Three-way Optimisation of Response Time, Subtask Dispersion and Energy Consumption in Split–Merge Systems

Tommi Pesu
Imperial College London
London, United Kingdom
ttp09@ic.ac.uk

Jani Kettunen
Google
Zurich, Switzerland
janikettunen@google.com

William J. Knottenbelt
Imperial College London
London, United Kingdom
wjk@doc.ic.ac.uk

Katinka Wolter
Freie Universität Berlin
Berlin, Germany
katinka.wolter@fu-berlin.de

ABSTRACT

This paper investigates various ways in which the triple trade-off metrics between task response time, subtask dispersion and energy can be improved in split-merge queueing systems. Four ideas, namely dynamic subtask dispersion reduction, state-dependent service times, multiple redundant subtask service servers and restarting subtask service, are examined in the paper. It transpires that all four techniques can be used to improve the triple trade-off, while combinations of the techniques are not necessarily beneficial.

CCS CONCEPTS

• **Mathematics of computing → Queueing theory**;

KEYWORDS

energy consumption, subtask dispersion, task response time, trade-off, optimisation

ACM Reference format:
Tommi Pesu, Jani Kettunen, William J. Knottenbelt, and Katinka Wolter. 2017. Three-way Optimisation of Response Time, Subtask Dispersion and Energy Consumption in Split–Merge Systems. In *Proceedings of Valuetools 2017, Venice, Italy, December 5-7, 2017 (VALUETOOLS 2017),* 8 pages.
https://doi.org/10.1145/3150928.3150934

1 INTRODUCTION

In real world applications quite often there are multiple criteria by which one may wish to be as good as possible. There is also a catchphrase saying "choose two out of three: fast, good and cheap", indicating that it is very difficult to satisfy all criteria. In this paper we investigate split–merge systems, a particular variety of parallel queueing systems that have three desired qualities. First we wish the system has a low subtask dispersion [6, 7, 9, 10] (difference in completion time between first and last subtasks to complete), low task response time (difference between task arrival and completion of service) and low energy usage. This paper discusses four techniques, which are compared with the method from [9].

Having a low subtask dispersion is important in many kinds of systems. For example in some online games it is important to have quick reflexes. If the lag between players varies too much the game becomes unfair as the players with lower lag are able to react faster and therefore are more likely to win. If subtask dispersion is reduced the playing experience will be more fair. This problem is analysed in [12].

A similar situation exists when trading on the stock market. Some actors in the industry have microwave links between the different exchanges, which allow them to be faster than people using fibre optic links. As a result a buy order spread over multiple exchanges may lead to high dispersion and a buyer may be front run by the high frequency traders. This means that the high frequency trader buys stock right before it arrives to the second exchange and then immediately sells it back to the customer at a slightly higher price.

Sec. 2 and 3 focus on two techniques, which are both based on dynamically adding delays so as to reduce the variation between the fastest and the slowest subtasks. The first technique inserts delays before the service of fast subtasks to make them take longer to complete with the ideal outcome that all subtasks complete at the same time. We will first show the mathematical formulation for dynamic subtask dispersion as introduced in Pesu and Knottenbelt [7]. The second technique is to select the delays inserted in front of subtasks based on the queue length of the system. Delays are determined with the help of Bayesian optimisation.

Sec. 4 introduces two techniques that use replication of subtask servers and service restart of subtasks. These techniques work best when the underlying subtask service time distribution has high variance, more precisely subtask service restart is beneficial only if the service time distribution is heavy-tailed. In addition, such tasks must be idempotent, i.e. replication, abortion and restart of the task must not have undesired side effects, such as a duplicated trade or transaction. Both methods are applicable only if tasks can be shifted, duplicated and repeated with no harm.

It is possible to use dynamic subtask dispersion if the time to transmit information that a sibling subtask has finished is small compared to the service time of subtasks. State-dependent delays

VALUETOOLS 2017, December 5-7, 2017, Venice, Italy

ACM ISBN 978-1-4503-6346-4/17/12…$15.00
https://doi.org/10.1145/3150928.3150934

have very little prerequisites and can be used almost always as they only require knowledge of the queue length.

