Imperial College London
Department of Computing

Reducing Subtask Dispersion in Parallel Queueing Systems

Iryna Tsimashenka

Supervised by Dr William J. Knottenbelt

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

Abstract

In various real-world parallel processing systems, incoming tasks divide into several subtasks that are processed independently by parallel servers. Queueing networks are a natural way to represent the flow and processing of tasks and subtasks in such systems. Two useful classes of queueing network representations are split-merge and fork-join systems. There are two main metrics of interest in these systems: task response time and subtask dispersion. These metrics are in tension with each other: when one is reduced, it tends to lead to an increase in the other. Generally, using the fork-join paradigm leads to low task response times but high subtask dispersion, while using the split-merge paradigm leads to low subtask dispersion but moderate to high task response times.

This thesis introduces methods for controlling subtask dispersion as well as for the trading off of subtask dispersion and task response time in parallel queueing systems.

In the context of split-merge systems with generally distributed service times, we show how to control mean subtask dispersion by the application of judiciously-chosen delays to subtask processing and extend it to control percentiles of the distribution of subtask dispersion. Our analysis is based on extensions to the theory of heterogeneous order statistics. While solely focusing on the reduction of subtask dispersion leads to a large increase in task response time, together with a corresponding decrease in maximum sustainable system throughput, aiming to reduce a product of mean subtask dispersion and mean task response time leads to a marginal increase in task response time while dramatically improving mean subtask dispersion. Fork-join systems are widely deployed in the real world, but are notoriously more difficult to analyse. In the context of fork-join systems with heterogeneous exponentially distributed service times, we present an on-line technique which improves on both the mean task response time and mean subtask dispersion achievable in an equivalent split-merge system. For split-merge systems we validate our results analytically, while for fork-join systems we validate the solutions against simulations.

We present case studies of different parts of our methodology in split-merge and fork-join systems with and without applications of the delays. These show the ability to reduce subtask dispersion while providing increasingly-sophisticated means to simultaneously control task response time.

Copyright

Acknowledgements

I would like to thank:

- My supervisor, Dr. William Knottenbelt for guiding and motivating me and being able always to find time for the discussions.
- My second supervisor Prof. Peter Harrison for his help with technical aspects and sharing his expertise.
- Dr. Richard Hayden for insightful discussions.
- My parents for their support, love and encouragement.
- My friends, you know who you are :)

To my parents.

Моим родителям посвящается.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In recent decades, sequential systems have increasingly been superseded by parallel ones. This applies as much to computer systems (e.g. processors and storage devices) as it does to physical systems which process customers and tasks (e.g the automated warehouses of online retailers). In many instances, incoming tasks are divided into subtasks which are concurrently processed by a set of parallel servers. Completed subtasks are held in an output buffer. When all subtasks have completed service, the task is complete and exits the system.

We term such systems *parallel processing systems*. System designers need rigorous mathematical formalisms and tools to tackle problems that relate to the performance optimisation and analysis of such systems.

Queueing systems have been one of the most widely adopted formalisms because they are a natural way to represent the flow and processing of tasks and resources in parallel processing systems. One of the most well-known queueing systems in a real world are fork-join systems, they are noted for their asynchronous operation and consequent low task response time and high maximum sustainable throughput. This operation is achieved because this kind of system supports several parallel queueing servers and each task forks into subtasks that go directly to

the parallel servers at the instant of arrival. The drawback of these systems is that it is very hard to analyse them; indeed there are no closed form analytical solutions for computing their associated performance metrics.

A more synchronised counterpart of the fork-join queueing system is a split-merge queueing system which are analytically tractable and, as we shall see, elements of their analysis may be extended to fork-join queueing systems. Both split-merge queueing systems and fork-join queueing systems are important queueing systems and have many applications in real world. This thesis is restricted and scoped to the analysis and optimisation of elementary parallel queueing systems in which there is only one level of subtask processing.

1.1.1 Performance Metrics in Parallel Queueing Systems

As more parallel queueing systems have been applied in real-life applications, the more they have gained scientific attention with respect to the analysis and optimisation of their associated performance metrics.

There are two important performance metrics in split-merge and fork-join queueing systems:

- **Task response time**, that is the time taken from the entry of a task into the system until its exit. This has been the primary focus of research effort over many decades (see e.g. [50, 37, 36, 75, 55]). The vast majority of this work targets the mean, and rarely higher moments and high/low bounds, of task response time, and/or the stationary distribution of the number of subtasks queued at parallel servers.

- **Subtask Dispersion**, that is the difference in time between the service completions of the first and last subtask of any given task to complete service. As discussed below, many applications require the times of arrival of a subtask and its siblings to be clustered as close together as possible. Research interest in this metric is relatively recent and has been mostly driven by publications associated with this dissertation, see e.g. [97, 99, 96, 98].

These metrics are in tension in the sense that taking action to reduce one usually results in an increase in the other (at least in the absence of structural reconfiguration or reparametrisation of the system); this is especially the case for high-intensity workloads. In this thesis we assume that structural reconfiguration or reparametrisation of the system is not possible. Classical fork-join queueing systems yield low task response times (and therefore higher maximum sustainable system throughput), but subtask dispersion is high under load. Conversely, split-merge queueing systems are characterised by low to moderate subtask dispersion, but can suffer from higher task response times (and therefore reduced maximum sustainable system throughput) under load. In this thesis we investigate minimisation of subtask dispersion and its impact on task response time.

1.1.2 Real-World Examples

There are enormous number of applications of fork-join and split-merge queueing systems in industry. Here we discuss only a few of them.

Examples of Split-Merge Queueing Systems

Air travel has provided a fertile ground for the application of queueing analysis ever since mass air transit became practical and affordable in 1950s [66]. As shown in Fig 1.1, air travellers and their hold baggage undergo two phases of split-merge processing. In the first phase passengers enter the departure hall with their hold baggage and join the queue for check-in. At check-in passengers and their hold baggage are separated (split). Passengers proceed through security and documentation checks while the baggage is securely screened and sorted. Passengers and their hold baggage are reunited (merged) aboard the aircraft, albeit the passengers sit in the seats above the floor level while their baggage is stored in unit load devices below floor level, as shown in Fig 1.1. Upon arrival on the destination airport the second phase of split-merge processing begins. Passengers undergo immigration checks while their baggage is offloaded and delivered to the baggage claim area. Once passengers pick up their baggage they can exit

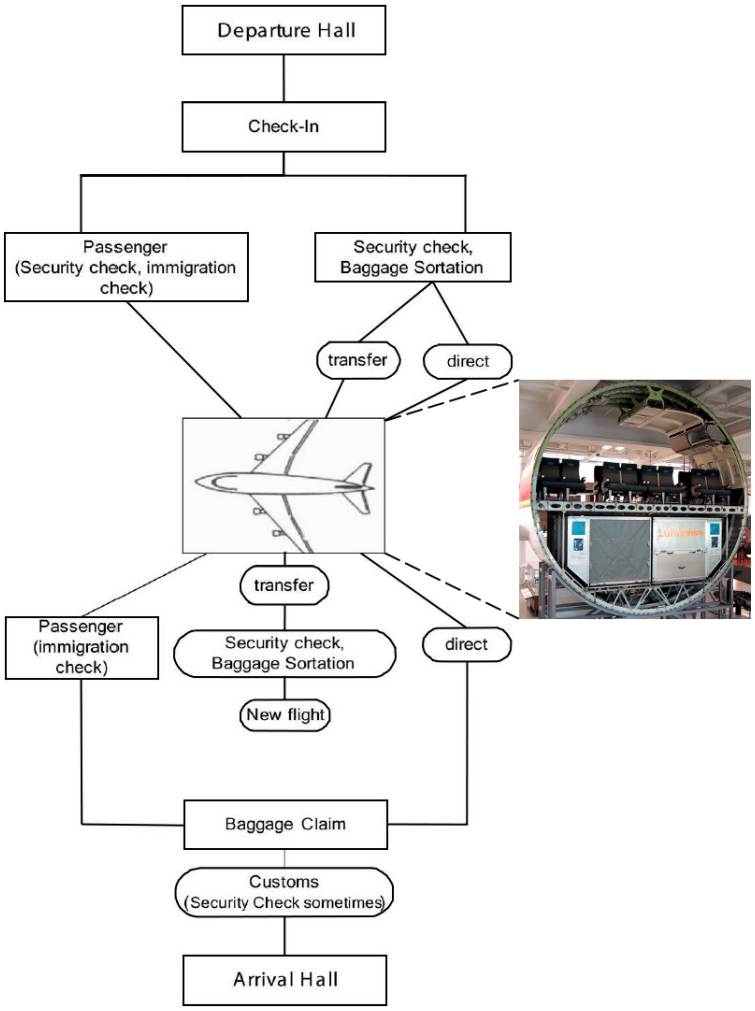

Figure 1.1: Structure of passengers and baggage processing in aircraft as an example of a two phase split-merge processing.

into the arrival hall via customs. If any part of this split-merge processing operation does not complete as it is anticipated there are large financial consequences for the airline: if the passengers are not on board when the aircraft has to depart then the whole plane is delayed; or if the passengers arrive and their luggage is not delivered then the airlines has to hire a courier company to deliver a luggage to the destination.

Another example of split-merge processing comes from Formula 1, when racing cars come in for pit stops (see Fig. 1.2). There are many different procedures that start at the instant the car is raised onto a jack, with different mechanics responsible for different tasks: replacing tires, refuelling, doing repairs and mechanical adjustments etc. All these tasks start at the same instant. A car leaves the pit stop only when all procedures have been completed.

While task response time is clearly the most important metric in the latter example, we now present an example from a completely different and very recent domain in which subtask dispersion plays a crucial role. The domain is a specific financial one, and relates to the placing of large orders across multiple exchanges which support High Frequency Trading (HFT). HFT is a relatively recent automated trading phenomenon which combines super fast networks, rapid analysis algorithms and high speed order placing. Every conceivable technology is exploited in the quest to reduce latency, for example the recent adoption of microwave rather then optic fibre to transmit data (since electromagnetic radiation travels 50% faster through air than light travels in the clearest glass) and the use of custom ASICs featuring so-called "tick-to-trade" times of less than 740 ns. Suppose a trader is in a market to buy a large amount of a given stock and simultaneously issues orders to the major exchanges to buy all shares of that stock available at or below some target price. The orders will not, however, arrive at the exchanges simultaneously, as shown on the left of the Fig 1.3, since geographical distance is a key determine factor in order arrival time [38]. HFT algorithms that detect the buying activity on the geographically nearest exchange can then "front run" the trade on the other exchanges by quickly buying any shares available below the target price and offering them for sale at the target price. (Another riskier but more profitable strategy is to buy up all shares available at or below the target price and offer them for sale at a price slightly above the target price.) In this case the orders set by the trader to the geographically more distant exchanges are returned unfilled and the trader must issue buy shares at the higher price to complete her purchase. What we see as to increasing subtask dispersion leads to increased latency-based information leakage with consequent lower profitability and lower fill rates for the trader. To counter this, the Royal Bank of Canada has developed a system called THOR for which a patent was filed at October 2011 [1]. This order routing system adds some heuristic delays to orders so that they arrive at the entire set of target exchanges nearly simultaneously. The authors claim that the resulting improvement in the execution quality is striking: they say that the fill rates are near 100% rather then 60% for conventional execution orders experience far fewer adverse ticks than average. It is intriguing that adding delays has become an important weapon in the battle to compete against the super fast high performance hardware and trading algorithms of HFT

systems. Therefore, the authors of THOR also exploit the notion that adding delays can have very beneficial impact on subtask dispersion.

Examples of Fork-Join Queueing Systems

Consider by way of example the processing of customer orders in an automated warehouse. Incoming orders (tasks) are made up of several items (subtasks), each of which must be retrieved from a different part of the warehouse. Partially completed orders must be held in an output buffer, and each order can only be released from the output buffer and dispatched to the customer when all items making up the order have been retrieved. The output buffer space is often limited and difficult to manage on account of its high utilisation, so it is important to keep subtask dispersion low. At the same time keeping task response time low (i.e. increasing system throughput) is an important concern.

Another example is a restaurant in which customer orders (tasks) consisting of different menu items (subtasks) must be concurrently prepared such that all dishes for a particular table of customers are ready at roughly the same time. In the mean time, partially completed orders for tables are held on a service counter (the output buffer), which should not be overburdened. Simultaneously a good standard of customer service dictates that customers should receive their orders in reasonable time. A real example showing how a restaurant customer complains about poor task response time and poor subtask dispersion is shown in Fig 1.4. In this review the customer complains that the food has arrived after a long wait (high task response time) and at different times (high subtask dispersion). Consequently the customer was unsatisfied with the service and will not come to this restaurant again.

Search engines use fork-join processing to answer queries with low response time [42], which in turn drives user satisfaction and revenues. Incoming queries are split into parallel subtasks, some of which relate to searching chunks of index data and some of which relate to preparing revenue generating content such as advertisement. Only when *all* of these subtasks have completed can the results page be generated and returned to the user. Users are very sensitive to even slight increases in delay [24].

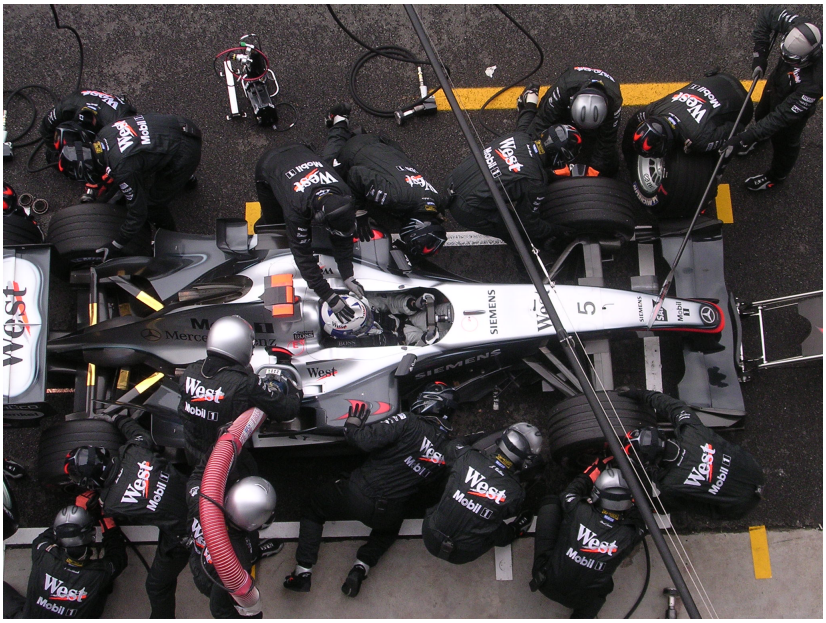

Figure 1.2: A pit stop in Formula 1 is an example of a split-merge queueing system, where a car can leave the pit stop only when all maintenance activities have been completed. Credit: Fabio Alessandro Locati.

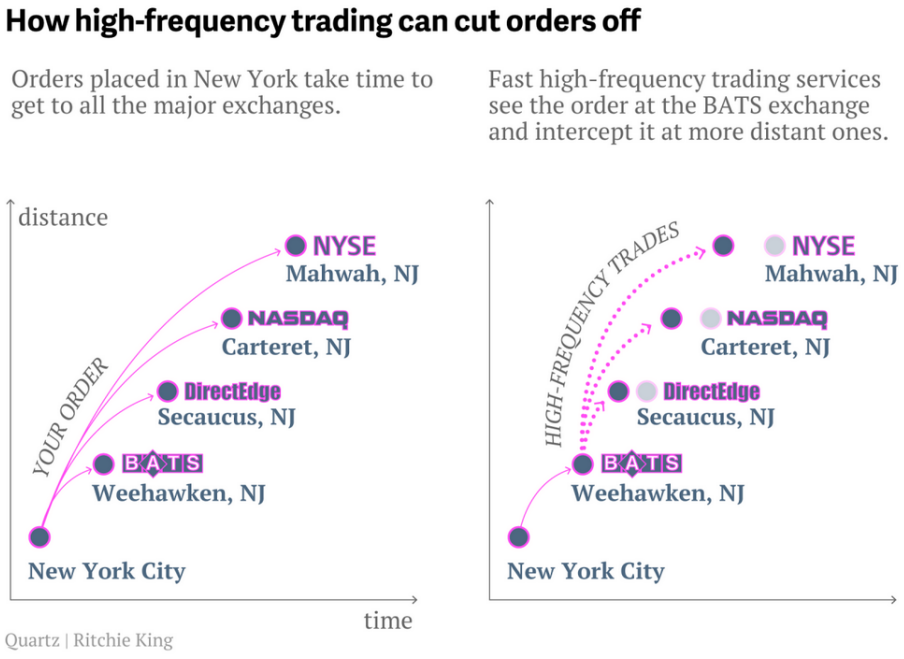

Figure 1.3: Orders sent to number of exchanges. How high-frequency trading can preempt orders. Credit: Quartz. Ritchie King.

Figure 1.4: Example of a customer review and complaint about a restaurant related to poor subtask dispersion and poor task response time. URL: `http://www.tripadvisor.co.uk/ShowUserReviews-g186337-d1986650-r122991388-JR_s_Bar-Liverpool_Merseyside_England.html`

A fork-join queueing policy applies in RAID systems. A RAID system is a Redundant Array of Independent Disks which are used to boost capacity, performance and reliability depending on the configuration of *RAID level* selected. For example, Fig. 1.5 shows RAID-5 set-up, which has n disks with the all information spread out on them, so that every stripe contains $n-1$ blocks of information and a parity block. When one of the disks fails, the information on it can be recovered by XORing the data on the remaining $n-1$ disks.

In general RAID, incoming I/O requests split in smaller sub-requests which are served by n disk, when all requests completed than the logical I/O request is completed; this policy corresponds to a fork-join processing policy across disks [27, 64]. There are some examples of applications

of split-merge queueing policy synchronisation in case of updating a replicated data, which includes updating mirrored disks [93], or replicated data bases in general [74].

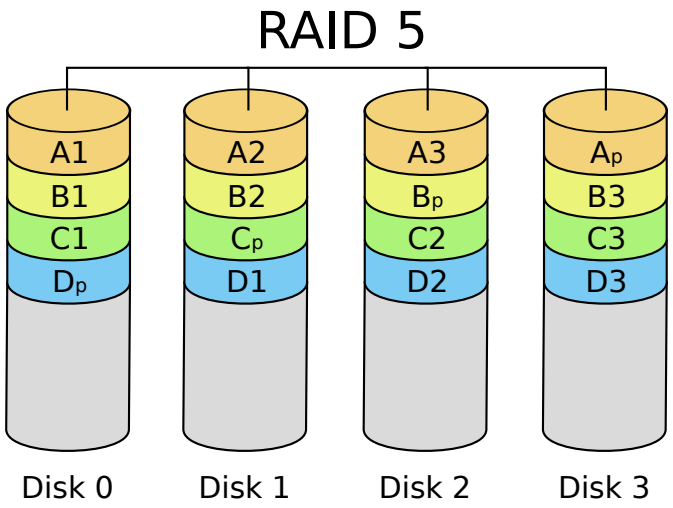

Figure 1.5: Structure of RAID-5. Credit: Colin M.L. Burnett.

1.2 Objectives

The primary hypothesis of this thesis is that it is possible to control the injection of subtasks into parallel queueing systems in such a way that subtask dispersion is reduced while impact on task response time can be quantified and ideally controlled. We do this by applying judiciously-chosen deterministic delays to the processing of subtasks. In order to achieve this, the following objectives must be fulfilled:

- Develop the theory of heterogeneous order statistics, particularly deriving the mean and the distribution of the range of heterogeneous order statistics, which will allow us to subsequently characterise the mean of the subtask dispersion and the distribution of subtask dispersion in elementary split-merge systems.

- Create a methodology for minimising mean subtask dispersion in elementary split-merge queueing systems with heterogeneous general service time distributions. This is based on applying judiciously-chosen deterministic delays to processing of the subtasks.

- Create an analogous methodology for reducing any given percentile of the distribution of subtask dispersion in elementary split-merge queueing systems with heterogeneous general service time distributions.

- Derive an analytical solution for the mean number of subtasks held in the output buffer of an elementary split-merge queueing systems, with and without subtask delays.

- Characterise the impact on task response time and maximum sustainable system throughput of our methodology of applying deterministic subtask delays optimised only for subtask dispersion.

- In order to control the adverse impact of applying delays to the processing of subtasks on task response time, develop a method for reducing the product of mean subtask dispersion and mean task response time in elementary split-merge queueing systems.

- Extend our methodology of reducing mean subtask dispersion in split-merge queueing systems to their asynchronous counterparts – fork-join queueing systems. We restrict our attention to fork-join queueing systems with heterogeneous exponential service time distributions and create a dynamic online algorithm for reducing mean subtask dispersion.

- Implement simulators for elementary split-merge and fork-join queueing networks in order to achieve mutual validation between methodologies presented in this thesis and the simulations.

1.3 Contributions

This dissertation presents a methodology for reducing subtask dispersion in parallel queueing systems, specifically elementary split-merge queueing systems and fork-join queueing systems by applying optimal deterministic delays to the processing of subtasks.

Our main contributions are fourfold, as detailed below.

1.3.1 Theory of Heterogeneous Order Statistics

We extend the classical theory of homogeneous order statistics to heterogeneous order statistics. Firstly, we define the mean of the range of heterogeneous order statistics. Secondly we derive the distribution of the range of heterogeneous order statistics. These formulae correspond to the mean and distribution of the subtask dispersion in elementary split-merge queueing systems. These analytical solutions form the foundations of our methodology for reducing subtask dispersion in parallel queueing systems in general.

1.3.2 Reducing Subtask Dispersion in Split-Merge Queueing Systems

We develop a methodology for reducing the mean or a given percentile subtask dispersion in split-merge queueing systems by applying judiciously-chosen deterministic delays to the processing of subtasks. We consider Poisson arrivals and heterogeneous general service time distributions. The methodology builds on the theory of heterogeneous order statistics developed above. We present numerical optimisation procedures for minimising mean subtask dispersion or any given percentile of distribution of subtask dispersion. We implement a split-merge queueing system simulator to validate the results. We derive an analytical solution for the mean number of subtasks present in the output buffer based on Little's law. We quantify the impact of applying delays to the processing of subtasks on task response time and maximum sustainable system throughput.

1.3.3 Trading off Mean Subtask Dispersion and Mean Task Response Time

We extend our methodology for reducing subtask dispersion alone to reducing a product of mean subtask dispersion and mean task response time. This trading off of two major performance metrics gives us means to control subtask dispersion while task response time is only marginally affected. We consider an elementary split-merge queueing system with Poisson arrivals and

general heterogeneous service time distributions. We also dramatically improve the run time of our previous optimisation methodology by applying Brent's method and a memoisation technique.

1.3.4 Reducing Mean Subtask Dispersion in Fork-Join Queueing Systems

Our final contribution relates to analytically intractable and asynchronous fork-join queueing systems. Since these systems are more difficult to analyse, we explore them under simplified assumptions of heterogeneous exponential service time distributions and Poisson arrivals. We create a dynamic online algorithm for reducing mean subtask dispersion in such systems. By exploiting the memoryless property of the exponential distribution, we are able to show an equivalence between the state of a fork-join queueing system at a certain time points and a split-merge queueing system of a particular state-dependent configuration from the perspective of selected subtasks. This allows us to exploit some of our previous results. We implement a fork-join queueing system simulator to validate our methodology.

1.4 Thesis Outline

Figure 1.6 shows the context of the chapters of this thesis.

The remainder of this dissertation is organised as follows:

- **Chapter 2** describes the background theory that relates to the present research. Some elementary probability theory is presented followed by the basics of the queueing theory and the theory of homogeneous order statistics, which is needed for the study of parallel processing queueing networks. Lastly, a literature survey about related work on parallel queueing networks is presented.

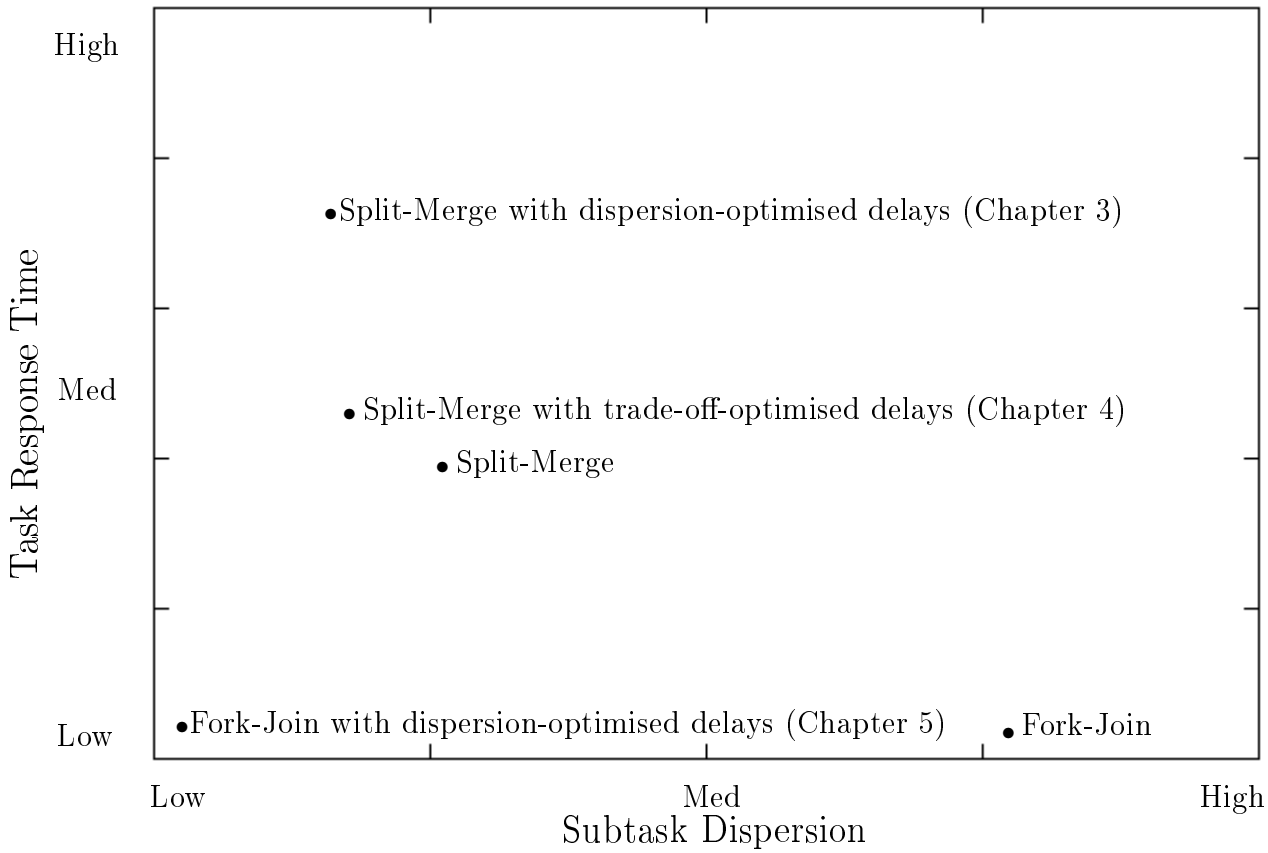

Figure 1.6: Thesis chapters in the context of our two target performance metrics.

- **Chapter 3** develops the theory of heterogeneous order statistics – an extension of classical homogeneous order statistics. We use this theory as a foundation of our methodology for reducing mean and percentiles of distribution of the subtask dispersion in elementary split-merge queueing systems. We provide three case studies to show the applications of the methodology and validate our results with simulations. We present an analytical solution for computing the mean number of subtasks present in the output buffer using Little's law. We show how to quantify the inevitable impact of the applied delays on task response time and maximum sustainable system throughput.

- **Chapter 4** extends our technique of solely reducing subtask dispersion to a methodology based on trading off mean subtask dispersion and mean task response time. This extension was done to mitigate the adverse effects of applying delays to the processing of subtasks on task response time and maximum sustainable system throughput. We define a new objective function – a product of mean subtask dispersion and mean task response time. We demonstrate of our introduced methodology in the context of a case study.

- **Chapter 5** explores ways to reduce mean subtask dispersion in elementary fork-join queueing networks. We present a dynamic online algorithm for reduction of mean subtask dispersion and show a case study. This analysis uses elements of the work from previous chapters but only to instantaneously analyse particular states of the system from the perspective of a particular subtask.

- **Chapter 6** concludes the dissertation by outlining and evaluating the research achievements and describing possible avenues for future work.

1.5 Publications and Statement of Originality

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

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

- **1st Imperial College Computing Student Workshop (ICCSW) 2011** [97] presents a technique for reducing mean subtask dispersion in split-merge systems with Poisson task arrivals and heterogeneous general service time distributions. We derive an expression for the mean of the range of heterogeneous order statistics based on random variables that arise from a certain service time distributions. We control subtask dispersion by introducing a vector specifies the deterministic delay applied to the processing of subtasks before service at a given parallel server. We update the formula of mean subtask dispersion to cater for an arbitrary delay vector. We describe a numerical optimisation procedure for finding that delay vector which gives the minimum mean subtask dispersion for given service time distributions and Poisson task arrivals. We present a case study of this technique based on an elementary split-merge queueing system with 3 parallel servers. The work in Section 3.2 is based on this paper.

- **19th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA) 2012** [99] presents a technique for reducing a given percentile of the range of subtask dispersion in split-merge queueing systems with Poisson task arrivals and heterogeneous general service time distributions. We characterise the full distribution of the range of subtask dispersion by deriving the distribution of the range of heterogeneous order statistics. We introduce a vector of delays into the system, where each element of the vector is applied to the processing of subtasks by a particular parallel server. We update the formula of the distribution of subtask dispersion to support an arbitrary vector of delays. We use inversion on a given percentile of a distribution function to find an appropriate vector of applied delays. We present a case study of this technique based on an elementary split-merge queueing system with 3 parallel servers. The work in Section 3.4 is based on this paper.

- **Trends in Parallel, Distributed, Grid and Cloud Computing for Engineering (PARENG) 2013** [58] gives additional examples of the real-world context of our work, and summarises techniques for reducing subtask dispersion in split-merge systems from [97, 99] with quantification of adverse impact of applying deterministic delays to the processing of subtasks. It also includes some speculative and preliminary ideas for

extending the approach to fork-join queueing systems. The work in Chapter 3 is based on this book chapter.

- **20th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA) 2013** [96] presents a technique for trading off a mean task response time and mean subtask dispersion in split-merge queueing systems with Poisson task arrivals and heterogeneous general service time distributions. The objective function is a product of mean subtask dispersion (defined using heterogeneous order statistics) and mean task response time (computed using the Pollaczek-Khinchine formula). The work in Chapter 4 is based on this paper.

- **10th European Workshop on Performance Engineering (EPEW) 2013** presents a dynamic online algorithm for reducing mean subtask dispersion in fork-join queueing systems with Poisson task arrivals and heterogeneous exponential service time distributions. In this algorithm we repeatedly compute and adjust the delays on processing of subtasks according to the current state of the fork-join queueing system. In order to do this we construct an equivalent split-merge queueing system that represents the view of certain set of subtasks at certain time instants. The work in Chapter 5 is based on this paper.

