Reduction of Subtask Dispersion in Fork-Join 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. Fork-join and split-merge queueing systems are well-known abstractions of parallel systems in which each incoming task splits into subtasks that are processed by a set of parallel servers. A task exits the system when all of its subtasks have completed service. Two key metrics of interest in such systems are task response time and subtask dispersion. This paper presents a technique applicable to a class of fork-join systems with heterogeneous exponentially distributed service times that is able to reduce subtask dispersion with only a marginal increase in task response time. Achieving this is challenging since the unsynchronised operation of fork-join systems naturally militates against low subtask dispersion. Our approach builds on our earlier research examining subtask dispersion and response time in split-merge systems, and involves the frequent application and updating of delays to the subtasks at the head of the parallel service queues. Numerical results show the ability to reduce dispersion in fork-join systems to levels comparable with or below that observed in all varieties of split-merge systems while retaining the response time and throughput benefits of a fork-join system.

Keywords: Fork-Join System, Subtask Dispersion, Task Response Time

1 Introduction

Nowadays parallel systems are becoming more prevalent than ever, with large automated warehouses, concurrent computing systems and distributed storage systems taking centre stage in the world of industry. Despite the fact that performance and operational efficiency are primary concerns in these systems, there are significant challenges from a modelling perspective in predicting and optimising their dynamic behaviour.

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. This paper focuses on two subclasses of queueing network models for parallel systems, namely *fork-join* and *split-merge* systems [3]. Definitions and operational characteristics of each of these two kinds of system are presented in the next section.

Two performance metrics of interest in these systems are *task response time* – that is the time taken from the entry of a task into the system until its exit – and *subtask dispersion* – that is the difference in time between the service completions of the first and last subtasks originating from a given task. Since subtasks in a fork-join system

are subject to less synchronisation than those in a split-merge system, the structure of a fork-join system naturally yields lower response times but higher subtask dispersion when compared to a split-merge system with similar parallel service time distributions.

In this paper we present an online technique for applying judiciously-chosen delays to subtask processing times in elementary fork-join systems with heterogeneous exponential service times. The technique reduces subtask dispersion significantly with only a marginal impact on task response time. Our method assumes non-preemptive scheduling; that is, once subtasks begin service they are executed to completion. Although preemption gives more flexibility for scheduling from a theoretical perspective, preemptive scheduling can lead to considerable overhead when applied in practice [12].

This paper makes the following specific contributions over our previous work exploring subtask dispersion and task response time in parallel systems [15, 24–26]:

1. We extend our modelling capability to fork-join systems, rather than split-merge systems. Since fork-join systems are more widely deployed in practice owing to their greater efficiency, this means our present technique is more applicable to the realistic modelling of modern parallel systems.
2. In contrast to our previous algorithms which were static, the method we present here is a dynamic online one that is sensitive to the current state of the system. Not only is it to be expected that a dynamic method will outperform any static one – at least in the absence of significant scheduling overhead (see e.g. [13, 16, 19, 21]), but also our dynamic method can support non-stationary workloads.

The remainder of the paper is organised as follows. Section 2 presents relevant preliminaries including details of the parallel processing systems considered and the theory of homogeneous and heterogenous order statistics (subsequently applied in computing state-dependent subtask delays). Section 3 elaborates on the two performance metrics we consider and recaps important results from the literature related to each metric and the trade-off between them. Section 4 describes our method for the online control of subtask dispersion. Section 5 presents numerical results showing the ability of our methodology to simultaneously achieve low subtask dispersion (better even that than achieved by the best static algorithm for reducing subtask dispersion in split-merge systems), and low response time (only slightly higher than a fork-join system without subtask delays). Section 6 concludes.

2 Preliminaries

2.1 Parallel Processing Systems

Fork-Join An *elementary fork-join system* (see Fig. 1) is composed of N parallel heterogeneous FCFS service queues, fork and join points and join queues (join buffers) for completed subtasks [3]. 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. Here we assume parallel server i processes its queue of subtasks according to an exponential service time distribution with mean service time

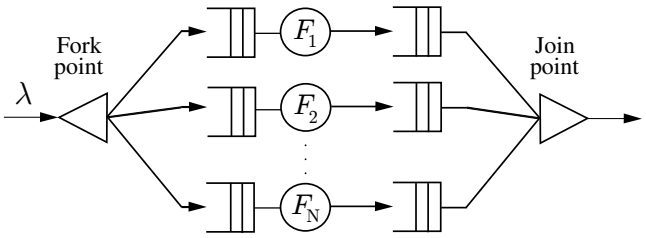

Fig. 1: Fork–Join queueing model.