We find that restarting subtasks and replicating service of subtasks is very helpful in cases where the subtask service time has high variance. However, formulating guidelines as to which of the methods to use best is not straightforward as it will depend on the system-specific parameters. Our results indicate that combining the two redundancy techniques is not necessarily able to further improve results. We attribute this to the fact that both techniques work by reducing the variance of the subtask service time distribution.

2 TRADE-OFFS USING DYNAMIC SUBTASK DISPERSION

In this section we first introduce the split–merge queueing model and then investigate how the trade-off delay scheme introduced in [9] can be improved by substituting the subtask-dispersion technique with the improved delay scheme introduced in [7].

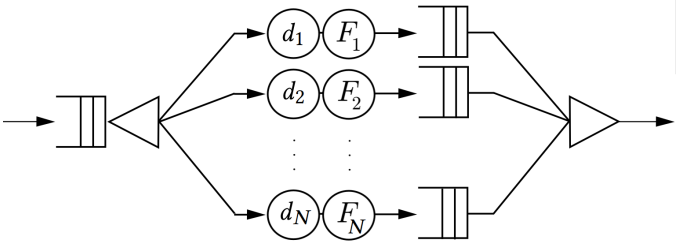

Figure 1: Visualisation of the split–merge system

In a split-merge system each time a new task enters service it is split into N subtasks. Those subtasks are served by N different parallel servers. Upon completion, the subtasks are reassembled into one task for further processing. This situation is illustrated in Fig. 1. Some of the parallel servers will finish processing their subtask much earlier than others which leads to undesired dispersion in the completion time of the subtasks. To avoid this, different measures can be taken. The most straightforward solution is to insert a delay before each subtask. These delays are specified in a delay vector $\mathbf{d}$.

The dynamic version of the algorithm sets delays to 0 once a sibling subtask completes service. In the regular algorithm delays are kept constant until completion of all subtasks. Sec. 2.1 and 2.2 discuss the dynamic algorithm. The regular subtask dispersion algorithm is from the paper [10]. For the dynamic algorithm intuition dictates that subtask dispersion should be reduced if delays are cut after a sibling subtask has completed. Similarly it dictates that task response time should decrease if delays are decreased.

An example of a potential operation by the algorithm is shown in Fig. 2. The advantage of dynamic subtask padding is shown by the results for an example in Fig. 3. We observe that removing delays once a sibling subtask finishes can dramatically reduce subtask dispersion.

The next subsection defines the optimisation problem that must be solved to determine suitable delays that minimise subtask dispersion. In Sec. 2.2 we formulate the corresponding optimisation problem for minimising the task response time. Finally, at the end

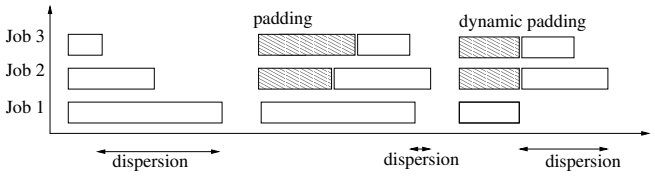

Figure 2: An example of processing a task in a three server split–merge system

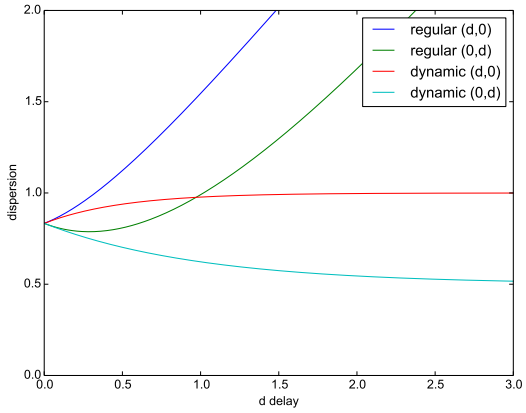

Figure 3: Dynamic and regular subtask dispersion techniques: A two server $\exp(\lambda = 1), \exp(\lambda = 2)$ example

of Sec. 2.2 we combine both into the formulation of the trade-off between task response time and subtask dispersion. The trade-off is specified in Eqn. (14).

2.1 Dynamic Subtask Dispersion

A technique to minimise dynamic subtask dispersion has been introduced in [7]. We briefly revisit the result here. This section introduces how subtask dispersion of a split-merge system can be computed for a given set of delays $\mathbf{d}$.