- **Annals of Operations Research 2014** [100] presents the extended version of paper [96] - a technique for trading off a mean task response time and mean subtask dispersion in split-merge queueing systems with Poisson task arrivals and heterogeneous general service time distributions. The objective function is a product of mean subtask dispersion (defined using heterogeneous order statistics) and mean task response time (computed using the Pollaczek–Khinchine formula). We show the impact of applying the optimal delays on system stability and task response time. Two case studies illustrate the applicability of our approach. The work in Section 3.2.4 and results of 3.4.2 based on this paper.

Chapter 2

Background Theory

2.1 Introduction

In this section we present background theory relevant to the research area of this thesis. This chapter contains two parts. In the first part we present selected theoretical topics; these include the application of random variables, stochastic processes, queueing networks, parallel processing systems, performance metrics, theory of order statistics and optimisation algorithms. The second part presents a review of related work in the area of performance analysis of parallel queueing systems, where the major focus is on approximations of bounds on task response time, scheduling algorithms for reducing response time and analysis of work-flows with directed acyclic graph structure.

2.2 Random Variables

A *random variable* X is a numerical variable whose value depends on a result of a random experiment [101]. A random variable X is a function whose domain is a *sample space*, that is a set of all mutually exclusive possible outcomes of a random experiment. The range of a random variable is a real line – it is often a natural way to represent discrete points into integers in a

real life [48, 101]. If a random variable X takes only discrete values then it is called a *discrete random variable*. If a random variable that takes on values real interval, $X \in [a, b]$ where $-\infty \leq a \leq X \leq b \leq \infty$, then the random variable is called a *continuous random variable* [77].

A discrete random variable can be described by its *probability mass function* (pmf). The probability that a discrete random variable takes on a particular value can be computed by:

$$p_X(x) = \mathbb{P}[X = x] = \mathbb{P}(x).$$

The function is called *probability mass function* if it satisfies the following conditions:

- $p_X(x)$ is defined for all values of x, but $p_X(x) \neq 0$ only at a finite set of points.
- $0 \leq p_X(x) \leq 1$ means that all values of $p_X(x)$ lie within interval $[0, 1]$.
- $\sum p_X(x) = 1$ where sum is taken across all values of x.

A continuous random variable X can be described by its *cumulative distribution function* (or cdf, or distribution function):

$$F_X(x) = \mathbb{P}(X \leq x)$$

This reflects that the probability that a sample from continuous random variable X is less than or equal to a particular value x [19]. If this distribution function is differentiable, then a *probability density function* can be derived by differentiating F_X:

$$f_X(x) = \frac{\partial F_X(x)}{\partial x}$$

where the density function has to satisfy the following conditions:

- $f(x)$ is defined for all values of x.
- $0 \leq f(x) < \infty$, that is all values of $f(x)$ lie in the interval $[0, \infty)$.
- $\mathbb{P}[a \leq X \leq b] = \int_a^b f_X(x)dx$

- $\int_{-\infty}^{\infty} f_X(x)dx = 1$, that is the area under the curve of the density function is one.

2.2.1 Expectations of Random Variables

The expected value or mean of a discrete random variable X, is a weighted average of all possible values it can take on [84]:

$$\overline{X} = \mathbb{E}[X] = \sum_{k=0}^{\infty} k\mathbb{P}(X = k)$$

The expectation value of a continuous random variable X, if its probability density function is known, is given by:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} xf(x)dx$$

If random variable X takes only positive values, then applying integration by parts we have

$$\mathbb{E}[X] = \int_{0}^{\infty} (-x)(-f_X(x))dx = [-x(1 - F(x))]_0^{\infty} + \int_{0}^{\infty} (1 - F(x))dx$$

this yields:

$$\mathbb{E}[X] = \int_{0}^{\infty} 1 - F(x)dx \tag{2.1}$$

Linearity Property of Expectation Operator

The expectation of the difference between two (possibly dependent) random variables X, Y is given by [45, 71]:

$$\mathbb{E}[X - Y] = \mathbb{E}[X] - \mathbb{E}[Y] \tag{2.2}$$

which means that the dependence between random variables is irrelevant when considering mean values due to the linearity property of expectation operator.

Monotonicity Property of Expectation Operator

For two (possibly dependent) random variables X and Y, a random variable X is less than a random variable Y (or $X \preceq Y$) in the usual stochastic order if $\mathbb{P}(X > x) \leq \mathbb{P}(Y > x)$ for all $x \in \mathbb{R}$, then:

$$\text{If} \quad X \preceq Y, \text{ then} \quad \mathbb{E}[X] \leq \mathbb{E}[Y] \tag{2.3}$$

which means that if X is less than or equal to the Y almost surely, then the expectation operators satisfy the same inequality [45].

2.2.2 Variance

Variance (or second central moment) characterised the expected spread of a set of samples drawn from a random variable X about its mean $\mathbb{E}[X]$. The variance of a discrete random variable X is [71]:

$$\sigma_X^2 = \text{Var}[X] = \overline{(X - \overline{X})^2} = \overline{X^2} - \overline{X}^2$$

Similarly to Eq. 2.1, in case of a continuous positive random variable the variance can be expressed through the cumulative distribution function as:

$$\text{Var}[X] = 2\int_0^\infty x\big(1 - F(x)\big)\,dx - \big(\int_0^\infty 1 - F(x)\,dx\big)^2. \tag{2.4}$$

Next, we describe examples of well-known discrete and continuous random variables [19] which are utilised in this thesis. Choice of continuous random variables is made to cover a full range of possible variances from very low to very high:

Bernoulli RandomVariable: If an experiment has only two possible outcomes: 0 or 1, the pmf of the random variable X is:

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

where $0 < p < 1$.

Binomial Random Variable: Consider an experiment with only two possible outcomes (0 or 1). If the random variable X is the number of times that 1 arises out of n independent trials of the experiment, the pmf of X is:

$$\mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \quad k = 0, 1, \ldots, n.$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Geometric Random Variable: Consider an experiment with two possible outcomes (0 or 1). The experiment is carried out several times. If the random variable X is the number of trials it takes before the outcome 1 occurs (including the current trial), then its pmf is:

$$\mathbb{P}(X = k) = p(1-p)^{k-1} \quad k = 1, 2, ...$$

Exponential Random Variable: If a random variable X is exponentially distributed with parameter λ then its cdf is:

$$F_X(x) = \begin{cases} 1 - e^{-\lambda x} & 0 \leq x < \infty, \\ 0 & x < 0. \end{cases}$$

Such random variables are often used to describe interarrival times in Poisson arrival streams or to describe Markovian service times.

Erlang Random Variable: Consider an experiment which consists of identical k exponential distributed time phases in tandem. Random variable X denotes the whole time span with the cdf calculated by:

$$F_X(x) = \begin{cases} 1 - e^{-\mu x} \sum_{j=0}^{k-1} \frac{(\mu x)^j}{j!} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

Each phase has a rate parameter μ.

Pareto Random Variable: Pareto random variable X with parameters k and α, has cdf:

$$F_X(x) = \begin{cases} 1 - (\frac{k}{x})^\alpha & x \geq k, \\ 0 & x < k. \end{cases}$$

where α is referred to as the shape of the cdf and k is referred to as its rate.

The Pareto distribution may be bounded in an interval $[a, b]$. Then the Pareto random variable X with parameters α, a, b has a cdf:

$$F_X(x) = \begin{cases} 0 & x < a, \\ \frac{1-a^\alpha x^{-\alpha}}{1-(\frac{a}{b})^\alpha} & a \leq x \leq b, \\ 1 & x > b. \end{cases}$$

where α corresponds to the shape, a denotes the minimum value and b denotes the maximum value of the distribution.

Uniform Random Variable: Consider a random variable X with parameters a, b which is distributed uniformly over the interval $[a, b]$, the cdf can be calculated by:

$$F_X(x) = \begin{cases} 0 & x < a, \\ \frac{x-a}{b-a} & a \leq x < b, \\ 1 & x \geq b. \end{cases}$$

Deterministic Random Variable: Consider an experiment where a random variable X takes only a particular value k; then its cdf is:

$$F_X(x) = \begin{cases} 0 & x < k, \\ 1 & x \geq k. \end{cases}$$

Sometimes $F_X(x)$ is called the Heaviside function.

2.3 Stochastic Processes

A stochastic process is a family of random variables $\{X(t), t \in T\}$, where t is a time parameter. A set of all possible values of $X(t)$ is the state space S [19]. In this section we are going to summarise properties of stochastic processes that are applied in this thesis.

2.3.1 Markov Processes

Consider a stochastic process $\{X(t), t \in T\}$ for all $0 = t_0 < t_1 < \cdots < t_n < t_{n+1}$ with a state space of the process is $S = \{s_i = i, i \in \mathbb{N}_0\}$. If the future state of the system depends only on its current state and not on the past history of the system then the system can be described by a *Markov process* and $X(t)$ is said to have the *Markov property* or *memoryless property* [18]. That is,

$$\mathbb{P}\{X(t) = s \mid X(t_n) = s_n, X(t_{n-1}) = s_{n-1}, \ldots, X(t_0) = s_0\} = \mathbb{P}\{X(t) = s \mid X(t_n) = s_n\}.$$

A Markov process is called *time-homogeneous* if a Markov process is invariant to shifts in time [18]:

$$\mathbb{P}(X(t + \widetilde{t}) = s \mid X(t_n + \widetilde{t}) = s_n) = \mathbb{P}(X(t) = s \mid X(t_n) = s_n)$$

2.3.2 Discrete Time Markov Chain

A stochastic process $\{X_0, X_1, \ldots, X_{n+1}, \ldots\}$ is a *Discrete Time Markov Chain* (DTMC) if the following relationship holds for all $n \in \mathbf{N}_0$ and all $s_i \in S$ [21]:

$$\mathbb{P}\{X_{n+1} = s_{n+1} \mid X_n = s_n, X_{n-1} = s_{n-1}, \ldots, X_0 = s_0\} = \mathbb{P}\{X_{n+1} = s_{n+1} \mid X_n = s_n\}.$$

2.3.3 Continuous Time Markov Chain

A given stochastic process $\{X(t), t \in T\}$ constitutes a *Continuous Time Markov Chain* (CTMC) if for any $t_i \in \mathbb{R}_0^+$, where $0 = t_0 < t_1 < \cdots < t_n < t_{n+1}, \forall n \in \mathbb{N}, \forall s_i \in S = \mathbb{N}_0$ the following relation holds:

$$\mathbb{P}\{X(t_{n+1}) = s_{n+1} \mid X(t_n) = s_n, X(t_{n-1}) = s_{n-1}, \ldots, X(t_0) = s_0\} = \mathbb{P}\{X(t_{n+1}) = s_{n+1} \mid X(t_n) = s_n\}.$$

2.3.4 Poisson Processes

Consider an experiment where events occur with exponentially distributed interarrival times, then the random variable $X(t)$ with Poisson distribution represents the number of events that occur in time interval $(0, t)$. The set of random variables $\{X(t), t > 0\}$ is a *Poisson process* [21, 56]

$$p(x) = \frac{e^{-\lambda}\lambda^x}{x!}.$$

2.4 Queueing Theory

Queueing theory relates to the study of lines of tasks which are waiting to receive service [21, 43]. The word queue comes from *French* **cue** which means **tail**, which comes from *Latin* **coda**. Queueing theory is a discipline in operations research which analyses and predicts the stochastic behaviour of queues in order to improve key performance metrics and overall utilisation of resources [77, 48]. Queueing theory is applied in many industrial areas such as telecommunications, computer networks, warehouses of online retailers, transportation and logistics etc. [102]. In computer systems, for example, queueing networks are often use to analyse and simulate performance metrics in RAID (Redundant Array of Independent Disks) systems [62, 61]. In wireless networks, queueing networks are used to model multimedia transmissions over downlinks, where queue has time or space priority [2].

In health centres, an example of a queue is where patients (tasks) have to wait (queue) in order

to see a doctor (server), possibly subject to some form of priority service [7, 8].

In 1953 David Kendall [53] developed a short notation to characterise a queueing system with a single waiting queue: $A/B/N/Y/Z$, where A describes an interarrival distribution, where the common types are M for Markov process and D for deterministic distribution and more sophisticated ones include *MMPP* (Markov Modulated Poisson Process) and *NHPP* (Non-Homogeneous Poisson Process). B defines a service time distribution; common distributions are M, G, D where G stands for general service time distribution. N expresses a number of parallel servers, Y is the system capacity and Z specifies a queueing discipline. Sometimes notation $A/B/N$ is used; this means that the system capacity is infinite and queueing discipline is First Come First Served (FCFS), rather than, for example, priority queueing or Last Come First Served (LCFS) [53].

2.4.1 *M/M/1* queue

In general a queueing network is a set of simple queues which are connected together. One of the well-studied and simplest queues to analyse is the *M/M/1* queue. In such queues interarrival times as well as service times have exponential distributions with rates λ and μ respectively [6]. Consequently, a *M/M/1* queue characterises as a continuous time Markov chain with whose state variable is a number of tasks in the queue. The queue's utilisation is $\rho = \frac{\lambda}{\mu}$, where $\rho < 1$ so $\lambda < \mu$ in order for queue to be stable. Using properties of Markov chains it is straightforward to compute some performance metrics, such as mean response time, which is given by:

$$\mathbb{E}[R] = \frac{1}{\mu - \lambda}$$

2.4.2 *M/G/1* queue

Another well-known queue is the *M/G/1* queue, which has exponential interarrival times and general service time distributions [21, 46]. These types of queues are highly applied in the real world because of their general service time distribution property. However this flexibility makes

them harder to solve analytically.

Although for *M/G/1* queues the theory of Markov process no longer holds [95], we can exploit the fact that arrivals still have the Poisson property. The *PASTA* (Poisson Arrivals See Time Averages) property states that the probability that an entering task sees a particular number of tasks in the queue is equal to the steady-state probability of the queue [110]. *M/M/1* queue is a special case of *M/G/1* queue; therefore a method that is applicable to *M/G/1* queue is applicable to *M/M/1* queue as well, but not vice versa. The queueing systems that we study in this thesis are mainly built on *M/G/1* and *M/M/1* queues.

2.5 Performance Metrics for Queues

Pollaczek-Khinchine Formula of Task Response Time

Mean task response time in a steady-state *M/G/1* queue with arrival rate (λ), mean service time distribution ($1/\mu$), variance ($\mathrm{Var}[X]$) of a service time and $\rho = \frac{\lambda}{\mu}$ is utilisation, characterised by a random variable X. At any random time instant mean response time can be calculated by Pollaczek-Khinchine formula [82, 54]:

$$\mathbb{E}[R] = \frac{\rho + \mu\ \lambda\ \mathrm{Var}[X]}{2\ (\mu - \lambda)} + \mu^{-1} \tag{2.5}$$

This result can be extended to a queueing system with Poisson arrivals and general service time distributions which can be conceptually represented as *M/G/1* queue with arrival rate (λ), service time distribution is maximum mean service time ($1/\mu$) across all parallel servers and variance.

Little's Law

Little's law is one of the most simple, but powerful results in queueing theory relating to steady-state performance measures. It had been an empirical rule for many years before it was proven

by J.D.C. Little in 1961 [77].

This law states that in a steady-state queueing system the average number of tasks including those in service (N) is the product of average arrival rate λ of the tasks entering the system and the average time (T) that tasks spend in the queueing system before they exit it [69]:

$$N = \lambda T \tag{2.6}$$

The beauty of this theorem is that it places no restriction on the arrival process and service time distribution; it also supports any queueing discipline and does not depend on the number of parallel servers in the queueing system.

2.5.1 Multi-class Queues

A multi-class queue serves different class of customers, each of which has its own service time distributions and has different routing behaviour.

2.5.2 Queues with Priorities

Consider a queue with a server and a FCFS waiting queue of tasks. At any time when a task comes into the queue, it goes at the back of FCFS waiting queue and waits while all tasks before it complete their service. Only after all tasks before the task have been served does the task start receiving service. Yet, in some queues tasks have priorities; if a task has a higher priority than others, then this task jumps the queue of tasks with lower priority and goes straight to a server. This kind of queues called *queues with priorities* [77]. In *preemptive queues with priorities* when a task with higher priority arrives, a server stops service of a current lower-priority task immediately and starts service of a task with a higher priority. In *non-preemptive queues with priorities* there is no interruption of service of a task that is in a service already, therefore a task with a high priority has to wait while a server finishes service of a current task and a task with a higher priority can start receive service [77].

2.5.3 Queueing Networks

A natural extension of a simple queue is a queueing network, which is a set of interacting queues. Some queues may be situated in parallel with each other, some may be set one after another such that, when tasks exit one queue they immediately enter the next queue. Usually a queueing network has to be analysed as a whole on account of the interaction and dependence between queues [77]. If a certain conditions are met, a queueing network has a product form solution, in which case each queue may be analysed separably; the answer can then be expressed as a product of all individual queues [47]. However such cases really encounter in practice and may even depend on the value of particular transition rates or particular service or arrival rates in the system [47]. An example of a simple queueing network is presented in Fig. 2.1 from [57].

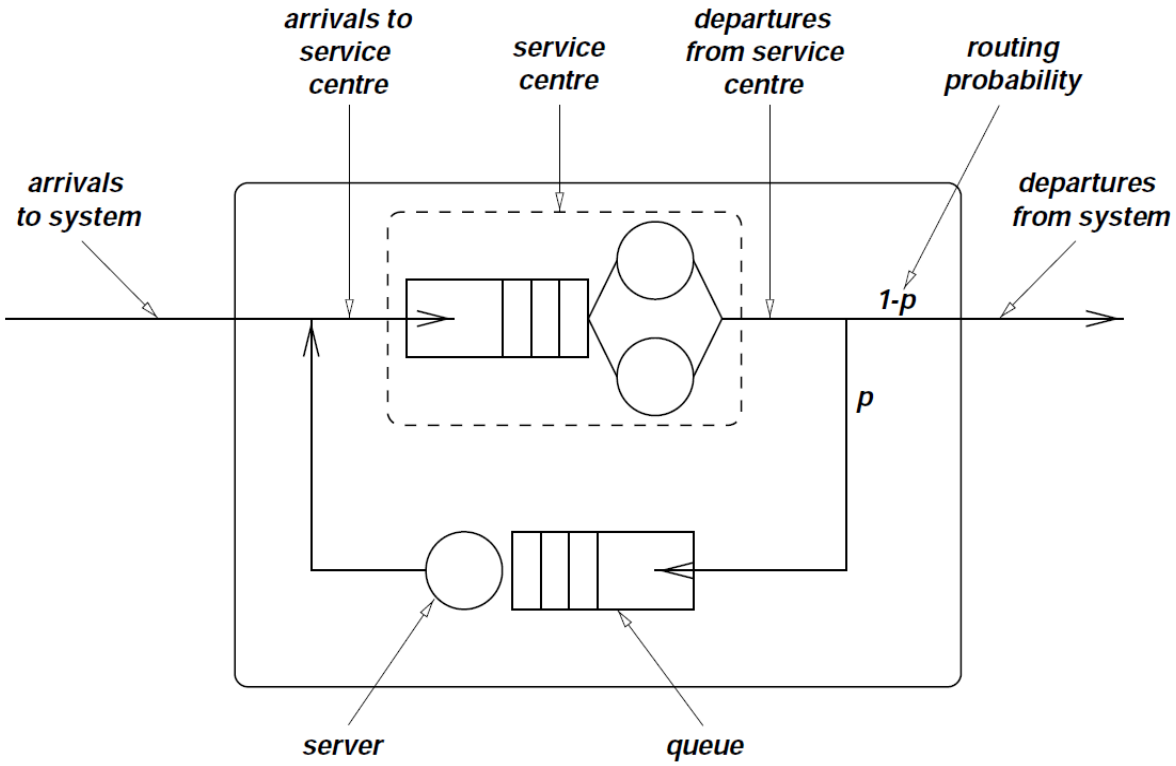

Figure 2.1: Example of a queueing network [57].

2.5.4 Open, Closed, Mixed Queueing Networks

Queueing networks can classified as open, closed and mixed. The major difference is where tasks are arriving from. If tasks arrive into a queueing system from one or more external sources then it is a *open queueing system*. By the flow conservation principle the total sum of arrival rates is equal to the total sum of departure rates in a steady–state queueing system [77].

In a *closed queueing system* tasks do not arrive from external sources nor do they exit the system. Instead tasks circulate inside the system. In a *mixed queueing system* there are several classes of tasks; some of the classes are closed while other classes are open.

2.6 Parallel Processing Systems

In this section we describe essential foundations of parallel processing systems which are related to this thesis. There are two primary areas in parallel processing systems: systems with synchronisation and systems where synchronisation is not required [21]. Since in this thesis we present research which is based on systems that require synchronisation, here we give description of only this kind of system.

Parallel processing systems are applied in areas such as parallel and distributed computing, computer networks and the warehouses of on-line retailers.

In area of concurrent and distributed computing Edsger W. Dijkstra was one of the most influential scientists in the computer science world. He presented the first solution for the mutual exclusion problem [33] which evolved into the area of concurrent and distributed computing and he also invented semaphore-based synchronisation [34]. He solved the shortest path problem for graphs with non-negative edge path costs [32]. Among his contributions are the self-stabilisation concept [35], programming languages, algorithms and protocols in concurrent and distributed computing.

Queueing network models are natural abstractions for representing the flow and processing of tasks in parallel systems in which high-level tasks split into subtasks which are concurrently

processed by a set of (heterogeneous) parallel servers. We present definitions of two subclasses of queueing network models for parallel processing systems that have been under research in this dissertation, namely *split-merge* and *fork-join* queueing systems.

2.6.1 Split-Merge Queueing System

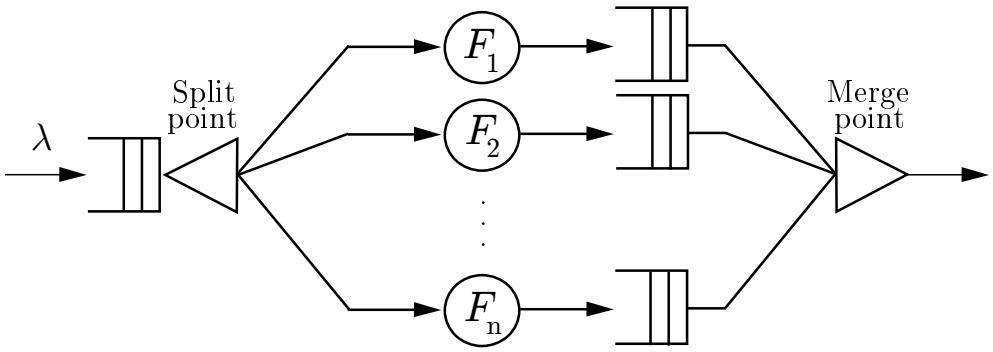

Figure 2.2: Split–Merge queueing model.

We present an *elementary split-merge system* (see Fig. 2.2), where the system processes only one task at a time. A split-merge system consists of split and merge points, a FCFS queue before the split point (split queue) and several heterogeneous parallel servers with queueing capability after service (merge buffers). When a task arrives in the system (usually assumed to happen according to a Poisson process with mean rate λ) it joins the split queue. Whenever all servers are idle and the split queue is not empty, a task is taken from the head of the split queue and is injected into the system, splitting into n subtasks at the split point. Each subtask enters the queue of its corresponding parallel server (where it is served according to a service time distribution with mean $1/\mu_i,\ i = 1, \ldots, n$). After service, a subtask enters a merge buffer. Only when all subtasks (of a particular task) are present in the merge buffers does the original task instantaneously exit the system via the merge point.

Since waiting tasks queue in the split queue while the service nodes do not have queueing capability, and only one task is processed by the servers at a time. We thus note that a split-merge queueing system might be conceptually treated as an *M/G/1* queue with service time set to be the maximum of the parallel service times.

2.6.2 Fork-Join Queueing System

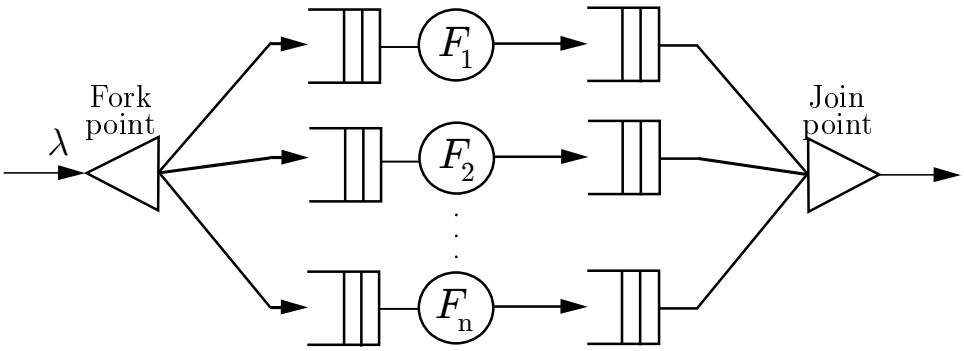

Figure 2.3: Fork–Join queueing model.

An *elementary fork-join system* is an asynchronous type of parallel queueing system (see Fig. 2.3). It is composed of n parallel heterogeneous FCFS service queues, fork and join points and join queues (join buffers) for completed subtasks [21]. When a task arrives in the system (usually assumed to happen according to a Poisson process with mean rate λ) it instantaneously enters the fork point, where it forks into n independent subtasks. Each subtask enters the queue of its corresponding parallel server. Parallel server i processes its queue of subtasks according to a heterogeneous general service time distribution with mean service time $1/\mu_i$, $i = 1, \ldots, n$. After service, a subtask enters a join queue. Only when all subtasks (of a particular task) are present in the join queues does the original task instantaneously exit the system via the join point.

Join and Merge Buffers

We note that in many real-life applications the join/merge buffers are managed as a single shared physical space set aside for the storage of partially completed subtasks. In such cases we term this space the *output buffer*. Careful management of the arrival times of subtasks into the output buffer is vital especially in circumstances where it occupies limited physical space and/or where it is highly utilised. One way to achieve this is to maintain low levels of subtask dispersion is to attempt to cluster the arrival times of the subtasks in the output buffer as close as possible.

2.6.3 Performance Metrics for Parallel Queueing Systems

In this dissertation we are concerned with four metrics related to the quality of service experienced by tasks and subtasks in parallel queueing systems. Generally, optimising performance of a queueing system means optimising a particular performance metric where one metric may have a priority upon others [51]. There are four performance metrics that are have been under attention for optimisation. Any of the last three of these metrics can be computed from two of the others by utilising Little's law.

Subtask Dispersion

We define *subtask dispersion* as the difference in time between the service completion of the first and last subtask which originated from the same task. Equivalently, this is the difference in time between the arrival of the first and last subtasks (of a given task) in the output buffer.

Task Response Time

Task response time is the time difference between to a task entering the system until it finishes its service and exits the system. Task response time is usually a sum of waiting time and service time.

System Throughput

System throughput is defined as the rate (tasks per unit time) at which the tasks can be served by the system. Usually, throughput of an unsaturated system increases as a workload of the system rises [51]. Since subtasks in a fork-join queueing system are subject to less synchronisation than those in a split-merge queueing system, the structure of a fork-join system naturally yields higher system throughput when compared to a split-merge queueing system with the same system configuration and parametrisation, such as the same service time distributions and service rates as well as arrival distribution and arrival rate.

Output Buffer Length

An output buffer queue length refers to the sum of subtasks present in each of the join or merge queues in fork-join queueing systems or merge queue in split-merge queueing systems. Keeping this metric under control is very important in a highly utilised queueing systems.

2.7 Theory of Order Statistics

In this section we recap the essentials of the theory of homogeneous order statistics which we are going to extend later to heterogeneous order statistics. The theory of order statistics studies the behaviour and properties of arranged random variables and the statistics derived from them [28]. Recently it has become more applied in many real world areas, such as auction theory [59], floods and droughts [91], fields of quality control in a problems of fatigue failure and breaking strength [30].

Definition 2.1. *Let the increasing sequence* $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$ *be a permutation of the real valued random variables* $X_1, X_2, \ldots, X_n$*, i.e. the* X_i *arranged in ascending order*

$$X_{(1)} \leqslant X_{(2)} \leqslant \ldots \leqslant X_{(n)}$$

Then $X_{(i)}$ *is called the* ith order statistic*, for* $i = 1, 2, \ldots, n$.

The random variables X_i are typically assumed to be identically and independently distributed (iid) with cumulative distribution function $F(t)$, but of course $X_{(i)}$ are dependent because of the ordering. The first and last order statistics, $X_{(1)}$ and $X_{(n)}$, are the minimum and maximum respectively, which are also called the *extremes*. $D = X_{(n)} - X_{(1)}$ is the *range*.

Distribution of the rth-Order Statistic (iid case)

If $X_1, X_2, \ldots, X_n$ are n independent random variables, the cumulative distribution function (cdf) of the maximum order statistic (the maximum) is given by:

$$F_n(x) = Pr\{X_{(n)} \leqslant x\} = Pr\{X_i \leqslant x, 1 \leqslant i \leqslant n\} = F^n(x)$$

Likewise, the cdf of the minimum statistic is:

$$F_1(x) = Pr\{X_{(1)} \leqslant x\} = 1 - Pr\{X_{(1)} > x\} =$$
$$1 - Pr\{X_i > x, 1 \leqslant i \leqslant n\} = 1 - [1 - F(x)]^n$$

These are special cases of the general cdf of the rth order statistic, $F_r(x)$, which can be expressed as:

$$\begin{aligned} F_r(x) = Pr\{X_{(r)} \leqslant x\} &= Pr\{\text{at least } r \text{ of the } X_i \leqslant x\} \\ &= \sum_{i=r}^{n} \binom{n}{i} F(x)^i [1 - F(x)]^{n-i} \end{aligned} \tag{2.7}$$

The pdf of X_r, $f_r(x) = F_r'(x)$, where the prime denotes the derivative with respect to x, when it exists, is then:

$$f_r(x) = \frac{n!}{(r-1)!1!(n-r)!} F^{r-1}(x) f(x) [1 - F(x)]^{n-r} \text{ [28].}$$

Multiplying both sides by "small" ϵ, this result follows intuitively from noting that we require one of the X_i to take a value in the interval $(x, x + \epsilon]$, exactly $r - 1$ of the X_i to be less than or equal to x and exactly $n - r$ of them to be greater than x. The coefficient $n!/((r-1)!1!(n-r)!)$ is the number of ways of doing this, given that the X_i are stochastically indistinguishable.

