueing at any queue in the fork-join queue. We approximate this will happen with a probability of $(\pi_0)^n$ for an n-queue system. In this case each queue with have an individual response time of only the service time X_1. These service times are independent of each other and therefore the response time of the system can be approximated by finding the maximum order statistic for these. Similarly if there is queueing at all the queues in the fork-join queue (with probability $(1 - \pi_0)^n$) we assume totally dependent queueing and find the maximum of the n service times. Since there is guaranteed to be queueing at each queue, we only consider service times for queues with 1 or more customers queueing. With this maximum service time, we find the response time of a single queue with this service time. We assume that these two problems are independent (whether there is queueing or not) and therefore for systems of queues with both queueing present in some queues and not in others, we find the maximum of the response times of the queues with queueing and those without. The response time of an n queue fork-join queue can then be approximated as (in terms of the random variables)

$$W_n = \sum_{i=0}^{n} \binom{n}{i} (1 - \pi_0)^{n-i} \pi_0^i \max(Q_{n-i}, \max_{j=1,\dots i} X_{1_j}) \tag{5.17}$$

where Q_{n-i} is the case of $n-i$ queues with queueing where the maximum has been found of their service times and $\max_{j=1,\dots i} X_1$ is the maximum of i service times with no queueing. In addition, to simplify the model, for the case of Q_{n-i}, we use a standard FCFS $M/G/1$ queue with a service time distribution that averages all the possible queue lengths given queueing (i.e. $\frac{\sum_{i=1}^{n} \pi_i X_i}{1-\pi_0}$).

The second difficulty with the RAID 01 RPO model is the implementation. As previously stated we use the Generalised Lambda Distribution to approximate service time and response time distributions and densities for this model. The GLD provides inverse distributions and densities. Equation (5.17) needs to find the maximum order statistic of cdfs derived using the GLD. This involves complicated manipulation of the inverse distribution and densities. The additional complications of using the GLD combined with this new method make it both labour intensive and difficult to apply. It is an area for future work.

Simulation

We incorporate RPO into our simulation by parameterising the service time distribution sampler with the current queue length of each queue. The sampler then takes as many combined samples of seek and rotation time as there are jobs in the queue at that moment and chooses the minimum of these to be the positioning time of the request starting service. This can be used for either single disk simulation or RAID simulation.

Validation

Service Time

In order to validate our service time model of Equation (5.16), we measured service times for various fixed queue lengths. Figure 5.22 plots measured and modelled mean service times against constant queue lengths. We observe moderate agreement between model and measurement with similar trends. We note that

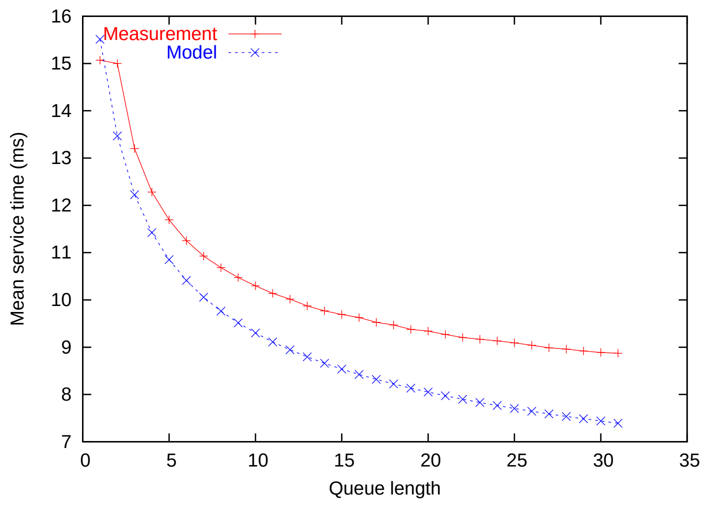


Figure 5.22: Comparison of measured and modelled mean 1-block read request service times for various fixed queue lengths.

these results are based on using a value of $p_{miss} = 0.05$ according to manufacturer advice. However, substantially better agreement is observed for a value of $p_{miss} = 0.17$. It is possible that this higher value may be correct in the context of our experiments, since our experiments consist entirely of random I/O workloads with no spatial or temporal locality and consequently will produce more aggressive seeks. One avenue of future work is to devise experiments to determine the exact value of p_{miss} for our specific disk drive and workload.

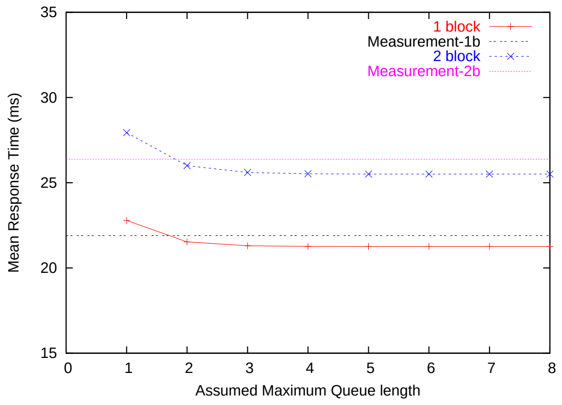
I/O Request Response Time

Figures 5.23 and 5.24 demonstrate the change in mean response time when different values are chosen for the queue length at which it is assumed that the service time distribution no longer changes for increasing queue lengths. A straight line is plotted to indicate the measured response time. For higher assumed maximum queue lengths, we observe excellent agreement between model and measurement for mean response times independent of arrival rate and request size. It can be observed, particularly for smaller sized requests and smaller arrival rates (e.g. Figures 5.23(a), 5.23(b), 5.24(a)), that the assumed maximum queue length does not have to be very high before convergence of the mean response times is observed. The impact of RPO on disk performance is magnified for larger request sizes and arrival rates. In many of these cases it can be observed that if RPO is not modelled (i.e. when the assumed maximum queue length is 1), the modelled mean response time is very high or the model is saturated (e.g. Figures 5.23(e), 5.24(c), 5.24(d)), whereas this does not occur in RPO-enabled measurements.