Let $T(i, t, \mathbf{d})$ be the probability that subtask i is the first to finish at time t. Then $E_r(i, t', \mathbf{d})$ is the expected completion time of the remaining subtasks, given that subtask i finished at time t'. $G_j(t, t', \mathbf{d_j})$ is the probability distribution of a subtask, given that a sibling subtask has finished already. $F_i(t)$ and $f_i(t)$ are the cdf and pdf of service time of ith subtask.

Those terms can be used to compute subtask dispersion for a given set of delays $\mathbf{d}$. To minimise dispersion a set of delays $\mathbf{d}$ must be found that minimise the $\text{Disp}(\mathbf{d})$ function.

$$\text{Disp}(\mathbf{d}) = \sum_{i=1}^{N} \int_0^\infty T(i, t, \mathbf{d}) E_r(i, t, \mathbf{d}) dt \tag{1}$$

where

$$T(i, t, \mathbf{d}) = f_i(t - d_i) \prod_{j \neq i} [1 - F_j(t - d_j)] \tag{2}$$

and

$$E_r(i, t', \mathbf{d}) = \int_0^\infty [1 - \prod_{j \neq i} G_j(t, t', d_j)]dt \tag{3}$$

with

$$G_j(t, t', d_j) = \begin{cases} F_j(t) & \text{if } t' < d_j \\ F_j(t + (t' - d_j)|t > 0) & \text{otherwise} \end{cases} \tag{4}$$

subject to conditions

$$\prod_{i=1}^{N} d_i = 0 \tag{5}$$

and

$$\forall i \; d_i \geq 0 \tag{6}$$

The Disp($\mathbf{d}$) function is used in the next section in the trade-off definition in Eqn. (14).

2.2 Dynamic Padding to Optimise Task Response Time and Dispersion

In this section we first derive an expression for the task response time in a split-merge system when using dynamic delay padding. Then we combine this result with the expression for dispersion derived in the previous section to formulate the trade-off between task response time and dispersion, which is one of the main results of this paper.

The task response time of a split–merge system can be computed using the Pollaczek–Khinchine (PK) formula known for M/G/1 queues. The split–merge queue is very similar to a M/G/1 queue, because the time to complete all N subtasks can be seen as the service time of a task in the M/G/1 queue as shown below:

$$\text{Resp}(\lambda, \mathbf{d}) = \frac{\rho + \mu\lambda \text{Var}[X_{(N)}]}{2(\mu - \lambda)} + \mu^{-1} \tag{7}$$

Here the arrival rate of incoming tasks is λ when the time between arrival of tasks is exponentially distributed, the task service rate is μ, ρ is the utilisation of the system given by λ/μ, and $\text{Var}[X_{(N)}]$ is the variance of the service time of last subtask to complete, equivalently the variance of the service time of the task.

We will now derive the mean and variance of the task service time. The first step is to derive a probability distribution for the service time of a task, which is shown below:

$$f(t, \mathbf{d}) = \sum_{i=1}^{N} \int_0^t T(i, t', \mathbf{d}) E(i, t', t - t', \mathbf{d}) dt' \tag{8}$$

The function $T(i, t, \mathbf{d})$ calculates the probability that subtask i is the first to finish at time t given the subtask delay vector $\mathbf{d}$.

$$T(i, t, \mathbf{d}) = f_i(t - d_i) \prod_{j \neq i} [1 - F_j(t - d_j)] \tag{9}$$

The function $E(i, t', t, \mathbf{d})$ describes the probability that the remaining subtasks finish in time t. It takes as priori i, which is the number of the subtask that finished first and the time t' when it finished.

$$E(i, t', t, \mathbf{d}) = \frac{d}{dt} \prod_{j \neq i} G_j(t, t', d_j) = \sum_{j \neq i} g_j(t, t', d_j) \prod_{k \neq i | j} G_k(t, t', d_j) \tag{10}$$

The function $G_j(t, t', d_j)$ renormalises the jth subtask service distribution to take into account that a sibling subtask has finished at time t'. If no other subtask has started service it immediately starts service. If the subtask has begun service the service time probability distribution is renormalised to take into account that it did not finish before time t'.

