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

Iryna Tsimashenka and William J. Knottenbelt

Imperial College London, 180 Queen's Gate,
London SW7 2AZ, United Kingdom,
Email: {it09,wjk}@doc.ic.ac.uk

Abstract. In many real-world systems incoming tasks split into subtasks which are processed by a set of parallel servers. In such systems two metrics are of potential interest: response time and subtask dispersion. Previous research has been focused on the minimisation of one, but not both, of these metrics. In particular, in our previous work, we showed how the processing of selected subtasks can be delayed in order to minimise expected subtask dispersion and percentiles of subtask dispersion in the context of split-merge systems. However, the introduction of subtask delays obviously impacts adversely on task response time and maximum sustainable system throughput. In the present work, we describe a methodology for managing the trade off between subtask dispersion and task response time. The objective function of the minimisation is based on the product of expected subtask dispersion and expected task response time. Compared with our previous methodology, we show how our new technique can achieve comparable subtask dispersion with substantial improvements in expected task response time.

Keywords: Split-Merge System, Subtask Dispersion, Response Time, Trade-Off

1 Introduction

In the modern world we are surrounded by systems in which incoming tasks can naturally be decomposed into subtasks that are processed by a set of parallel servers. Completed subtasks are held in an output buffer pending arrival of its sibling subtasks. When all child subtasks of a given task have completed service and are present in the output buffer, the task is deemed to have completed service. Two classes of performance metrics are of concern in such systems: those related to task response time and those related to subtask dispersion (i.e. the time between the arrival of the first and last subtasks originating from a given task in the output buffer). The former class has been the subject of extensive research [2, 3, 6, 7, 9–11, 13, 14, 16, 18, 21, 22], while the latter class has received less attention [19, 20], although reducing subtask dispersion can be of critical importance in certain real-life contexts.

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.

Parallel queueing systems are natural modelling abstractions for these kinds of systems. Here we focus on an analytically tractable subclass of such systems, namely split-merge systems. In our previous research [19,20] we showed that reduction of subtask dispersion in such systems can be achieved through the application of judiciously chosen delays to subtask processing. This naturally has an adverse impact on task response time (as characterised in [12]), and therefore on maximum sustainable system throughput. This adverse impact increases with workload intensity, and can even result in previously stable systems becoming unstable. There is therefore an urgent need to define an analtyical framework which allows effective balancing of subtask dispersion and response time concerns. To this end, we formally characterise the subtask dispersion–response time trade-off in split-merge systems as an optimisation problem. The objective function is the product of the expected subtask dispersion and expected task response times. We were inspired in this research by the survey on multi-objective optimisation techniques by Marler and Arora [15], in particular the exposition of product methods, as well as the work of Gandhi, Harchol-Balter et al. [8] which explored energy–performance trade-offs in server farms by means of an objective function based on the energy–response time product (ERP).

The remainder of this paper organised as follows. Section 2 presents definitions of split-merge queueing systems, and important results related to the theory of homogeneous and heterogeneous order statistics. Section 3 characterises the subtask dispersion–response time trade-off in split-merge systems and presents a methodology for its optimisation based on a modified Newton's method. Section 4 presents a case study which illustrates the benefits of our proposed approach in comparison to approaches based on the unique minimisation of either subtask dispersion or task response time. Section 5 concludes and considers directions for future work.

2 Preliminaries

2.1 Split-Merge Systems

As shown in Fig. 1, a *split-merge system* consists of split and merge points, a queue before the split point (a split queue) and N heterogeneous parallel servers with queueing capability after service (merge buffers). Tasks enter the split queue according to a Poisson process with mean rate λ. Whenever all parallel 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 is sent to its allocated parallel server where it is served according to a general service time distribution with

mean $1/\mu_i,\ i = 1, ..., N$. Completed subtasks enter a merge buffer. When all subtasks (originating from a particular task) are present in the merge buffers, the original task exits the system via the merge point.