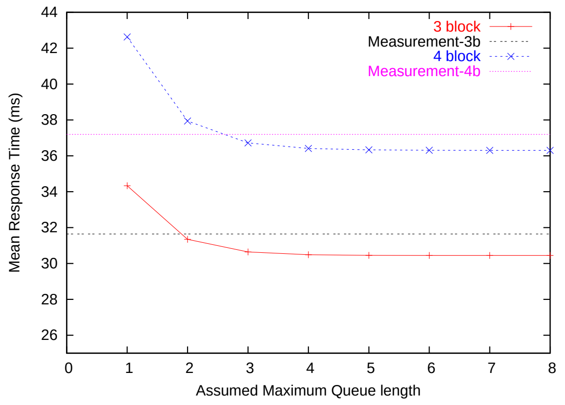
Although the mean response times show excellent agreement between model and measurement, our modelled variances compare less favourably with measurements. Table 5.5 presents variances for the same cases as Figures 5.23 and 5.24 using an assumed maximum queue length chosen at the length that the respective mean response time converges. For increasing arrival rates, the model presents significantly smaller variances than the measurements. Inevitably, this will affect skew and kurtosis (input parameters for the GLD with the mean and variance) to an even greater degree.

To test the accuracy of the GLD approximation that we use to approximate our response time densities, we first compare the approximation with a known pdf. In Figure 5.25, we compare our single disk model (from Chapter 3) with the GLD approximation of it, for single block transfers and arrival rate 0.01 requests/ms. We observe good agreement between approximate and exact models.

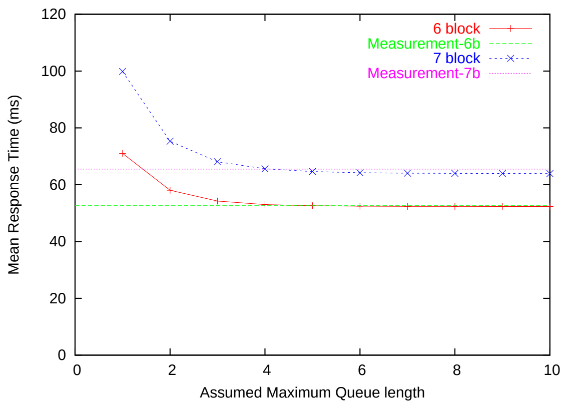
In Figures 5.26 and 5.27 we present GLD approximations of the I/O request re-



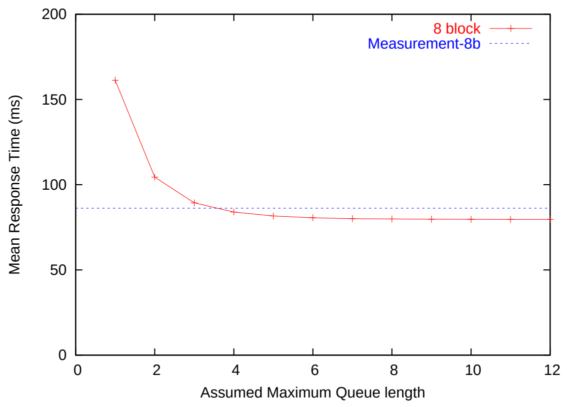
(a)



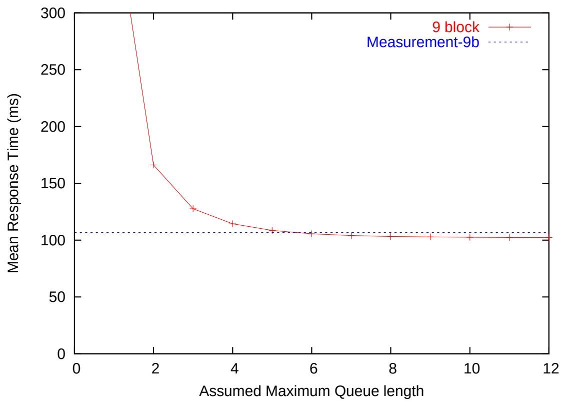
(b)



(c)

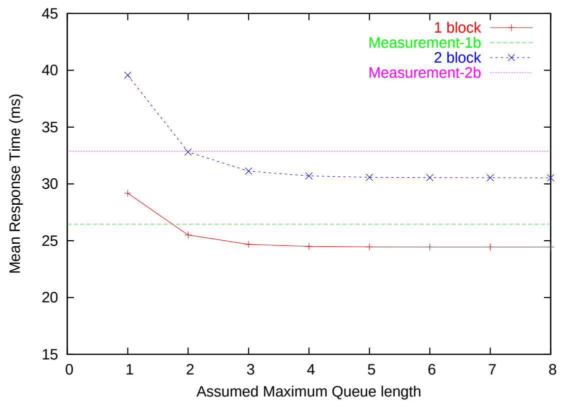


(d)

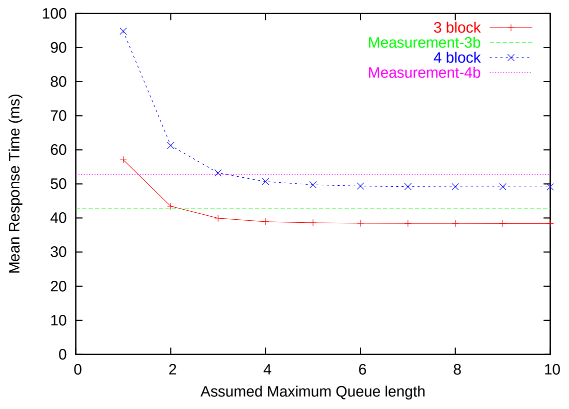


(e)

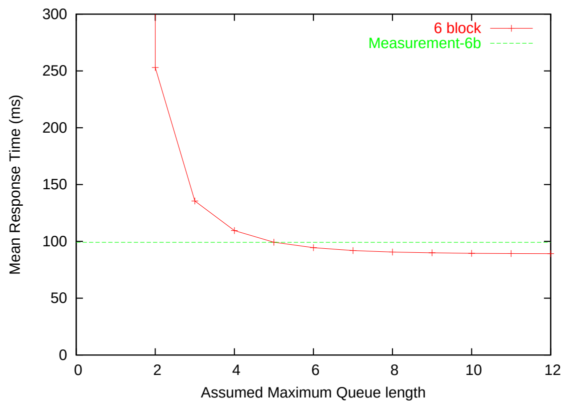
Figure 5.23: Mean response time against assumed maximum queue length and measurements for different sized read requests on a single disk with arrival rate 0.03 requests/ms.



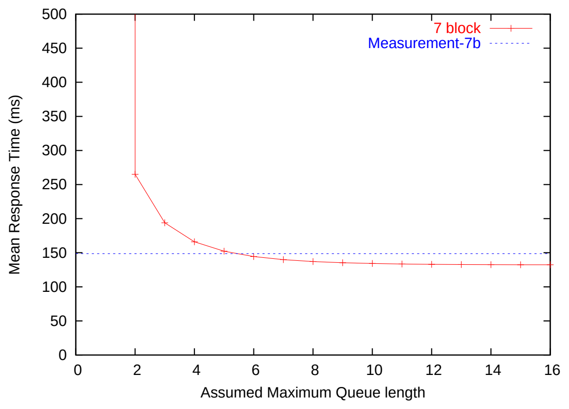
(a)



(b)



(c)



(d)

Figure 5.24: Mean response time against assumed maximum queue length and measurements for different sized read requests on a single disk and arrival rate 0.04 requests/ms.