$$G_j(t, t', d_j) = \begin{cases} F_j(t) & \text{if } t' < d_j \\ F_j(t + (t' - d_j) \mid t > 0) & \text{otherwise} \end{cases} \tag{11}$$

Once the probability distribution of the service time of a task has been defined its mean and variance can be derived in a standard way:

$$\mu^{-1} = \text{E}(X_{(N)}(\mathbf{d})) = \int_0^\infty t f(t, \mathbf{d}) dt \tag{12}$$

$$\text{Var}[X_{(N)}](\mathbf{d}) = \int_0^\infty (t - \mu)^2 f(t, \mathbf{d}) dt \tag{13}$$

After deriving the mean and variance of the response time the algorithm from paper [9] can be applied. The optimal delay vector $\mathbf{d}$ can be found by solving the following optimisation problem:

$$\arg\min_{\mathbf{d} \geq 0} \text{Resp}(\lambda, \mathbf{d}) \text{Disp}(\mathbf{d}) \tag{14}$$

When solving the equation in practice, the integrals are solved using numerical integration techniques and the minimisation of Eqn. (14) can be done using an optimisation algorithm such as Nelder–Mead.

Details on how to solve Eqn (14) are discussed in Sec. 5.1. This trade-off is evaluated in Sec. 6.1.

3 STATE-DEPENDENT DELAY VECTORS

While Sec. 2 has focused on how the trade-off between subtask dispersion and task response time can be optimised using a technique which ignores system load, this section introduces the concept of state-dependent delay vectors, i.e. where the delay vector is determined based on how many tasks are currently in the queue waiting to be served.

3.1 Background

There is a large body of research on state-dependent M/G/1 and M/M/1 queues [1, 5, 8], as well as analytical solutions. However the analytical solutions contain Laplace transforms, infinite sums and recursive equations which make computation of accurate results very difficult. We were unable to produce a stable and quick enough computations so that we could perform optimality search over the function in sufficiently high dimensional space. We therefore adopted numerical techniques.

Our intuition is that delays should be small when the queue length is large and delays should be larger for a short queue. Reducing delays of subtasks may increase subtask dispersion for the corresponding task, but it benefits the response time of all subsequent tasks in the queue.

Next we present a two-step technique to find optimal delays for different queue lengths. A naïve search would optimise on an $O(N \times m)$ dimensional search space, where N is number of subtasks and m is the number of tasks in the queue for which a unique delay

vector is specified. Our technique reduces this to a $O(\max(n, m))$ dimensional search space. The search space reduction is especially necessary as our objective function evaluations are expensive.

3.2 The Algorithm

With the first step of the algorithm we wish to construct a function that takes one parameter and then maps to a delay vector which optimises the product of expected subtask dispersion and expected task response time:

$$f(\lambda, \mathbf{d}) = \text{Resp}(\lambda, \mathbf{d})\text{Disp}(\mathbf{d}) \tag{15}$$

Tsimashenka's trade-off technique [9] can be used to determine an optimal delay vector for a given λ. Varying λ from 0 until utilisation of the system is 1 gives us a parametric curve of delays, as shown in Fig. 4. We can see that at low utilisations relatively large delays are added in order to reduce subtask dispersion, while at high utilisations, the system concentrates on the protection of response time by minimising delays. If the technique does not produce a clear line, regression analysis can be applied.

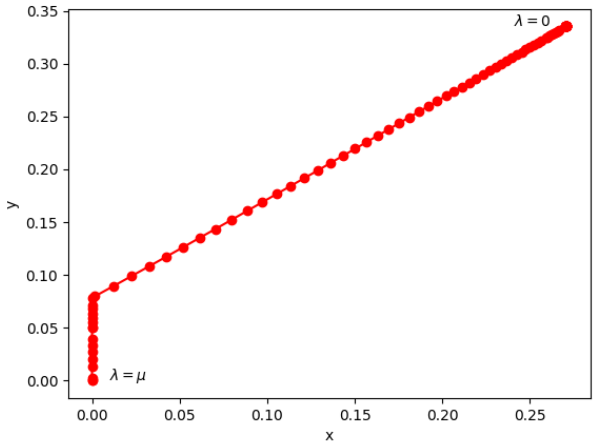

Figure 4: Showing the parametric curve of delays of the form (0,x,y). Formed by application of Tsimashenka's trade-off technique [9] for varying range of utilisation, on a three server split–merge system $exp(\lambda = 1), exp(\lambda = 5), exp(\lambda = 10)$