The joint density function of the rth and sth order statistics $X_{(r)}, X_{(s)}$, where $(1 \leqslant r < s \leqslant n)$, is:

$$f_{rs}(x, y) = S_{rs} F^{r-1}(x) f(x) [F(y) - F(x)]^{s-r-1} f(y) [1 - F(y)]^{n-s} \tag{2.8}$$

where $S_{rs} = \frac{n!}{(r-1)!(s-r-1)!(n-s)!}$, by similar reasoning. The corresponding joint cdf $F_{rs}(x, y)$ of $X_{(r)}$ and $X_{(s)}$ may be obtained by integration of the pdf or, alternatively, for $x < y$ we have:

$$\begin{aligned} F_{rs}(x,y) &= Pr\{\text{at least } r \text{ of the } X_i \leqslant x, \text{ at most } n-s \text{ of the } X_i > y\} \\ &= \sum_{j=s}^{n} \sum_{i=r}^{j} Pr\{\text{exactly } i \text{ of the } X_i \leqslant x, \text{ exactly } n-j \text{ of the } X_i > y\} \\ &= \sum_{j=s}^{n} \sum_{i=r}^{j} \frac{n!}{i!(j-i)!(n-j)!} F^i(x)[F(y)-F(x)]^{j-i}[1-F(y)]^{n-j} \end{aligned}$$

Finally, the joint pdf for the k order statistics $X_{(n_1)}, \ldots, X_{(n_k)}$, $1 \leqslant n_1 < \ldots < n_k \leqslant n)$, is, similarly, for $x_1 \leqslant \ldots \leqslant x_k$:

$$\begin{aligned} f_{n_1,\ldots,n_k}(x_1,\ldots,x_k) = S_{n_1,\ldots,n_k} F^{n_1-1}(x_1) f(x_1)[F(x_2)-F(x_1)]^{n_2-n_1-1} f(x_2) \cdots \\ [F(x_k)-F(x_{k-1})]^{n_k-n_{k-1}-1} f(x_k)[1-F(x_k)]^{n-n_k} \end{aligned}$$

where $S_{n_1,\ldots,n_k} = \frac{n!}{(n_1-1)!(n_2-n_1-1)!\ldots(n_k-n_{k-1}-1)!(n-n_k)!}$.

Distribution of the Range

The pdf $f_{D_{rs}}(x)$ of the interval $D_{rs} = X_{(s)} - X_{(r)}$ follows from the joint pdf of the rth and sth order statistics in Eq. 2.8 by setting $y = x + t_{rs}$ and integrating over x, giving:

$$f_{D_{rs}}(t_{rs}) = S_{rs} \int_{-\infty}^{\infty} F^{r-1}(x) f(x)[F(x+t_{rs})-F(x)]^{s-r-1} f(x+t_{rs})[1-F(x+t_{rs})]^{n-s} dx$$

In the special case when $r = 1$ and $s = n$, D_{rs} is the range $D = X_{(n)} - X_{(1)}$ and the pdf simplifies to:

$$f_D(t) = n(n-1) \int_{-\infty}^{\infty} f(x)[F(x+t)-F(x)]^{n-2} f(x+t) dx$$

The cdf of D then follows by integrating inside the integral with respect to x, giving:

$$\begin{aligned} F_D(t) &= n\int_{-\infty}^{\infty} f(x)\int_0^t (n-1)f(x+t')[F(x+t')-F(x)]^{n-2}dt'dx \\ &= n\int_{-\infty}^{\infty} f(x)\big[[F(x+t')-F(x)]^{n-1}\big]_{t'=0}^{t'=t}dx \\ &= n\int_{-\infty}^{\infty} f(x)[F(x+t)-F(x)]^{n-1}dx \end{aligned} \tag{2.9}$$

As noted in [28], this equation follows intuitively by noting that the integrand (multiplied by an infinitesimal quantity dx) is the probability that X_i falls into the interval $(x, x+dx]$ (for some i) and the remaining $n-1$ of the $X_j, j \neq i$ fall into $(x, x+t]$. There are n ways of choosing i, giving the factor n.

2.8 Numerical Optimisation Algorithms

Area of numerical optimisation algorithms is a study of optimisation algorithms which make use of numerical approximations, that is a non-symbolic analysis. These kind of algorithms aim to obtain fast approximate solutions while keeping a tight error bounds.

Since in this thesis we utilise only numerical algorithms to minimise objective functions, therefore we recall only this type of algorithms. Assume we have an objective function $f(\mathbf{x})$ which converges to its minimum at some point $\mathbf{x}_{\min}$:

$$\mathbf{x}_{\min} = \arg\min_{\mathbf{x}} f(\mathbf{x}) \tag{2.10}$$

The aim of a numerical optimisation algorithm is to find a $\mathbf{x}_{\min}$.

Desirable computational properties in such procedures include: high speed, utilising marginal amount of memory and minimising the number of $f(\mathbf{x})$ evaluations [83].

2.8.1 Newton's Method

Newton's method is a gradient based iterative method of finding local minimum/maximum of an objective function $f(\mathbf{x})$, provided that it is a continuous and twice-differentiable [79]. Usually Newton's method is used rather than the well-known gradient descent method, because the former requires fewer steps to converge to a minimum due to its utilisation of a Hessian matrix, which describes the local curvature of a function. Each step in Newton's method is more computationally expensive than a step in gradient descent but Newton's method typically needs fewer steps to find the minimum.

The method calculates the gradient $\nabla f(\mathbf{x}_k)$ as a direction from the current point $\mathbf{x}_k$ and utilises curvature information to take a more direct course:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma[H_{f(\mathbf{x}_k)}]^{-1}\nabla f(\mathbf{x}_k) \tag{2.11}$$

where $0 < \gamma \leq 1$ is a constant introduced to satisfy Wolfe's conditions. The step size γ needs to be chosen to be small enough to support convergence, yet large enough to make rapid progress towards the minimum, these conditions we discuss in the next section. $H_{f(\mathbf{x_k})}$ is a Hessian matrix of a second order partial derivatives:

$$H = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \quad \forall i, j.$$

Each iteration of the algorithm requires calculations of the gradient $\nabla f(\mathbf{x}_k)$ and the Hessian matrix $H_{f(\mathbf{x}_k)}$ at a point $\mathbf{x}_k$. We explore the surface of the function with applied current vector $\mathbf{x}_k$ and compare $f(\mathbf{x}_k)$ against previous value $f(\mathbf{x}_{k-1})$, the calculations continue while the current value of function is greater than previous one ($f(\mathbf{x}_k) > f(\mathbf{x}_{k-1})$), than the previous vector $\mathbf{x}_{k-1} = \mathbf{x}_{\min}$ is the desired vector.

2.8.2 Wolfe Conditions

We discuss the Wolfe's conditions which can be applied in iterative optimisation methods where a search direction towards the minimum of the objective function is based on a gradient descent. Previously, in Newton's method Eq. 2.11 we utilised $0 < \gamma \leq 1$, which is a constant introduced to satisfy Wolfe conditions [108, 109]. In this section we look how to rigorously compute step length γ to satisfy a trade-off between providing substantial reduction of $f(\mathbf{x})$, but at the same time make a rapid progress [79].

A descent direction for Newton's method from Eq. 2.11 is:

$$\mathbf{p}_k = -H^{-1}_{f(\mathbf{x}_k)} \nabla f(\mathbf{x}_k)$$

There are two inequalities namely the Wolfe conditions which must hold to ensure correct step size. First condition stipulates that γ has *sufficient decrease* in the objective function $f(\mathbf{x})$:

$$\begin{aligned} f\Big(\mathbf{x}_k + \gamma\big(-H^{-1}_{f(\mathbf{x}_k)} \nabla f(\mathbf{x}_k)\big)\Big) \leq \\ f(\mathbf{x}_k) + c_1 \gamma \nabla f^T(\mathbf{x}_k)\big(-H^{-1}_{f(\mathbf{x}_k)} \nabla f(\mathbf{x}_k)\big) \end{aligned} \tag{2.12}$$

where $c_1 \in (0, 1)$ is a constant. The inequality means that the reduction of $f(\mathbf{x})$ should be proportional to both the step size γ and the directional derivative $\nabla f^T(\mathbf{x}_k)\mathbf{p}_k$ [79]. Eq. (2.12) above corresponds to the Armijo rule [5].

The sufficient decrease condition is not enough to make sure that the optimisation algorithm makes fast progress, as it satisfies for all sufficiently small values of γ. In order to reject all unacceptable short steps the second condition, namely *curvature condition* should be satisfied:

$$\begin{aligned} \nabla f^T\big(\mathbf{x}_k + \gamma(-H^{-1}_{f(\mathbf{x}_k)} \nabla f(\mathbf{x}_k))\big)\big(-H^{-1}_{f(\mathbf{x}_k)} \nabla f(\mathbf{x}_k)\big) \geq \\ c_2 \nabla f^T(\mathbf{x}_k)\big(-H^{-1}_{f(\mathbf{x}_k)} \nabla f(\mathbf{x}_k)\big) \end{aligned} \tag{2.13}$$

where c_2 are constants, which should be chosen such that $0 < c_1 < c_2 < 1$ and $c_1 \ll c_2$. Practically, for the purposes of Newton method it is recommended to set c_1 as 10^{-4} and for c_2

to take on 0.9 [79].

The former inequality means that if we look at the left-hand side as $\frac{\partial f(\mathbf{x}_k-\gamma\mathbf{p}_k)}{\partial\gamma}$, than the curvature condition ensures that the slope of f at γ is greater than $c_2\frac{\partial f(\mathbf{x}_k-\gamma\mathbf{p}_k)}{\partial\gamma}|_{\gamma=0}$. This condition works because a strongly negative left-hand-side illustrates that f can be significantly reduced by moving further in the chosen direction. But if left-hand side is just marginally negative or positive, than it means that there is no more decrease in further direction of the optimisation method [80].

2.8.3 Nelder-Mead Method

Nelder-Mead method is an iterative simplex-reflection technique for minimising an objective function $f(\mathbf{x})$ where its derivatives may not be known, therefore it is gradient-free optimisation algorithm. The method was created in 1965 by John Nelder and Roger Mead [73], the main idea is in n dimension problem of minimising $f(\mathbf{x})$ to keep track of $n+1$ points which form a simplex. Assume a simplex S is given, where vertices are

$$S = \{s_1, s_2, ..., s_{n+1}\}.$$

From the vector of vertices associated matrix formed

$$V(S) = [s_2 - s_1, s_3 - s_1, \ldots, s_{n+1} - s_1].$$

In each iteration for the set of current vertices $\{x_1, x_2, \ldots, x_{n+1}\}$, which are in order such that $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})$, the algorithm replaces the vertex x_{n+1} which gives the highest value of the objective function with a new vertex, that provides best value of the function. The new vertex is computed by reflecting, contracting or expanding the simplex along the line joining the worst point with the centroid point $(\bar{x})$ of remaining points [78, 10].

2.9 Related Work in Parallel Queueing Systems

Parallel processing systems have been in scientific interest for many decades. Usually research on fork-join and split-merge queueing systems is concerned with approximations of mean or lower/upper bounds on task response time. The analysis of fork-join queueing systems is hard even under Markovian assumptions because firstly, a fork-join queueing system is a non-product form since arrival process is correlated therefore the systems can not be decomposed. Secondly, because of the asynchronous properties of fork-join queueing systems, its difficult to analyse it analytically, except very simple cases where analytical solutions were found [36, 37]. The main difference between researches is specifications of the queueing systems which researches were based on. Fork-join queueing systems may be classified by different number of sources (single or multiple, although most fork-join queueing systems have one source of arrival tasks), open or closed systems, considering only fork or only join points, multi-class systems etc.

By contrast, the focus of the present dissertation is not response time computation; rather it concerns ways to control the dispersion of subtask completion time which affects a split/join queue length in parallel queueing systems such as split-merge and fork-join queueing systems.

2.9.1 Performance Analysis of Parallel Systems

The importance of performance prediction in *split-merge* queueing systems and their less synchronised – but analytically much less tractable – counterparts, *fork-join* queueing systems, has been long appreciated by performance modellers. Understandably, for both kinds of model, the primary focus of research work to date has been on the computation of the stationary distribution of the number of subtasks queued at each parallel server and on moments of task response time, most especially the mean.

Related Literature Review on Subtask Dispersion

There are only a few items in the literature which consider the issue of subtask dispersion in parallel systems; and most of this work treats the metric indirectly through an examination of the mean number of subtasks in the output buffer. We summarise the main results below.

Balsamo and Mura [16] consider fork-join queueing networks with Poisson arrivals and general service time distributions represented as a Coxian distribution. They claim any general service time distribution that has rational Laplace transform can be approximated arbitrary closely using this technique. They derive two approximations of the join queue length (i.e. the number of subtasks in each join queue) distribution and closed–form expressions for all moments for this distribution with a proof of lower and upper bounds on the moments of the join queue length. The technique is based on an analysis of the Markov process underlying the fork-join queueing system. They identify a regular process structure by choosing an appropriate state ordering which allows to apply a matrix-geometric method [16].

Bolch proposed performance metrics for fork-join queueing systems that relate to join queues: firstly, *synchronisation overhead* – that is ratio of subtasks' mean waiting times in a join queues to the mean task response time, secondly *blocking factor* – that is an average number of subtasks in the join queues (join queue length) and lastly, *speed-up*, which is a ratio of mean task response time with n subtasks that serve sequentially to a mean task response time with n subtasks that are served by a parallel queueing system with a fork-join policy [20].

In the Map-Reduce community, Zaharia et al. study the scheduling of Map-Reduce tasks in clusters, particularly sharing a cluster between users while keeping data locality, that is the placement of computation near its input data in the system. Locality of data is an essential performance issue in large clusters because network bandwidth is a bottleneck [31]. They propose a *delay scheduling* algorithm, whereby some scheduled tasks wait for a small amount of time, so that other tasks can finish their service. This simple method was tested on a 600-server Hadoop cluster at Facebook. This delayed scheduling of tasks can counter-intuitively lead to a greater fairness and a higher level of data locality [111].

Related Literature Review on Task Response Time

In this section we make attempt to recall some of major research on task response time in chronological order.

Heidelberger and Trivedi developed two highly accurate approximation methods for mean queue length and mean response time prediction in closed parallel queueing systems based on M/M/1-queues where primary tasks fork into two or more secondary subtasks [50]. Flatto et al. derived exact analytical solutions for the stationary distribution of the number of subtasks in each queue in a two-node fork-join queueing systems with exponential task arrivals and heterogeneous exponential service time distributions [36, 37]. For fork-join systems with homogeneous exponential service time distributions, Nelson and Tantawi describe a technique which yields approximate lower and upper bounds on mean task response time as a function of the number of servers [75]. Nelson et al. consider fork-join queueing networks with Poisson arrival bulks of tasks, exponential service time distributions, they found analytical expression for mean task response time in [76]. Kim and Agrawala derive an approach which approximates mean task response time and state-occupancy probabilities in multi-server fork-join systems with Erlang service time distributions [55]. Baccelli et al. derive bounds on various transient and steady-state performance measures for (predominantly homogeneous) fork-join queueing systems by stochastically comparing a given system with constructed queueing systems with simpler structure but identical stability characteristics [13, 14].

Towsley et al. develop mathematical models for mean task response time of fork-join parallel programs executing on a shared memory multiprocessor under Poisson tasks arrivals and two different scheduling policies [94]. Under the *task scheduling* policy, the authors derive lower and upper bounds for mean task response time, while under the *job scheduling* policy, standard birth-death theory leads to an exact expression for mean task response time. Liu and Perros in [70] consider a closed fork-join queueing system where they use decomposition algorithm for approximation of mean task response time and system throughput. Furthermore, the result converges to exact in case of 3 servers system. Varma and Makowski use interpolation between light and heavy traffic modes to approximate the mean response time for a homogeneous fork-

join system of M/M/1 queues with Erlang and hyper-exponential arrival distributions [105]. Balsamo et al. consider fork-join queueing network. They found task response distributions which are represented by homogeneous irreducible Markov processes with a Quasi-Birth-Death structure [15]. Varki considers closed fork-join queueing network with Poisson arrivals exponential service time distributions where she approximates mean task response time, queue length and throughput of the system. The method which is based on the mean value equation for fork-join queueing networks for response time which computes lower performance bounds [103].

Fork-join queueing system based on homogeneous M/M/1 queues with Poisson arrivals was considered in [62], where a maximum order statistic provides an easily-computable upper bound on response time. Harrison and Zertal present an approximation for moments of the maximum of response times in a split-merge queueing system with Poisson task arrivals and general heterogeneous subtask service times [49]; this gives an exact result in the case of exponential subtask service time distributions. Varki et al. present bounds on mean response time in a fork-join queueing system with Poisson arrivals and exponential subtask service time distributions and variable number of subtasks that fork from a task [104]. Lebrecht et al consider fork-join queueing networks with general service time distributions and bulk arrivals for applications in zoned disks in RAID. They found response time distribution for a randomly placed request in bulk arrival [63]. Another recent approach by Sun and Peterson presented in the context of parallel program execution time analysis – but with ready application to the analysis of split-merge systems – approximates the expectation of the maximum value of a set of random variables drawn from certain distribution classes by solving for the domain value at which the inverse cdf of the maximum order statistic is equal to a constant (0.570376002) [92].

Chapter 3

Reducing Subtask Dispersion in Split-Merge Systems

In this chapter, we consider an elementary split-merge queueing system with assumptions of Poisson task arrivals, heterogeneous general service time distributions and non-preemptive subtask service as presented in Fig. 2.2. We aim to reduce subtask dispersion, which is defined as the difference in time of completion between the first and the last subtasks from an original task without structural reconfiguration or reparametrisation of the system. We do this firstly, by extending the theory of heterogeneous order statistics to desire the mean and distribution of the range of heterogeneous order statistics.

We describe subtask dispersion as a range of heterogeneous order statistics based on random variables that derived from service time distributions of parallel nodes. Secondly, we introduce delays to the processing of subtasks before their service – this has a side effect of reducing merge buffer utilisation in case of judiciously-chosen delays. We update our results for the mean of the range of subtask dispersion and distribution of the range of subtask dispersion by introducing delay vector into them. These formulae form objective functions for minimising mean subtask dispersion [97] and reducing a given percentile of the distribution of subtask dispersion [99, 58]. Thirdly, we apply these objective functions into optimisation schemes which calculate a vector of optimal delays. Each element i of the vector is applied to the processing of subtasks before

they start service at parallel server i. We derive an analytical solution for the mean number of subtasks in the output buffer. Furthermore, we quantify the inevitable effects of applying our methodology on task response time and maximum sustainable system throughput. Lastly, we illustrate the application of our techniques using three case studies and validate our results with a simulator, written in C++, although we note that split-merge queueing systems are also is analytically tractable.

3.1 Theory of Heterogeneous Order Statistics

In this section we extend the theory of classical homogeneous order statistics [28] to heterogeneous order statistics [29]. We give a definition of the concept theory, derive the mean of the range of heterogeneous order statistics, derive the joint density function of a pair of heterogeneous order statistics, and hence derive the distribution of the range of heterogeneous order statistics. These results will be useful in our analysis of parallel queueing systems in which the parallel servers have heterogeneous service time distributions.

Definition 3.1. *We consider n independent, real-valued random variables $X_1, \ldots, X_n$ where each X_i has an arbitrary probability distribution $F_i(x)$ and probability density function $f_i(x) = F_i'(x)$. In this case of "heterogeneous" (or independent, but not necessarily identically distributed) random variables, we call the order statistics* heterogeneous order statistics *and written as:*

$$X_{(1)}, X_{(2)}, ..., X_{(n-1)}, X_{(n)}$$

The first and last heterogeneous order statistics, $X_{(1)}$ and $X_{(n)}$, are the minimum and maximum respectively, which are also called the *extremes*. $D = X_{(n)} - X_{(1)}$ is the *range*.

Recent decades have seen increasing consideration given to heterogeneous order statistics in the literature. Key theoretical results relating to the distributions and density functions of heterogeneous order statistics are summarised in [29]. This includes the work of Sen [88], who derived bounds on the median and the tails of the distribution of heterogeneous order

statistics. Vaughan et al. [106] expressed the permanent joint and marginal density functions of *innid* variates. Guilbaud [44] derived the probability of functions of independent random variables with not necessary identical distributions as a linear combination of functions with *iid* variates. Boncelet et al. [22] dealt with linear and non-linear recurrence relations for computing distributions of order statistics in *innid* or Markov cases. Bapat and Beg [17] derived the joint pdf and cdf of order statistics of *innid* variates. Sathe and Dixit in [87] derived a recurrence relation for the joint distribution function of heterogeneous order statistics. Practical issues related to the numerical computation of the ith heterogeneous order statistic were considered in [26], with special consideration of recurrence relations among distribution functions of order statistics. Pearson et al. [81] present linear function of order statistics and its expectations, but a more straightforward approach is given in [85].

In the following sections we derive the mean and distribution of the range of heterogeneous order statistics.

3.1.1 Mean of the Range of Heterogeneous Order Statistics

We assume that a random variable X takes only non-negative values. The rth heterogeneous order statistic has the following cdf:

$$\begin{aligned} F_{(r)}(x) &= Pr\{X_{(r)} \leqslant x\} = Pr\{\text{at least } r \text{ of the } X_i \leqslant x\} \\ &= \sum_{i=r}^{n} \sum_{\{\vec{\ell}_1,\vec{\ell}_2\}\in\mathcal{P}_i} \prod_{k=1}^{i} F_{\ell_{1k}}(x) \prod_{k=1}^{n-i} [1 - F_{\ell_{2k}}(x)] \end{aligned} \tag{3.1}$$

where $\mathcal{P}_i$ is the set of all two-set partitions $\{D, E\}$ of $\{1, 2, \ldots, n\}$ with $|D| = i$ and $|E| = n-i$, and ℓ_{hk} is the kth component of the vector $\vec{\ell_h}$ for $h = 1, 2$.

Similarly to the homogeneous case, the minimum and maximum order statistics are respectively

given by:

$$F_{(1)}(x) = Pr\{X_{(1)} \leqslant x\} = 1 - Pr\{X_{(1)} > x\} =$$
$$1 - Pr\{X_i > x \mid 1 \leq i \leq n\} = 1 - \prod_{i=1}^{n}[1 - F_i(x)],$$

and

$$F_{(n)}(x) = Pr\{X_{(n)} \leqslant x\} = Pr\{X_i \leqslant x \mid 1 \leq i \leq n\} = \prod_{i=1}^{n} F_i(x).$$

The expectation of the rth heterogeneous order statistic $X_{(r)}$ can be expressed in terms of $F_{(r)}$:

$$\mathbb{E}\big[X_{(r)}\big] = \int_0^\infty 1 - F_{(r)}(x)\ dx \tag{3.2}$$

Similarly, the expectations of $X_{(1)}$ and $X_{(n)}$ can be expressed in terms of $F_{(1)}$ and $F_{(n)}$ respectively as:

$$\mathbb{E}\big[X_{(1)}\big] = \int_0^\infty 1 - F_{(1)}(x)\ dx = \int_0^\infty 1 - (1 - \prod_{i=1}^{n}[1 - F_i(x)])\ dx = \int_0^\infty \prod_{i=1}^{n}\big(1 - F_i(x)\big)\ dx \tag{3.3}$$

and

$$\mathbb{E}\big[X_{(n)}\big] = \int_0^\infty 1 - F_{(n)}(x)\ dx = \int_0^\infty 1 - \prod_{i=1}^{n} F_i(x)\ dx \tag{3.4}$$

Utilising the linearity property of the expectation operator over the difference of two random variables from Eq. 2.2, the mean of the range $D = X_{(n)} - X_{(1)}$, as presented in [97] is given by:

$$\begin{aligned}\mathbb{E}[D] &= \mathbb{E}\big[X_{(n)} - X_{(1)}\big] = \mathbb{E}\big[X_{(n)}\big] - \mathbb{E}\big[X_{(1)}\big] \\ &= \int_0^\infty 1 - F_{(n)}(x)\ dx - \int_0^\infty 1 - F_{(1)}(x)\ dx \\ &= \int_0^\infty 1 - \prod_{i=1}^{n} F_i(x)\ dx - \int_0^\infty \prod_{i=1}^{n}\big(1 - F_i(x)\big)\ dx\end{aligned} \tag{3.5}$$

This result can be applied to determine mean of the dispersion of subtask completion times in an elementary split-merge queueing system with heterogeneous service time distributions; this formula will form the basis for our numerical optimisation techniques relating to mean subtask dispersion as considered in the next section and following chapters. Practically, we use

truncated numerical integration to take advantage of the fact that these two terms that being integrated are monotonically decreasing functions.

3.1.2 Joint Density of Two Heterogeneous Order Statistics

We apply previously derived the cdf of the rth heterogeneous order statistic in Eq. 3.1. Differentiating it and simplifying yields the pdf:

$$f_{(r)}(x) = \sum_{i=r}^{n} \sum_{\{\vec{\ell}_1,\vec{\ell}_2\}\in\mathcal{P}_i} \left[\sum_{j=1}^{i} \prod_{k=1,k\neq j}^{i} F_{\ell_{1k}}(x) \prod_{k=1}^{n-i} [1-F_{\ell_{2k}}(x)] f_{\ell_{1j}}(x) - \right.$$
$$\left. \sum_{j=1}^{n-i} \prod_{k=1}^{i} F_{\ell_{1k}}(x) \prod_{k=1,k\neq j}^{n-i} [1-F_{\ell_{2k}}(x)] f_{\ell_{2j}}(x) \right]$$

$$= \sum_{i=r}^{n}\sum_{h=1}^{n} \left[\sum_{\{\vec{\ell}_1,\vec{\ell}_2\}\in\mathcal{P}_{i-1}^{h-}} \prod_{k=1}^{i-1} F_{\ell_{1k}}(x) \prod_{k=1}^{n-i} [1-F_{\ell_{2k}}(x)] f_h(x) - \right.$$
$$\left. I_{i<n} \sum_{\{\vec{\ell}_1,\vec{\ell}_2\}\in\mathcal{P}_i^{h-}} \prod_{k=1}^{i} F_{\ell_{1k}}(x) \prod_{k=1}^{n-i-1} [1-F_{\ell_{2k}}(x)] f_h(x) \right]$$

$$= \sum_{h=1}^{n} f_h(x) \left[\sum_{i=r}^{n} \sum_{\{\vec{\ell}_1,\vec{\ell}_2\}\in\mathcal{P}_{i-1}^{h-}} \prod_{k=1}^{i-1} F_{\ell_{1k}}(x) \prod_{k=1}^{n-i} [1-F_{\ell_{2k}}(x)] - \right.$$
$$\left. \sum_{i=r+1}^{n} \sum_{\{\vec{\ell}_1,\vec{\ell}_2\}\in\mathcal{P}_{i-1}^{h-}} \prod_{k=1}^{i-1} F_{\ell_{1k}}(x) \prod_{k=1}^{n-i} [1-F_{\ell_{2k}}(x)] \right]$$

$$= \sum_{h=1}^{n} f_h(x) \sum_{\{\vec{\ell}_1,\vec{\ell}_2\}\in\mathcal{P}_{r-1}^{h-}} \prod_{k=1}^{r-1} F_{\ell_{1k}}(x) \prod_{k=1}^{n-r} [1-F_{\ell_{2k}}(x)]$$

where $I_\bullet$ is the indicator function and $\mathcal{P}_i^{h-}$ is the set of all 2-set partitions of $\{1,2,\ldots,n\}\setminus\{h\}$ with i elements in the first set and $1 \leq h \leq n$. In fact this result also follows from an intuitive argument over the infinitesimal interval $(x, x+\epsilon]$, as in the homogeneous case.

The joint density function $f_{rs}(x,y)$ of two order statistics, $X_{(r)}$ and $X_{(s)}$, for $1 \leqslant r < s \leqslant n$,

follows similarly as:

$$f_{(r)(s)}(x,y) = \sum_{1\le h_1\neq h_2\le n} f_{h_1}(x)f_{h_2}(y) \sum_{\{\vec{\ell}_1,\vec{\ell}_2,\vec{\ell}_3\}\in\mathcal{P}^{h_1-,h_2-}_{r-1,s-r-1}} \prod_{k=1}^{r-1} F_{\ell_{1k}}(x) \times \prod_{k=1}^{s-r-1}[F_{\ell_{2k}}(y)-F_{\ell_{2k}}(x)] \prod_{k=1}^{n-s}[1-F_{\ell_{3k}}(y)] \tag{3.6}$$

where $\mathcal{P}^{h_1-,h_2-}_{i_1,i_2}$ is the set of all 3-set partitions of $\{1,2,\ldots,n\}\setminus\{h_1,h_2\}$ with i_1 elements in the first set, i_2 elements in the second set, and so $n-i_1-i_2-2$ in the third, and $1\le h_1\neq h_2\le n$.

3.1.3 Distribution of the Range for Heterogeneous Order Statistics

From the joint pdf of two heterogeneous order statistics in Eq. 3.6, we obtain the pdf of the interval $D_{rs}=X_{(r)}-X_{(s)}$ by setting $t_{rs}=y-x$:

$$f_{(r:s)}(t_{rs}) = \sum_{1\le h_1\neq h_2\le n}\int_{-\infty}^{\infty} f_{h_1}(x)f_{h_2}(x+t_{rs}) \sum_{\{\vec{\ell}_1,\vec{\ell}_2,\vec{\ell}_3\}\in\mathcal{P}^{h_1-,h_2-}_{r-1,s-r-1}} \prod_{k=1}^{r-1} F_{\ell_{1k}}(x) \prod_{k=1}^{s-r-1}[F_{\ell_{2k}}(x+t_{rs})-F_{\ell_{2k}}(x)] \prod_{k=1}^{n-s}[1-F_{\ell_{3k}}(x+t_{rs})]dx \tag{3.7}$$

For the range, we want the special case in which $r=1,\ s=n$ and $D=X_{(n)}-X_{(1)}$, giving the pdf:

$$\begin{aligned} f_{(1:n)}(t) &= \sum_{1\le h_1\neq h_2\le n}\int_{-\infty}^{\infty} f_{h_1}(x)f_{h_2}(x+t) \sum_{\{\vec{\ell}_1,\vec{\ell}_2,\vec{\ell}_3\}\in\mathcal{P}^{h_1-,h_2-}_{0,n-2}} \prod_{k=1}^{n-2}[F_{\ell_{2k}}(x+t)-F_{\ell_{2k}}(x)]dx \\ &= \sum_{1\le h_1\neq h_2\le n}\int_{-\infty}^{\infty} f_{h_1}(x)f_{h_2}(x+t) \prod_{k\neq h_1,h_2}[F_k(x+t)-F_k(x)]dx \end{aligned} \tag{3.8}$$

The cdf now follows by integration (inside the sum and integral with respect to x):

$$
\begin{aligned}
F_{(1:n)}(t) &= \sum_{1\le h_1\neq h_2\le n} \int_{-\infty}^{\infty} f_{h_1}(x) \int_0^t f_{h_2}(x+t') \prod_{k\neq h_1,h_2} [F_k(x+t') - F_k(x)]dx\ dt' \\
&= \sum_{1\le h_1\le n} \int_{-\infty}^{\infty} f_{h_1}(x) \prod_{k\neq h_1} [F_k(x+t) - F_k(x)]dx \qquad (3.9)
\end{aligned}
$$

Given a particular choice of $i = 1, 2, \ldots, n$, the integrand (multiplied by an infinitesimal quantity dx) is the probability that X_i falls into the interval $(x, x+dx]$ and the other $X_j, j \neq i$ fall into $(x, x+t]$. Of course there are n ways of choosing i,and so we have to sum over n terms; in the homogeneous case, all these terms are the same, which gave the factor n in Eq. 2.9. For heterogeneous order statistics, we therefore obtain:

$$
F_D(t) = F_{(1:n)}(t) = \sum_{i=1}^{n} \int_{-\infty}^{\infty} f_i(x) \prod_{j=1,j\neq i}^{n} [F_j(x+t) - F_j(x)]\, dx \qquad (3.10)
$$

This is a useful result, which requires a sum of only n terms. It can be directly applied to determine the distribution of the dispersion of subtask completion times in any split-merge system with heterogeneous service time distributions, and will form the basis for distribution-based optimisation as considered in Section 3.4.