	$\lambda = 0.03$		$\lambda = 0.04$	
# Blocks	Measured	Modelled	Measured	Modelled
1	129.3658	71.4639	234.3871	105.61
2	208.1058	110.839	383.8498	184.18
3	320.1849	175.285	822.2696	330.98
4	628.6987	280.2	2081.566	614.12
6	1568.488	737.56	10598.82	2494.2
7	3055.687	1229.4	25867.46	5745.9
8	6824.624	2106.4	sat	sat
9	11976.34	3809.4	sat	sat

Table 5.5: Measured and modelled variances for read request response times on a single disk with different sized requests and arrival rate λ requests/ms.

sponse time density of various request sizes and arrival rates of 0.03 and 0.04 requests/ms. Again we use a maximum queue length chosen at the length that the respective mean response time converges. We generally observe good agreement between model and measurement. However, the increase in difference between measured and modelled variances for larger request sizes causes increasing disagreement between model and measurement, despite still having excellent agreement for mean response time.

The simulation model works in a very similar way to the analytical model, finding the minimum joint seek and rotation time of a number of samples. In Figure 5.28, we compare some of the cases shown previously for the analytical model with the simulation model and device measurements. In Figure 5.28(a) we consider the case of 6-block reads to a single disk with an arrival rate of 0.03 requests/ms. In Figure 5.26(b) we observed excellent agreement for this case for pdfs of the analytical model and device measurements. Here we observe that the analytical and simulation models are very close and accurately model the device measurements. We then consider a case where the analytical model was less impressive. In Figure 5.27(d) we observed poor agreement for pdfs of the analytical model

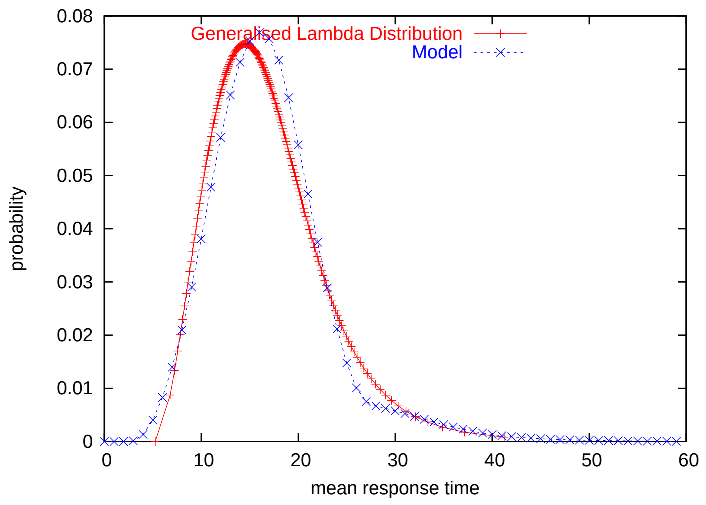
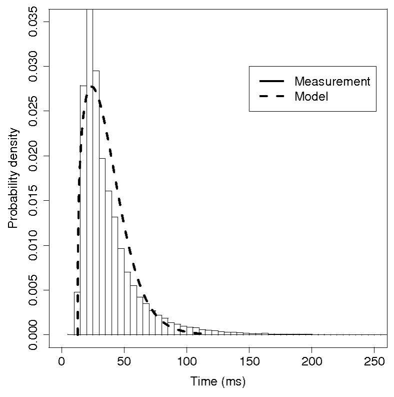


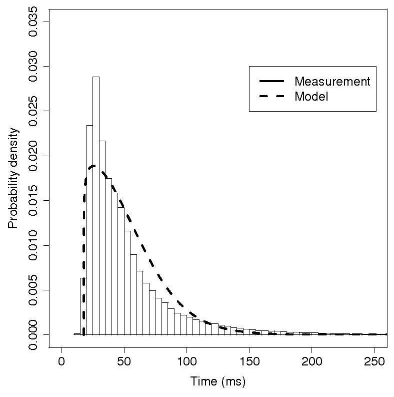
Figure 5.25: Comparison of actual model and Generalised Lambda Distribution approximation of the response time model for a 1-block read request to a single disk, with an arrival rate of 0.01 requests/ms.

and device measurements for the case of a 7-block read with an arrival rate of 0.04 requests/ms. Figure 5.28(b) compares the cdfs for this case. The simulation and analytical model are still very close but do not follow the trends of the device measurements.

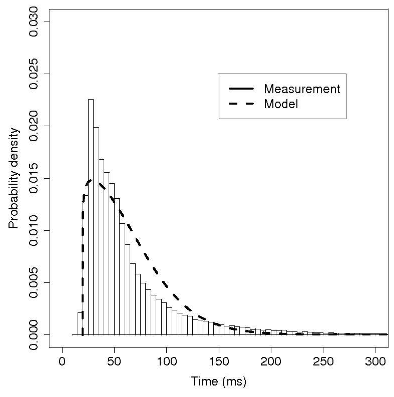
It is very straightforward to extend the simulation for RAID 01 with RPO enabled. Figure 5.29 involves a high arrival rate at the array (0.06 requests/ms), such that RPO should be expected. We plot two simulation cdfs, one with RPO enabled on the simulator and the second with RPO disabled. It is clear from the graph that for large arrival rates (and hence long queue lengths) incorporating RPO into any RAID or disk model is crucial. In addition we observe excellent agreement between the simulation and measurements.