The second step of our algorithm makes application of delay vectors state-dependent by mapping queue length thresholds onto points on the parametric curve of delays. The objective function of our optimisation takes as input an m element vector. The ith element of the vector is a real number in the same range as λ in the previous step. The ith element maps to a delay vector via the parametric curve of delays, using interpolation between evaluated points where necessary. This delay vector is applied when there are i tasks waiting in the queue. If there are more than m tasks in the queue, the last element of the vector is used.

In the objective function, the split–merge system is simulated with a large number of tasks to obtain an estimate of the product of mean subtask dispersion and mean task response time when applying state-dependent subtask delays using the m element vector.

Due to the noisy nature of the optimisation function we chose Bayesian optimization, a well-established technique for optimizing noisy, non-convex functions. GpyOpt [2] was used for implementing the Bayesian optimisation.

Experimental setup and results for this technique are presented in Sec. 5.1 and 6.1, where it used in Methods 3 and 4.

4 ENERGY METRIC AND SERVICE TIME MANIPULATIONS

This section introduces subtask service restart and subtask service replication as potential techniques for improving trade-offs between performance metrics in parallel queueing systems. It also introduces the energy metric as a necessary addition to the previously-used metrics for two reasons. Firstly, energy consumption considerations are typically a critical consideration in large scale service delivery. Secondly, if there is no cost associated with service restart and replication, it is often possible to arbitrarily improve performance in terms of subtask dispersion and task response time by increasing the number of server replications without a limit.

4.1 Server Replication

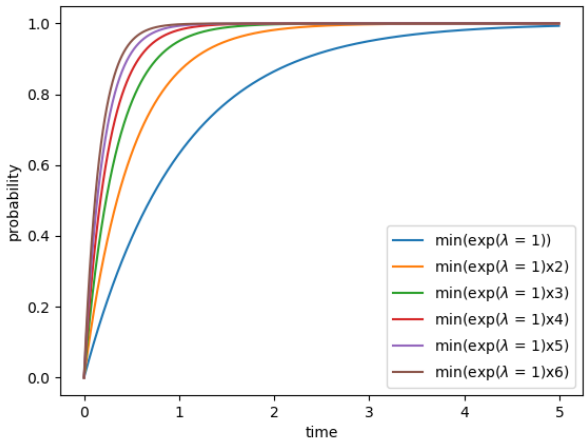

Figure 5: Changes in cumulative probability distribution of subtask service time when the number of servers is varied from 1 **to** 6**.**

Given a non-deterministic service time distribution with high variance and assuming no correlation between consecutive service attempts, it is possible to improve the service time of a subtask using multiple servers. Several copies of the subtask are served in parallel by these servers and the result of the fastest server is used. An example of how server replication improves service time can be see in Fig. 5.

The cumulative distribution function of the minimum service time of n subtasks can be calculated in the following way:

$$F_n(t) = 1 - (1 - F(t))^n \tag{16}$$

The corresponding probability density function is the following:

$$f_n(t) = n(1 - F(t))^{n-1} f(t). \tag{17}$$

Details on how server replication is used are discussed in Sec. 5.2. This technique is evaluated in Sec. 6.2.

4.2 Service Restart

An alternative to serving redundant copies of subtasks is to use service restart if service has not completed by a given target time τ. Restart comes with a time cost c. Restart will not always provide an improvement in terms of service time. However, there exists a class of service time distributions where the conditional remaining service time of a job increases as time progresses. It has been shown that the average service time of such tasks can be decreased by periodically restarting service [11].