3.2 Reducing Mean Subtask Dispersion

This section presents a technique for reducing mean subtask dispersion in elementary split-merge queueing systems. In the following we suppose random variable X_i with distribution $X_i \sim F_i(x), \forall i$ describes the service time distribution of the i^{th} parallel server. Taking into account that in an elementary split-merge queueing system all subtasks originate from the same original task injected into the system and start service at the *same instant*, we can apply the theory of heterogeneous order statistics.The heterogeneous order statistics $X_{(i)}$ then describe arranged subtask completion times across all parallel servers. Particularly the minimum heterogeneous order statistic $X_{(1)}$ denotes the completion time of the first subtask and the maximum

heterogeneous order statistic $X_{(n)}$ denotes completion time of the last subtask (corresponding to task response time) which were form from the same original task. The dispersion or range $D = X_{(n)} - X_{(1)}$ corresponds to subtask dispersion.

In the following, we use our derived result of mean of the range of heterogeneous order statistics in Eq. 3.5, which was derived utilising the linearity property of the expectation operator.

3.2.1 Incorporation of Deterministic Delays

In order to control subtask dispersion we apply judiciously-chosen deterministic delays to processing times of subtasks before they start service. We now introduce a vector of deterministic delays $\mathbf{d}$:

$$\mathbf{d} = \big(d_1, d_2, ..., d_n\big) \tag{3.11}$$

Each element d_i denotes the deterministic delay that is applied before a subtask begins its service at a parallel server i.

Now $X_i^{\mathbf{d}} \sim F_i(x - d_i)$ defines the delayed-adjusted service time distribution of parallel server i. The variates $X_1^{\mathbf{d}}, X_2^{\mathbf{d}}, \ldots, X_n^{\mathbf{d}}$ have corresponding heterogeneous order statistics $X_{(1)}^{\mathbf{d}}, X_{(2)}^{\mathbf{d}}, \ldots, X_{(n)}^{\mathbf{d}}$.

We can now express the objective function for mean subtask dispersion in an elementary split-merge queueing system from Eq. 3.5 as a function of a given delay vector $\mathbf{d}$:

$$\begin{aligned}
\mathbb{E}[D_{\mathbf{d}}] &= \mathbb{E}\big[X_{(n)}^{\mathbf{d}}\big] - \mathbb{E}\big[X_{(1)}^{\mathbf{d}}\big] \\
&= \int_0^\infty \Big(1 - \prod_{i=1}^{n} F_i(x - d_i)\Big)\, dx - \int_0^\infty \Big(1 - \big(1 - \prod_{i=1}^{n}(1 - F_i(x - d_i))\big)\Big)\, dx \\
&= \int_0^\infty 1 - \prod_{i=1}^{n} F_i(x - d_i)\; dx - \int_0^\infty \prod_{i=1}^{n} \big(1 - F_i(x - d_i)\big)\; dx
\end{aligned} \tag{3.12}$$

in Appendix A we prove the convexity of this objective function. The fact that this function is convex will guarantee the optimality of the optimisation procedure that is presented next.

3.2.2 Optimisation Procedure of Mean Subtask Dispersion

Our aim is to find that vector $\mathbf{d}$ which minimises the objective function for mean subtask dispersion given by $\mathbb{E}[D_{\mathbf{d}}]$.

That is, we aim to solve for $\mathbf{d}_{\min}$ in:

$$\mathbf{d}_{\min} = \arg\min_{\mathbf{d}} \mathbb{E}[D_{\mathbf{d}}] \tag{3.13}$$

We can explore the surface of $\mathbb{E}[D_{\mathbf{d}}]$ for a minimum utilising Newton's method from Eq. 2.11. That is we apply the iteration:

$$\mathbf{d}_{i+1} \quad = \quad \mathbf{d}_i - \left[H_{\mathbb{E}[D_{\mathbf{d}_i}]}\right]^{-1} \nabla \mathbb{E}[D_{\mathbf{d}_i}],\ i \geq 0 \tag{3.14}$$

with initial vector

$$\mathbf{d}_0 = (\max_i \mathbb{E}\big[X_i\big] - \mathbb{E}\big[X_1\big], \max_i \mathbb{E}\big[X_i\big] - \mathbb{E}\big[X_2\big], \ldots, \max_i \mathbb{E}\big[X_i\big] - \mathbb{E}\big[X_N\big])$$

Here the intuitive minimum obtained by subtracting the mean service time of each server from the maximum mean service time and is multiplied by the difference between full utilisation and current utilisation. Therefore $\mathbf{d}_0$ attempts to begin the search in the expected locality of the minimum, although we remark that Newton's algorithm is capable of finding the minimum from any initial vector for convex objective functions. We note that other plausible initial guesses are possible, e.g. median initial approximations [11]. Here we use Newton's method rather than the well-known gradient descent method, because it requires fewer steps to converge to its minimum due to utilisation of a Hessian matrix (which is a square matrix of second order partial derivatives: $H = \frac{\partial^2 \mathbb{E}[D_{\mathbf{d}}]}{\partial d_i \partial d_j}$), which defines a more direct path to the extreme point. Consequently, each step in Newton's method is more computationally expensive than a step of the gradient descent method, but Newton's method usually finds the minimum in considerably fewer steps than the gradient descent method. We found that the run time of Newton's method is typically $3-4$ times less than the run time of the gradient descent method.

We implemented this numerical optimisation procedure in C++. For each step of Newton's method our program evaluates the objective function of Eq. 3.12, where the integrals involved in the computation of the expectations of the maximum and minimum heterogeneous order statistics are evaluated numerically by the trapezoidal rule. As we explore the surface of the objective function with current delay vector $\mathbf{d}_i$, we compare $\mathbb{E}[D_{\mathbf{d}_i}]$ against its previous value $\mathbb{E}[D_{\mathbf{d}_{i-1}}]$. We cease iterating when the current value of the objective function is no better than the previous one i.e. $(\mathbb{E}[D_{\mathbf{d}_i}] - \mathbb{E}[D_{\mathbf{d}_{i-1}}] \leq \xi)$. In order to avoid invalid and unnecessary delays we apply constraints $d_i \geq 0$ for all i and $\prod_i d_i = 0$. The latter condition ensures that no delays are added to a bottleneck server(s).

3.2.3 Analytical Solution for Mean Number of Subtasks in Output Buffer

Here we derive the mean number of subtasks in output buffer in an unsaturated elementary split-merge queueing system utilising Little's Law (Eq. 2.6). According to this law, the mean number of subtasks is a product of average arrival rate of subtasks into the output buffer and mean waiting time of a subtask in the output buffer. The former term can be derived from a fact that if the mean arrival rate of tasks in the system is λ, then the mean arrival rate of subtasks in the output buffer is $n\lambda$, where n is number of subtasks that are born from a task, and $\lambda = \min(\lambda, \mu_i)\ \forall i$ in unsaturated systems. The latter term can be found from the service policy of a split-merge queueing system; that is all subtasks must wait for all their siblings to finish the service. It means that the first subtask that enters the output buffer is waiting for $X_{(n)} - X_{(1)}$ time units to exit the system, the waiting time of the second arrival is $X_{(n)} - X_{(2)}$ time units and so on. The last subtask does not wait at all – at an instant of its arrival all siblings that originated from the task are merged together and exit the system. Therefore, the

mean number of subtasks in output buffer is:

$$\mathbb{E}_{OutputBuffer} = \frac{(X_{(n)} - X_{(1)}) + ... + (X_{(n)} - X_{(n-1)})}{n} n\lambda$$
$$= \big((n-1)X_{(n)} - \sum_{i=1}^{n-1} X_{(i)}\big)\lambda \quad (3.15)$$

In practice, computing this formula involves calculations of all means of heterogeneous order statistics, which is tedious. An alternative way to compute the mean number of subtasks in output buffer, albeit inexact, is by simulation.

3.2.4 Impact of Applied Delays on System Performance

In this chapter our concern is solely on the minimising of mean subtask dispersion by introducing delays to processing of the subtasks. However, the introduction of such delays has negative implications for system stability and task response time, as quantified below [100].

Impact on System Stability

We observe first that, at a high-level, any given elementary split-merge queueing system with a set of parallel servers is conceptually equivalent to a single M/G/1 queue whose service time is equal to the maximum of the service times of the replaced parallel servers, i.e. $\max\{X_1, X_2, ..., X_n\} = X_{(n)}$. This is of course applicable for a split-merge system with delays as well, in which case we have $\max\{X_1^{\mathbf{d}}, X_2^{\mathbf{d}}, ..., X_n^{\mathbf{d}}\} = X_{(n)}^{\mathbf{d}}$. Secondly, due to the fact that distributions are monotonically increasing functions:

$$F_i(x - d_i) \leq F_i(x), \forall i \text{ (given } d_i \geq 0 \ \forall i)$$

It follows that

$$\prod_{i=1}^{n} F_i(x - d_i) \leq \prod_{i=1}^{n} F_i(x)$$

and

$$\int_0^\infty 1 - \prod_{i=1}^{n} F_i(x - d_i)\, dx \geq \int_0^\infty 1 - \prod_{i=1}^{n} F_i(x)\, dx$$

Thus from the expectation operator of random variable computed via its cdf as in Eq. 2.1 we have:

$$\mathbb{E}[X^{\mathbf{d}}_{(n)}] \geq \mathbb{E}[X_{(n)}]$$

which proves the intuition that applying *any* delay to any subtask maintains or increases the mean task service time.

Exploiting the well-known stability condition for an M/G/1 queue, and denoting the maximum arrival rate at which the system with and without delays remains stable as $\lambda^{\mathbf{d}}_{\max}$ and $\lambda_{\max}$ respectively, we have:

$$\lambda^{\mathbf{d}}_{\max} = \frac{1}{\mathbb{E}[X^{\mathbf{d}}_{(n)}]} \leq \lambda_{\max} = \frac{1}{\mathbb{E}[X_{(n)}]} \tag{3.16}$$

Thus, adding any delay to any subtask adversely impacts the maximum arrival rate at which the system remains stable.

Impact on Response Time

In order to quantify the impact of applying delays to processing of subtasks on mean task response time we use the following metric:

$$\textit{Response Penalty}_\lambda = \frac{\mathbb{E}[R_{\mathbf{d}=\mathbf{d}_{\min},\lambda}] - \mathbb{E}[R_{\mathbf{d}=\mathbf{0},\lambda}]}{\mathbb{E}[R_{\mathbf{d}=\mathbf{0},\lambda}]} * 100 \tag{3.17}$$

where $\mathbb{E}[R_{\mathbf{d}=\mathbf{d}_{min},\lambda}]$ and $\mathbb{E}[R_{\mathbf{d}=\mathbf{0},\lambda}]$ correspond for a given arrival rate λ to mean task response time with optimal delays and without any delays respectively. *Response Penalty* corresponds to the percentage by which task response time increases after application of optimal delays to the processing of subtasks.

The expected task response time in a split-merge queueing system is conceptually equivalent to the expected task response time in a single M/G/1 queue, whose service time is given by

the maximum of the service times of the parallel servers. Consequently, we can apply the Pollaczek-Khinchine formula for mean task response time in an M/G/1 queue:

$$\mathbb{E}[R_{\mathbf{d}=\mathbf{0},\lambda}] = \frac{\rho + \mu\ \lambda\ \mathrm{Var}[X_{(n)}]}{2\ (\mu - \lambda)} + \mu^{-1} \tag{3.18}$$

where λ is arrival rate, $1/\mu$ is the mean service time (with $\mu = \mathbb{E}[X_{(n)}]$), and $\mathrm{Var}[X_{(n)}]$ is the variance of service time. The latter can be calculated from Eq. 2.4 as:

$$\mathrm{Var}[X_{(n)}] = 2\int_0^\infty x\big(1 - \prod_{i=1}^{n} F_i(x)\big)\,dx - \big(\int_0^\infty 1 - \prod_{i=1}^{n} F_i(x)\,dx\big)^2$$

Straightforward modification of the above formulae yields the expected task response time with applied delays:

$$\mathbb{E}[R_{\mathbf{d}=\mathbf{d}_\alpha,\lambda}] = \frac{\rho^{\mathbf{d}} + \mu^{\mathbf{d}}\ \lambda\ \mathrm{Var}[X^{\mathbf{d}}_{(n)}]}{2\ (\mu^{\mathbf{d}} - \lambda)} + (\mu^{\mathbf{d}})^{-1} \tag{3.19}$$

where λ is arrival rate, $1/\mu^{\mathbf{d}}$ is the mean service time with applied delay vector $\mathbf{d}$ ($\mu^{\mathbf{d}} = \mathbb{E}[X^{\mathbf{d}}_{(n)}]$), and $\mathrm{Var}[X^{\mathbf{d}}_{(n)}]$ is the variance of service time with applied delays:

$$\mathrm{Var}[X^{\mathbf{d}}_{(n)}] = 2\int_0^\infty x\big(1 - \prod_{i=1}^{n} F_i(x - d_i)\big)\,dx - \big(\int_0^\infty 1 - \prod_{i=1}^{n} F_i(x - d_i)\,dx\big)^2$$

3.3 Numerical Results

3.3.1 Split-Merge Simulation

Simulations and analytical models are often used to mutually validate each other [64, 65, 107]. To this end, we have developed a simulator of parallel queueing systems written in C++ which provides rapid event-driven simulation. Input for the simulator are specified distributions for task inter-arrival time and the service times of the set of parallel nodes. The simulations allow for the computation of numerous performance-related metrics without and with subtask delays, e.g.: mean task response time, mean subtask dispersion, mean number of subtasks in output

buffer, task throughput and distribution of subtask dispersion.

As shown in Fig. 3.1, our simulator of a split-merge queueing system is based on several classes, e.g.: a **SPLIT** point with a queueing capability before it (split queue), which is linked to several parallel **NODES**, which are connected to a merge queue and followed by a **MERGE** point. **TASKS** arrive at any time instant according to a Poisson process; they go to the back of the split queue. When the system is empty and all parallel nodes are idle a task from the head of the split queue is injected into the split point, where the task splits into several (**N**) **SUBTASKS**. Each subtask goes via a link to its allocated node. The number of tasks that is injected into the system is **TRIALS**. All variables and procedures that relate to statistical calculations are stored in the **NETWORK** class. For the simulation of a split-merge queueing system with optimised subtask dispersion, the vector of optimal delays is calculated beforehand. During the simulations the optimal delays are applied to processing of subtasks at the instant of their arrival at a parallel node.

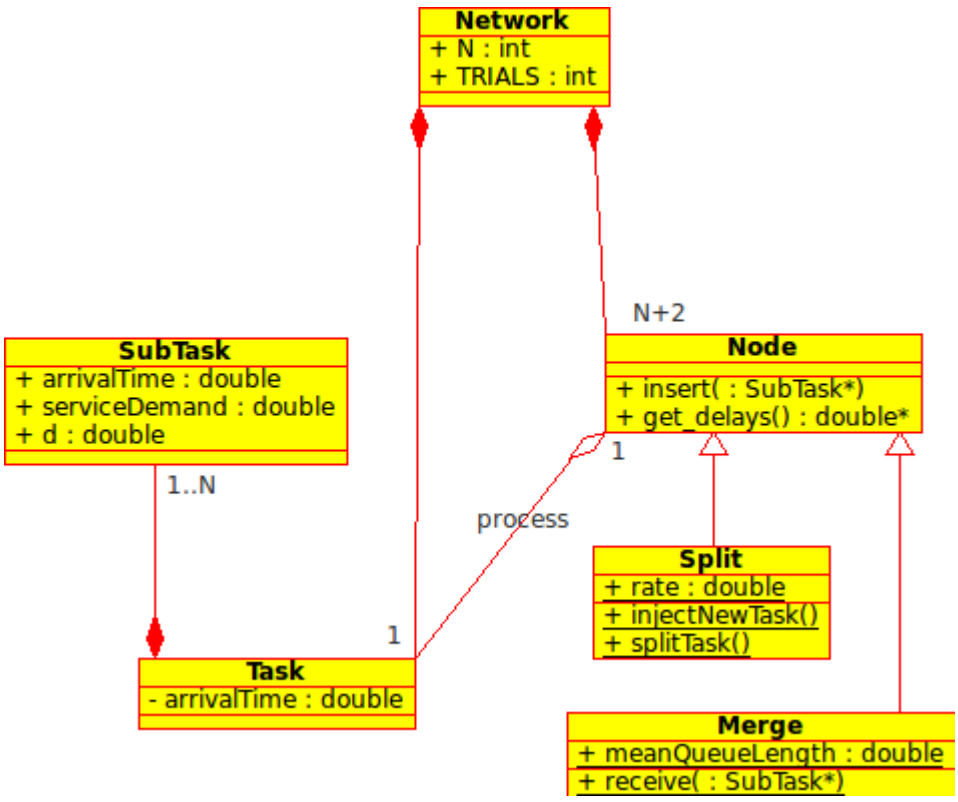

Figure 3.1: UML class diagram for simulator of an elementary split-merge queueing network.

3.3.2 Case Study

We present a case study of a split-merge queueing system, which is analysed using our method and validated by simulation. The simulations were performed on a 3.5GHz Intel Core-i5 workstation with 8GB RAM. The simulations were run by processing 10 000 000 tasks after a warm-up period of 1 000 000 tasks. Results are reported to three significant figures. For all following case studies we use the same experimental set up. We consider a split-merge queueing system with task arrival rate $\lambda = 1.0$ (tasks/time unit) and 3 parallel servers having heterogeneous service time density functions:

$$\begin{aligned} X_1 &\sim \text{Exponential}(\lambda = 5) \\ &\quad (\mathbb{E}[X_1] = 0.2,\ \text{Med}[X_1] = 0.139,\ \text{Var}[X_1] = 0.04) \\ X_2 &\sim \text{Erlang}(n = 2, \lambda = 3) \\ &\quad (\mathbb{E}[X_2] = 0.667,\ \text{Med}[\text{X}_2] = 0.559,\ \text{Var}[X_2] = 0.222) \\ X_3 &\sim \text{Uniform}(0.2, 0.5) \\ &\quad (\mathbb{E}[X_3] = 0.35,\ \text{Med}[X_3] = 0.35,\ \text{Var}[X_3] = 0.00750) \end{aligned}$$

Without adding any delays the mean subtask dispersion is $\mathbb{E}[D] = 0.576$ time units, the expected task response time is $\mathbb{E}[R_{\mathbf{d}=\mathbf{0},1.0}] = 2.026$ time units and maximum sustainable arrival rate is $\lambda_{\max} = 1.377$ tasks/time unit. The simulations of the system without delays show that the mean number of subtasks in output buffer is 0.963 (95% CI $[0.962, 0.964]$) subtasks. The means of the corresponding heterogeneous order statistics are:

$$\mathbb{E}[X_{(1)}] = 0.153,\ \mathbb{E}[X_{(2)}] = 0.340,\ \mathbb{E}[X_{(3)}] = 0.726.$$

Mean output buffer length from Eq. 3.15 is 0.962 subtasks giving confidence in the accuracy of the simulation. Applying Newton's method from Eq. 3.14 we find after 32 seconds of runtime

the vector of optimal delays which satisfies Eq. 3.13 as:

$$\mathbf{d}_{\min} = (0.393, 0.0, 0.204)$$

Fig. 3.2 presents a surface plot of mean subtask dispersion for various d_1 and d_3 values. After applying $\mathbf{d}_{\min}$ to subtask processing, mean subtask dispersion becomes $\mathbb{E}[D_{\mathbf{d}_{\min}}] = 0.448$ time units, which represents a 22% improvement. The expected task response time becomes $\mathbb{E}[R_{\mathbf{d}_{\min},1.0}] = 3.69$ time units, which is 82% higher than before the application of delays. After applying subtask delays the maximum sustainable task arrival rate drops to $\lambda^{\mathbf{d}}_{\max} = 1.178$ tasks/time unit, as shown in Fig.3.5. Simulations of the split-merge system with applied delays show that the mean number of subtasks in output buffer becomes 0.733 (95% CI $[0.732, 0.734]$) subtasks, representing 24% fewer subtasks than in the system without delays. Heterogeneous order statistics with applied delays become:

$$\mathbb{E}[X^{\mathbf{d}}_{(1)}] = 0.400,\ \mathbb{E}[X^{\mathbf{d}}_{(2)}] = 0.564,\ \mathbb{E}[X^{\mathbf{d}}_{(3)}] = 0.849.$$

Mean number of subtasks in the output buffer from Eq. 3.15 with applied delays is thus 0.734 subtasks. We present application of analytical solution for calculating mean length of a merge buffer from Eq. 3.15 only in this case study because it is analytically tedious to calculate means of all heterogeneous order statistics. Therefore for next case studies we present mean number of subtasks in the output buffer based on simulations.

It is shown in the Fig. 3.3 that at a distributional level the subtask dispersion of the split-merge queueing systems with optimised delays applied to processing of the subtasks dominates the subtask dispersion of the split-merge queueing system with no applied delays. However, Fig. 3.4 shows opposite affect on task response time.

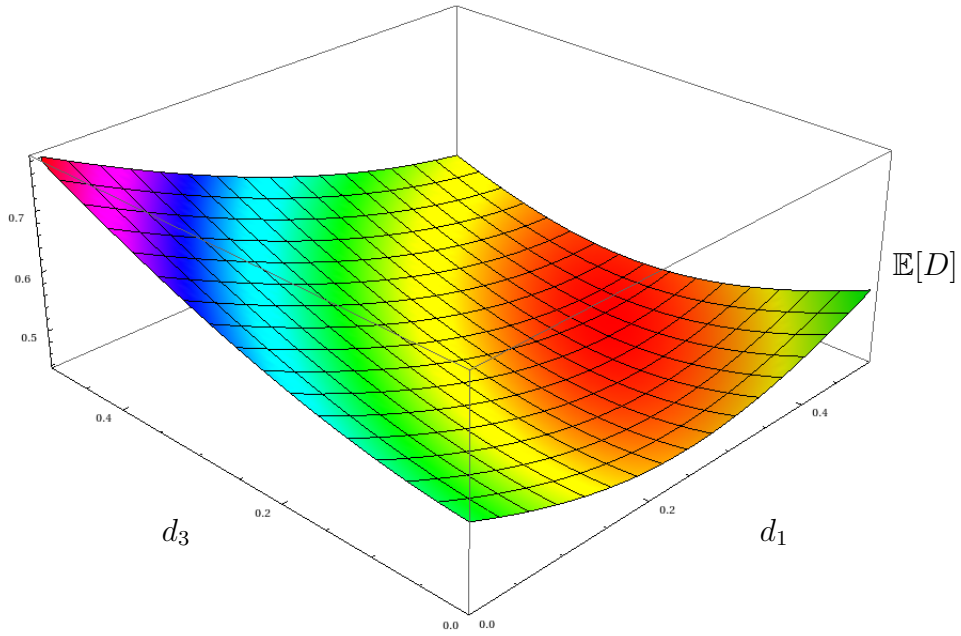

Figure 3.2: Surface plot of mean subtask dispersion against delays.

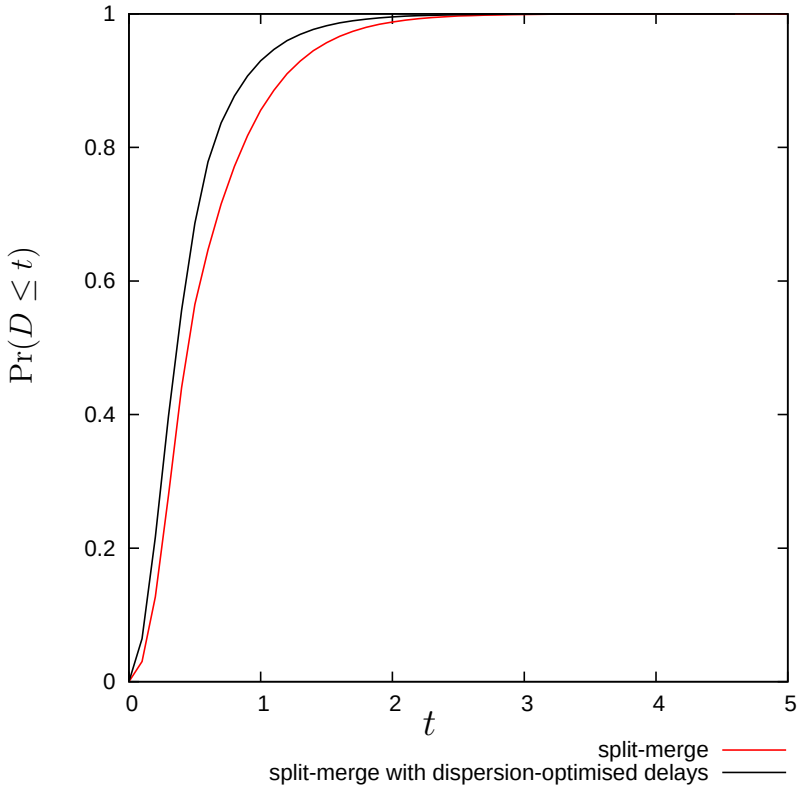

Figure 3.3: Distributions of subtask dispersion in split-merge queueing system with and without delays optimised for mean subtask dispersion ($\mathbb{E}[D_{\mathbf{d}}]$). $\lambda = 1.0$ (task/unit time).

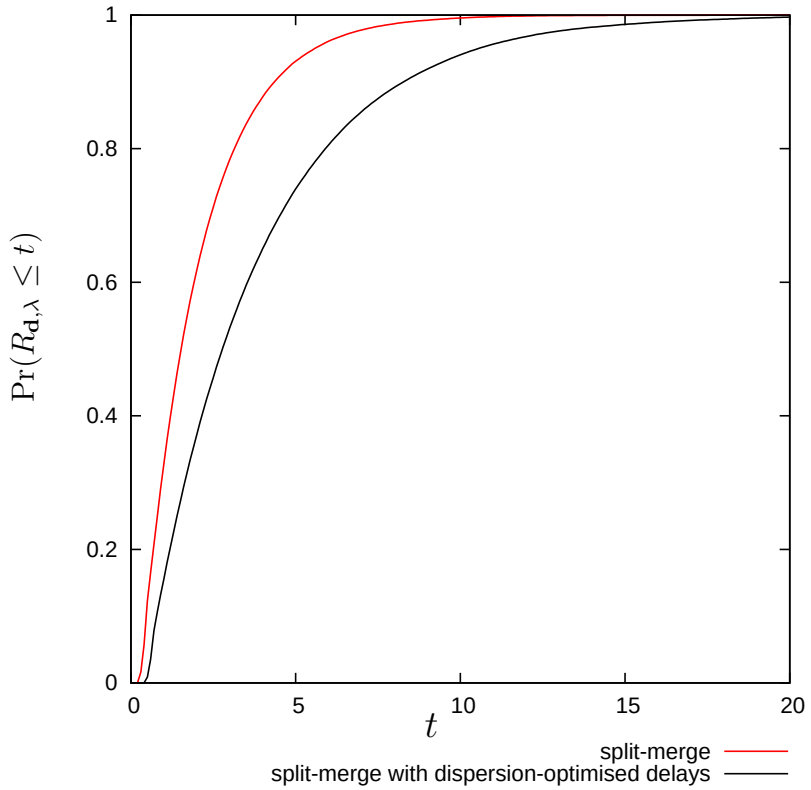

Figure 3.4: Distributions of task response time in split-merge queueing system with and without delays optimised for mean subtask dispersion ($\mathbb{E}[D_{\mathbf{d}}]$). $\lambda = 1.0$ (task/unit time).

3.4 Reducing Percentiles of the Distribution of the Range of Subtask Dispersion

In this section we move towards a more sophisticated framework for finding delay vectors which provide soft (probabilistic) guarantees on subtask dispersion. Previously we derived the distribution of the range of heterogeneous order statistics in Eq. 3.10 which equates to the range of subtask dispersion for elementary split-merge queueing systems. Here we make use of these results in a numerical optimisation procedure. In order to control a given percentile of the distribution of subtask dispersion we apply deterministic delays to processing of subtasks.

For a given probability α, $\alpha \in (0, 1]$ we aim to minimise the 100αth percentile of subtask dispersion with respect to $\mathbf{d}$. That is, we aim to solve for $\mathbf{d}_\alpha$ in:

$$\mathbf{d}_\alpha = \arg\min_{\mathbf{d}} F_D^{-1}(\alpha, \mathbf{d}) \tag{3.20}$$

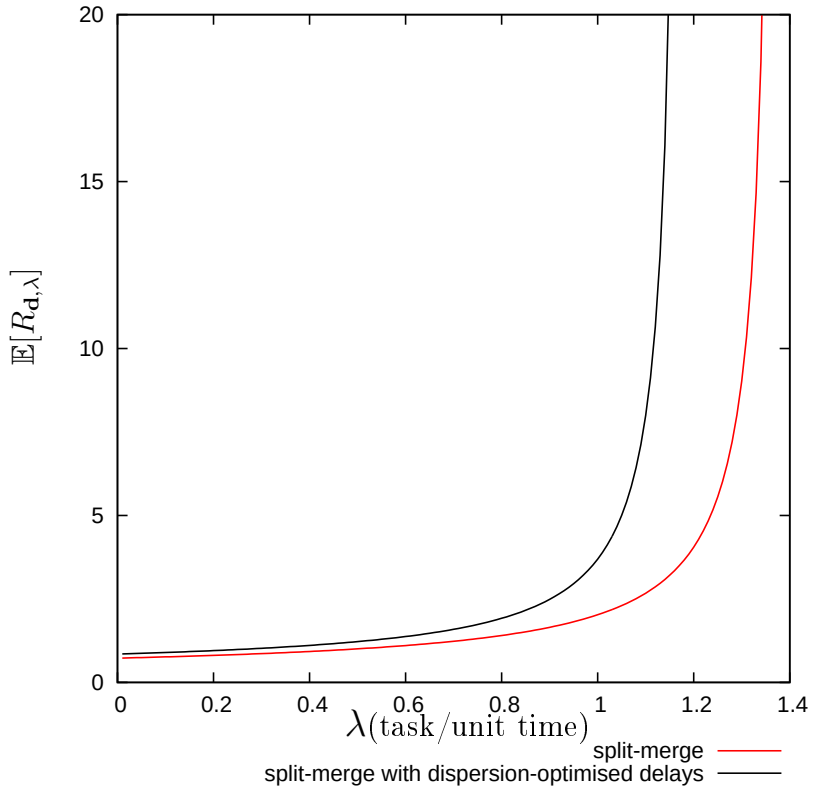

Figure 3.5: Expected response time of case study split-merge queueing system for various customer arrival rates without any delays (red line), with delays optimised for mean subtask dispersion ($\mathbb{E}[D_{\mathbf{d}}]$) (black line).

where $F_D^{-1}(\alpha, \mathbf{d})$ is the solution for t in:

$$F_D(t, \mathbf{d}) = \sum_{i=1}^{n} \int_{-\infty}^{\infty} f_i(x - d_i) \prod_{j=1, j\neq i}^{n} [F_j(x + t - d_j) - F_j(x - d_j)]\, dx = \alpha. \tag{3.21}$$

which is Eq. 3.10 updated to reflect the application of delays. Put another way, we aim to find that vector $\mathbf{d}_\alpha$ which yields the lowest value for the 100αth percentile of the difference in the completion times of the first and the last subtasks in an elementary split-merge queueing system.