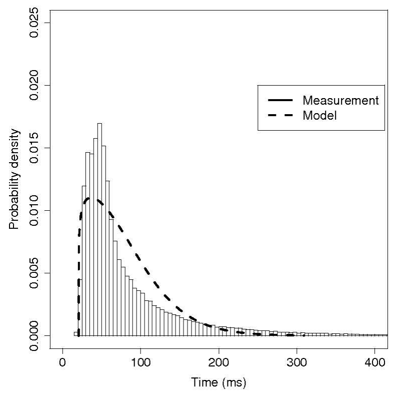
(a) 4 block



(b) 6 block

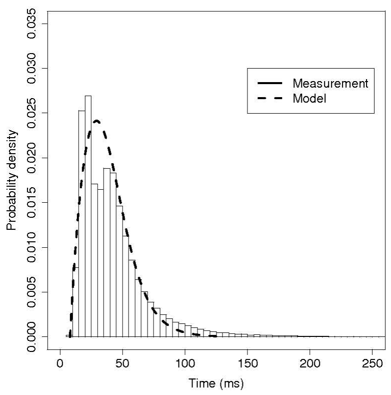


(c) 7 block

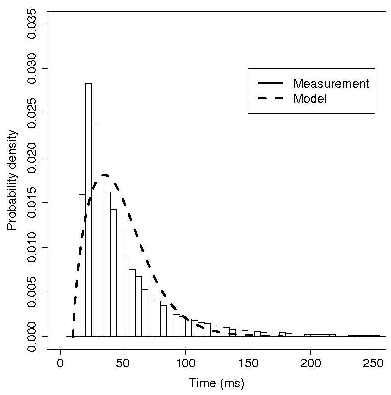


(d) 8 block

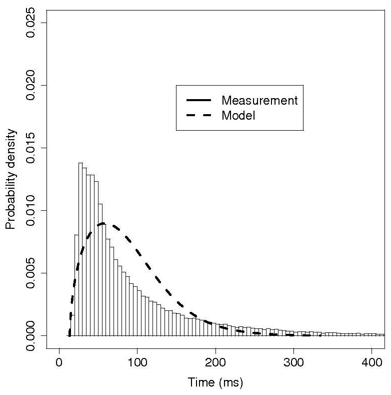
Figure 5.26: Comparison of measurements and approximations of the modelled pdfs for response times of different sized read requests to a single disk with arrival rate 0.03 requests/ms.



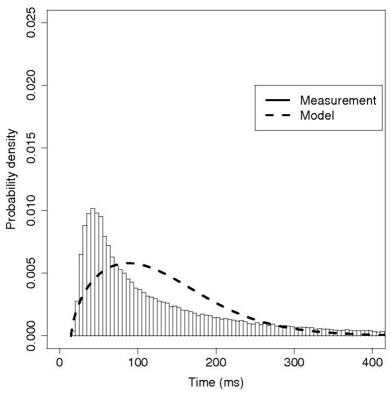
(a) 3 block



(b) 4 block

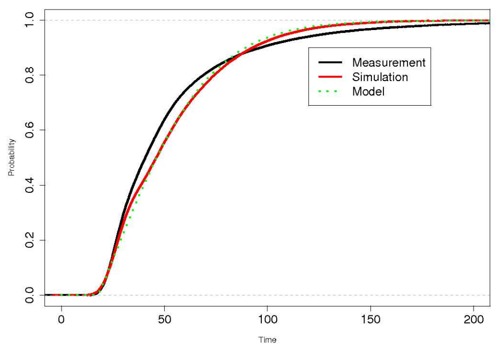


(c) 6 block

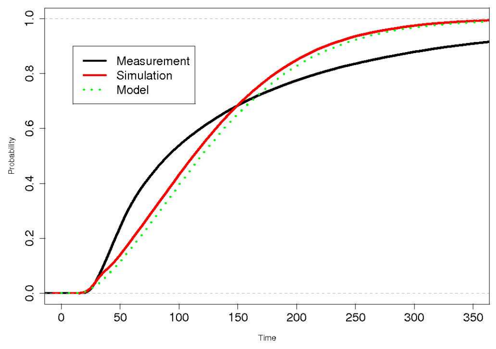


(d) 7 block

Figure 5.27: Comparison of measurements and approximations of the modelled pdfs for response times of different sized read requests to a single disk with arrival rate 0.04 requests/ms.



(a) 6-block , $\lambda = 0.03$



(b) 7-block , $\lambda = 0.04$

Figure 5.28: Response time cdf of simulation and analytical models against measurement for read requests on a single disk with RPO enabled and arrival rate λ requests/ms.

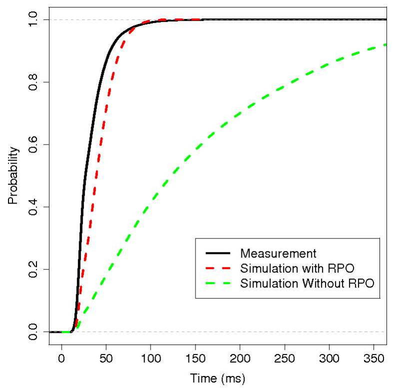


Figure 5.29: I/O request response time distributions of 4-disk RAID 01 with 4-block read requests and an arrival rate of 0.06 requests/ms.

Chapter 6

Conclusion

6.1 Summary of Thesis Achievements

The main objective of this thesis has been to create models of I/O request response time in zoned RAID systems. We have presented a comprehensive set of models of zoned RAID systems for various different RAID levels and expected workloads. No single work prior to this has presented a unified approach for deriving a disk and RAID model of different levels under different workloads. Furthermore, by illustrating how the model can be adapted for different RAID levels and workloads it is clear that the model can be extended if necessary for other possible situations not currently modelled in this thesis. Another unique attribute of this thesis is that all results are presented as full response time distributions or densities. Prior work has focused only on deriving mean response times.

The disk drive model presented in this thesis builds on existing work [139] to create a full response time distribution of a zoned disk drive. This was a key contribution as no analytical queueing network model existed of a zoned disk drive that provided the full response time distribution. All the work that follows in the thesis employs and adapts this initial disk drive model.

We presented a summary of methods for approximating the response time of fork-join queues and chose the maximum order statistic approximation. Again, this ensures that full response time distributions can be calculated as opposed to other work which only provides approximations for the mean response time. This can be used in tandem with the zoned disk drive model to calculate response time distributions of disk arrays.

With these foundations in place we introduced our models of RAID 0, 01 and 5. We introduced extensions to the fork-join queueing model to enable modelling the intricacies of RAID systems. RAID 5 write requests specifically require more complicated extensions which provided another novel contribution of this thesis. We studied several different approaches to modelling partial stripe write RAID 5 requests, by considering the pre-read and partial stripe together and averaging their service times and implementing multiclass and priority queueing networks and compared the benefits of each method.

We initially created a model for Markovian arrival streams consisting of requests of a constant size. We then relaxed this and considered requests whose size is determined from a specified probability distribution. We derived response time distributions for disk and RAID systems with Markovian bulk arrival streams. We assumed requests arrive according to a First Come First Served queueing discipline and then attempted to implement a more realistic model where scheduling is decided by the shortest positioning time. In developing models for these situations we have had to develop some new results and approximations in queueing theory, specifically in the area of queues with bulk arrivals and queues with state dependent service time distributions. In addition we were faced with the non-trivial task of applying these new results to the specific requirements of our existing disk drive and RAID models.