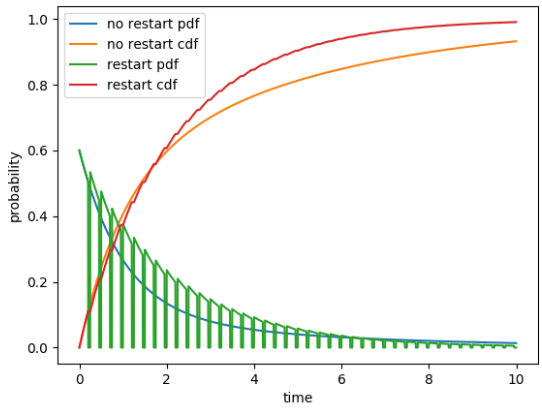

Figure 6: Comparison of a hyper-exponential distribution with/without service restart. $\tau = 0.2$, $c = 0.05$, $f(x) = 0.5\exp(\lambda = 1.0) + 0.5\exp(\lambda = 0.2)$

The new probability density function of service time is:

$$f_\tau(t) = \begin{cases} (1 - F(\tau))^k f(t - k(\tau + c)) & \text{if } k(\tau + c) \le t < k(\tau + c) + \tau \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

for $k = 0, 1, 2, \ldots$.

The corresponding cumulative distribution function is:

$$F_\tau(t) = \begin{cases} 1 - (1 - F(\tau))^k (1 - F(t - k(\tau + c))) & \text{if } k(\tau + c) \le t \\ & < k(\tau + c) + \tau \\ 1 - (1 - F(\tau))^{k+1} & \text{otherwise} \end{cases}, \tag{19}$$

for $k = 0, 1, 2, \ldots$.

The higher moments of a probability distribution with heavy tail can be decreased when service restart is incorporated. The new distribution will then be similar to a geometric distribution. This improves both task response time of the system and subtask dispersion. An example of the effect of service restart on a distribution can be seen in Fig. 6.

Details on how server restart is used are also discussed in Sec. 5.2. This technique is evaluated in Sec. 6.2.

4.3 Energy Metric

This section introduces the energy metric, which measures how much energy is consumed by a split–merge system. Split–merge systems in practice come at the energy cost of running several servers. Improving performance may likely increase the energy cost. This paper uses the following cost function:

$$\text{Energy}(\mathbf{n}) = \sum_{i=1}^{N} n_i(\lambda/\mu_i C_H + (1 - \lambda/\mu_i)C_L) \tag{20}$$

The constants defined in the equation are as follows. The rate of incoming tasks is λ, the service rate of subtask i is μ_i. The idle operational cost of a server is C_L, while the cost for service, restart and cool down is C_H. The number of servers serving the ith subtask is n_i, the number of subtasks each task is split into is N.

4.4 Trade-off Metric

Energy trade-off metrics have been investigated in the past [3, 4]. For computing the triple trade-off between subtask dispersion, task response time and energy the third power of the product, the RDE^3 metric is used. This is because we found that the RDE and RDE^2 metrics sometimes indicate that an infinite replication would be optimal (see appendix for analysis). The trade-off is computed by first applying the multiple servers and/or restart transforms to the subtask service time distributions. Then the subtask dispersion, task response time and energy cost are derived. The optimal delay vector $\mathbf{d}$, server replication factors $\mathbf{n}$ and server restart intervals τ can be found by solving the following optimisation problem:

$$\arg\min_{\mathbf{d}\ge 0, \mathbf{n}, \tau} \text{Resp}(\lambda, \mathbf{d})\text{Disp}(\mathbf{d})\text{Energy}^3(n), \tag{21}$$

In our example we allow each subtask to have up to 10 replicated servers (including the original). For each possible server configuration (i.e. for every permutation of $\mathbf{n}$) we ran the Nelder–Mead algorithm 50 times, using random start vectors for τ and $\mathbf{d}$. Finally, we selected the $\mathbf{n}$,τ and $\mathbf{d}$ that minimised Eqn. (21).

Details on how the triple trade-off is used in our experiments are presented in Sec. 5.2. The trade-off is evaluated in Sec. 6.2.

5 EXPERIMENTAL SETUP

5.1 Exploring response time and dispersion

This section explains the methods used in the results of Sec. 6.1.

5.1.1 ***Method 1***. For this method, analytical formulas (22) and (7) were used to find the delay vector which optimises the product of subtask dispersion and task response time, according to the method presented in [9].

5.1.2 ***Method 2***. This method uses the concept of dynamic subtask dispersion from Sec. 2.1 and dynamic delay padding from Sec. 2.2 to find the delay vector which optimises the trade-off between task response time and subtask dispersion. For computation of subtask dispersion and task response time we use Eqn. (1) and (7).