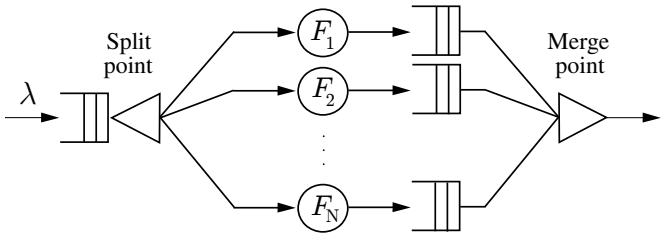

Fig. 1: Split–Merge queueing model.

In the Fig. 1 the merge buffers have been shown as separate entities but in many real applications they share the same physical space, which we will term the *output buffer*.

2.2 Theory of Order Statistics [5].

The theory of order statistics studies the behaviour and properties of arranged random variables and the statistics derived from them [5].

Definition 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 increasing order $X_{(1)} \leqslant X_{(2)} \leqslant \ldots \leqslant X_{(n)}$. Then $X_{(i)}$ is 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 the $X_{(i)}$ are dependent because of the ordering. The extreme values $X_{(1)}$ and $X_{(n)}$ are the minimum and maximum order statistics respectively. $T = X_{(n)} - X_{(1)}$ is the *range*.

2.3 Theory of Heterogeneous Order Statistics

Relaxing the assumption that the X_i should be identically distributed leads to the theory of heterogeneous order statistics [4, 20].

Definition 2. *We consider n independent, real-valued random variables $X_1, \ldots, X_n$ where each X_i has an arbitrary probability distribution $F_i(t)$ and probability density function $f_i(t) = F_i'(t)$. In this case of "heterogeneous" (or independent, but not necessarily identically distributed) random variables, we call the corresponding order statistics* heterogeneous order statistics.

The cumulative distribution functions of the minimum and maximum order statistics are:

$$F_{(1)}(t) = Pr\{X_{(1)} \leqslant t\} = 1 - \prod_{i=1}^{n}[1 - F_i(t)],$$

and

$$F_{(n)}(t) = Pr\{X_{(n)} \leqslant t\} = \prod_{i=1}^{n} F_i(t).$$

In addition the cumulative distribution function of the range $X_{(n)} - X_{(1)}$ is [20]:

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

3 Subtask Dispersion–Response Time Trade-off

3.1 Metrics for split-merge systems

Two important metrics of operational interest in split-merge systems are subtask dispersion, defined as the difference between the arrival times of the first and last subtasks originating from a given task in the output buffer, and task response time, defined as the difference between the arrival time of a task in the split queue and the time at which the task leaves the system via the merge point. There is a conflicting tension between these two metrics. Indeed, in previous research we have considered how to minimise mean subtask dispersion [12, 19] and percentiles of subtask dispersion [20] by delaying the processing of the subtasks in such a way as to cluster subtask completion times. 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 (as quantified in [12]), with a corresponding reduction in maximum sustainable system throughput.

3.2 Application of heterogeneous order statistics to split-merge systems

We consider a split-merge system with i parallel servers. Suppose $X_i \sim F_i(x)$ is a random variable that describes the (heterogenous) 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 start service at the same time). 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 range $X_{(n)} - X_{(1)}$ corresponds to subtask dispersion.

3.3 Introducing deterministic subtask delays

As in our previous research, 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 the delay applied to the processing times of 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 $X_{(1)}^{\mathbf{d}}, X_{(2)}^{\mathbf{d}}, \ldots, X_{(n)}^{\mathbf{d}}$.