To complement this analytical model, we developed a queueing-based discrete event simulation that mimicked the situations that the analytical model supports. This can be used to validate and improve the analytical model and as a standalone simulation model. We consistently show that the simulation model compares favourably with device measurements and the analytical model compares favourably with both simulation and device measurements. There is very little work in the performance analysis of any application area that compares device measurements, simulation and analytical model and even less that shows good agreement between the three (as illustrated in the study in [110]).

6.2 Applications

The research in this thesis has direct applications for storage system design and analysis. There is an unrelenting business demand for fast, reliable storage. Much of this data is ultimately stored on RAID systems, which are deployed either as standalone storage solutions or as the building blocks of virtualised storage infrastructures. The detailed understanding of RAID system performance is therefore critical to determining whether or not application-level quality of service demands will be met by a given storage infrastructure.

The disk drive and RAID performance models presented in this thesis can be applied to any hard disk drive and hardware RAID system. The models can be easily parameterised from disk and RAID system specifications and will provide response time distributions from which it can be observed if a certain choice of disks, RAID level and combination of the two best meets the needs of the user, fulfilling service level agreements and quality of service performance expectations while lowering costs and providing a required standard of reliability.

A performance model of a virtualised storage system could be developed from an effective performance model of a disk array. The physical resources underlying a virtualised storage system consist of tiers of disk arrays. Each tier has a different cost per capacity ratio. A performance model of virtualised storage architectures can therefore be developed from the existing physical model, with intelligent data management that minimises physical storage size over the data lifecycle.

Expanding storage capacity requirements must be met both with new storage technology and data placement strategies for optimal space efficiency. The development of a definitive performance model of the physical storage system can establish and refine data placement strategies.

The simulation presented in this thesis is a useful extension to the existing JINQS software. It can be applied to any fork-join queue simulation, hard disk drive or RAID simulation.

The analytical queueing results developed here do not only have applications in disk storage systems. The work with bulk arrivals has many application areas, for example hospital arrivals during major incidents and passengers boarding trains. Similarly there are often situations in which queueing jobs consist of subtasks to be serviced and the work here on different sized requests will be applicable. One other application area for state dependent service times occurs with cell discarding in ATM networks [27]. The analytical results presented in this thesis can all be directly applied for these and other application areas.

6.3 Future Work

There are a number of possible extensions to the work presented in this thesis:

This thesis always assumes that there is a constant stream of *random* I/O requests. We do this since random requests are more difficult to model than sequential requests. However the model could be easily adapted in all instances for sequential requests. This would enable the model to represent commonly occurring disk access patterns with temporal and spatial locality.

The RAID 5 write analytical and simulation models presented are stripe-aligned for a constant sized request in the analytical model and for any sized request in the simulation. These need to be extended so a RAID 5 write request can start in any position in a stripe to better represent actual RAID procedures.

The model for RAID 01 with bulk arrivals currently assumes only full stripe read or write requests. It should be extended for any fixed request size. A more complex task would be to extend the bulk arrival disk model for a RAID 5 system. In addition the bulk arrival model could be adapted to accept the variety of workloads that our standard model accepts: mixtures of read and write requests, mixtures of random and sequential access and a variety or request sizes.

Similarly, the disk model that employs rotational positioning ordering should be extended to match the workload conditions of our other models. As discussed in the thesis, extending the disk model with RPO to RAID systems is a non-trivial problem, but should be persevered with. Again, either disk or RAID model should take workloads that consist of mixtures of read and write requests and different sizes. It is particularly important to adapt this model to support bulk arrivals, since bulk arrivals will always provide larger queue lengths requiring RPO.

The simulation presented in this thesis offers a number of areas for improvement. It can be made more general to represent other RAID levels and configurations and combinations of read and write requests and random and sequential I/O. Simulation results currently underestimate the device measurements because RAID controller overheads are not currently incorporated into the model. This has been because they are hard to distill from device measurements and we have had difficulty discovering the value of these overheads from RAID system manufacturers.

We have assumed throughout that each disk or RAID system has a Markovian arrival stream with or without bulk geometric arrivals. It would be interesting to compare this assumption with real I/O traces to analyse the justification for this assumption. If necessary the model could be developed to accept non-Markovian arrival streams. Our analytical model for Markovian bulk arrivals permits any discrete probability distribution for the batch size. In this work we have chosen a geometric distribution to clearly illustrate our model; however, it would be an interesting and constructive addition to define a discrete probability distribution based on I/O traces of bulk arrivals to a typical RAID system which would easily fit into our existing analytical result.

Caching is an interesting and practically useful aspect that merits further investigation and integration into our model for both the disk and RAID models.

Throughout this work we have validated our models against the same type of hard disk drive and RAID system. To verify our models, it would be useful to validate them against other models of disks and RAID systems produced by different manufacturers.

With a satisfactory performance model, a future project based on this work would be to extend the RAID model to be component parts in a performance model of a virtualised tiered storage system. Using the performance model, optimal data placement strategies can be devised for a disk drive, RAID system and tiered storage system.

Bibliography

[1] J. Abate, G. Choudhury, and W. Whitt. An introduction to numerical transform inversion and its application to probability models. In W. K. Grassman, editor, *Computational probability*, chapter 8, pages 257–323. Springer, 2000.

[2] J. Abate and W. Whitt. The Fourier-series method for inverting transforms of probability distributions. *Queueing Systems Theory and Applications*, 10(1-2):5–88, 1992.

[3] J. Abate and W. Whitt. Numerical inversion of Laplace transforms of probability distributions. *INFORMS Journal on Computing*, 7(1):36, 1995.

[4] D. Anderson. You don't know jack about disks. *Queue*, 1(4):20–30, 2003.

[5] D. Anderson and W. Whittington. Disk drive technology. Tutorial at 5th USENIX Conference on File and Storage Technologies (FAST), February 2007.

[6] M. Andrews, M. A. Bender, and L. Zhang. New algorithms for the disk scheduling problem. In *Proc. 37th Annual Symposium on Foundations of Computer Science*, pages 550–559, Oct 1996.

[7] H. P. Anvin. The mathematics of RAID 6, March 2007. `http://www.kernel.org/pub/linux/kernel/people/hpa/raid6.pdf`.

[8] S. W. M. Au-Yeung. *Response Times in Healthcare Systems*. PhD thesis, Imperial College London, January 2008.

[9] S. W. M. Au-Yeung, N. J. Dingle, and W. J. Knottenbelt. Efficient approximation of response time densities and quantiles in stochastic models. In *4th ACM Workshop on Software and Performance (WOSP)*, pages 151–155, January 2004.