$1/\mu_i, i = 1, \ldots, N$. 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.

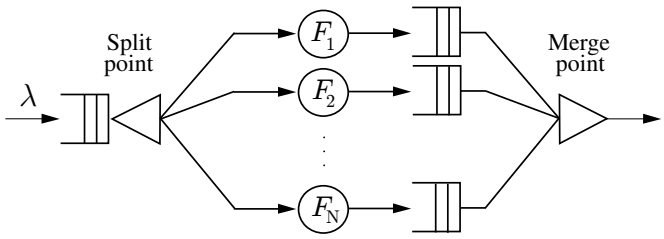

Fig. 2: Split–Merge queueing model.

Split-Merge A more synchronised type of parallel system is the *split-merge system* (see Fig. 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 time $1/\mu_i,\ i = 1, \ldots, N$). 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.

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.

2.2 Theory of Order Statistics

Ordinary (homogeneous) order statistics [6] enable reasoning about *sorted* samples drawn from independent random variables having the same underlying distribution.

Definition 1. *If iid random variables $X_1, X_2, \ldots, X_n$ each having distribution $F(x)$ are arranged in the increasing order $X_{(1)} \leq X_{(2)} \leq \ldots \leq X_{(n)}$, then $X_{(i)}$ is the ith* order statistic *($1 \leq i \leq n$).*

The extremes are given by $X_{(1)}$ (the minimum order statistic), and $X_{(n)}$ (the maximum order statistic). $X_{(n)} - X_{(1)}$ is the range.

2.3 Theory of Heterogeneous Order Statistics

Heterogeneous order statistics [7] enable reasoning about sorted samples drawn from independent, but not necessarily identically distributed (inid) random variables.

Definition 2. *If inid random variables $X_1, X_2, \ldots, X_n$ each having distribution $F_i(x)$, are arranged in the increasing order $X_{(1)} \leq X_{(2)} \leq X_{(3)} \leq \ldots \leq X_{(n)}$, then $X_{(k)}$ is the k^{th}* heterogeneous order statistic *having corresponding cdf $F_{(k)}(x)$ ($1 \leq k \leq n$).*

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

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

and

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

The cumulative distribution function of the range $X_{(n)} - X_{(1)}$ is [26]:

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

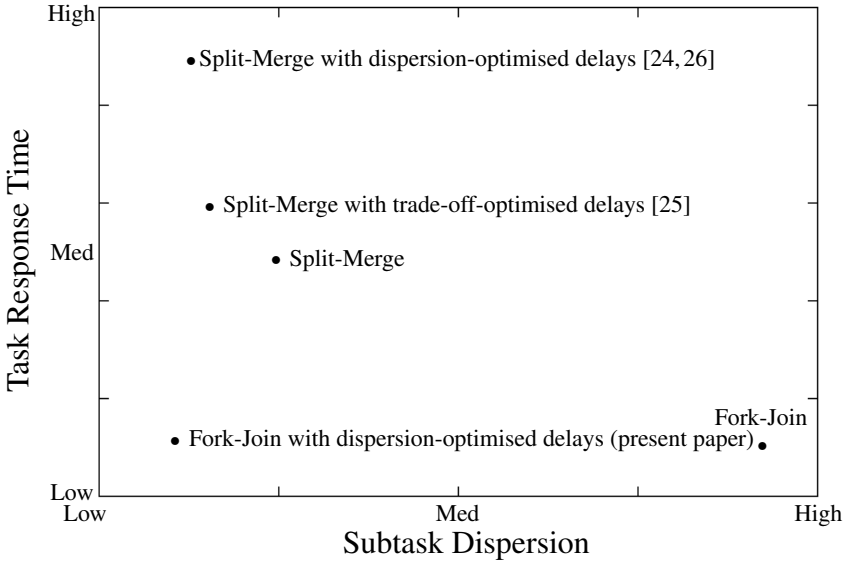

Fig. 3: Task response time vs. subtask dispersion of fork-join and split-merge queues with and without optimised subtask delays.