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

In order to express the subtask dispersion–response time trade-off for a split-merge system subject to subtask delay vector $\mathbf{d}$ we create an objective function formed from the product of mean task response time ($\mathbb{E}[R_{\lambda,\mathbf{d}}]$) and mean subtask dispersion ($\mathbb{E}[D_{\mathbf{d}}]$). The former metric is computed using the Pollaczek-Khinchine formula for mean task response time in an $M/G/1$ queue with service distribution $X_{(n)}^{\mathbf{d}}$. This is because a split-merge 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 $\mu^{-1} = \mathbb{E}[X_{(n)}^{\mathbf{d}}]$). The latter metric is computed as the expected difference between the maximum and minimum heterogeneous order statistics. We note dependence between order statistics is irrelevant when considering mean values due to the linearity property of expectation operator, i.e. $\mathbb{E}[D_{\mathbf{d}}] = \mathbb{E}[X_{(n)}^{\mathbf{d}} - X_{(1)}^{\mathbf{d}}] = \mathbb{E}[X_{(n)}^{\mathbf{d}}] - \mathbb{E}[X_{(1)}^{\mathbf{d}}]$.

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

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

The variance of $X_{(n)}^{\mathbf{d}}$ can be computed as:

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

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. We note that, in line with the treatment of weighted product methods in [15], each component can be raised to a different exponent (> 1) in order to express a preference about the relative importance of the components.

3.5 Optimising the objective function

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

$$
\mathbf{d}_{\text{min}} = \arg\min_{\mathbf{d}} T(\mathbf{d}, \lambda) \qquad (3)
$$

We can apply Newton's method to find $\mathbf{d}_{\text{min}}$ iteratively:

$$\mathbf{d}_{k+1} \;=\; \mathbf{d}_k - \gamma\big[H_{T(\mathbf{d}_k,\lambda)}\big]^{-1}\nabla T(\mathbf{d}_k,\lambda),\; k \geq 0 \tag{4}$$

where $H_{T(\mathbf{d}_k,\lambda)}$ 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:

$$\mathbf{d}_0 = ((\max_i \mathbb{E}\big[X_i\big] - \mathbb{E}\big[X_1\big])(1-\rho), \ldots, (\max_i \mathbb{E}\big[X_i\big] - \mathbb{E}\big[X_n\big])(1-\rho)) \tag{5}$$

where $\rho = \lambda/\mu$.

In Eq. (4) $\gamma < 1$ is a constant introduced to satisfy the Wolfe conditions [23, 24]. 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. There are two inequalities which must hold to ensure this. Firstly:

$$\begin{aligned} T\Big(\mathbf{d}_k + \gamma\big(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)\big),\lambda\Big) \leq \\ T(\mathbf{d}_k,\lambda) + c_1\gamma\nabla T^T(\mathbf{d}_k,\lambda)\big(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)\big) \end{aligned} \tag{6}$$

And secondly:

$$\begin{aligned} \nabla T^T\big(\mathbf{d}_k + \gamma(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)),\lambda\big)\big(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)\big) \geq \\ c_2\nabla T^T(\mathbf{d}_k,\lambda)\big(-H^{-1}_{T(\mathbf{d}_k,\lambda)}\nabla T(\mathbf{d}_k,\lambda)\big) \end{aligned} \tag{7}$$

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, for the purposes of Newton and quasi-Newton methods it is recommended to set c_1 as 10^{-4} and for c_2 to take on 0.9 [17]. Inequality (6) above corresponds to the Armijo rule [1] which guarantees that the step size γ will decrease the objective function sufficiently, while Inequality (7) ensures the curvature condition.

3.6 Implementation

We have implemented the above optimisation technique in C++. Using Newton's method, we begin by initialising $\mathbf{d}_0$ according to Eq. (5) and choose an appropriate γ satisfying Inequalities (6) and (7). 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). The vector of optimal delays is deemed to be found as $\mathbf{d}_k$ when

$$\frac{\partial T(\mathbf{d}_k,\lambda)}{\partial \mathbf{d}_k} = 0.$$

4 Case Study

Consider a split-merge system with 3 parallel servers having heterogeneous service time density functions (as considered in [20]):

$$f_1(t) = \text{Pareto}(\alpha = 3, b = 3.5) \quad (E[X_1] = 5.25,\ \text{Med}[X_1] = 4.41,\ \text{Var}[X_1] = 9.19)$$
$$f_2(t) = \text{Erlang}(n = 2, \lambda = 1) \quad (E[X_2] = 2,\ \text{Med}[\text{X}_2] = 1.68,\ \text{Var}[X_2] = 2)$$
$$f_3(t) = \text{Det}(5) \quad (E[X_3] = 5,\ \text{Med}[X_3] = 5,\ \text{Var}[X_3] = 0)$$

Without adding any subtask delays, mean subtask dispersion is $\mathbb{E}[D_{\mathbf{d}=\mathbf{0}}] = 3.576$ time units and maximum sustainable task throughput is $\lambda_{\text{max}} = 0.182$ tasks per time unit.