3.4.1 Optimisation Procedure of the Distribution of Subtask Dispersion

Practically, we developed a full version of this numerical optimisation procedure in C++. Evaluation of Eq. 3.21 for a given α and $\mathbf{d}$ is performed by means of numerical integration using the trapezoidal rule. While this is adequate for almost all continuous service time density functions, complications arise in the case of the pdf of deterministic service time density functions because of its infinitely thin, infinitely high impulse. We resolve this by replacing the deterministic pdf with delay parameter a by the Gaussian approximation:

$$f_{\mathrm{Det}(a)}(x) \approx \frac{1}{c\sqrt{\pi}} e^{-\frac{(x-a)^2}{c^2}}$$

which becomes exact as $c \to 0$; in practice we set $c = 0.01$.

In order to invert Eq. 3.21 for a given α and $\mathbf{d}$, we first applied the Bisection method [25] in our previous work [99] because of its excellent robustness characteristics. Subsequently, we have applied Brent's method [23] which combines bisection, secant and inverse quadratic interpolation. This delivers considerably higher computational efficiency without loss of robustness.

Finally, we explore the optimisation surface of $F_D^{-1}(\alpha, \mathbf{d})$ with the initial vector $\mathbf{d}_0 = \{0, \ldots, 0\}$ using a numerical optimisation procedure. We constrain the search such that $d_i \geq 0$ for all i

and $\prod_i d_i = 0$. In our implementation, we have used a simple Nelder-Mead optimisation technique [73], which is based on the simplex method. We note that a range of more sophisticated (and correspondingly considerably more complex to implement) gradient-free optimisation techniques are also available; see e.g. [3, 68].

3.4.2 Numerical Results

Case Study A

We present a three dimensional case study, that is chosen because its easily visualisable as well as provide an intuition to the reader about the technique and its followed by higher dimensional case study to show the scalability of the algorithm.

Consider a split-merge system with task arrival rate $\lambda = 0.1$ (tasks/time unit) and 3 parallel servers having heterogeneous service time density functions:

$$\begin{aligned} X_1 &\sim \text{Pareto}(\alpha = 3, l = 3.5, h = 10) \\ &\quad (\mathbb{E}[X_1] = 4.81,\ \text{Med}[X_1] = 1.2,\ \text{Var}[X_1] = 24.96) \\ X_2 &\sim \text{Erlang}(n = 2, \lambda = 1) \\ &\quad (\mathbb{E}[X_2] = 2,\ \text{Med}[\text{X}_2] = 1.68,\ \text{Var}[X_2] = 2) \\ X_3 &\sim \text{Det}(5) \\ &\quad (\mathbb{E}[X_3] = 5,\ \text{Med}[X_3] = 5,\ \text{Var}[X_3] = 0.0) \end{aligned}$$

The choice of distributions was made to illustrate our methodology against service time distributions with different variances: Deterministic distribution has no variance, Pareto distribution has very high variance and Erlang distribution has intermediate variance.

Without adding any extra delays, it is straightforward to apply Eq. 3.10 in a root-finding algorithm (e.g. Brent's method) to compute the 50th ($\alpha = 0.5$) and 90th ($\alpha = 0.9$) percentiles of subtask dispersion as $t_{0.5} = 3.62$ and $t_{0.9} = 5.21$ time units respectively. Simulations of the split-merge queueing system show that the mean number of subtasks in output buffer is

0.4647 (95% CI [0.4645, 0.4649]) subtasks. Incorporating delays into the distribution of subtask dispersion as per Eq. 3.21, and executing a Nelder-Mead optimisation (suitably constrained so that $\prod_i d_i = 0$) to solve Eq. 3.20 given $\alpha = 0.5$ yields

$$\mathbf{d}_{0.5} = (0.533, 3.50, 0.0)$$

as shown in Fig. 3.6. The optimisation procedure's run time (using Brent's method) is 13 seconds in comparison with 46 seconds for the previous optimisation procedure where we use bisection method. We note that in this case the "bottleneck" server is server 3. With the incorporation of the optimal delays, the 50th percentile of subtask dispersion $t_{0.5} = 1.29$ time units, representing an improvement of 64% over the original system configuration without delays. The mean number of subtasks in output buffer is 0.289 (95% CI [0.288, 0.290]), a 38% reduction.

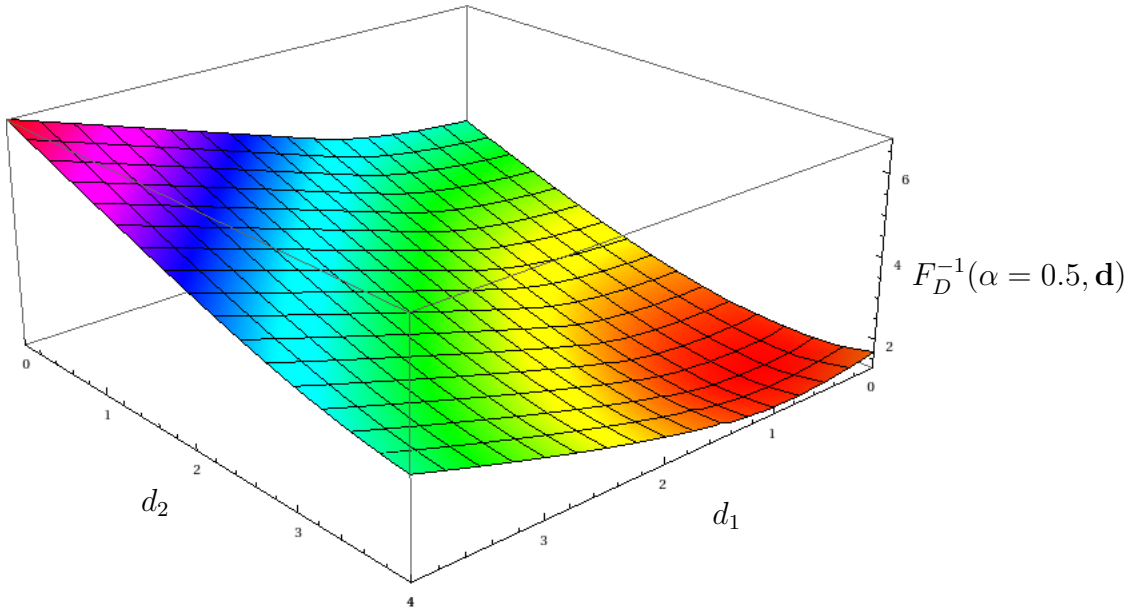

Figure 3.6: Surface plot of the 50th percentile of subtask dispersion of split-merge queueing system for various deterministic processing delays. The optimal delay vector is $\mathbf{d}_{0.5} = (0.533, 3.50, 0.0)$.

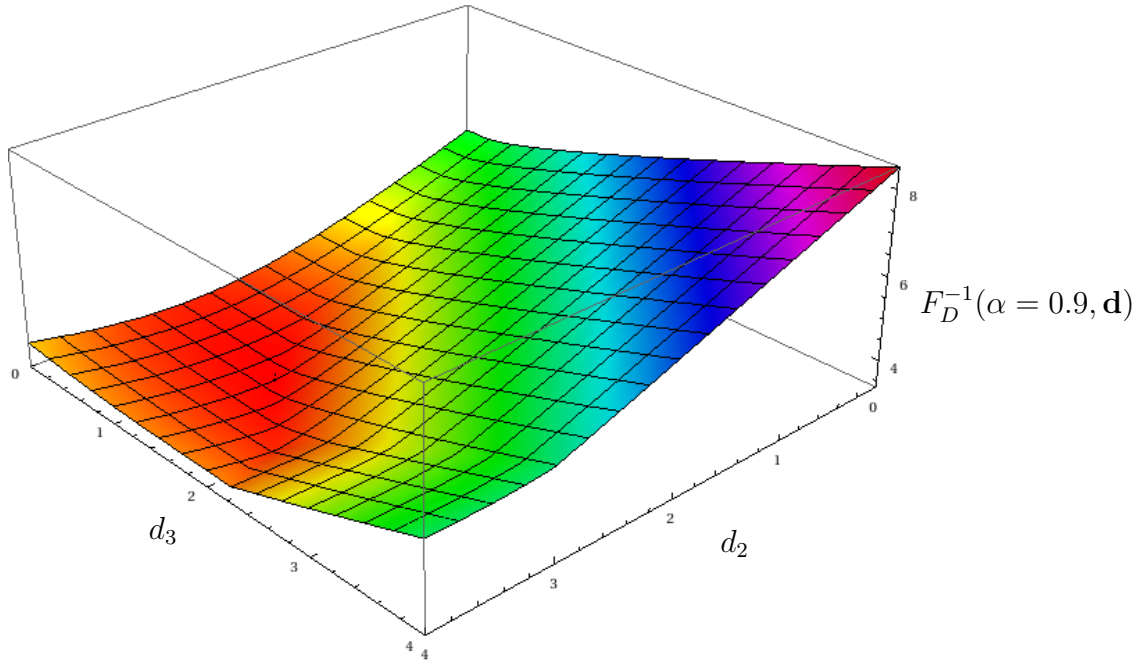

Figure 3.7: Surface plot of the 90th percentile of subtask dispersion of split-merge queueing system for various deterministic processing delays. The optimal delay vector is $\mathbf{d}_{0.9} = (0.0, 2.82, 1.16)$.

For $\alpha = 0.9$ we obtain

$$\mathbf{d}_{0.9} = (0.0, 2.82, 1.16)$$

as shown in Figure 3.7. We note that for this percentile the "bottleneck" switches from server 3 to server 1, despite the fact that server 3 has a higher mean service time and median than server 1, but server 3 does not have any variance, while server 1 has naturally very high variance, since it has a heavy-tailed service time distribution. With the incorporation of optimal delays, the 90th percentile of subtask dispersion becomes $t_{0.9} = 3.35$ time units, representing an improvement of 36% over the original system configuration without delays. The mean number of subtasks in output buffer is 0.3792 (95% CI $[0.3790, 0.3794]$) subtasks, a 18% reduction.

Figure 3.8 shows how the distribution of subtask dispersion changes according to the value of α. We note that a change of α can have a significant impact on the quantiles of $F_D(t, \mathbf{d})$, and may result in the shifting of the "bottleneck" server.

Without delays, the expected task response time is $\mathbb{E}[R_{\mathbf{d}=\mathbf{0},0.1}] = 8.93$ time units. After

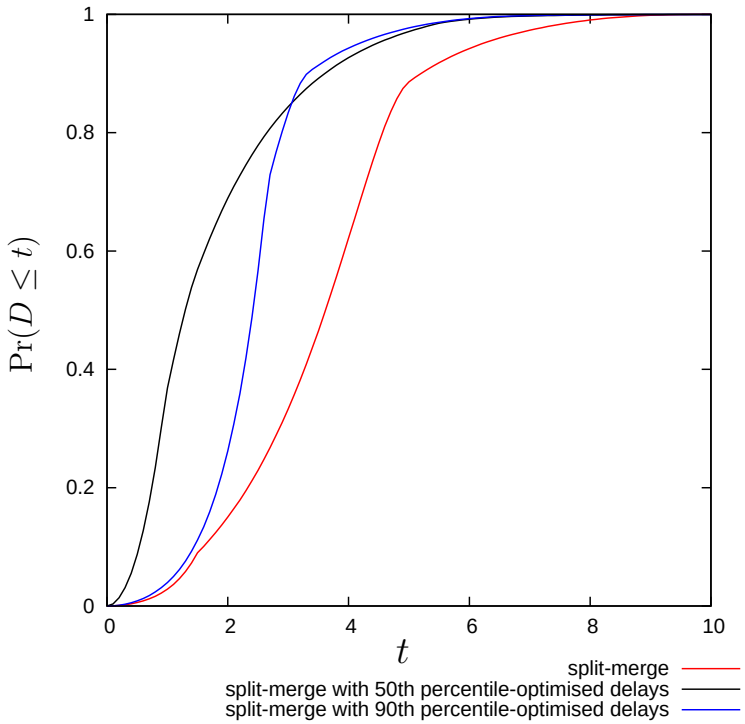

Figure 3.8: Distributions of subtask dispersion of split-merge queueing system without any delays (blue line), with delays optimised under $\alpha = 0.5$ (black line) and delays optimised under $\alpha = 0.9$ (blue line).

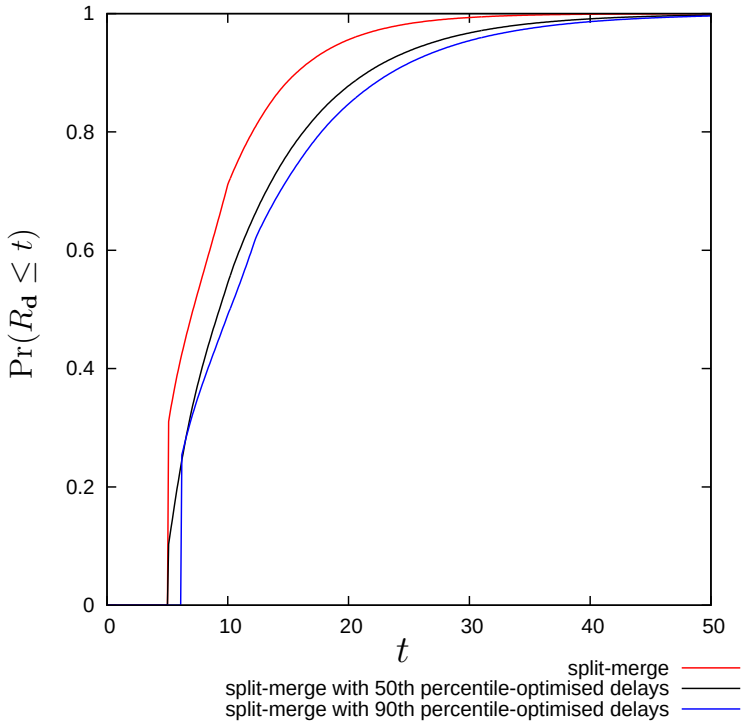

Figure 3.9: Distributions of task response time of split-merge queueing system given $\lambda = 0.1$, without any delays (red line), with delays optimised under $\alpha = 0.5$ (black line) and delays optimised under $\alpha = 0.9$ (blue line).

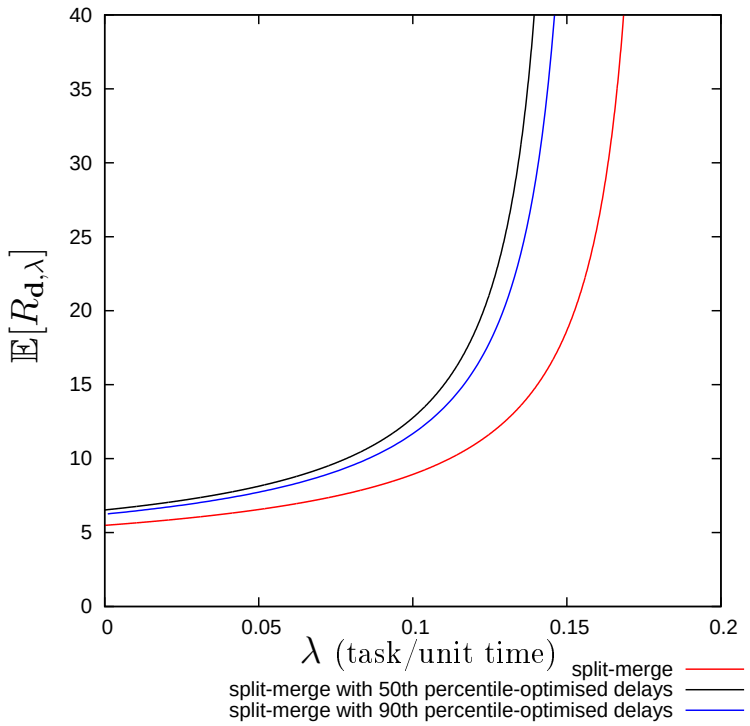

Figure 3.10: Expected response time of split-merge queueing system for various customer arrival rates without any delays (red line), with delays optimised under $\alpha = 0.5$ (green line) and delays optimised under $\alpha = 0.9$ (blue line).

introducing optimal subtask delays, the expected task response time becomes $\mathbb{E}[R_{\mathbf{d}=\mathbf{d}_{0.5},0.1}] = 11.7$ time units for $\alpha = 0.5$ and $\mathbb{E}[R_{\mathbf{d}=\mathbf{d}_{0.9},0.1}] = 12.8$ time units for $\alpha = 0.9$. The percentage increases in expected task response time from Eq. 3.17 are 31% and 43% respectively. Figure 3.9 presents the corresponding distributions of task response time. For the system without delays the maximum sustainable arrival rate from Eq. 3.16 is $\lambda_{\max} < 0.182$ tasks/time unit. After introducing subtask delays to minimise the 50th and 90th percentiles of subtask dispersion, this drops to 0.160 and 0.153 tasks/time unit respectively. This is illustrated in Figure 3.10. Although the task response time corresponds to the maximum order statistic and no delays are added to it, the task response time is affected by applying delays because the maximum order statistic is not guaranteed to be always the largest due to the randomness of the distributions. Therefore, if adding delays produces changes in the maximum order statistic then it affects the task response time.

Case Study B

We present a higher dimensional case study with eight nodes to show that our methodology scales. Consider a split-merge queueing system with arrival rate $\lambda = 0.1$ tasks/time unit and service nodes with the following heterogeneous service time distributions:

$$
\begin{aligned}
X_1 &\sim \text{Exponential}(\lambda = 1) \\
&\quad (\mathbb{E}[X_1] = 1,\ \text{Med}[X_1] = 0.693,\ \text{Var}[X_1] = 1) \\
X_2 &\sim \text{Erlang}(n = 2, \lambda = 5) \\
&\quad (\mathbb{E}[X_2] = 0.4,\ \text{Med}[\text{X}_2] = 0.336,\ \text{Var}[X_2] = 0.08) \\
X_3 &\sim \text{Det}(2) \\
&\quad (\mathbb{E}[X_3] = 2,\ \text{Med}[X_3] = 2,\ \text{Var}[X_3] = 0) \\
X_4 &\sim \text{Pareto}(n = 2.1, a = 3.5, b = 10) \\
&\quad (\mathbb{E}[X_4] = 4.89,\ \text{Med}[\text{X}_4] = 4.63,\ \text{Var}[X_4] = 28.81) \\
X_5 &\sim \text{Exponential}(\lambda = 4) \\
&\quad (\mathbb{E}[X_5] = 0.25,\ \text{Med}[\text{X}_5] = 0.173,\ \text{Var}[X_5] = 0.0625) \\
X_6 &\sim \text{Erlang}(n = 3, \lambda = 3) \\
&\quad (\mathbb{E}[X_6] = 1,\ \text{Med}[\text{X}_6] = 0.891,\ \text{Var}[X_6] = 0.333) \\
X_7 &\sim \text{Exponential}(\lambda = 8) \\
&\quad (\mathbb{E}[X_7] = 0.125,\ \text{Med}[\text{X}_7] = 0.0866,\ \text{Var}[X_7] = 0.0156) \\
X_8 &\sim \text{Pareto}(n = 2.5, a = 3.5, b = 10) \\
&\quad (\mathbb{E}[X_8] = 4.98,\ \text{Med}[\text{X}_8] = 4.49,\ \text{Var}[X_8] = 26.97)
\end{aligned}
$$

Applying Eq. 3.10 in a root-finding algorithm, we compute the 50th ($\alpha = 0.5$) and 90th ($\alpha = 0.9$) percentiles of subtask dispersion as $t_{0.5} = 5.43$ time units and $t_{0.9} = 8.26$ time units respectively. Simulations show that the mean number of subtasks in output buffer is 3.196 (95% CI $[3.194, 3.198]$) subtasks.

Applying our methodology for determining optimal subtasks delays under $\alpha = 0.5$ yields:

$$\mathbf{d}_{0.5} = (3.84, 4.13, 2.32, 0.0, 4.71, 3.55, 4.88, 0.279)$$

For this case the improved optimisation procedure (using Brent's method) takes 2 minutes 5 seconds, whereas the procedure where we apply the bisection method from [99] takes 29 minutes. We see the "bottleneck" server is server 4. With the incorporation of the optimal delays, the 50th percentile of subtask dispersion becomes $t_{0.5} = 2.02$ time units, representing an improvement of 63% over the original system configuration without delays. The mean number of subtasks in output buffer is 1.277 (95% CI $[1.276, 1.278]$) subtasks, a 60% reduction.

For $\alpha = 0.9$ we obtain:

$$\mathbf{d}_{0.9} = (4.92, 5.97, 3.16, 0.0, 5.02, 4.37, 7.04, 0.758)$$

Here we see the "bottleneck" remains server 4. After adding optimal delays, the 90th percentile of subtask dispersion becomes $t_{0.9} = 4.29$ time units, representing an improvement of 48% over the original system configuration without delays. The mean number of subtasks in output buffer is 1.357 (95% CI $[1.356, 1.357]$), a 58% reduction.

Figure 3.11 shows how the distribution of subtask dispersion changes according to the value of α. Without delays, the expected task response time is $\mathbb{E}[R_{\mathbf{d}=\mathbf{0},\lambda}] = 10.3$ time units. After introducing optimal subtask delays, the expected task response time becomes $\mathbb{E}[R_{\mathbf{d}_{0.5},\lambda}] = 12.4$ time units for $\alpha = 0.5$ and $\mathbb{E}[R_{\mathbf{d}_{0.9},\lambda}] = 19.2$ time units for $\alpha = 0.9$. The percentage increases in expected task response time are 20% and 86% respectively. Figure 3.12 presents the corresponding distributions of task response time.

For the system without delays the maximum sustainable arrival rate from Eq. 3.16 is $\lambda_{\max} < 0.171$ tasks/time unit. After introducing subtask delays to minimise 50th and 90th percentile of subtask dispersion, this drops to 0.156 and 0.133 tasks/time unit respectively. This is illustrated in Figure 3.13.

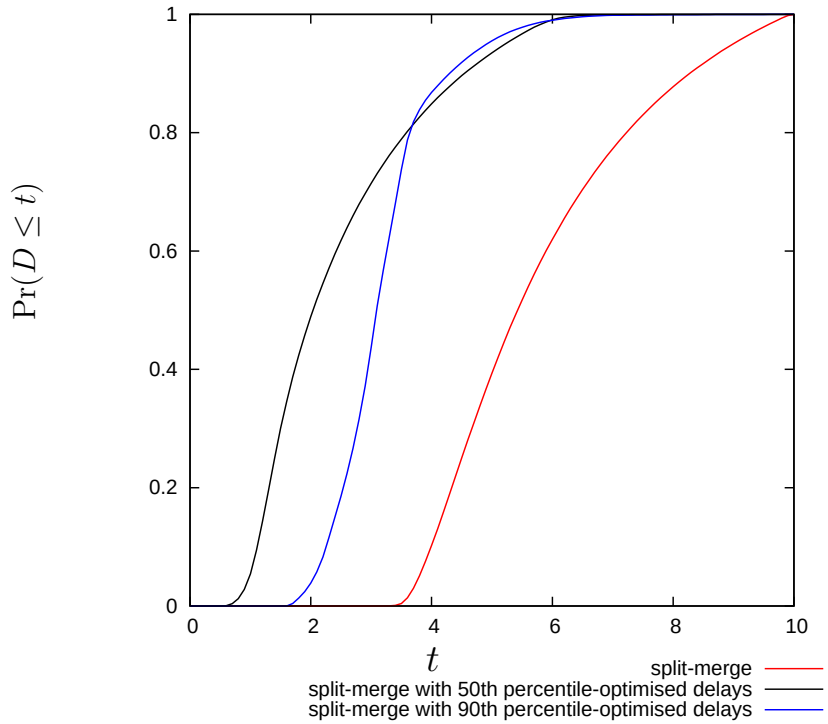

Figure 3.11: Distributions of subtask dispersion of split-merge queueing system without any delays (red line), with delays optimised under $\alpha = 0.5$ (black line) and delays optimised under $\alpha = 0.9$ (blue line).

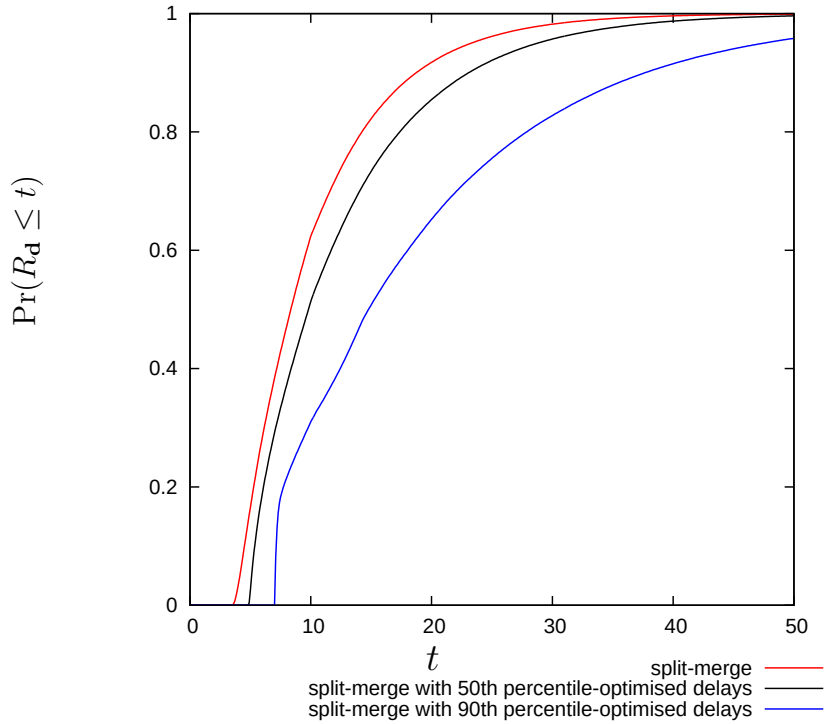

Figure 3.12: Distributions of task response time of split-merge queueing system given $\lambda = 0.1$ without any delays (red line), with delays optimised under $\alpha = 0.5$ (black line) and delays optimised under $\alpha = 0.9$ (blue line).

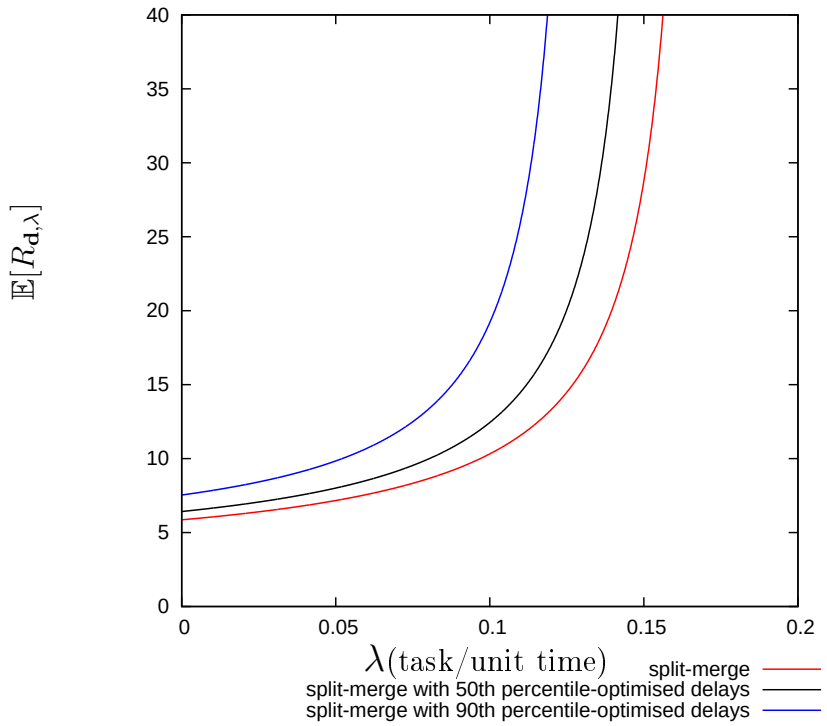

Figure 3.13: Expected response time of split-merge queueing system for various customer arrival rates without any delays (red line), with delays optimised under $\alpha = 0.5$ (green line) and delays optimised under $\alpha = 0.9$ (blue line).

3.5 Summary

This chapter has presented a methodology for minimising the mean of the range of subtask dispersion and controlling a given percentile of the distribution of subtask dispersion in elementary split-merge queueing systems. Here subtask dispersion is defined in terms of the difference between first and last arrival times of subtasks in the merge buffers, and is controlled through the application of judiciously-chosen deterministic delays to subtask service times. The methodology has four main components. The first is the extension of the classical theory of heterogeneous order statistics, which yields the mean and distribution of the range of heterogeneous order statistics. The second component is the introduction of deterministic delays into the aforementioned expressions, this yielding objective functions for the third component – numerical optimisation procedures which yield vectors of subtask delays which either minimise the mean of the range distribution of subtask dispersion or reduce a given percentile of the distribution of subtask dispersion. We presented a case study for the methodology of minimising mean of the range of subtask dispersion and two case studies for the methodology of controlling

a given percentile of the distribution of subtask dispersion which showed that the choice of percentile can have a significant impact on the optimal delay vector and the "bottleneck" server. We derived analytical solutions for mean number of subtasks in the output buffer. Lastly, in the fourth component we quantified the negative impact of adding delays into processing of subtasks on task response time and maximum sustainable throughput of the system.

In this chapter we focused solely on reduction of subtask dispersion and did not pay attention to inevitable impact of our methodologies on other performance related metrics. From the case studies we observed that the task response time and the maximum sustainable throughput increased dramatically after applying the deterministic delays for reducing subtask dispersion. One possible direction for future work is to develop an initial subtask delay vector based on medians rather than means of the service distributions, as it discussed in [11]. In the next chapter we are going to improve our methodology by considering the impact of delays on the trade-off between subtask dispersion and task response time.

Chapter 4

Trading off Subtask Dispersion and Task Response Time in Split-Merge Systems

4.1 Introduction

In this chapter we present a technique for managing a trade-off between *mean subtask dispersion* and *mean task response time*. Mean subtask dispersion is defined as the difference between the completion times of the first and last subtasks originating from a given task. Mean task response time is defined as the difference between the arrival time of a task into the split queue and the time at which the task leaves the system via the merge point. The methodology of trading off subtask dispersion and task response time is presented in [96].

In our analysis we assume elementary split-merge queueing systems (Fig. 2.2) with Poisson arrivals and heterogeneous generally distributed service times. As discussed in Section 2.9.1 and Chapter 3 previous research has been concerned about minimisation of one but not both of these measures [12, 4]. Particularly, in Chapter 3 we demonstrated how the processing of selected subtasks can be delayed in order to reduce mean subtask dispersion and percentiles of subtask dispersion for elementary split-merge queueing systems. Whilst we apply a constraint to ensure that no delay is added to the bottleneck server, the introduction of subtask delays does naturally have an adverse impact on mean task response time, with a corresponding reduction

in maximum sustainable system throughput. The adverse impact expands as workload intensity rises, and even can make a previously stable system become unstable. In an absence of the ability to reconfigure or reparameterise the system there is a conflicting tension between these two metrics – if we try to lower one metric, the other will be increased. Thus, trading off these two metrics allows effective balancing of subtask dispersion and response time. We formally characterise the subtask dispersion–response time trade-off in elementary split-merge queueing systems as an optimisation problem.