3 Metrics

There are two important metrics in fork-join and split-merge 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. [1, 2, 4, 5, 8–11, 14, 17, 18, 20, 22, 23, 27–30]). The vast majority of this work targets the mean (and rarely higher moments) of task response time and/or the stationary distribution of the number of tasks queued at parallel servers.
- **Subtask Dispersion**, that is the difference in time between the service completions of the first and last subtasks of a given task. This is an especially important metric in the context of automated warehouses which process orders made up of multiple items. In such systems the first arrival of a subtask in the output buffer triggers reservation of physical space for that subtask and its siblings. Only when the final subtask belonging to a task has arrived in the output buffer can the space be freed. Efficient management of the output buffer space therefore requires the times of arrival of a subtask and its siblings in the output buffer to be clustered as close together as possible. It is also a consideration in other environments like full service restaurants, where customer satisfaction requires that the food for each course ordered by each table arrives at nearly the same time, and that each dish is hot (if appropriate) and freshly prepared. Research interest in this metric is relatively recent, see e.g. [24–26].

As illustrated in Fig. 3 these metrics are in tension in the sense that taking action to reduce one usually results in an increase in another; this is especially the case for high-intensity workloads. Unmodified fork-join systems yield low task response times (and therefore higher maximum sustainable system throughput), but subtask dispersion is high under load. Conversely, unmodified split-merge 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. As we have shown in our previous work, adding delays to subtask processing times in split-merge systems can help to reduce mean subtask dispersion [24] and/or percentiles of subtask dispersion [26], but the sole focus on subtask dispersion only serves to exacerbate the problem of poor task response times under load. One solution is to apply load-dependent subtask delays which minimise the product of expected task response time and expected subtask dispersion [25]. 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 in the present work is to find a way to reduce dispersion in fork-join systems to levels comparable with or *below* that observed in all varieties of split-merge systems while retaining the response time and throughput benefits of a fork-join system.

4 On Online Technique for Reducing Subtask Dispersion in Fork-Join Systems

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 $n_i(t)$ denote the number subtasks present in parallel server queue i; as such $N(\max_i n_i(t)) - \sum_i n_i(t)$ 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 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 are still present in the parallel queues and we apply delays to S and 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 system in which parallel server i has service time distribution $\mathrm{Erlang}(q_i(t) + 1, \mu_i)$, where $q_i(t)$ is a number of subtasks in front of S or its sibling subtask in parallel queue i at time t. The $q_i(t)$ form vector $\mathbf{q}(t) = \big(q_1(t), q_2(t), \ldots, q_N(t)\big)$. We can then exploit the optimisation method we developed in our previous work [24–26] to determine a

vector of (near-)optimal deterministic subtask delays $\mathbf{d} = \left(d_1, d_2, \ldots, d_N\right)$. 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, computed as the difference between the maximum and minimum heterogeneous order statistics of the split-merge-equivalent system with delays. Utilising the linearity property of the expectation operator over dependent variables, we have:

$$\begin{aligned}
\mathbb{E}[D_{\mathbf{d}}] &= \left(\mathbb{E}[X^{\mathbf{d}}_{(N)} - X^{\mathbf{d}}_{(1)}]\right) \\
&= \left(\mathbb{E}[X^{\mathbf{d}}_{(N)}] - \mathbb{E}[X^{\mathbf{d}}_{(1)}]\right) \\
&= \int_0^\infty \left(1 - \prod_{i=1}^{N} F_i(x - d_i)\right) dx - \\
&\quad \int_0^\infty \left(1 - \left(1 - \prod_{i=1}^{N}(1 - F_i(x - d_i))\right)\right) dx
\end{aligned} \tag{2}$$

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

When optimising, we solve for:

$$\mathbf{d}_{\min} = \arg\min_{\mathbf{d}} \mathbb{E}[D_{\mathbf{d}}] \tag{3}$$

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

The optimisation procedure itself is based on Newton's method. Practically, it utilises numerical integration to evaluate 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 Numerical Results

In this section we present results from C++ simulations of fork-join and split-merge queueing systems that employ the dynamic optimisation of the present paper for fork-join systems and the static optimisation techniques developed in our previous work [24,25] for split-merge systems. The simulations collect a range of performance-related statistics, e.g. mean task response time, mean subtask dispersion, mean output utilisation of join/merge buffers, task throughput and distributions of subtask dispersion. The

simulations were performed on a 3.5GHz Intel Core-i5 workstation with 8GB RAM. 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 requires approximately one second to run each replica, and for the dynamic optimisation of fork-join simulator it takes around 7.5 minutes for each replica. The replicas are used to put 95% confidence intervals (CIs) on all measures. Results are reported to three decimal places.