For $\lambda = 0.01$, mean task response time is $\mathbb{E}[R_{\mathbf{d}=\mathbf{0},0.01}] = 5.651$ time units, while for $\lambda = 0.15$, mean task response time is $\mathbb{E}[R_{\mathbf{d}=\mathbf{0},0.15}] = 18.658$ time units.

Using our previously developed optimisation technique designed to minimise mean subtask dispersion without regard to impact on response time [19] we obtain the vector of optimal subtask delays:

$$\mathbf{d}_D = (0.608, 3.372, 0)$$

as shown in Fig. 2.

After applying these delays, maximum sustainable throughput drops to $\lambda_{\text{max}} = 0.161$ and mean subtask dispersion improves to $\mathbb{E}[D_{\mathbf{d}_D}] = 1.72354$. For $\lambda = 0.01$, mean task response time rises to $\mathbb{E}[R_{\mathbf{d}_D,0.01}] = 6.434$, a 14% increase. For $\lambda = 0.15$ mean task response time dramatically increases to $\mathbb{E}[R_{\mathbf{d}_D,0.15}] = 51.382$, an increase of 175%.

Now we optimise the same system except under the subtask dispersion–response time trade-off developed in the present paper. With $\lambda = 0.01$ we obtain the following vector of optimal subtask delays:

$$\mathbf{d}_T = (0.453, 3.002, 0)$$

as shown in Fig. 3. Mean subtask dispersion becomes $\mathbb{E}[D_{\mathbf{d}_T}] = 1.755$ time units, which is only 1.8% higher than the dispersion obtained under delay vector $\mathbf{d}_D$. Mean task response time is $\mathbb{E}[R_{\mathbf{d}_T,001}] = 6.24$ (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 $\lambda = 0.15$ the vector of optimal subtask delays drops to:

$$\mathbf{d}_T = (0.0928, 2.043, 0.0)$$

as shown in Fig. 4. Mean subtask dispersion is now $\mathbb{E}[D_{\mathbf{d}_T}] = 2.079$, a 21% increase over the dispersion obtained under delay vector $\mathbf{d}_D$, but still a good improvement over the system without added delays (3.576). The corresponding distributions of subtask dispersion are shown in Fig. 5. 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 $\mathbb{E}[R_{\mathbf{d}_T,0.15}] = 22.875$, which is 23% worse than the system without delays but a 55% improvement on mean task response time under delay vector $\mathbf{d}_D$. The corresponding distributions of response time are shown in Fig. 6. It is apparent that the trade-off is able to maintain competitive response times as compared to the system with no delays, in stark contrast with our previous methodology.