Our inspiration for this technique of trading off of two major performance metrics originates from two sources; firstly, the survey on continuous non-linear multi-objective optimisation techniques by Marler and Arora [72], which explores a variety of different optimisation techniques that may be suitable for solving multi-objective problems (e.g. weighted global criterion, weighted sum, lexicographic, weighted min/max, exponential weighted and bounded objective function). In particular they presented a weighted product method which inspires our choice of a product for our objective function.

Secondly, we were inspired by the work of Gandhi, Harchol-Balter et al. [39, 40] where they explore energy-performance trade-offs in server farms by means of an objective function based on energy-response time product (ERP). In server farms with a large number of servers, each server can be placed into one of several states, such as on/off and several sleep or standby modes. A problem occurs when a large setup time and energy penalty (setup cost) are needed to switch a server from one state to another. Thus, a trade-off arises when a potential server switch between states occurs. For example leaving an idle server on will reduce mean response time at the cost of extra energy use while turning it off will case power but adversely impacts on mean response time.

Choosing product of two performance metrics instead of weighted sum was also made because the product objective function tends to penalise poor performance on one metric more heavily than does the additive weighting sum objective function [52].

We are now going to demonstrate how our technique can reduce mean subtask dispersion while keeping mean task response time under control. The objective function of the trade-off is based

on the product of expected subtask dispersion and expected task response time. We compare the effects of our previous methodology of solely reducing mean subtask dispersion with the technique of trading-off mean subtask dispersion and mean task response time.

4.2 Application of heterogeneous order statistics to split-merge systems

As before, we consider an elementary split-merge queueing system with n parallel servers. Suppose $X_i \sim F_i(x)$ is a random variable that describes the (heterogeneous) general service time distribution of the ith parallel server. Then the heterogeneous order statistics $X_{(i)}$ correspond to ordered subtask completion times (since in a split-merge system all subtasks originating from a given task are injected into the system from split point at the same instant). The minimum heterogeneous order statistic $X_{(1)}$ corresponds to the time of first subtask completion and the maximum heterogeneous order statistic $X_{(n)}$ corresponds to the time of last subtask completion (equivalently, task response time). The dispersion or range $D = X_{(n)} - X_{(1)}$ corresponds to subtask dispersion.

As in Chapter 3, in order to provide a means to control the subtask dispersion–response time trade-off we introduce a vector of deterministic delays $\mathbf{d} = (d_1, d_2, ..., d_n)$, where element d_i represents a deterministic delay that is applied to the processing times of the subtasks that are served by the ith parallel server. The random variables that describe the parallel service times with applied delays are: $X_i^{\mathbf{d}} \sim F_i(x - d_i)\ \forall i$ with corresponding heterogeneous order statistics $X_{(1)}^{\mathbf{d}}, X_{(2)}^{\mathbf{d}}, \ldots, X_{(n)}^{\mathbf{d}}$.

4.3 An objective function for the subtask dispersion–response time trade-off

In order to express the subtask dispersion–response time trade-off for an elementary split-merge queueing system subject to subtask delay vector $\mathbf{d}$ we create an objective function formed from the product of mean subtask dispersion ($\mathbb{E}[D_{\mathbf{d}}]$) and mean task response time $\mathbb{E}[R_{\lambda,\mathbf{d}}]$. The former metric is computed as the expected difference between the maximum and minimum heterogeneous order statistics of subtask service times – the mean of the range (Eq. 3.5). As we stated in Eq. 2.2 the dependence between order statistics is irrelevant when considering mean values due to the linearity property of expectation operator, i.e. $\mathbb{E}[D_{\mathbf{d}}] = \mathbb{E}[X^{\mathbf{d}}_{(n)} - X^{\mathbf{d}}_{(1)}] = \mathbb{E}[X^{\mathbf{d}}_{(n)}] - \mathbb{E}[X^{\mathbf{d}}_{(1)}]$. The latter metric is computed by applying the Pollaczek-Khinchine formula from Eq. 2.5 for mean task response time in an $M/G/1$ queue with service time distribution $X^{\mathbf{d}}_{(n)}$. This is because an elementary split-merge queueing system is conceptually equivalent to an $M/G/1$ queue whose service time is the maximum of its set of parallel service times (giving mean service time $\mu^{-1} = \mathbb{E}[X^{\mathbf{d}}_{(n)}]$).

We thus express the trade-off as a function of $\mathbf{d}$ and λ:

$$
\begin{aligned}
T(\mathbf{d},\lambda) &= \mathbb{E}[R_{\mathbf{d},\lambda}]\mathbb{E}[D_{\mathbf{d}}] = \mathbb{E}[R_{\mathbf{d},\lambda}](\mathbb{E}[X^{\mathbf{d}}_{(n)}] - \mathbb{E}[X^{\mathbf{d}}_{(1)}]) \qquad (4.1)\\
&= \Big(\frac{\rho + \lambda\mu \mathrm{Var}[X^{\mathbf{d}}_{(n)}]}{2(\mu-\lambda)} + \mu^{-1}\Big)\\
&\quad \times \Big(\int_0^\infty 1 - \prod_{i=1}^{n} F_i(x-d_i)dx - \int_0^\infty \big(1 - (1 - \prod_{i=1}^{n}(1 - F_i(x-d_i)))\big)dx\Big)\\
&= \Big(\frac{\rho + \lambda\mu \mathrm{Var}[X^{\mathbf{d}}_{(n)}]}{2(\mu-\lambda)} + \mu^{-1}\Big)\\
&\quad \times \Big(\int_0^\infty 1 - \prod_{i=1}^{n} F_i(x-d_i)dx - \int_0^\infty \prod_{i=1}^{n} \big(1 - F_i(x-d_i)\big)dx\Big)
\end{aligned}
$$

The variance of $X^{\mathbf{d}}_{(n)}$ can be computed from Eq. 2.4:

$$
\mathrm{Var}[X^{\mathbf{d}}_{(n)}] = 2\int_0^\infty x\big(1 - \prod_{i=1}^{n} F_i(x-d_i)\big)\,dx - \Big(\int_0^\infty 1 - \prod_{i=1}^{n} F_i(x-d_i)\,dx\Big)^2
$$

In the above we have assumed that each major component of the objective function (i.e. mean subtask dispersion and mean task response time) should be given equal weighting since we regard these metrics to have equal significance. We note that, in line with the treatment of weighted product methods in [72], each component can be raised to a different exponent (> 1) in order to express a preference about the relative importance of the components.

4.4 Optimisation Procedure of a Product of Subtask Dispersion and Task Response Time

We seek the vector of subtask delays $\mathbf{d}_{\min}$ which minimises $T(\mathbf{d}, \lambda)$. That is,

$$\mathbf{d}_{\min} = \arg\min_{\mathbf{d}} T(\mathbf{d}, \lambda) \tag{4.2}$$

We can apply Newton's method (see Eq. 2.11) to find $\mathbf{d}_{\min}$ iteratively:

$$\mathbf{d}_{k+1} \quad = \quad \mathbf{d}_k - \gamma\big[H_{T(\mathbf{d}_k,\lambda)}\big]^{-1}\nabla T(\mathbf{d}_k, \lambda),\ k \geq 0 \tag{4.3}$$

where $\nabla T(\mathbf{d}_k, \lambda)$ is a gradient of the objective function $T(\mathbf{d}_k, \lambda)$, $H_{T(\mathbf{d}_k,\lambda)}$ is the Hessian matrix (matrix of second order partial derivatives) of the objective function $T(\mathbf{d}_k, \lambda)$ and $\mathbf{d}_k$ is the kth iterate of the subtask delay vector. The initial subtask delay vector is chosen heuristically as:

$$\mathbf{d}_0 = ((\max_i \mathbb{E}\big[X_i\big] - \mathbb{E}\big[X_1\big])(1-\rho), \ldots, (\max_i \mathbb{E}\big[X_i\big] - \mathbb{E}\big[X_n\big])(1-\rho)) \tag{4.4}$$

where $\rho = \lambda/\mu$. Here the intuitive minimum obtained by subtracting the mean service time of each server from the maximum mean service time and is multiplied by the difference between full utilisation and current utilisation.

In Eq. 4.3 $\gamma \in (0, 1)$ is a constant introduced to satisfy the Wolfe conditions from Eq. 2.12 and Eq. 2.13. The step size γ needs to be chosen to be small enough to support convergence yet large enough to make rapid progress towards the minimum. For the objective function in

Eq. 4.1 Wolfe's conditions must hold to ensure this. Firstly:

$$T\Big(\mathbf{d}_k + \gamma\big(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)\big),\lambda\Big) \leq T(\mathbf{d}_k,\lambda) + c_1\gamma\nabla T^T(\mathbf{d}_k,\lambda)\big(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)\big) \tag{4.5}$$

And secondly:

$$\nabla T^T\big(\mathbf{d}_k + \gamma(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)),\lambda\big)\big(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)\big) \geq c_2\nabla T^T(\mathbf{d}_k,\lambda)\big(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)\big) \tag{4.6}$$

Here c_1 and c_2 are constants, which should be chosen such that $0 < c_1 < c_2 < 1$, $c_1 \ll c_2$. Practically, we set c_1 as 10^{-4} and c_2 as 0.5. Eq. 4.5 above corresponds to the Armijo rule [5] which guarantees that the step size γ will decrease the objective function sufficiently, while Eq. 4.6 ensures the curvature condition.

4.5 Numerical Results

4.5.1 Implementation

We have implemented the above optimisation technique in C++. Using Newton's method, we begin by initialising $\mathbf{d}_0$ according to Eq. 4.4 and choose an appropriate γ satisfying Eq. 4.5 and 4.6. On each iteration k the method calculates the inverse Hessian matrix and gradient of the objective function, which gives a direction for the updated vector $\mathbf{d}_{k+1}$. Evaluation of the objective function involves computation of mean task response time using the Pollaczek-Khinchine formula and computation of mean subtask dispersion, which in turn involves evaluating the expected value of the minimum and maximum heterogeneous order statistics by numerical integration (using the trapezoidal rule).

4.5.2 Case Study

Consider an elementary split-merge queueing system with 3 parallel servers having heterogeneous service time density functions:

$$X_1 \sim \text{Pareto}(\alpha = 3, l = 3.5, h = 10) \quad (E[X_1] = 5.25,\ \text{Med}[X_1] = 4.41,\ \text{Var}[X_1] = 9.19)$$
$$X_2 \sim \text{Erlang}(n = 2, \lambda = 1) \quad (E[X_2] = 2,\ \text{Med}[\text{X}_2] = 1.68,\ \text{Var}[X_2] = 2)$$
$$X_3 \sim \text{Det}(5) \quad (E[X_3] = 5,\ \text{Med}[X_3] = 5,\ \text{Var}[X_3] = 0)$$

As we noted in Section 3.3.1, performance measures from split-merge queueing systems can be obtained analytically or via simulation. In this case study we chose analytical computation of performance measures instead of simulations.

Without adding any subtask delays, mean subtask dispersion is $\mathbb{E}[D_{\mathbf{d}=\mathbf{0}}] = 3.58$ time units and maximum sustainable task throughput is $\lambda_{\max} = 0.182$ tasks/time unit.

For $\lambda = 0.01$ tasks/time unit, mean task response time is $\mathbb{E}[R_{\mathbf{d}=\mathbf{0},0.01}] = 5.65$ time units, while for $\lambda = 0.15$ tasks/time unit, mean task response time is $\mathbb{E}[R_{\mathbf{d}=\mathbf{0},0.15}] = 18.7$ time units.

Using our previously developed optimisation technique designed to reduce mean subtask dispersion without regard to impact on response time from Chapter 3 we obtain the vector of optimal subtask delays:

$$\mathbf{d}_D = (0.608, 3.372, 0.0)$$

as shown in Fig. 4.1.

After applying these delays, maximum sustainable throughput drops to $\lambda_{\max} = 0.161$ tasks/time unit and mean subtask dispersion improves to $\mathbb{E}[D_{\mathbf{d}_D}] = 1.73$ time units, improving by 62%. For $\lambda = 0.01$ tasks/time unit, mean task response time rises to $\mathbb{E}[R_{\mathbf{d}_D,0.01}] = 6.44$ time units, a 14% increase. For $\lambda = 0.15$ tasks/time unit mean task response time dramatically increases to $\mathbb{E}[R_{\mathbf{d}_D,0.15}] = 51.4$ time units, an increase of 175%.

Now we optimise the same system but under the subtask dispersion–response time trade-off

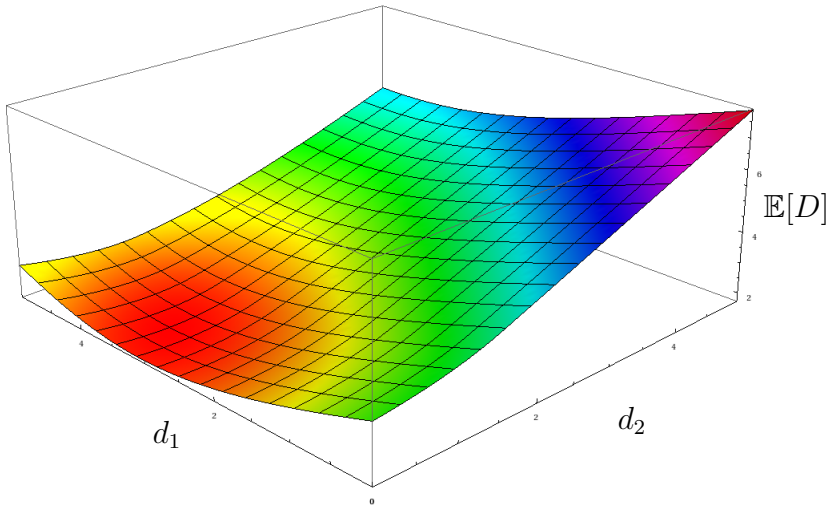

Figure 4.1: Surface plot of mean subtask dispersion of split-merge queueing system against subtask delays using our previous methodology from Chapter 3.

presented in this chapter. With $\lambda = 0.01$ tasks/unit time we obtain the following vector of optimal subtask delays:

$$\mathbf{d}_T = (0.453, 3.002, 0.0)$$

as shown in Fig. 4.2. Mean subtask dispersion becomes $\mathbb{E}[D_{\mathbf{d}_T,0.01}] = 1.76$ time units, which is only 1.8% higher than the dispersion obtained under delay vector $\mathbf{d}_D$. Mean task response time is $\mathbb{E}[R_{\mathbf{d}_T,001}] = 6.24$ time units (a 10% rise in comparison to a system without delays, but a 3% reduction compared to the response time under delay vector $\mathbf{d}_D$).

With $\lambda = 0.15$ tasks/unit time the vector of optimal subtask delays drops to:

$$\mathbf{d}_T = (0.0928, 2.04, 0.0)$$

as shown in Fig. 4.3. Mean subtask dispersion is now $\mathbb{E}[D_{\mathbf{d}_T,0.15}] = 2.08$ time units, a 21% increase over the dispersion obtained under delay vector $\mathbf{d}_D$, but still a good improvement over the system without added delays (3.58 time units). The corresponding distributions of subtask dispersion are shown in Fig. 4.4. It is apparent that the trade-off is able to maintain competitive dispersion with our previous methodology, especially for high percentiles. Mean task response time is $\mathbb{E}[R_{\mathbf{d}_T,0.15}] = 22.9$ time units, which is 23% worse than the system without delays but

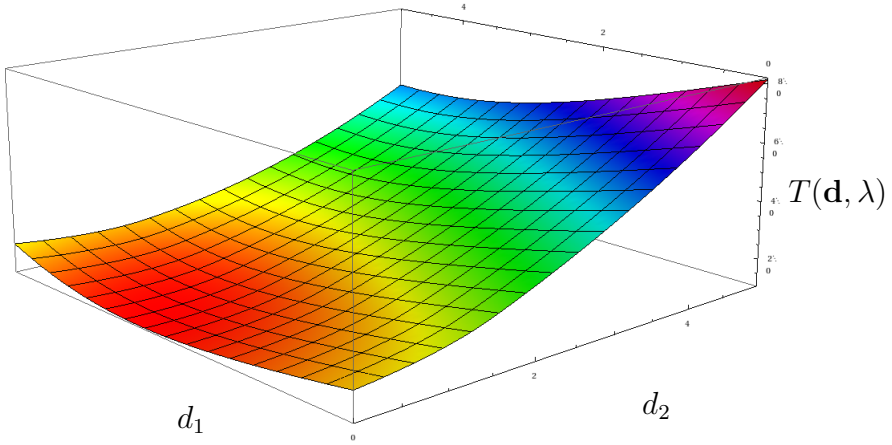

Figure 4.2: Surface plot of subtask dispersion–response time trade-off objective function of split-merge queueing system against subtask delays for $\lambda = 0.01$ (tasks/time unit).

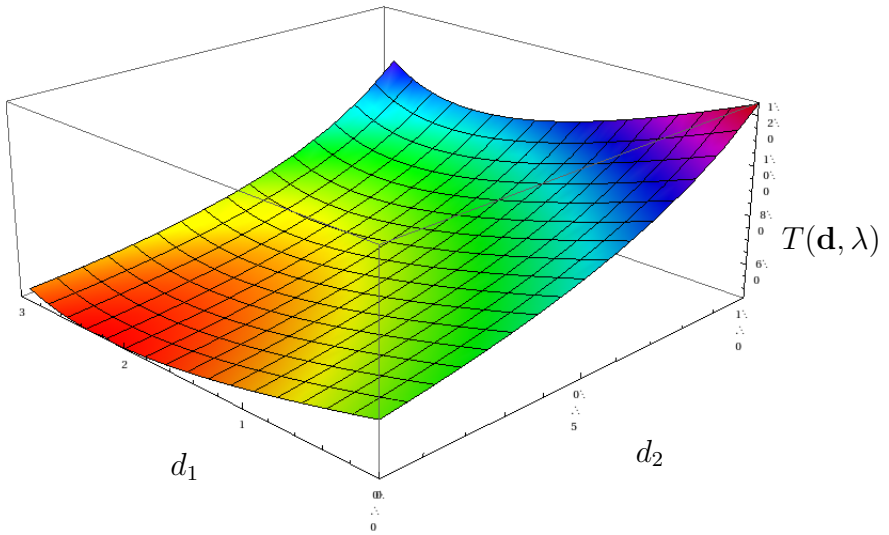

Figure 4.3: Surface plot of subtask dispersion–response time trade-off objective function of split-merge queueing system against subtask delays for $\lambda = 0.15$ (tasks/ time unit).

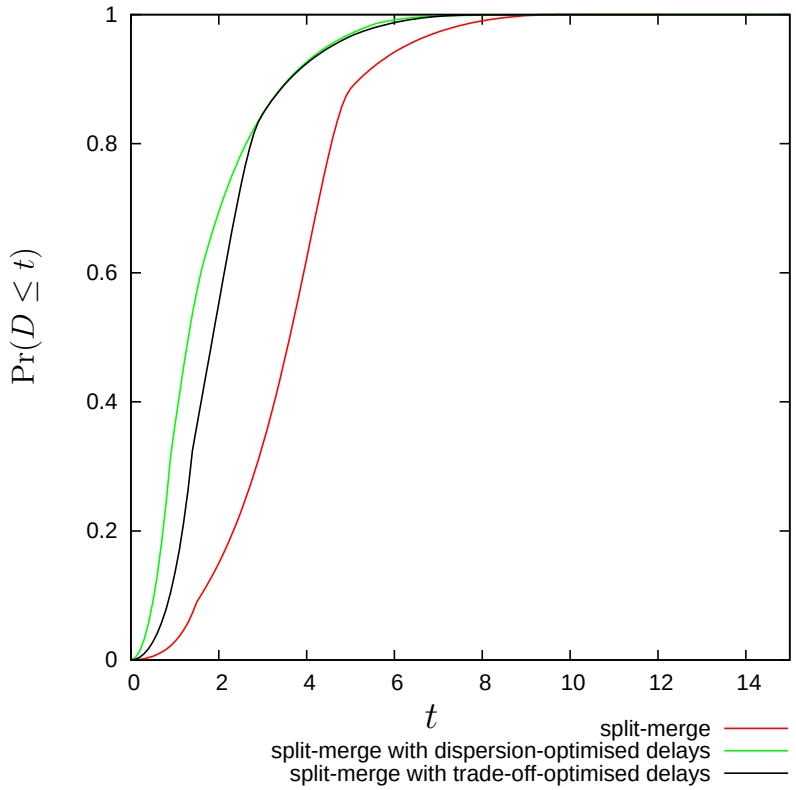

Figure 4.4: Distributions of subtask dispersion of split-merge queueing system with $\lambda = 0.15$ (tasks/time unit) without any delays (red line) with delays optimised for $T(\mathbf{d}, 0.15)$ (green line) and for $\mathbb{E}[D_{\mathbf{d}}]$ (blue line).

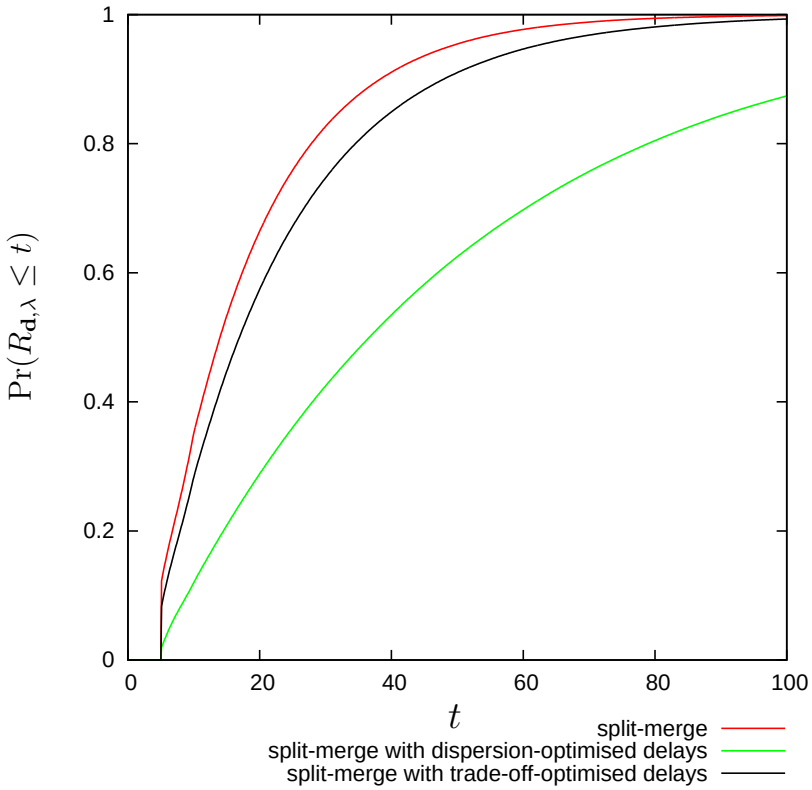

Figure 4.5: Distributions of task response time of split-merge queueing system given $\lambda = 0.15$ (tasks/time unit), without any delays (red line), with delays optimised for $T(\mathbf{d}, 0.15)$ (green line) and delays optimised for $\mathbb{E}[D_{\mathbf{d}}]$ (blue line).

a 55% improvement on mean task response time under delay vector $\mathbf{d}_D$. The corresponding distributions of task response time are shown in Fig. 4.5. It is apparent that the trade-off is able to maintain competitive task response times as compared to the system with no delays, in stark contrast with our previous methodology.

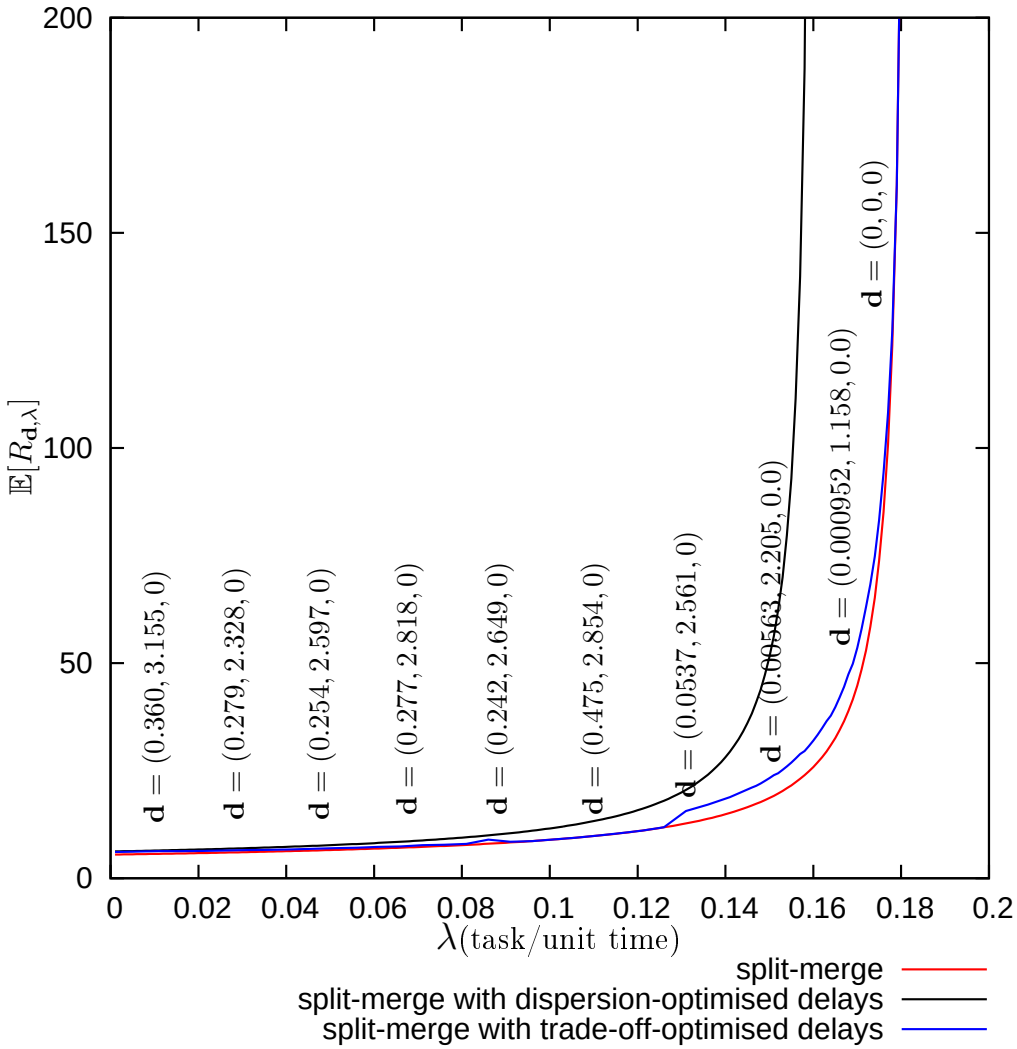

Figure 4.6: Expected response time of split-merge queueing system for various customer arrival rates without any delays (red line), with delays optimised for subtask dispersion–task response time trade-off (green line), and with delays optimised for mean subtask dispersion alone (blue line). Subtask delay vectors are also shown for the subtask dispersion–task response time trade-off.

It is interesting to note that under our trade-off as λ converges to the maximum sustainable throughput of the split-merge system without delays (cf. [58]), the vector of optimal subtask delays tends to a vector of zeros. Indeed, optimising the case study system for $\lambda = 0.18$ leads

to the vector of optimal subtask delays:

$$\mathbf{d} = (0.0, 0.0, 0.0)$$

Fig. 4.6 shows how the vector of optimal delays changes with λ, and how it converges to the zero-vector as λ approaches $\lambda_{\max} = 0.182$ task/time unit. We note that the maximum sustainable throughput of the system optimised under our previous methodology from Chapter 3 is rather less than that of the system without delays, whereas the maximum sustainable throughput of the system without delays is maintained using the present methodology.

4.6 Summary

In this chapter we have described a framework for delaying the dispatch of subtasks to parallel servers in elementary split-merge queueing systems in order to manage the trade-off between response time and subtask dispersion. At the core of our technique is an objective function computed as the product of expected subtask dispersion and expected task response time. Previous research has concentrated on the optimisation of each one of these metrics in isolation, frequently resulting in significant deterioration in the other. By contrast, the present approach is able to achieve an excellent compromise between the two metrics, without adversely affecting the maximum sustainable throughput of the system.

Chapter 5

Reducing Mean Subtask Dispersion in Fork-Join Systems

5.1 Introduction

Fork-join queueing systems are well-known asynchronous counterparts of split-merge queueing systems. This asynchronous nature is achieved by having queueing capabilities at service nodes, which is missing in split-merge queueing systems.

In this chapter we consider an elementary fork-join queueing systems (see Fig 2.3) with assumptions of Poisson task arrivals, heterogeneous exponentially distributed service times and non-preemptive subtask service. Its detailed structure and operation was presented in Section 2.6.2.

As we did for elementary split-merge queueing systems, for optimising elementary fork-join queueing systems we consider the same two metrics of interest: *task response time*, or sojourn time is the total time that a task spends in the system. This measure has been the major target of research effort over many decades [49, 9] and *subtask dispersion*, that is the difference in time between completions of service of the first and last subtasks that originated from the same task. This is an especially important metric in certain real world applications. Reduction

of this metric has a side effect – it reduces utilisation of mean number of subtasks in output buffer.

As illustrated in Fig. 1.6 these metrics are in tension in the sense that trying to reduce one usually results in an increase in another; this is especially the case for high-intensity workloads. Elementary fork-join queueing systems have low task response times (and therefore higher maximum sustainable system throughput), but subtask dispersion is high and worsens under load. Conversely, split-merge queueing systems with the same parallel server configuration are characterised by low subtask dispersion, but usually suffer from a high task response times (and therefore reduced maximum sustainable system throughput) under load. As we have shown previously in Chapters 3 and 4, adding delays to subtask processing times in elementary split-merge queueing systems can help to minimise mean subtask dispersion [97] and/or percentiles of subtask dispersion [99], but the sole focus on subtask dispersion serves to exacerbate the problem of poor task response times under load. One solution described in Chapter 4 is to apply load-dependent subtask delays which reduce the product of expected task response time and expected subtask dispersion [96]. This is highly effective at achieving a balance between the metrics; however, maximum sustainable system throughput is still limited to that achievable under an unmodified split-merge system. Our goal is to find a way to reduce mean subtask dispersion in elementary fork-join queueing systems to levels comparable with or *below* that observed in all varieties of elementary split-merge queueing systems while retaining the task response time and throughput benefits of a classical fork-join queueing system.

We present a technique that is able to reduce subtask dispersion with only a marginal increase in task response time [98]. Achieving this is challenging since the unsynchronised operation of fork-join queueing systems naturally militates against low subtask dispersion. Our approach builds on the elements of our earlier research examining subtask dispersion and response time in elementary split-merge queueing systems in Chapters 3 and 4. The methodology involves the frequent application and updating of delays to the processing of subtasks which are at the head of the parallel service queues. We present numerical results that show the ability to reduce subtask dispersion in elementary fork-join queueing systems to levels comparable with or below that observed in all varieties of elementary split-merge queueing systems while retaining the

task response time and throughput benefits of a classical fork-join queueing system.