5.1.3 ***Method 3***. This method uses the state-dependent delay vectors introduced in Sec. 3 to optimise the trade-off between subtask dispersion and task response time. No dynamic delay padding is used.

5.1.4 ***Method 4***. This method uses the state-dependent delay vectors introduced in Sec. 3 to optimise the trade-off between subtask dispersion and task response time. In this method dynamic delay padding is applied.

5.2 Exploring response time, dispersion and energy

This section explains the methods used in the results of Sec. 6.2.

All methods the results were computed analytically. For computing subtask dispersion we used the formula from [9]:

$$\text{Disp}(\mathbf{d}) = \int_0^\infty 1 - \prod_{i=1}^{N} F_i(x - d_i) - \prod_{i=1}^{N}(1 - F_i(x - d_i))\text{dx} \quad (22)$$

Task response time is computed with the PK formula of Eqn (7).

Optimisation is done by applying the formulas from [9] for subtask dispersion and task response time. For the computation of energy metric we used the results from Sec. 4.3. If service restart was used we transformed the distributions with the formula in Sec. 4.2. If server replication was used we transformed the distributions with the formula in Sec. 4.1 and allowed each subtask to have up to 10 servers. We then individually computed the optima for each case and chose the best server configuration.

6 RESULTS

We performed two experiments, the first using the Subtask Dispersion vs. Response Time trade-off, and the second using the Response Time vs. Subtask Dispersion vs. Energy trade-off.

6.1 Subtask Dispersion vs. Response Time trade-off

We use a three-server split–merge system with exponentially distributed incoming tasks that have an arrival rate of $\lambda = 0.78$ tasks per time unit. The subtask service times are distributed as:

$$\begin{aligned} X_1 &\sim \text{Exp}(\lambda = 1) \\ X_2 &\sim \text{Exp}(\lambda = 5) \\ X_3 &\sim \text{Exp}(\lambda = 10) \end{aligned}$$

Method 1 The resulting delays for the subtasks are: $\mathbf{d} = (0, 0, 0.068)$. Corresponding performance metrics are:

Task response time:	5.286 time units
Subtask dispersion:	0.946 time units
Trade-off:	4.999 (time units)^2

Method 2 The resulting delays for subtasks are: $\mathbf{d} = (0, 0.019, 1.879)$. Corresponding performance metrics are:

Task response time:	5.437 time units
Subtask dispersion:	0.867 time units
Trade-off:	4.718 (time units)^2

Method 3 and 4 The results can be seen in Fig. 7, 8 and 9. The number of delays in the figures indicate how many delays there are to choose from when picking the delay based on a queue length.

The results indicate that the dynamic subtask dispersion technique introduced in Sec. 2 is better than the technique introduced by Tsimashenka [9]. **Method 2** is able to produce a 6% decrease in the trade-off cost function when compared against **Method 1**.