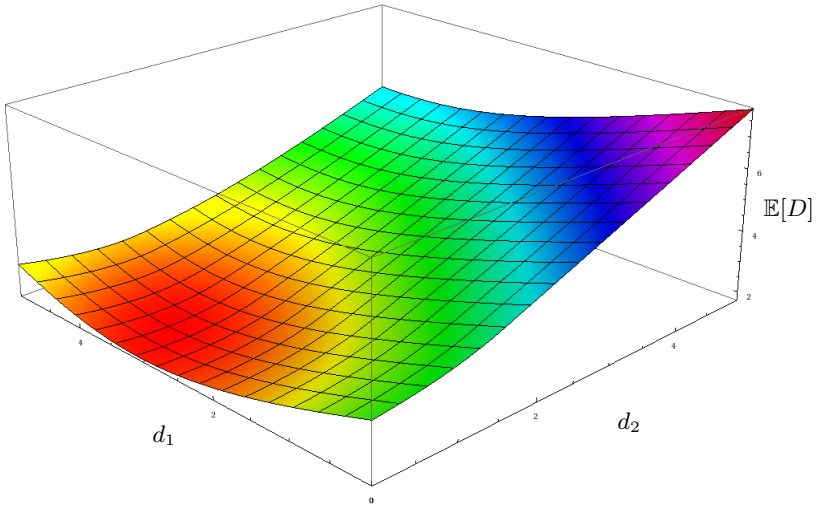

Fig. 2: Surface plot of mean subtask dispersion against subtask delays using our previous methodology from [19].

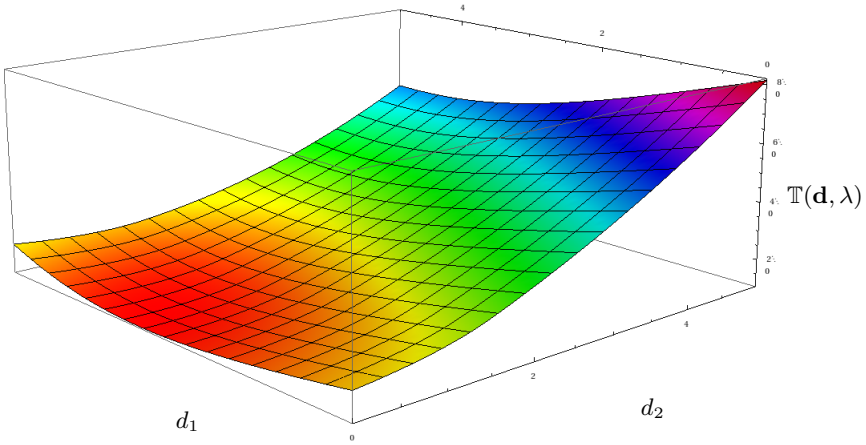

Fig. 3: Surface plot of subtask dispersion–response time trade-off objective function against subtask delays for $\lambda = 0.01$.

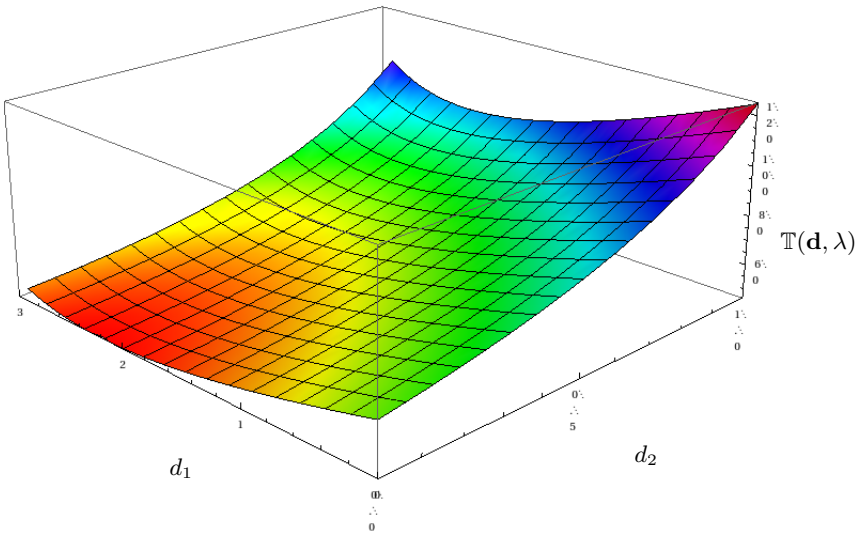

Fig. 4: Surface plot of subtask dispersion–response time trade-off objective function against subtask delays for $\lambda = 0.15$.