[10] O. I. Aven, E. G. Coffman, and Y. A. Kogan. *Stochastic Analysis of Computer Storage*. D. Riedel Publishing Company, 1987.

[11] E. Bachmat. Average case analysis of disk scheduling, increasing subsequences and spacetime geometry. *Algorithmica*, 49(3):212–231, November 2007.

[12] S. Balsamo, L. Donatiello, and N. M. Van Dijk. Bound performance models of heterogeneous parallel processing systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):1041–1056, 1998.

[13] J. Banks, J. S. Carson II, B. L. Nelson, and D. M. Nicol. *Discrete-Event Simulation*. Prentice Hall, 3rd edition, 2001.

[14] P. Biswas, K. K. Ramakrishnan, and D. Towsley. Trace driven analysis of write caching policies for disks. *SIGMETRICS Performance Evaluation Review*, 21(1):13–23, 1993.

[15] D. Bitton and J. Gray. Disk shadowing. In *Proc. 14th International Conference on Very Large Data Bases (VLDB '88)*, pages 331–338, San Francisco, CA, USA, 1988. Morgan Kaufmann Publishers Inc.

[16] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modelling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, Inc., 1998.

[17] O. J. Boxma, G. M. Koole, and Z. Liu. Queueing-theoretic solution methods for models of parallel and distributed systems. In *Proc. Performance Evaluation of Parallel and Distributed Systems Solution Methods*, pages 1–24, 1994.

[18] P. H. Brill and M. J. M. Posner. Level crossings in point processes applied to queues: Single server case. *Operations Research*, 25(4):662–674, 1977.

[19] J. S. Bucy, G. R. Ganger, and Contributors. *The DiskSim Simulation Environment Version 3.0 Reference Manual*. School of Computer Science, Carnegie Mellon University, 3.0 edition, January 2003.

[20] W. A. Burkhard and J. D. Palmer. Rotational position optimization (RPO) disk scheduling. Technical Report CS2001-0679, University of California at San Diego, La Jolla, CA, 2001.

[21] M. L. Chaudhry and J. G. C. Templeton. *A First Course in Bulk Queues*. John Wiley & Sons, 1983.

[22] P. M. Chen and E. K. Lee. Striping in a RAID level 5 disk array. *SIGMETRICS Performance Evaluation Review*, 23(1):136–145, 1995.

[23] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[24] S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley. Performance evaluation of two new disk scheduling algorithms for real-time systems. *Real-Time Systems*, 3(3):307–336, 1991.

[25] S. Chen and D. Towsley. The design and evaluation of RAID 5 and parity striping disk array architectures. *IEEE Transactions on Parallel and Distributed Systems*, 17(1-2):58–74, 1993.

[26] S. Chen and D. Towsley. A performance evaluation of RAID architectures. *IEEE Transactions on Computers*, 45(10):1116–1130, 1996.

[27] D. I. Choi, C. Knessl, and C. Tier. A queueing system with queue length dependent service times, with applications to cell discarding in ATM networks. *Journal of Applied Mathematics and Stochastic Analysis*, 12(1):35–62, 1999.

[28] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley Publishing Company, 1967.

[29] D. R. Cox. *Renewal Theory*. Chapman and Hall, 1962.

[30] A. E. Dashti, S. H. Kim, and R. Zimmermann. Zoned-RAID for multimedia database servers. In *Proc. 10th International Conference on Database Systems for Advanced Applications (DASFAA)*, April 2005.

[31] H. A. David. *Order Statistics*. John Wiley and Sons, Inc, 1981.

[32] P. J. Denning. Effects of scheduling on file memory operations. In *Proc. AFIPS Spring Joint Computer Conference*, volume 31, pages 9–21, 1967.

[33] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys (CSUR)*, 10(3):225–261, September 1978.

[34] H. Diepers. Key-parameters of vertical magnetic recording. In *Proc. VLSI and Microelectronic Applications in Intelligent Peripherals and their Interconnection Networks*, May 1989.

[35] N. J. Dingle. *Parallel Computation of Response Time Densities and Quantiles in Large Markov and Semi-Markov Models*. PhD thesis, Imperial College London, 2004.

[36] A. Duda and T. Czachórski. Performance evaluation of fork and join synchronization primitives. *Acta Informatica*, 24(5):525–553, 1987.

[37] F. Durbin. Numerical inversion of Laplace transforms: an efficient improvement to Dubner and Abate's method. *Computer Journal*, 17(4):371–376, 1974.

[38] A. J. Field. *JINQS: An Extensible Library for Simulating Multiclass Queueing Networks*. Imperial College London, August 2006. `http://www.doc.ic.ac.uk/~ajf/Research/manual.pdf`.

[39] Fujitsu. *MAN3367FC Series Disk Drives- Product/Maintenance Manual*. Fujitsu, April 2002.

[40] S. Ghandeharizadeh, D. J. Ierardi, D. Kim, and R. Zimmermann. Placement of data in multi-zone disk drives. In *Proc. 2nd International Baltic Workshop on Databases and Information Systems*, June 1996.

[41] M. E. Gomez and V. Santonja. Characterizing temporal locality in I/O workload. In *Proc. International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, San Diego, CA, 2002.

[42] W. J. Gordon and G. F. Newell. Closed queuing systems with exponential servers. *Operations Research*, 15(2):254–265, 1967.

[43] C. C. Gotlieb and G. H. MacEwen. Performance of movable-head disk storage devices. *Journal of the ACM*, 20(4):604–623, 1973.

[44] W. J. Gray, P. Wang, and M. Scott. An M/G/1-type queuing model with service times depending on queue length. *Applied Mathematical Modelling*, 16(12):652 – 658, 1992.

[45] D. Gross and C. M. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons, 3rd edition, 1998.

[46] E. J. Gumbel. The maxima of the mean largest value and of the range. *The Annals of Mathematical Statistics*, 25(1):76–84, March 1954.

[47] T. R. Haining and D. D. E. Long. Management policies for non-volatile write caches. In *Performance, Computing and Communications Conference (IPCCC '99)*, pages 321–328. IEEE, February 1999.

[48] C. M. Harris. Queues with state-dependent stochastic service rates. *Operations Research*, 15(1):117–130, 1967.

[49] P. G. Harrison. Teaching M/G/1 theory with extension to priority queues. *IEE Proc. Computers and Digital Techniques*, 147(1):23–26, January 2000.

[50] P. G. Harrison and W. J. Knottenbelt. Quantiles of sojourn times. In E. Gelenbe, editor, *Computer System Performance Modelling in Perspective: A Tribute to the Work of Professor Kenneth C. Sevcik*, chapter 10, pages 156–194. Imperial College Press, 2006.