As our case study, consider a parallel system with Poisson arrivals with rate parameter $\lambda = 0.78$ tasks/time unit and 3 parallel service nodes with exponential service time density functions: Exp(1), Exp(5), Exp(10).

In this context, we compute measures of subtask dispersion and of task response time of 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 $\mathbb{E}[D_{\mathbf{d}=\mathbf{0}}] = 4.490$ (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 compared with the fork-join system without subtask delays, we observe mean task response time 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% CI $[5.1526, 5.271]$) time units and mean subtask dispersion is $\mathbb{E}[D_{\mathbf{d}=\mathbf{0}}] = 0.976$ (95% CI $[0.975, 0.977]$) time units. The mean number of subtasks in the output buffer is 1.416 (95% 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 disperstion [24]. The vector of optimised delays is:

$$\mathbf{d}_{\min} = (0, 0.553, 0.617)$$

   Mean task response time is $\mathbb{E}\left[R_{\mathbf{d}_{\min},\lambda=0.78}\right] = 63.02$ (95% CI $[58.21, 67.83]$) time units and mean subtask dispersion is $\mathbb{E}[D_{\mathbf{d}_{\min}}] = 0.783$ (95% CI $[0.780, 0.785]$) time units. The mean number of subtasks in the output buffer is 1.029 (95% 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 [25]. 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% 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.

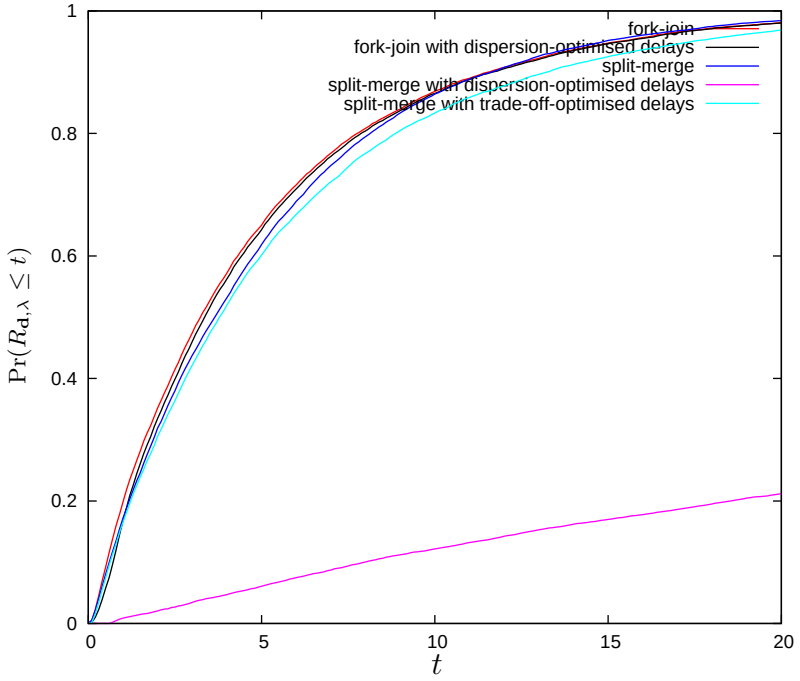

Fig. 4: Distributions of task response time for fork-join and split-merge queueing systems with and without optimised subtask delays. $\lambda = 0.78$

Turning now to distributions of task response time, Fig. 4 demonstrates that the task response time cdf of the fork-join system with dispersion-optimised delays is very close to that of the fork-join system without subtask delays. Here, the response time cdf of the split-merge system without subtask delays is marginally worse than that of the fork-join system, but after applying dispersion-optimised delays response time suffers heavily. Applying delays optimised for the subtask dispersion–task response time trade-off impacts only marginally on task response time.

Fig. 5 shows the corresponding distributions of subtask dispersion. The poor subtask dispersion of the fork-join 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 system without delays. The subtask dispersion profile of the fork-join system with dispersion-optimised delays is competitive with that of the split-merge system with dispersion-optimised delays, and even dominates it for percentiles of subtask dispersion below 70%.