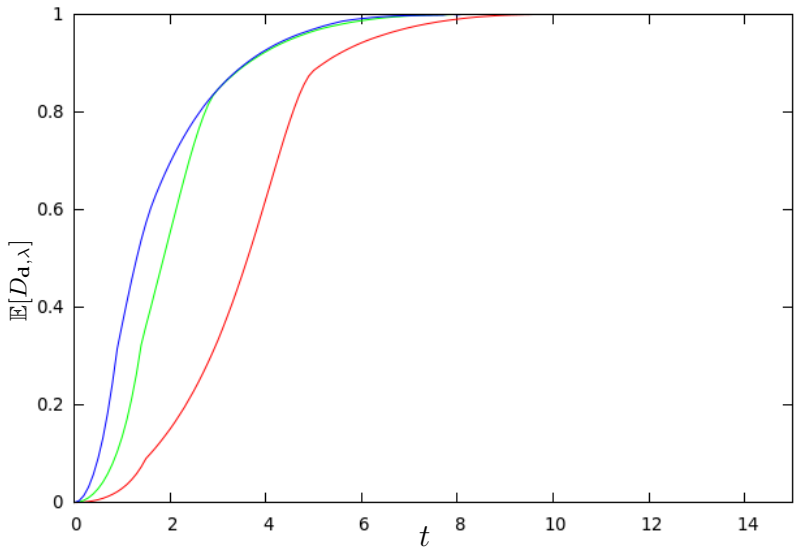

Fig. 5: Distributions of subtask dispersion with $\lambda = 0.15$ without any delays (red line) with delays optimised for $\mathbb{E}[T_{\mathbf{d},0.15}]$ (green line) and for $\mathbb{E}[D_{\mathbf{d}}]$ (blue line).

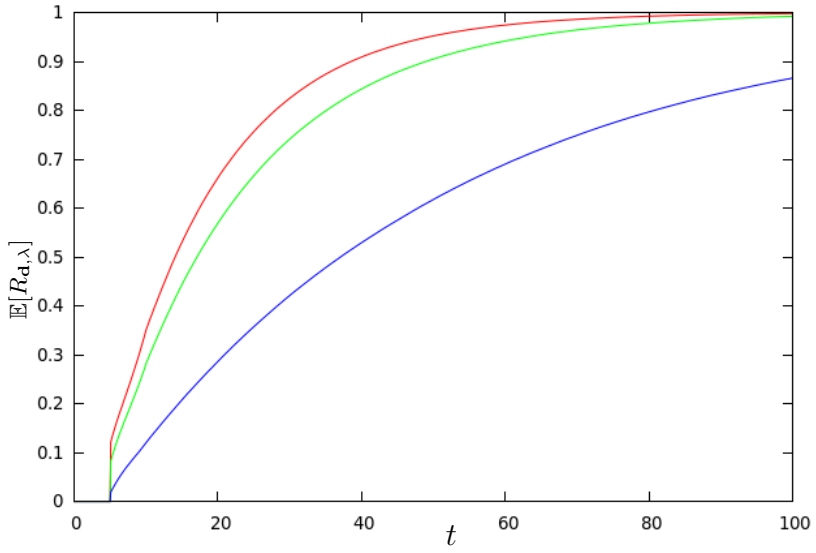

Fig. 6: Distributions of task response time given $\lambda = 0.15$, without any delays (red line), with delays optimised for $\mathbb{E}[T_{\mathbf{d},0.15}]$ (green line) and delays optimised for $\mathbb{E}[D]$ (blue line).