5.2 Dynamic Online Algorithm for Reduction of Subtask Dispersion

In the following we consider a fork-join system with n parallel heterogeneous servers, the ith of which has an exponential service time distribution with rate parameter μ_i, i.e. $F_i(x) = 1 - e^{-\mu_i x}$. To describe the state of the system at time t let $k_i(t)$ denote the number subtasks present in parallel server queue i; as such $n(\max_i k_i(t)) - \sum_i k_i(t)$ subtasks will be present in the join queues (or output buffer) at time t.

Our strategy is to let the system operate in its normal fork-join fashion, but to delay the start of service of certain of the subtasks that are at the head of the parallel service queues. In particular, at *every* time instant t at which a *hitherto-unserviced* subtask S reaches the front of a parallel queue, we take the following control actions:

1. If any of the siblings of S have already completed service then the best mean subtask dispersion and task response time with respect to S's task are simultaneously achieved by immediately beginning service of S and also of any of its siblings that are at the front of their parallel queue.

2. Otherwise all siblings of S are still present in the parallel queues and we apply delays to S and those of its siblings that are at the front of their parallel queues and which have not yet entered service. We choose appropriate delays (which may include zero delays) by observing that, from the point of view of subtask S and its siblings, the system at that instant is equivalent to an n-server split-merge queuing system (see Fig. 5.1) in which parallel server i has service time distribution Erlang$(q_i(t)+1, \mu_i)$, where number of stages in tandem in Erlang distribution, that is $q_i(t)$ is a number of subtasks in front of S or its sibling subtask in parallel queue i plus residual service time (exploiting memoryless property of the exponential distribution). We can then exploit the optimisation methods

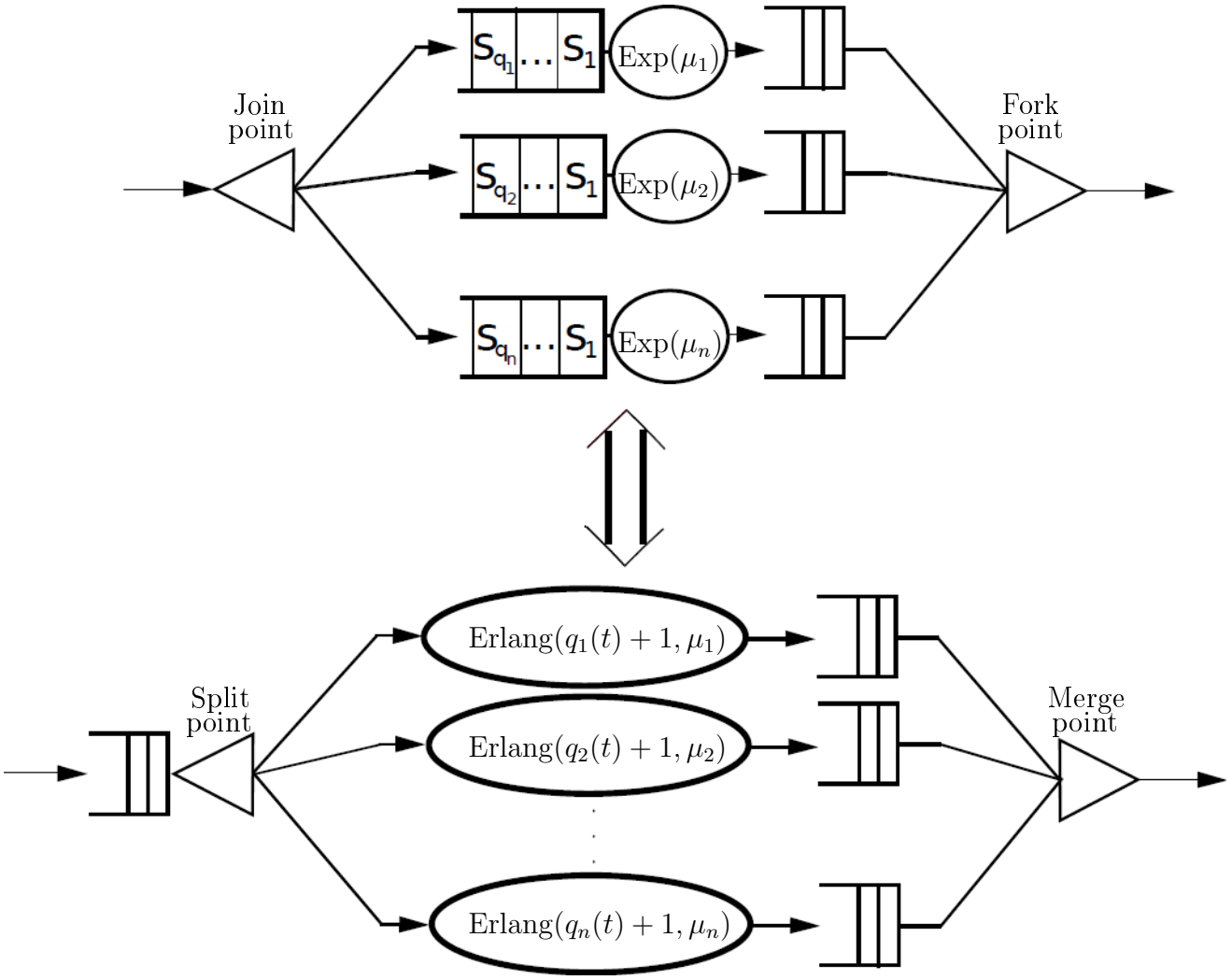

Figure 5.1: How a fork-join queueing system can be viewed as a split-merge queueing system from the point of view of a subtask and its siblings at time instant t.

we developed in Chapter 3 and Chapter 4 to determine a vector of (near-)optimal deterministic subtask delays $\mathbf{d} = \big(d_1, d_2, \ldots, d_n\big)$. Here element d_i denotes the deterministic delay which should notionally be applied to parallel server i. In fact we only adopt the delays corresponding to S and its siblings that are at the front of their parallel queues and which have not yet entered service (note this may involve overwriting a currently pending delay).

Similarly at time instants at which a subtask S enters a join queue (or output buffer) then we immediately begin service of any of the siblings of S that are at the front of their parallel queues.

The objective function of the optimisation is mean subtask dispersion based on Eq. 3.5, computed as the difference between the maximum and minimum heterogeneous order statistics of the split-merge-equivalent system with delays observed by queueing subtasks. Utilising the linearity property of the expectation operator over dependent variables from Eq. 2.2, we have:

$$\begin{aligned}
\mathbb{E}[D_{\mathbf{d}}] &= \Big(\mathbb{E}[X^{\mathbf{d}}_{(n)} - X^{\mathbf{d}}_{(1)}]\Big) \\
&= \Big(\mathbb{E}[X^{\mathbf{d}}_{(n)}] - \mathbb{E}[X^{\mathbf{d}}_{(1)}]\Big) \\
&= \int_0^\infty \big(1 - \prod_{i=1}^{n} F_i(x - d_i)\big)\, dx - \\
&\quad \int_0^\infty \big(1 - \big(1 - \prod_{i=1}^{n}(1 - F_i(x - d_i))\big)\big)\, dx \qquad (5.1) \\
&= \int_0^\infty 1 - \prod_{i=1}^{n} F_i(x - d_i)\, dx - \int_0^\infty \prod_{i=1}^{n} \big(1 - F_i(x - d_i)\big)\, dx
\end{aligned}$$

where $F_i(x - d_i)$ is a shifted Erlang$(q_i(t) + 1, \mu_i)$ cumulative distribution function.

When optimising, we solve for:

$$\mathbf{d}_{\min} = \arg\min_{\mathbf{d}} \mathbb{E}[D_{\mathbf{d}}] \qquad (5.2)$$

while additionally applying the constraint $(\prod_i d_i = 0)$ to avoid the addition of superfluous

delays to bottleneck queues into the queueing system.

The optimisation procedure itself is based on Newton's method (Eq. 2.11) and Wolfe conditions (Inequalities 2.12 and 2.13). Practically, the procedure utilises utilises numerical integration when evaluating the objective function and exploits a disk-based memoisation technique to dramatically reduce the time cost of computing optimised delay vectors for system states that have already been encountered in the current execution or in some previous execution.

5.3 Complexity

The computational effort of the dynamic online algorithm for finding $\mathbf{d}_{\min}$ increases dramatically as the number of parallel servers rises. There are two reasons for this: firstly, numerical optimisation techniques for finding the minimum of an objective function, such as Newton's method or the Nelder–Mead method increase in complexity and computational cost as dimension of the problem increases this is because it is a hard class of combinatorial optimisation problem. Secondly, the memoisation technique applied for parallel queueing systems is most affective on systems with small dimension. When the size of the input vector $\mathbf{d}$ grows, the memoisation algorithm sees the same permutation of numbers of subtasks in each queues much more rarely.

5.4 Numerical Results

5.4.1 Fork-Join Simulation

In this section we present an event-driven simulator for elementary fork-join queueing networks which is written in C++. The simulator collects a range of performance-related statistics, e.g.: mean task response time, mean subtask dispersion, mean number of subtasks in output buffer, task throughput and distributions of subtask dispersion.

This is an expanded version of the split-merge queueing system simulator. Knowing that the main difference between split-merge queueing systems and fork-join queueing systems is that the former can process more than one task at a time – we first validated correctness of the fork-join queueing system simulator by comparing it with the split-merge queueing system simulator where the arrival rate is very small (to make it very likely that there is only one task at a time in both systems) and the same service time distributions in both simulators. Fig. 5.3 and Fig. 5.4 present the distributions of subtask dispersion and task response time in split-merge queueing systems and fork-join queueing systems with the same configurations. Under light workload the distributions of task response time and subtask dispersion are the same. But under heavy workload task response time of the split-merge queueing system simulator was larger then task response time from the fork-join queueing system simulator. Subtask dispersion under heavy workload is lower for the split-merge queueing system simulator than for the fork-join queueing system simulator.

As shown in Fig 5.2, our fork-join queueing system simulator is based on several classes, e.g.: a FORK point which is connected to N parallel QUEUEINGNODEs, with a queueing capability (FIFO). Each queueing node is connected to a JOINBUFFER (an output buffer) followed by a JOIN point. A TASK come to the system via the FORK point, where it forks immediately into N SUBTASKs. Each subtask arrives at the back of a queue of its allocated QUEUEINGNODE. After been served a subtask goes to the JOINBUFFER where it waits for all of its siblings. All SUBTASKs, that belong to the same original task; join together at the JOIN point and leave the system.

In order to embed the dynamic online algorithm for reduction of subtask dispersion into the fork-join queueing system simulator we introduce several functions and variables: a QUEUEINGNODE class has functionality of calculation of the vector of optimal delays by the dynamic online algorithm (GETDELAYS). The delays are applied to subtasks that reach a front of parallel queues in the QUEUEINGNODEs. A function GETDELAYS also updates delays for the SUBTASKs that are already in a parallel service nodes and have been already delayed but have not start their service. When the first SUBTASK from a particular task arrives into the JOINBUFFER a flag fires and all delays that relate to its siblings are deleted by the function

DELETEDELAYS and no new delays are applied for these siblings.

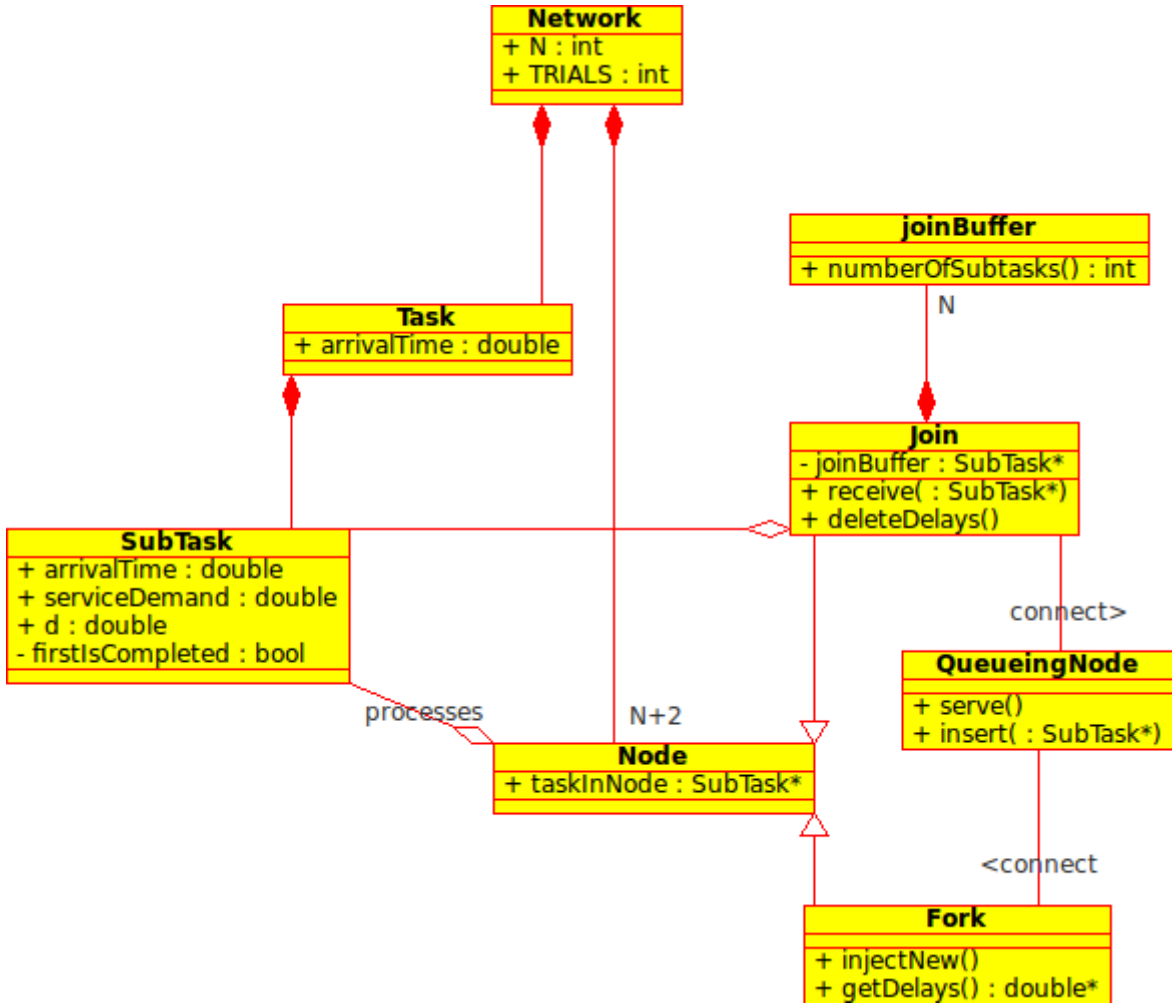

Figure 5.2: UML class diagram for simulator of elementary fork-join queueing network.

5.4.2 Case Study

In this section we present results from C++ simulations of fork-join queueing systems and split-merge queueing systems. We compare the dynamic optimisation algorithm for elementary fork-join queueing systems and the static optimisation techniques developed in Chapter 3 and Chapter 4 for elementary split-merge queueing systems.

For our case study each simulation run is made up of 10 replicas, and each replica consists of a warm-up period of the processing of 250 000 tasks followed by an measurement period of the processing of 250 000 tasks. For the static optimisation techniques, it takes approximately one second to run each replica, and for the dynamic optimisation using the fork-join simulator it

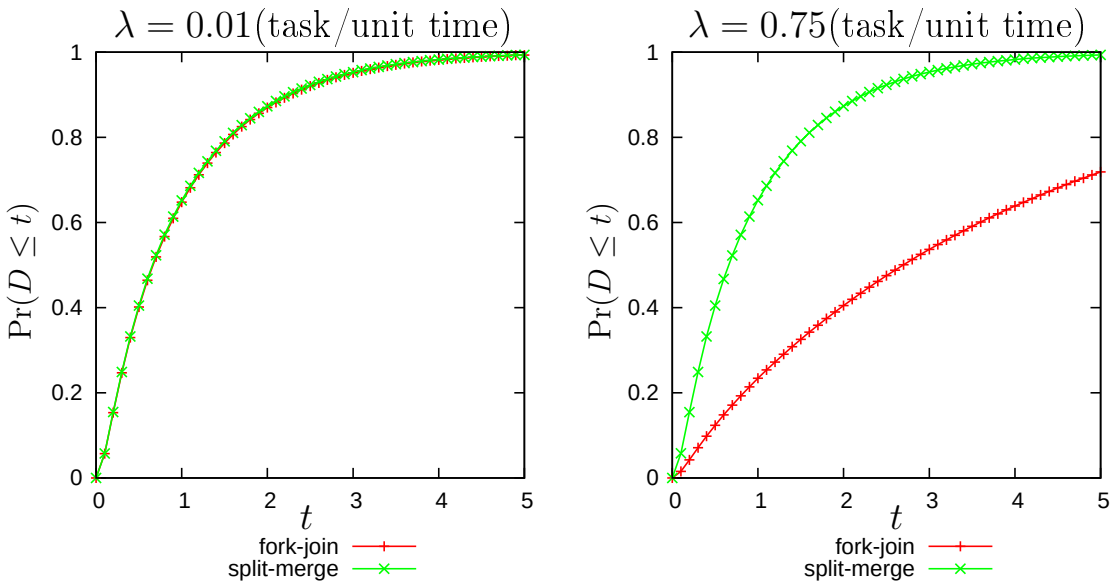

Figure 5.3: Distributions of subtask dispersion in fork-join and split-merge queues with the same configurations but different mean arrival rates.

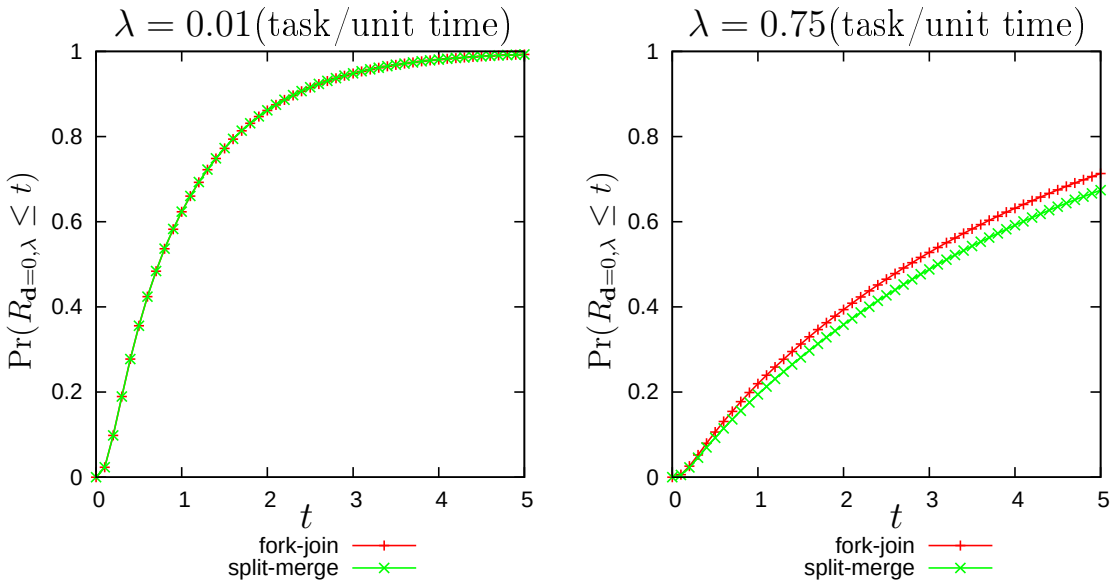

Figure 5.4: Distributions of task response time for fork-join queueing systems and split-merge queueing systems with the same configurations but different mean arrival rates.

takes around 7.5 minutes. The replicas are used to put 95% confidence intervals (CIs) on all measures. Results are reported to three decimal places.

Consider a parallel queueing system with Poisson arrivals with rate parameter $\lambda = 0.78$ tasks/-time unit and 3 parallel service nodes with exponential service time density functions:

$$\begin{aligned} X_1 &\sim \text{Exponential}(\lambda = 1) \\ &\quad (\mathbb{E}[X_1] = 1,\ \text{Med}[X_1] = 0.693,\ \text{Var}[X_1] = 1) \\ X_2 &\sim \text{Exponential}(\lambda = 5) \\ &\quad (\mathbb{E}[X_2] = 0.2,\ \text{Med}[X_2] = 0.139,\ \text{Var}[X_2] = 0.04) \\ X_3 &\sim \text{Exponential}(\lambda = 10) \\ &\quad (\mathbb{E}[X_3] = 0.1,\ \text{Med}[X_3] = 0.0693,\ \text{Var}[X_3] = 0.01) \end{aligned}$$

In this context, we compute measures of subtask dispersion and of task response time in five different types of fork-join and split-merge queueing systems:

1. A fork-join queueing system (without subtask delays). Here the mean task response time is $\mathbb{E}[R_{\mathbf{d}=\mathbf{0}}] = 4.553$ (95% CI $[4.504, 4.602]$) time units and mean subtask dispersion is $\mathbb{E}[D_{\mathbf{d}=\mathbf{0}}] = 4.49$ (95% CI $[4.429, 4.54]$) time units. The mean number of subtasks in the output buffer is 6.862 (95% CI $[6.79, 6.93]$).

2. A fork-join queueing system utilising our dynamic online algorithm for reducing mean subtask dispersion. Here mean task response time is $\mathbb{E}[R_{\mathbf{d}_{\min}}] = 4.703$ (95% CI $[4.586, 4.819]$) time units and mean subtask dispersion is $\mathbb{E}[D_{\mathbf{d}_{\min}}] = 0.752$ (95% CI $[0.745, 0.759]$) time units. The mean number of subtasks in the output buffer is 1.081 (95% CI $[1.071, 1.091]$). When comparing this modified fork-join system with the fork-join system without subtask delays, we observe that mean task response time has increased very slightly by 3.3% but mean subtask dispersion dropped very dramatically by 83%. Similarly, the mean number of subtasks in the output buffer decreased by 84%.

3. A split-merge queueing system (without subtask delays). Mean task response time is $\mathbb{E}\left[R_{\mathbf{d}=\mathbf{0},\lambda=0.78}\right] = 5.212\,(95\%\text{ CI }[5.1526, 5.271])$ time units and mean subtask dispersion is $\mathbb{E}[D_{\mathbf{d}=\mathbf{0}}] = 0.976\,(95\%\text{ CI }[0.975, 0.977])$ time units. The mean number of subtasks in the output buffer is $1.416\,(95\%\text{ CI }[1.415, 1.418])$. This method is thus completely dominated by our dynamic online algorithm for each of these metrics, by factors of 11%, 30% and 31% respectively.

4. A split-merge queueing system with delays applied to reduce mean subtask dispersion from Chapter 3. The vector of optimised delays is:

$$\mathbf{d}_{\min} = (0.0, 0.553, 0.617)$$

Mean task response time is $\mathbb{E}\left[R_{\mathbf{d}_{\min},\lambda=0.78}\right] = 63.02\,(95\%\text{ CI }[58.21, 67.83])$ time units and mean subtask dispersion is $\mathbb{E}[D_{\mathbf{d}_{\min}}] = 0.783\,(95\%\text{ CI }[0.780, 0.785])$ time units. The mean number of subtasks in the output buffer is $1.029\,(95\%\text{ CI }[1.027, 1.031])$. This method is dominated by our dynamic online algorithm with respect to the mean task response time and mean subtask dispersion metrics, by factors of 1240% and 4% respectively. There is however a 5% improvement with respect to the mean number of subtasks in the output buffer.

5. A split-merge queueing system with delays applied to optimise the product of mean task response time and mean subtask dispersion from Chapter 4. The vector of optimised delays is:

$$\mathbf{d}_{\min} = (0.0, 0.0398, 0.0673)$$

Mean task response time is $\mathbb{E}\left[R_{\mathbf{d}_{\min},\lambda=0.78}\right] = 5.329\,(95\%\text{ CI }[5.272, 5.385])$ time units and mean subtask dispersion is $\mathbb{E}[D_{\mathbf{d}_{\min}}] = 0.9343\,(95\%\text{ CI }[0.9336, 0.9349])$ time units. The mean number of subtasks in the output buffer is $1.355\,(95\%\text{ CI }[1.353, 1.357])$. While improving dramatically on the mean task response of the previous case, the method is completely dominated by our dynamic online algorithm for each metric, by factors of 13%, 24% and 25% respectively.

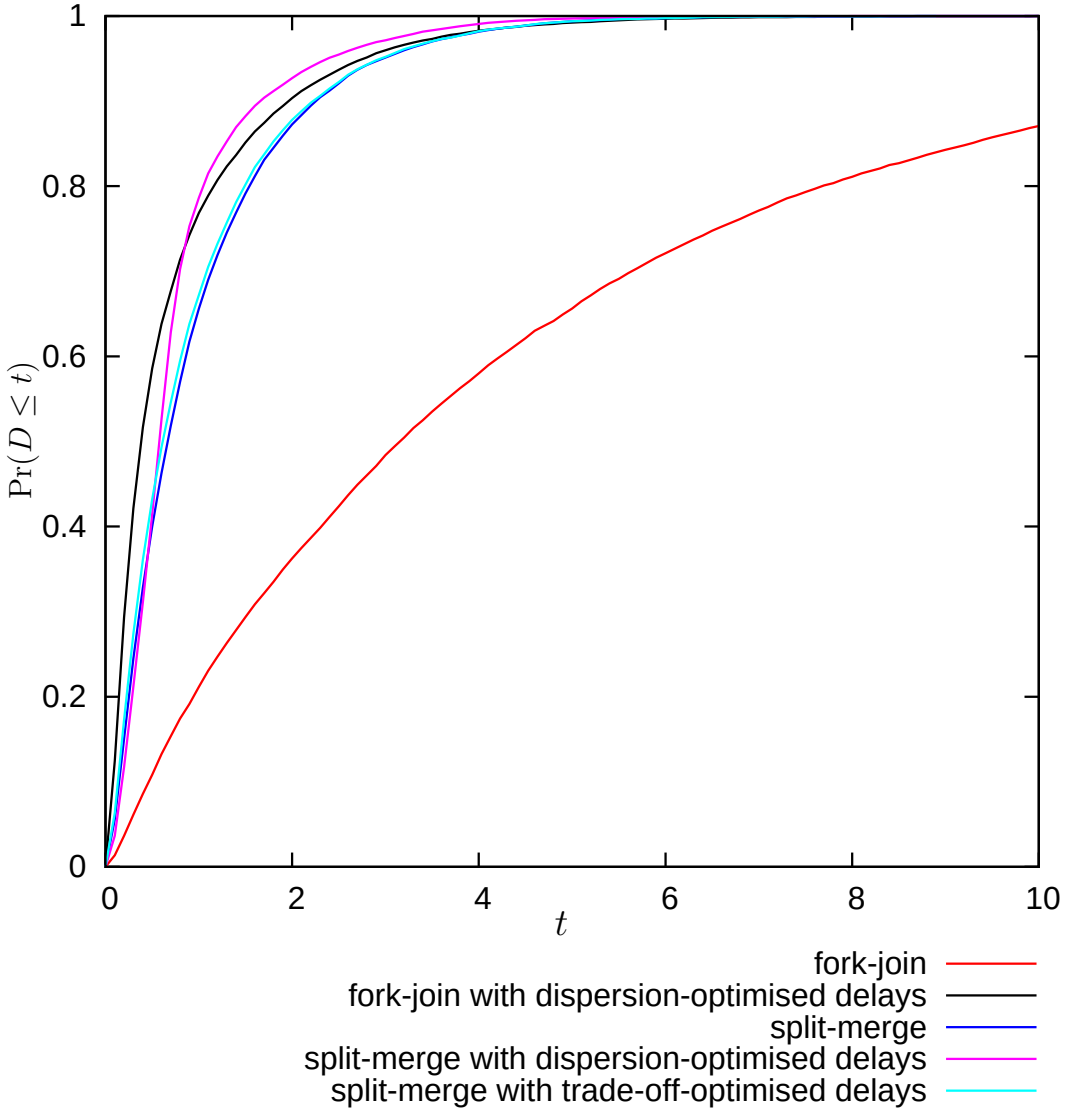

Figure 5.5: Distributions of subtask dispersion in fork-join queueing systems and split-merge queueing systems with and without optimised subtask delays. $\lambda = 0.78$ (task/unit time).

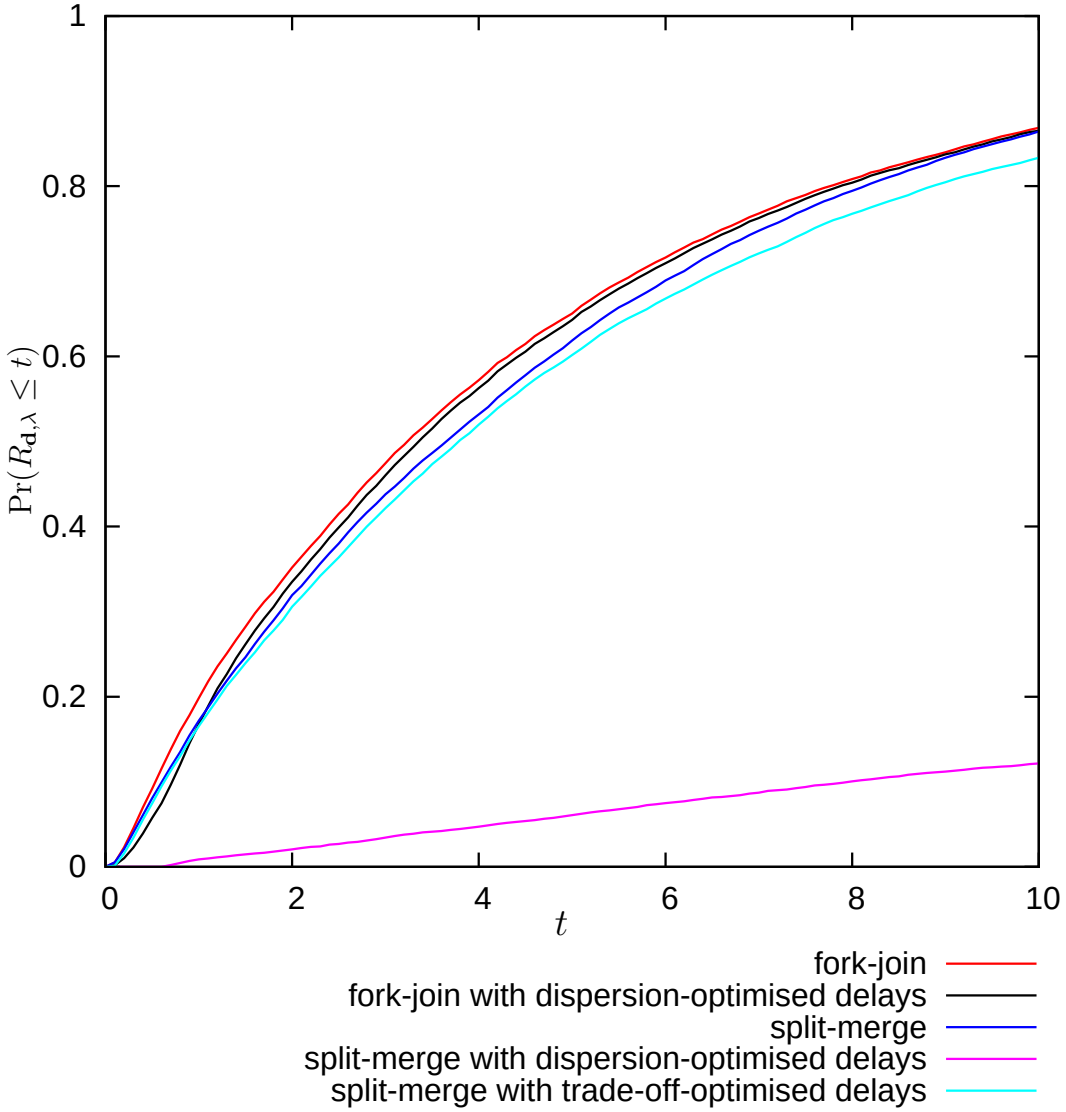

Figure 5.6: Distributions of task response time for fork-join queueing systems and split-merge queueing systems with and without optimised subtask delays. $\lambda = 0.78$ (task/unit time).