[51] P. G. Harrison and N. M. Patel. *Performance Modelling of Communication Networks and Computer Architectures*. Addison-Wesley, 1993.

[52] P. G. Harrison and S. Zertal. Queueing models with maxima of service times. In *Proc. 13th International Conference on Computer Performance Evaluations, Modelling Techniques and Tools*, pages 152–168, 2003.

[53] P. G. Harrison and S. Zertal. Calibration of a queueing model of RAID systems. In *Proc. Workshop on Practical Applications of Stochastic Modelling (PASM)*, 2004.

[54] P. G. Harrison and S. Zertal. Queueing models of RAID systems with maxima of waiting times. *Performance Evaluation*, 64(7-8):664–689, August 2007.

[55] H. O. Hartley and H. A. David. Universal bounds for mean range and extreme observation. *The Annals of Mathematical Statistics*, 25(1):85–99, March 1954.

[56] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 4th edition, 2007.

[57] W. W. Hsu and A. J. Smith. The performance impact of I/O optimizations and disk improvements. *IBM Journal of Research and Development*, 48(2):255–289, 2004.

[58] A. Huffman and J. Clark. Serial ATA Native Command Queueing. Whitepaper, Intel Corporation and Seagate Technology, July 2003.

[59] S. S. Isukapalli. *Uncertainty analysis of transport-transformation models*. PhD thesis, Rutgers, The State University of New Jersey, 1999.

[60] G. L. Choudhury J. Abate and W. Whitt. On the Laguerre method for numerically inverting Laplace transforms. *INFORMS Journal on Computing*, 8(4):413–427, 1996.

[61] J. R. Jackson. Networks of waiting lines. *Operations Research*, 5(4):518–521, 1957.

[62] D. M. Jacobson and J. Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7rev1, HP Laboratories, 1991.

[63] R. G. Miller Jr. Priority queues. *The Annals of Mathematical Statistics*, 31(1):86–103, 1960.

[64] R. H. Katz, D. W. Gordon, and J. A. Tuttle. Storage system metrics for evaluating disk array organizations. Technical Report UCB/CSD-91-611, EECS Department, University of California, Berkeley, Dec 1990.

[65] D. G. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain. *The Annals of Mathematical Statistics*, pages 338–354, 1953.

[66] C. Kim and A. K. Agrawala. Analysis of the fork-join queue. *IEEE Transactions on Computers*, 38(2):250–255, 1989.

[67] M. Y. Kim and A. N. Tantawi. Asynchronous disk interleaving: Approximating access delays. *IEEE Transactions on Computers*, 40(7):801–810, July 1991.

[68] S. H. Kim, H. Zhu, and R. Zimmermann. Zoned-RAID. *ACM Transactions on Storage (TOS)*, 3(1):1–17, March 2007.

[69] L. Kleinrock. *Queueing Systems - Volume I: Theory*. John Wiley and Sons, 1975.

[70] A. Kuratti and W. H. Sanders. Performance analysis of the RAID 5 disk array. In *Proc. IEEE International Computer Performance and Dependability Symposium (IPDS)*, pages 236–245, Erlangen, Germany, 1995.

[71] A. Lakhany and H. Mausser. Estimating the parameters of the Generalized Lambda Distribution. *Algo Research Quarterly*, 3(3):47–58, December 2000.

[72] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt. Modelling and validation of response times in zoned RAID. In *Proc. 16th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, September 2008.

[73] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt. A response time distribution model for zoned RAID. In *Proc. 15th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA)*, pages 144–157, June 2008.

[74] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt. Validation of large zoned RAID systems. In *Proc. 24th UK Performance Engineering Workshop (UKPEW)*, pages 246–261, July 2008.

[75] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt. Modelling zoned RAID systems using fork-join queueing simulation. In *Proc. 6th European Performance Engineering Workshop (EPEW)*, pages 16–29, July 2009.

[76] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt. A performance model of zoned disk drives with I/O request reordering. In *Proc. 6th International Conference on Quantitative Evaluation of Systems (QEST)*, pages 97–106, September 2009.

[77] A. S. Lebrecht, N. J. Dingle, W. J. Knottenbelt, P. G. Harrison, and S. Zertal. Using bulk arrivals to model I/O request response time distributions in zoned disks and RAID systems. In *Proc. 4th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, October 2009.

[78] A. S. Lebrecht and W. J. Knottenbelt. Response time approximations in fork-join queues. In *Proc. 23rd UK Performance Engineering Workshop (UKPEW)*, July 2007.

[79] E. K. Lee. *Performance Modeling and Analysis of Disk Arrays*. PhD thesis, University of California at Berkeley, 1993.

[80] E. K. Lee and R. H. Katz. An analytic performance model of disk arrays. *SIGMETRICS Performance Evaluation Review*, 21(1):98–109, 1993.

[81] J. D. C. Little. A proof for the queueing formula: $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

[82] Y. C. Liu and H. G. Perros. A decomposition procedure for the analysis of a closed fork/join queueing system. *IEEE Transactions on Computers*, 40(3):365–370, March 1991.

[83] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11:431–441, 1963.

[84] R. Van Meter. Observing the effects of multi-zone disks. In *Proc. USENIX Annual Technical Conference*, pages 19–30, 1997.

[85] D. W. Miller and D. T. Harper. Performance analysis of disk cache write policies. *Microprocessors and Microsystems*, 19(3):121–130, April 1995.

[86] I. Mitrani. *Probabilistic Modelling*. Cambridge University Press, 1998.

[87] J. A. Morrison. Sojourn and waiting times in a single-server system with state-dependent mean service rate. *Queueing Systems*, 4(3):213–235, 1989.

[88] R. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739–743, June 1988.

[89] R. Nelson, D. Towsley, and A. N. Tantawi. Performance analysis of parallel processing systems. *IEEE Transactions on Software Engineering*, 14(4):532–540, 1988.

[90] S. Park and H. Shin. Rigorous modeling of disk performance for real-time applications. In *Proc. International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 486–498, 2003.

[91] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. International Conference on Management of Data (SIGMOD)*, 1988.

[92] D. A. Patterson and J. L. Hennessy. *Computer organization and design: the hardware/software interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 1993.

[93] H. G. Perros. *Queueing networks with Blocking*. Oxford University Press, 1994.

[94] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Technical Report CS-96-332, University of Tennessee, 1997.

[95] J. S. Plank. Erasure codes for storage applications. Tutorial at 4th USENIX Conference on File and Storage Technologies (FAST), 2005.