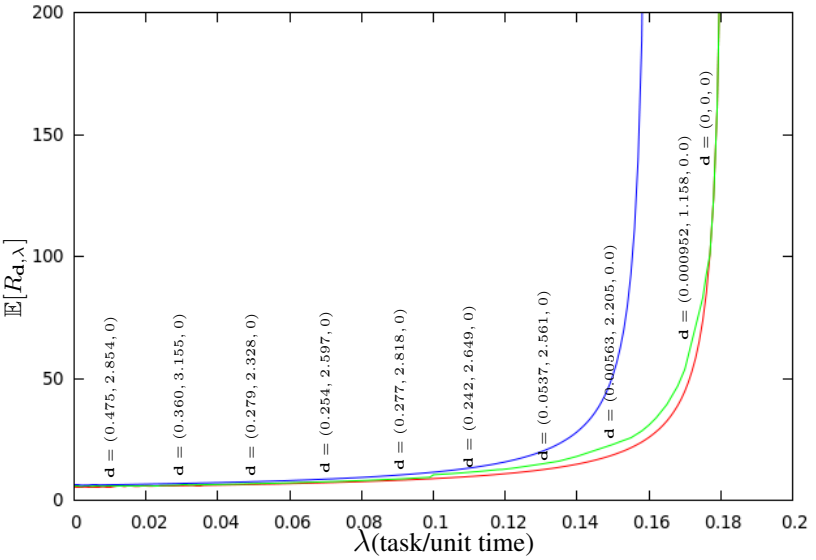

Fig. 7: Expected response time of case study split-merge system for various customer arrival rates without any delays (red line), with delays optimised for subtask dispersion–response time trade-off (green line), and with delays optimised for mean subtask dispersion alone (blue line). Subtask delay vectors are also shown for the subtask dispersion–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. [12]), 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. 7 shows how the vector of optimal delays changes with λ, and how it converges to the zero-vector as λ approaches $\lambda_{\text{max}} = 0.182$. We note that the maximum sustainable throughput of the system optimised under our previous methodology 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.

5 Conclusion

In this paper we have described a framework for delaying the dispatch of subtasks to parallel servers in split-merge 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 response time. Previous research has concentrated on the optimisation of each one of these metrics in isolation, frequently resulting in signficant 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.

One possible direction for future work is to develop similar techniques for the optimisation of fork-join systems, which are less synchronised – and much less analytically tractable – relatives of split-merge systems. Although this is expected to be a very challenging exercise, it would open up the application of our technique to an even broader range of real-world systems.

References

1. Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.
2. 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.
3. F. Baccelli, W. A. Massey, and D. Towsley. Acyclic fork-join queuing networks. *J. ACM*, 36(3):615–642, July 1989.
4. 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.
5. H. A. David and H. N. Nagaraja. *Order Statistics*. Wiley Series in Probability and Mathematical Statistics. John Wiley, third edition, 2003.
6. L. Flatto and S. Hahn. Two parallel queues created by arrivals with two demands I. *SIAM Journal on Applied Mathematics*, 44(5):pp.1041–1053, 1984.
7. Leopold Flatto. Two parallel queues created by arrivals with two demands ii. *SIAM Journal on Applied Mathematics*, 45(5):pp. 861–878, 1985.

8. Anshul Gandhi, Varun Gupta, Mor Harchol-Balter, and Michael A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010.
9. 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.
10. 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.
11. C. Kim and A.K. Agrawala. Analysis of the fork-join queue. *IEEE Transactions on Computers*, 38(2):250–255, Feb 1989.
12. 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.
13. A. Lebrecht and W. J. Knottenbelt. Response Time Approximations in Fork-Join Queues. In *23rd Annual UK Performance Engineering Workshop (UKPEW)*, July 2007.
14. J. C. S. Lui, R. R. Muntz, and D. Towsley. Computing performance bounds of fork-join parallel programs under a multiprocessing environment. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):295 –311, 1998.
15. R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.
16. R. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739 –743, 1988.
17. Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer verlag, 1999.
18. 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.
19. I. Tsimashenka and W. J. Knottenbelt. Reduction of Variability in Split-Merge Systems. In *Imperial College Computing Student Workshop (ICCSW 2011)*, pages 101–107, 2011.
20. 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.
21. E. Varki. Response time analysis of parallel computer and storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1146–1161, Nov 2001.
22. S. Varma and A. M. Makowski. Interpolation approximations for symmetric fork-join queues. *Performance Evaluation*, 20(13):245–265, 1994.
23. Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.
24. Philip Wolfe. Convergence conditions for ascent methods. II: Some corrections. *SIAM review*, 13(2):185–188, 1971.