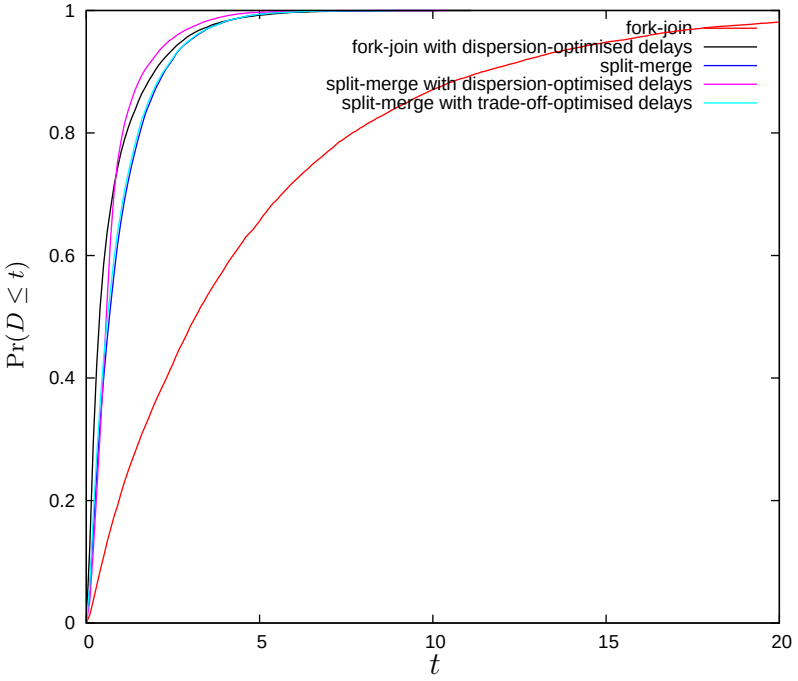

Fig. 5: Distributions of subtask dispersion in fork-join and split-merge queues with and without optimised subtask delays. $\lambda = 0.78$

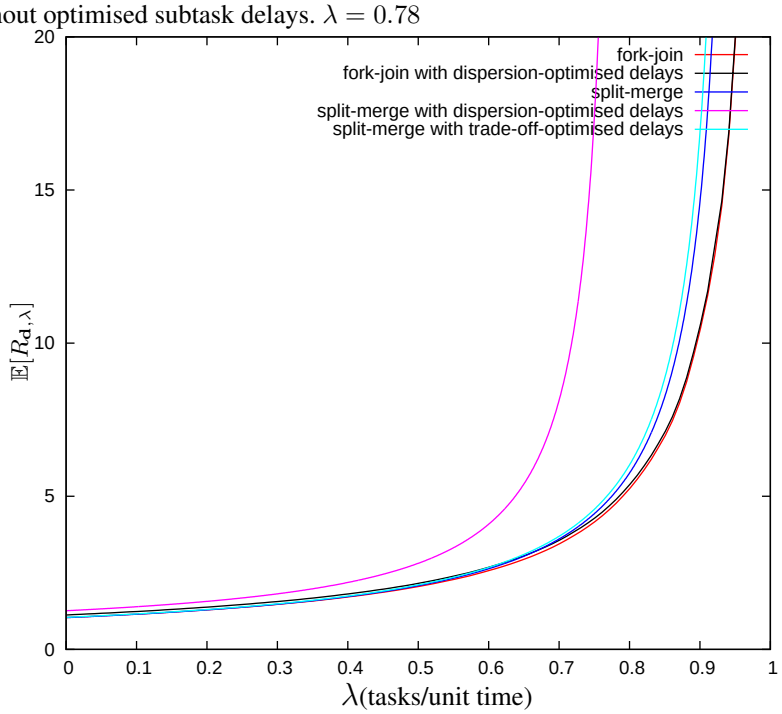

Fig. 6: Expected response time of case study of fork-join and split-merge systems for various customer arrival rates.

Fig. 6 shows how mean task response time varies with various task arrival rates under the various policies. We observe the split-merge system with dispersion-optimised delays has the lowest maximum sustainable system throughput, followed by the split-merge system with delays optimised for the subtask dispersion–task response time trade-off, and then the split-merge system without delays. The highest maximum sustainable system throughput is provided by the fork-join system utilising dispersion-optimised subtask delays and the fork-join system without subtask delays.