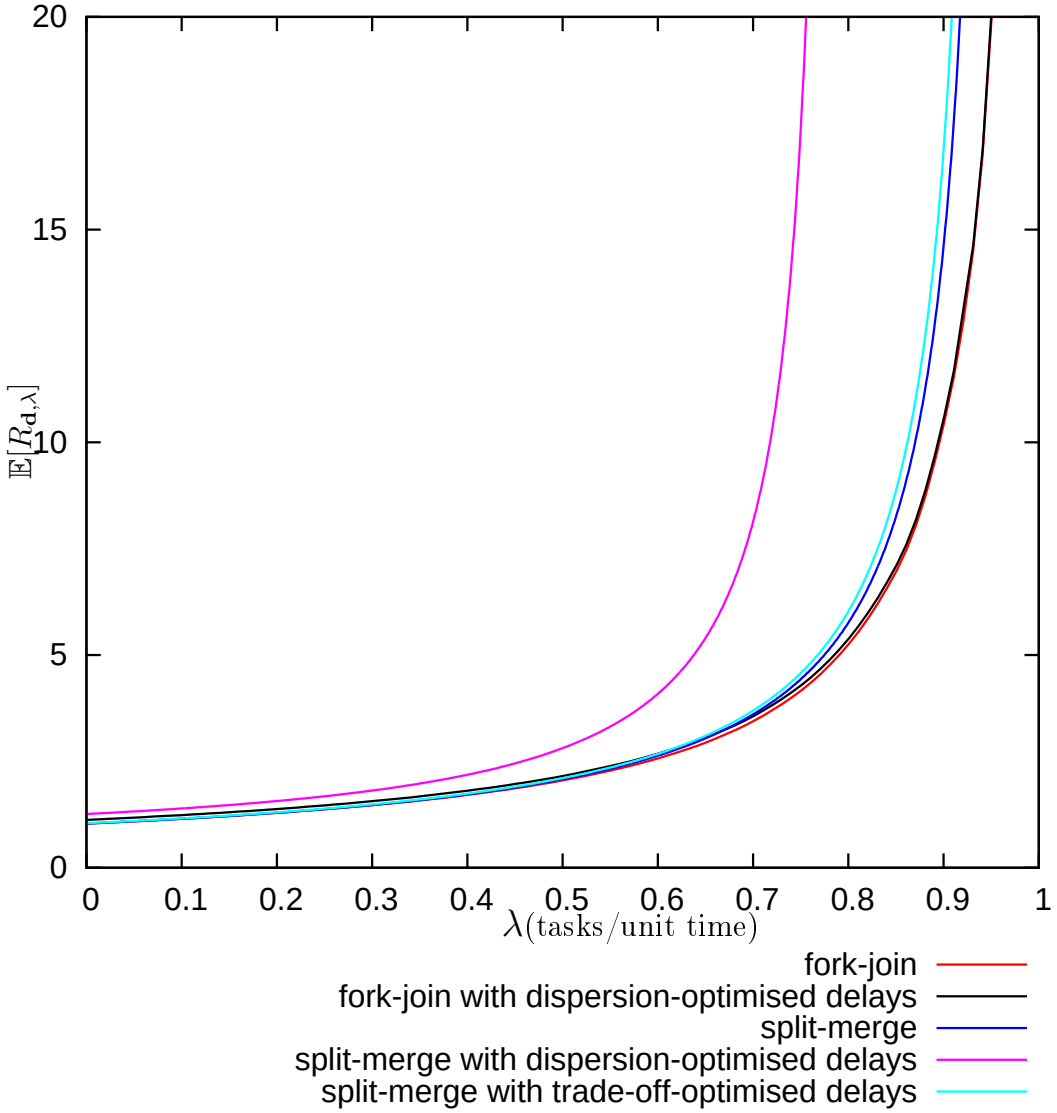

Figure 5.7: Expected response time of fork-join queueing systems and split-merge queueing systems for various customer arrival rates.

Fig. 5.5 shows the corresponding distributions of subtask dispersion. The poor subtask dispersion of the fork-join queueing system without subtask delays is evident. Applying subtask delays optimised for the subtask dispersion–task response time trade-off yields a similar subtask dispersion profile to that of the split-merge queueing system without delays. The subtask dispersion profile of the fork-join queueing system with dispersion-optimised delays is competitive with that of the split-merge queueing system with dispersion-optimised delays, and even dominates it for percentiles of subtask dispersion below 70%.

Turning now to distributions of task response time, Fig. 5.6 demonstrates that the distribution of task response time of the fork-join queueing system with dispersion-optimised delays is very close to that of the fork-join queueing system without subtask delays. Here, the distribution of response time of the split-merge queueing system without subtask delays is marginally worse than that of the fork-join queueing system, but after applying dispersion-optimised delays the response time suffers heavily. Applying delays optimised for subtask dispersion–task response time trade-off impacts only marginally on the task response time.

Fig. 5.7 shows how mean task response time varies with various task arrival rates under the various queueing policies. We observe the split-merge queueing system with the dispersion-optimised delays has the lowest maximum sustainable system throughput, followed by the split-merge queueing system with dispersion-optimised delays and the split-merge queueing system without delays. The highest maximum sustainable system throughput is provided by the fork-join system without subtask delays and the fork-join system utilising dispersion-optimised subtask delays.

5.5 Summary

In this chapter we considered a problem of reducing subtask dispersion in fork-join queueing systems. To control this metric, we derived an online algorithm which dynamically computes and applies state-dependent delays to the processing of subtasks and their siblings at various time instants.

We demonstrated our algorithm on a case study, a parallel system subjected to five different kinds of split-merge and fork-join queueing policies. The results show how the technique proposed in this chapter is able to deliver low subtask dispersion competitive with split-merge-based queueing systems while simultaneously delivering low task response times competitive with fork-join-based queueing systems.

Our current research can no doubt be extended to apply a fork-join queueing system with non-exponential services time distributions. Certainly extension to Erlang and phase-type service time distributions is likely to be straightforward given appropriate extensions to the system state vector. Further possible directions are to extend the dynamic online algorithm to elementary fork-join queueing systems with heterogeneous general service time distributions where residual service times may be defined as in [11].

Chapter 6

Conclusion

6.1 Summary of Achievements

The main aim in this dissertation was to investigate ways in which subtask dispersion can be reduced in parallel processing queueing systems by means of introducing judiciously-chosen deterministic delays to the processing of subtasks. We have derived an analytical toolbox and a set of numerical techniques to achieve reduction of subtask dispersion in parallel processing systems.

We began by investigating a straightforward method for minimising subtask dispersion in split-merge queueing systems with heterogeneous general service time distributions. To achieve this we extended the theory of heterogeneous order statistics by deriving the mean and the distribution of the range of heterogeneous order statistics. The former formula corresponds to mean subtask dispersion in elementary split-merge queueing system, so it became an objective function in an optimisation procedure for minimising mean subtask dispersion. The latter formula corresponds to the distribution of the subtask dispersion in elementary split-merge queueing systems, so it was applied in an optimisation procedure for reducing a given percentile of subtask dispersion. This allows us to control the system's dispersion with soft (probabilistic) guarantees. The results showed that our techniques were very effective in terms of minimising subtask dispersion, but adversely impacted task response time and maximum

system sustainable throughput.

Because task response time suffers from applying our previous methodology, we expanded it to use a new objective function, that is a product of mean subtask dispersion and mean task response time. Consequently, we started trading-off mean subtask dispersion and mean task response time. The results were very effective – mean subtask dispersion was reduced while mean task response time was affected much less.

Although we could reduce subtask dispersion while having task response time under control in split-merge queueing systems – the achieved methodology still left us with a problem that split-merge queueing systems naturally have lower level of maximum sustainable throughput and higher task response time than fork-join queueing systems with the same parallel server configuration. Therefore, we were looking for ways to relax our set of techniques to apply it in fork-join queueing systems. But we faced a considerable challenge because fork-join systems are notoriously analytically intractable. To overcome this, we devised a dynamic online algorithm for elementary fork-join queueing systems with heterogeneous exponential service time distributions. At certain instants, selected subtasks see the fork-join system queueing system with heterogeneous exponentially distributed service times as being equivalent to a split-merge system with heterogeneous Erlang service time distributions. This idea gives to us the opportunity to apply our developed theory for split-merge queueing systems.

Finally we have achieved the goal of simultaneously reducing subtask dispersion (which is compatible with the best subtask dispersion only minimising algorithm for split-merge queueing systems) and achieving maximum sustainable system throughput and task response time which are very close to the maximum sustainable system throughput and task response time of a fork-join queueing system without any delays.

6.2 Applications

We discuss the applications enabled by our methods for reducing subtask dispersion in: split-merge queueing systems and fork-join queueing systems.

An example where split-merge queueing systems can be applied is the finance domain. In order to increase the likelihood of filling orders across multiple markets and experience fewer adverse ticks, delays can be applied to the orders so that they arrive at all target exchanges nearly simultaneously. This has been shown to increase execution quality. The applications of applied delays to orders was a subject of a recent book [67] and the foundation of a new exchange IEX. Results of minimising mean subtask dispersion from Chapter 3 and trading off mean subtask dispersion and mean task response time from Chapter 4 can be applied in this applications. The random variables are the distributions of the time it takes to send an order to each of the exchanges. The delays computed by Eq. 3.13 or Eq. 4.2 can be applied to the orders that are sent to the nearest exchanges, and no delays have to be applied to the orders that are sent to the exchanges that have the longest distance. This methodology will ensure that all orders arrive to the exchanges with different geographical distances simultaneously.

Secondly, in fork-join queueing systems, our methodology of applying delays from Chapter 5 can be applied in warehouses of online retailers, where each order is made up from a set of items, that have to be retrieved from a different place in the warehouse. Our methodology can be installed into the control system of the warehouse where optimal delays can be introduced before each item retrieval is initiated. It will apply delays computed by the dynamic online algorithm from Section 5.2 to some items that have a low retrieval time so that these items can finish their service at around the same time as their slower siblings resulting in *all* items arriving in the packing area at approximately at the same time. This will considerably lower the utilisation of the packing area.

Another application is in restaurants where we can develop an intelligent order management system based on Chapter 5 which suggests to the staff working in a kitchen when they have to start preparing dishes, so that all food comes to each table of customers at the same time. Obviously the application here is not direct, because preparing n copies of a dish takes less time that preparing them sequentially; the methodology needs some modification to take this feature into account, so that the service time in the queue depends on the number of waiting items in the queue. This order management system will ensure that all food will come to each table of customers at the same time without unnecessary delays.

6.3 Future Work

There are several possible extensions of this research. The following sections present future work for split-merge queueing systems and fork-join queueing systems, and discuss investigation of subtask dispersion reduction in general work-flows described by directed acyclic graphs.

6.3.1 Split-Merge Queueing System

Split-merge queueing systems with single class of tasks have been a focus of this thesis. However it would be interesting to investigate subtask dispersion in split-merge queueing systems with multiple classes of tasks. Previously only mean task response time was considered in multi-class parallel queueing networks [103, 4].

One more potential way to extend our research is to consider a split-merge queueing system with several classes of tasks, each of which splits into a different number of subtasks. This describes more accurately real-world systems such as warehouses of online retailers. In such warehouses usually each order consists of a number of items, which is less than the number of picking stations in the system.

Lastly, split-merge queueing system may be considered with very large number of parallel servers. Consequently this kind of system will bring an issue of complexity, where the complexity of the optimisation algorithm grows faster than the dimension of the problem (for discussion see Section 5.3).

6.3.2 Fork-Join Queueing System

In this thesis we considered elementary fork-join queueing systems with exponential service time distributions. This assumption of service times can perhaps, be extended to general service time distributions for elementary fork-join queueing systems, where the residual service times of general distributions can be computed from [11].

Furthermore, elementary fork-join queueing systems with very large number of parallel servers may be investigated, as in the split-merge case.

Throughout our research we have investigated reduction of subtask dispersion and compared it against simulations of parallel queueing systems. A possible extension is to validate our methodology in the context of real systems, such as warehouses of online retailers, restaurants and trading systems.

6.3.3 Directed Acyclic Graph Work-flows

The elementary queueing systems that we have considered in this thesis have a simple structure: there is a fork/split point, then there is *one* level of parallel servers and a join/merge point. In practice, workloads can follow a more complicated work-flow, which can be represented by a directed acyclic graph (DAG).

Not every subtask is processed concurrently in a system represented by a DAG structure. For example, in scheduling of a building construction, certain kinds of work can be done in parallel – for example, building several walls at the same time. Yet, a roof can be built only after a foundation and walls have been installed.

The generalisation of parallel processing systems into DAGs provides a new area of applications – project scheduling [86]. However, the complexity of some optimisation problems in this environment is NP-complete [41, 89, 90].

In comparison with parallel processing queueing systems, DAG work-flows have their own subtask dispersion-related metric called slack time. The aim is to keep this as low as possible. Under Just-In-Time scheduling, necessary products have to be produced and delivered at the instant where it is just needed; this scheduling has been adopted in many real-life applications such as [60].

Bibliography

[1] D. Aisen, B. Katsuyama, R. Park, J. Schwall, R. Steiner, A. Zhang, and T. L. Popejoy. Synchronized processing of data by networked computing resources. *US Patent 8489747 B2*, October 2011.

[2] K. Al-Begain, A. Dudin, and V. Mushko. Novel queuing model for multimedia over downlink in 3.5 G wireless networks. *Journal of Communications Software and Systems*, 2(2):68–80, 2006.

[3] M. M. Ali and M. N. Gabere. A simulated annealing driven multi-start algorithm for bound constrained global optimization. *Journal of Computational and Applied Mathematics*, 223(10):2661–2674, 2010.

[4] F. Alomari and D. Menascé. Efficient response time approximations for multiclass fork and join queues in open and closed queuing networks. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1–1, 2013.

[5] L. Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.

[6] S. Asmussen. *Applied probability and queues*, volume 2. Springer New York, 2003.

[7] S. Au-Yeung, P. Harrison, and W. Knottenbelt. A queueing network model of patient flow in an accident and emergency department. In *Proc. 20th Annual European and Simulation Modelling Conference*, pages 60–67, 2006.

[8] S. Au-Yeung, P. Harrison, and W. Knottenbelt. Approximate queueing network analysis of patient treatment times. In *Proceedings of the 2nd international conference on performance evaluation methodologies and tools*, page 45. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[9] S. W. M. Au-Yeung. *Response Times in Healthcare Systems*. PhD thesis, Imperial College London, January 2008.

[10] M. Avriel. *Nonlinear programming*. Dover Publications, 2003.

[11] T. Awagu. Variance reduction in manufacturing systems. *MEng Individual Project, Imperial College London*, 2013.

[12] P. Axer, S. Quinton, M. Neukirchner, R. Ernst, B. Dobel, and H. Hartig. Response-time analysis of parallel fork-join workloads with real-time constraints. In *25th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 215–224, 2013.

[13] F. Baccelli, A. M. Makowski, and A. Shwartz. The fork-join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds. *Advances in Applied Probability*, 21(3):pp. 629–660, 1989.

[14] F. Baccelli, W. A. Massey, and D. Towsley. Acyclic fork-join queuing networks. *J. ACM*, 36(3):615–642, July 1989.

[15] S. Balsamo and I. Mura. Approximate response time distribution in fork and join systems. *ACM SIGMETRICS Performance Evaluation Review*, 23(1):305–306, 1995.

[16] S. Balsamo and I. Mura. On queue length moments in fork and join queuing networks with general service times. In *Computer Performance Evaluation Modelling Techniques and Tools*, pages 218–231. Springer, 1997.

[17] R. B. Bapat and M. I. Beg. Order statistics for nonidentically distributed variables and permanents. *Sankhya: The Indian Journal of Statistics, Series A (1961-2002)*, 51(1):pp. 79–93, 1989.

[18] F. Bause and P. S. Kritzinger. *Stochastic Petri Nets: An Introduction to the Theory*, volume 26. ACM, New York, NY, USA, August 1998.

[19] K. Begain, G. Bolch, and H. Herold. *Practical performance modeling: application of the MOSEL language*. Kluwer Academic Publishers, 2001.

[20] G. Bolch et al. Algorithms for non-product-form networks. In *Queueing Networks and Markov Chains*, chapter 10, pages 421–556. J. Wiley & Sons, Inc., 2006.

[21] G. Bolch et al. *Queueing Networks and Markov Chains*. J. Wiley & Sons, Inc., 2006.

[22] C. G. Boncelet, JR. Algorithms to computer order statistic distributions. *SIAM Journal on Scientific and Statistical Computing*, 8(5):868–876, 1987.

[23] R. P. Brent. *Algorithms for Minimization Without Derivatives*. Dover Books on Mathematics. Dover Publications, 2002.

[24] J. Brutlag. Speed matters for google web search. `www.tribler.org/trac/raw-attachment/wiki/LivePlaylists/delayexp.pdf`, 2009.

[25] E. F. Burden and R. L. Burden. *Numerical Methods 3rd edition*. Cram101 Textbook Outlines. Academic Internet Publishers, 2006.

[26] G. Cao and M. West. Computing distributions of order statistics. *Communicat. in Statistics – Theory and Methods*, 26(3):755–764, 1997.

[27] 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 (CSUR)*, 26(2):145–185, 1994.

[28] H. A. David. *Order Statistics*. Wiley Series in Probability and Mathematical Statistics. John Wiley, 1980.

[29] H. A. David and H. N. Nagaraja. The non-IID case. In *Order Statistics*, chapter 5, pages 95–120. J. Wiley & Sons, Inc., 3rd edition, 2003.

[30] H. A. David and H. N. Nagaraja. *Order Statistics.* Wiley Series in Probability and Mathematical Statistics. John Wiley, third edition, 2003.

[31] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[32] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[33] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569, Sept. 1965.

[34] E. W. Dijkstra. The structure of the "THE"-multiprogramming system. *Communications of the ACM*, 11(5):341–346, May 1968.

[35] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, Nov. 1974.

[36] L. Flatto. Two parallel queues created by arrivals with two demands II. *SIAM Journal on Applied Mathematics*, 45(5):861–878, 1985.

[37] L. Flatto and S. Hahn. Two parallel queues created by arrivals with two demands I. *SIAM Journal on Applied Mathematics*, 44(5):1041–1053, 1984.

[38] S. Forman. How the âĂIJnavy sealsâĂİ of trading are taking on wall street‘s predatory robots. `http://qz.com/138388/how-the-navy-seals-of-trading-are-taking-on-wall-streets-predatory-robots/`, October 2013.

[39] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010.

[40] A. Gandhi, M. Harchol-Balter, and I. Adan. Server farms with setup costs. *Performance Evaluation*, 67(11):1123 – 1138, 2010.

[41] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

[42] M. Ghodsi and K. Kant. Performance analysis of parallel search algorithms on multiprocessors. In *Proceedings of the 14th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*, pages 407–421. North-Holland Publishing Co., 1990.

[43] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris. *Fundamentals of queueing theory*. Wiley, 2013.

[44] O. Guilbaud. Functions of non-iid random vectors expressed as functions of iid random vectors. *Scandinavian Journal of Statistics*, 9(4):pp. 229–233, 1982.

[45] R. W. Hamming. *The art of probability for Scientists and Engineers*. Addison Wesley Publishing Company, 1991.

[46] M. Harchol-Balter. *Performance modelling and design of computer systems*. Cambridge University press, 2013.

[47] P. G. Harrison. Reversed processes, product forms and a non-product form. *Linear Algebra and Its Applications*, 386:359–381, 2004.

[48] P. G. Harrison and N. M. Patel. *Performance Modelling of Communication Networks and Computer Architectures*. Addison-Wesley Longman Publishing Co., Inc., 1992.

[49] P. G. Harrison and S. Zertal. Queueing models of RAID systems with maxima of waiting times. *Perf. Evaluation*, 64(7–8):664–689, Aug 2007.

[50] P. Heidelberger and K. S. Trivedi. Analytic queueing models for programs with internal concurrency. *IEEE Transactions on Computers*, C-32(1):73–82, Jan 1983.

[51] R. Jain. *The art of computer systems performance analysis*. John Wiley & Sons, 1991.

[52] C. Kahraman. Multi-criteria decision making methods and fuzzy sets. In *Fuzzy Multi-Criteria Decision Making*, pages 1–18. Springer, 2008.

[53] 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.

[54] A. Y. Khinchin. The mathematical theory of a stationary queue. Technical report, DTIC Document, 1967.

[55] C. Kim and A. Agrawala. Analysis of the fork-join queue. *IEEE Transactions on Computers*, 38(2):250–255, Feb 1989.

[56] L. Kleinrock. Queueing systems. volume 1: Theory. Wiley-Interscience, 1975.

[57] W. J. Knottenbelt. *Parallel performance analysis of large Markov models.* PhD thesis, Imperial College London (University of London), 2000.

[58] W. J. Knottenbelt, I. Tsimashenka, and P. G. Harrison. Reducing subtask dispersion in parallel systems. In *Trends in Parallel, Distributed, Grid and Cloud Computing for Engineering*. Saxe-Coburg Publications, 2013.

[59] V. Krishna. *Auction theory.* Academic Press, 2009.

[60] M. Laguna and J. Velarde. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2(4):253–260, 1991.

[61] A. Lebrecht, N. Dingle, and W. Knottenbelt. A response time distribution model for zoned RAID. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 144–157. Springer, 2008.

[62] A. Lebrecht and W. J. Knottenbelt. Response Time Approximations in Fork-Join Queues. In *23rd Annual UK Performance Engineering Workshop (UKPEW)*, July 2007.

[63] A. S. Lebrecht, N. J. Dingle, P. G. Harrison, W. J. Knottenbelt, and S. Zertal. Using bulk arrivals to model I/O request response time distributions in zoned disks and RAID systems. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, page 23. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[64] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt. Modelling zoned RAID systems using fork-join queueing simulation. In *Computer Performance Engineering*, pages 16–29. Springer, 2009.

[65] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt. Analytical and simulation modelling of zoned RAID systems. *The Computer Journal*, 54(5):691–707, 2011.

[66] A. Lee and P. Longton. Queueing processes associated with airline passenger check-in. *OR*, pages 56–71, 1959.

[67] M. Lewis. *Flash Boys*. Penguin UK, 2014.

[68] R. M. Lewis, A. Shepherd, and V. Torczon. Implementing generating set search methods for linearly constrained minimization. *SIAM Journal on Scientfic Computing*, 29(6):2507–2530, 2007.

[69] J. Little. A proof for the queuing formula: L = λw. *Operations Research*, 9(3):383–387, 1961.

[70] Y. 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, 1991.

[71] M. Loève. *Probability Theory.* New York: D. Van Nostrand Company, 1955.

[72] R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[73] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[74] R. Nelson and B. Iyer. Analysis of a replicated data base. *Performance Evaluation*, 5(3):133–148, 1985.

[75] R. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739 –743, 1988.

[76] R. Nelson, D. Towsley, and A. N. Tantawi. Performance analysis of parallel processing systems. *Software Engineering, IEEE Transactions on*, 14(4):532–540, 1988.

[77] C. H. Ng. *Queueing Modelling Fundamentals*. John Wiley & Sons, 1997.

[78] J. Nocedal and S. J. Wright. Derivative-free optimization. In *Numerical optimization*, chapter 3, pages 220–244. Springer Series in Operations Research and Financial Engineering, 2nd edition, 1999.

[79] J. Nocedal and S. J. Wright. Line search methods. In *Numerical optimization*, chapter 3, pages 30–65. Springer Series in Operations Research and Financial Engineering, 2nd edition, 1999.

[80] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering, 2nd edition, 1999.

[81] E. Pearson and C. Sekar. The efficiency of statistical tools and a criterion for the rejection of outlying observations. *Biometrika*, 28(3/4):308–320, 1936.

[82] F. Pollaczek. Über eine Aufgabe der Wahrscheinlichkeitstheorie. I. *Mathematische Zeitschrift*, 32(1):64–100, 1930.

[83] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical recipes in C: The art of scientific computing, 1992.

[84] S. M. Ross. *Introduction to probability models*. Academic Press, 6th edition, 2006.

[85] T. Rychlik. Projecting statistical functionals. In *Lecture Notes in Statistics*, volume 160. Springer, 2001.

[86] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. Dikaiakos. Scheduling workflows with budget constraints. In S. Gorlatch and M. Danelutto, editors, *Integrated Research in Grid Computing*, pages 189–202. Springer, 2007.

[87] Y. Sathe and U. Dixit. On a recurrence relation for order statistics. *Statistics & probability letters*, 9(1):1–4, 1990.

[88] P. K. Sen. A note on order statistics for heterogeneous distributions. *The Annals of Mathematical Stat.*, 41(6):pp. 2137–2139, 1970.

[89] Z. Shi and J. J. Dongarra. Scheduling workflow applications on processors with different capabilities. *Future Generation Computer Systems*, 22(6):665 – 675, 2006.

[90] O. Sinnen and L. Sousa. Communication contention in task scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, 16(6):503–515, 2005.

[91] J. Smith. Regional flood frequency analysis using extreme order statistics of the annual peak record. *Water Resources Research*, 25(2):311–317, 1989.

[92] J. Sun and G. D. Peterson. An effective execution time approximation method for parallel computing. IEEE Transactions of Parallel and Distributed Systems, January 2012.

[93] D. Towsley, S. Chen, and S.-P. Yu. Performance analysis of a fault tolerant mirrored disk system. In *Proceedings of the 14th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*, pages 239–253. North-Holland Publishing Co., 1990.

[94] 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.

[95] K. Trivedi. *Probability and statistics with reliability, queuing, and computer science applications*. Prentice-hall Englewood Cliffs, 1982.

[96] I. Tsimashenka and W. Knottenbelt. Trading off subtask dispersion and response time in split-merge systems. In *Analytical and Stochastic Modeling Techniques and Applications (ASMTA'13)*, Lecture Notes in Computer Science, 2013.

[97] I. Tsimashenka and W. J. Knottenbelt. Reduction of Variability in Split-Merge Systems. In *Imperial College Computing Student Workshop (ICCSW 2011)*, pages 101–107, September 2011.

[98] I. Tsimashenka and W. J. Knottenbelt. Reduction of subtask dispersion in fork-join systems. In *Computer Performance Engineering*, pages 325–336. Springer, 2013.

[99] I. Tsimashenka, W. J. Knottenbelt, and P. Harrison. Controlling variability in split-merge systems. In *Analytical and Stochastic Modeling Techniques and Applications (ASMTA'12)*, volume 7314 of *Lecture Notes in Computer Science*, pages 165–177. Springer, June 2012.

[100] I. Tsimashenka, W. J. Knottenbelt, and P. G. Harrison. Controlling variability in split–merge systems and its impact on performance. *Annals of Operations Research*, pages 1–20.

[101] L. G. Underhill. *Introstat*. Juta and Company Ltd, 1987.

[102] N. M. van Dijk. Why queuing never vanishes. *European Journal of Operational Research*, 99(2):463–476, 1997.

[103] E. Varki. Mean value technique for closed fork-join networks. In *Proc. of the ACM SIGMETRICS 1999*, SIGMETRICS '99, pages 103–112, New York, NY, USA, 1999. ACM.

[104] 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*, 2008.

[105] S. Varma and A. M. Makowski. Interpolation approximations for symmetric fork-join queues. *Performance Evaluation*, 20(1âĂŞ3):245–265, 1994.

[106] R. J. Vaughan and W. N. Venables. Permanent expressions for order statistic densities. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):pp. 308–310, 1972.

[107] F. Wan, N. Dingle, W. Knottenbelt, and A. Lebrecht. Simulation and modelling of RAID 0 system performance. In *22nd Annual European Simulation and Modelling Conference (ESM)*, pages 145–149. Citeseer, 2008.

[108] P. Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.

[109] P. Wolfe. Convergence conditions for ascent methods. II: Some corrections. *SIAM review*, 13(2):185–188, 1971.

[110] R. Wolff. Poisson arrivals see time averages. *Operations Research*, 30:223–231, 1982.

[111] M. Zaharia et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proc. 5th European Conference on Computer Systems (EuroSys '10)*, pages 265–278, 2010.

Appendix A

Proof of the Mean of Range of Heterogeneous Order Statistics Convexity

This proof was done in collaboration with Dr. Richard Hayden. Here we prove that the mean of the range of heterogeneous order statistics

$$\mathbb{E}[D] = \mathbb{E}\big[X_{(n)}\big] - \mathbb{E}\big[X_{(1)}\big]$$

is convex. Let

$$g(\mathbf{X}, \mathbf{d}) = \max_i\{X_i + d_i\} - \min_i\{X_i + d_i\}$$

So the cost function $\mathbb{E}[D_{\mathbf{d}}] = \mathbb{E}\big[g(\mathbf{X}, \mathbf{d})\big]$.

Claim: For fixed $\mathbf{x}$ we have $g(\mathbf{x}, d_i)$ is convex

Proof. Firstly, $x_i + d_i$ is trivially convex.

Secondly, for maximum $\max\{x_1 + d_1, \ldots, x_n + d_n\}$ is convex as maximum of convex functions is convex.

For the minimum: $-\min\{x_1 + d_1, \ldots, x_n + d_n\}$ is convex because $-\min\{x_1 + d_1, \ldots, x_n + d_n\} = \max\{-x_1 - d_1, \ldots, -x_n - d_n\}$, maximum is convex and $-x_i - d_i$ is convex.

Consequently, $\max\{x_1 + d_1, \ldots, x_n + d_n\} - \min\{x_1 + d_1, \ldots, x_n + d_n\}$ is convex as required.

For the convex functions the following inequality should hold:

$$g(\mathbf{x}, \lambda\mathbf{d} + (1-\lambda)\mathbf{d}') \leq \lambda g(\mathbf{x}, \mathbf{d}) + (1-\lambda) g(\mathbf{x}, \mathbf{d}')$$

Therefore if the function $g(\mathbf{x}, \cdot)$ is convex then $\mathbb{E}\big[g(\mathbf{X}, \cdot)\big]$ is convex:

$$\mathbb{E}\big[g(\mathbf{X}, \lambda\mathbf{d} + (1-\lambda)\mathbf{d}')\big] \leq \lambda \mathbb{E}\big[g(\mathbf{X}, \mathbf{d})\big] + (1-\lambda)\mathbb{E}\big[g(\mathbf{X}, \mathbf{d}')\big]$$

□