[96] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[97] M. Reiser and S. Lavenburg. Mean-value analysis of closed multichain queuing networks. *Journal of the ACM*, 27(2):313–322, April 1980.

[98] A. Riska and E. Riedel. Disk drive level workload characterization. In *Proc. USENIX '06 Annual Technical Conference (ATEC)*, Boston, MA, 2006.

[99] S. M. Ross. *Introduction to Probability Models*. Academic Press, 6th edition, 1997.

[100] C. Ruemmler and J. Wilkes. Unix disk access patterns. In *Proc. Usenix Winter Conference*, pages 405–420, San Diego, CA, 1993.

[101] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *Computer*, 27(3):17–28, 1994.

[102] J. Rydning and D. Reinsel. Worldwide hard disk drive 2009–2012 forecast: Navigating the transitions for enterprise applications. IDC Market Analysis, Document 216394, February 2009.

[103] R. A. Sahner and K. S. Trivedi. Performance and reliability analysis using directed acyclic graphs. *IEEE Transactions on Software Engineering*, 13(10):1105–1114, 1987.

[104] R. A. Scranton, D. A. Thompson, and D. W. Hunter. The access time myth. Technical Report RC10197, IBM, 1983.

[105] Seagate. Barracuda ES Data Sheet. `http://www.seagate.com/docs/pdf/datasheet/disc/ds_barracuda_es.pdf`.

[106] Seagate. Economies of Capacity and Speed: Choosing the most cost-effective disc drive size and RPM to meet IT requirements. Whitepaper, Seagate, May 2004. `http://www.seagate.com/content/pdf/whitepaper/economies_capacity_spd_tp.pdf`.

[107] M. Seltzer, P. Chen, and J. Ousterhout. Disk Scheduling Revisited. In *Proc. USENIX Winter Technical Conference*, pages 313–324. USENIX Association, 1990.

[108] E. Shriver, A. Merchant, and J. Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 182–191. ACM, 1998.

[109] A. J. Smith. Cache memories. *ACM Computing Surveys*, 14(3):473–530, 1982.

[110] A. Symington and P. Kritzinger. A hardware test bed for measuring IEEE 802.11g distribution coordination function performance. In *Proc. 17th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 61–67, September 2009.

[111] L. Takács. Priority queues. *Operations Research*, 12(1):63–74, 1964.

[112] A. Talbot. The accurate numerical inversion of Laplace transforms. *Journal of the Institute of Mathematical Applications*, 23:97–120, 1979.

[113] A. Thomasian and G. Fu. Anticipatory disk arm placement to reduce seek time. *International Journal of Computer Systems Science & Engineering*, 21(3):173–182, 2006.

[114] A. Thomasian and C. Han. Affinity-based routing in zoned mirrored disks. *The Computer Journal*, 48(3):292–299, 2005.

[115] A. Thomasian and A. N. Tantawi. Approximate solutions for M/G/1 fork/join synchronization. In *Proc. 26th Conference on Winter Simulation (WSC '94)*, pages 361–368, San Diego, CA, USA, 1994.

[116] D. Towsley, C. G. Rommel, and J. A. Stankovic. Analysis of fork-join program response times on multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):286–303, July 1990.

[117] R. Treiber and J. Menon. Simulation study of cached RAID 5 designs. In *Proc. 1st IEEE Symposium on High-Performance Computer Architecture*, pages 186–197, 1995.

[118] P. Triantafillou, S. Christodoulakis, and C. Georgiadis. Optimal data placement on disks: A comprehensive solution for different technologies. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):324–330, 2000.

[119] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Prentice Hall, 1982.

[120] M. Uysal, G. A. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. In *Proc. 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE Computer Society, 2001.

[121] D. Vadala. *Managing RAID on Linux*. O'Reilly, 2003.

[122] Nico M. van Dijk. Why queuing never vanishes. *European Journal of Operational Research*, 99(2):463–476, 1997.

[123] E. Varki. Mean value technique for closed fork-join networks. In *Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 103–112, 1999.

[124] E. Varki. Response time analysis of parallel computer and storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1146–1161, November 2001.

[125] E. Varki and L. W. Dowdy. Analysis of balanced fork-join queueing networks. In *Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 232–241, 1996.

[126] E. Varki, A. Merchant, and H. Chen. The M/M/1 fork-join queue with variable sub-tasks. Unpublished – `http://www.cs.unh.edu/˜varki/publication/open.pdf`.

[127] E. Varki, A. Merchant, and X. Qiu. An analytical model of disk arrays under synchronous I/O workloads. Technical report, Univ. of New Hampshire, Jan 2003.

[128] E. Varki, A. Merchant, J. Xu, and X. Qiu. Issues and challenges in the performance analysis of real disk arrays. *IEEE Transactions on Parallel and Distributed Systems*, 15(6):559–574, June 2004.

[129] E. Varki and S. X. Wang. A performance model of disk array storage systems. In *Proc. Computer Measurement Group International Conference (CMG)*, December 2000.

[130] S. Varma and A. M. Makowski. Interpolation approximations for symmetric fork-join queues. In *Proc. 16th IFIP Working Group 7.3 International Symposium on Computer Performance Modeling Measurement and Evaluation (Performance)*, pages 245–265, 1994.

[131] F. Wan, N. J. Dingle, W. J. Knottenbelt, and A. S. Lebrecht. Simulation and modelling of RAID 0 system performance. In *Proc. 22nd Annual European Simulation and Modelling Conference (ESM)*, pages 145–149, September 2008.

[132] M. Wang, A. Ailamaki, and C. Faloutsos. Capturing the spatio-temporal behavior of real traffic data. *Performance Evaluation*, 49(1-4):147–163, 2002.

[133] Wikipedia. Standard RAID levels, April 2009. `http://en.wikipedia.org/wiki/Standard_RAID_levels`.

[134] R. W. Wolff. Poisson arrivals see time averages. *Operations Research*, 30(2):223–231, 1982.

[135] S. Wolfram. *The Mathematica Book*. Wolfram Media, 5th edition, 2003.

[136] T. M. Wong and J. Wilkes. My cache or yours? Making storage more exclusive. In *Proc. USENIX Annual Technical Conference*, pages 161–175. USENIX Association, 2002.

[137] W. S. Wong and R. J. T. Morris. Benchmark synthesis using the LRU cache hit function. *IEEE Trans. Computers*, 37(6):637–645, 1988.

[138] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling algorithms for modern disk drives. In *Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 241–251, 1994.

[139] S. Zertal and P. G. Harrison. Multi-RAID queueing model with zoned disks. In *Proc. High Performance Computing and Simulation Conference (HPCS'07)*, June 2007.