6 Conclusion

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

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

Our current research can no doubt be extended to apply to fork-join systems with non-exponential services times. Certainly extension to Erlang and phase-type service time distributions is likely to be straightforward given appropriate extensions to the system state vector.

References

1. 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.
2. F. Baccelli, W. A. Massey, and D. Towsley. Acyclic fork-join queuing networks. *Journal of ACM*, 36(3):615–642, July 1989.
3. G. Bolch et al. *Queueing Networks and Markov Chains*. J. Wiley & Sons, Inc., 2006.
4. R. J. Chen. A hybrid solution of fork/join synchronization in parallel queues. *IEEE Transactions on Parallel and Distributed Systems*, 12(8):829–845, August 2001.
5. R. J. Chen. An upper bound solution for homogeneous fork/join queuing systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(5):874–878, May 2011.
6. H. A. David. *Order Statistics*. Wiley Series in Probability and Mathematical Statistics. John Wiley, 1980.
7. H. A. David and H. N. Nagaraja. *Order Statistics*. Wiley Series in Probability and Mathematical Statistics. John Wiley, 3rd edition, 2003.
8. L. Flatto. Two parallel queues created by arrivals with two demands II. *SIAM Journal on Applied Mathematics*, 45(5):pp. 861–878, 1985.
9. 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.
10. P. G. Harrison and S. Zertal. Queueing models of RAID systems with maxima of waiting times. *Performance Evaluation*, 64(7–8):664–689, August 2007.

11. P. Heidelberger and K. S. Trivedi. Analytic queueing models for programs with internal concurrency. *IEEE Transactions on Computers*, C-32(1):73–82, January 1983.
12. K. Jeffay, D.F. Stanat, and C.U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proc. 12th Real-Time Systems Symposium*, pages 129–139, 1991.
13. H. Kameda, J. Li, C. Kim, and Y. Zhang. A comparison of static and dynamic load balancing. In *Optimal Load Balancing in Distributed Computer Systems*, Telecommunication Networks and Computer Systems, pages 225–240. Springer, 1997.
14. C. Kim and A.K. Agrawala. Analysis of the fork-join queue. *IEEE Transactions on Computers*, 38(2):250–255, February 1989.
15. W. J. Knottenbelt and I. Tsimashenka. Reducing subtask dispersion in parallel systems. In *Trends in Parallel, Distributed, Grid and Cloud Computing for Engineering*, chapter 9, pages 203–227. Saxe-Coburg Publications, April 2013.
16. Y. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, December 1999.
17. A. Lebrecht and W. J. Knottenbelt. Response Time Approximations in Fork-Join Queues. In *23rd Annual UK Performance Engineering Workshop (UKPEW 2007)*, July 2007.
18. 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 Distributed Systems*, 9(3):295–311, March 1998.
19. I. Mitrani. Management of server farms for performance and profit. *Computer Journal*, 53(7):1038–1044, 2010.
20. R. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739 –743, 1988.
21. J. Slegers, I. Mitrani, and N. Thomas. Static and dynamic server allocation in systems with on/off sources. *Annals of Operations Research*, 170:251–263, 2009.
22. J. Sun and G. D. Peterson. An effective execution time approximation method for parallel computing. *IEEE Transactions on Parallel and Distributed Systems*, 23(11):2024–2032, 2012.
23. 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.
24. 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.
25. I. Tsimashenka and W. J. Knottenbelt. Trading off subtask dispersion and response time in split-merge systems. In *Analytical and Stochastic Modeling Techniques and Applications (ASMTA'13)*, volume 7984 of *Lecture Notes in Computer Science*, pages 431–442. Springer, July 2013.
26. I. Tsimashenka, W. J. Knottenbelt, and P. G. 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.
27. E. Varki. Response time analysis of parallel computer and storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1146–1161, Nov 2001.
28. S. Varma and A. M. Makowski. Interpolation approximations for symmetric fork-join queues. *Performance Evaluation*, 20(13):245–265, 1994.
29. T. Yang and A. Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, Sep 1994.
30. H. Zhao, C. H. Xia, Z. Liu, and D. Towsley. A unified modeling framework for distributed resource allocation of general fork and join processing networks. In *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2010)*, pages 299–310, New York, NY, USA, 2010. ACM.