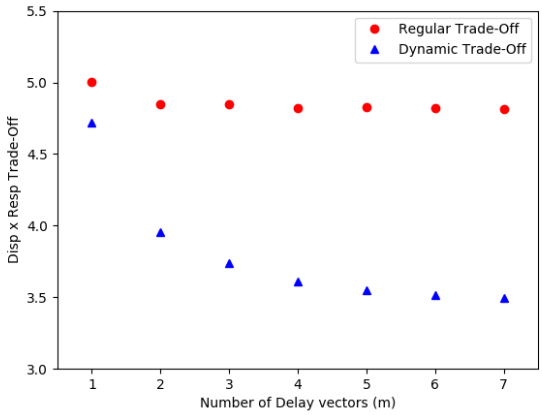

Figure 7: Trade-Off Results of Methods 3 and 4

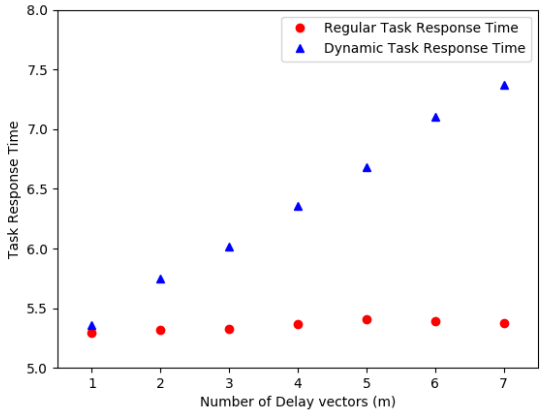

Figure 8: Task Response time of Methods 3 and 4

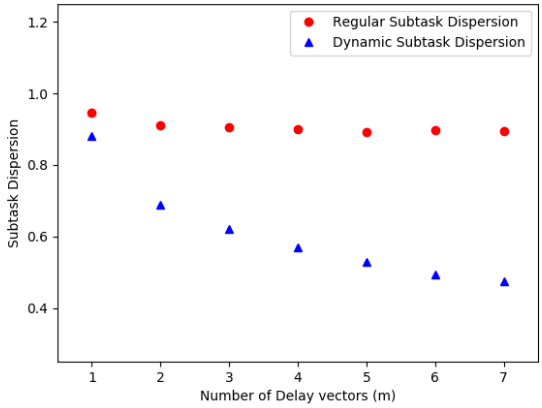

Figure 9: Subtask Dispersion of Methods 3 and 4

When **Methods 3 and 4** were compared against their single delay vector counterparts **Methods 1 and 2**, the state-dependent delay vectors introduced in Sec. 3 reduced the trade-off cost functions by 4% and 25% respectively.

6.2 Response Time vs. Subtask Dispersion vs. Energy Trade-Off

In our example case we have a split–merge system with Poisson task arrivals at rate $\lambda = 1.05$ tasks per time unit. The energy metric cost for high utilisation is 1.0 energy units and low utilisation cost is 0.15 energy units. The cost of a restart is 0.1 time units. Subtask service times are distributed as:

$$\begin{aligned} X_1 &\sim \text{Erlang(k} = 5, \lambda = 10) \\ X_2 &\sim \text{Uniform(a} = 0.25, \text{b} = 0.3) \\ X_3 &\sim 0.5 * \text{Exp}(\lambda = 1) + 0.5 * \text{Exp}(\lambda = 5) \end{aligned}$$

Method 1 No service restart or multiple servers were used. The resulting delays for subtasks are: $\mathbf{d} = (0.0, 002, 0)$. Corresponding performance metrics are:

Task response time:	5.729 time units
Subtask dispersion:	0.637 time units
Energy Cost:	1.677 energy units
RDE^3 Trade-off:	17.226 units

Method 2 Subtask server replication factors $\mathbf{n} = (2, 1, 4)$ and no subtask restart was used. The resulting delays for subtasks are: $\mathbf{d} = (0, 0.028, 0.233)$. Corresponding performance metrics are:

Task response time:	0.633 time units
Subtask dispersion:	0.174 time units
Energy Cost:	2.314 energy units
RDE^3 Trade-off:	1.362 units

Method 3 No multiple subtask service servers were used. Optimal restart subtask service for subtask 3 was ($\tau_3 = 0.363$). The resulting delays for subtasks are: $\mathbf{d} = (0, 0.046, 0)$. Corresponding performance metrics are:

Task response time:	2.241 time units
Subtask dispersion:	0.511 time units
Energy Cost:	1.577 energy units
RDE^3 Trade-off:	4.495 units

Method 4 Subtask server replication factors $\mathbf{n} = (2, 1, 4)$. The optimal restart subtask service for subtask 3 was ($\tau_3 = \infty$). The resulting delays for subtasks are: $\mathbf{d} = (0, 0.028, 0.233)$. The corresponding performance metrics are:

Task response time:	0.633 time units
Subtask dispersion:	0.174 time units
Energy Cost:	2.314 energy units
RDE^3 Trade-off:	1.362 units

The results indicate that using multiple servers as described in Sec. 4.1 does improve the subtask dispersion, task response time, and energy consumption. Our results show a huge 92% drop in cost associated with the triple trade-off metric as shown by comparison of **Method 1** and **Method 2**. Also it can be seen that the subtask service restart in Sec. 4.2 is also a considerable decrease at 73% as shown by comparison of **Method 1** and **Method 3**. However no further improvement was to be gained with the combination of subtask service restart and multiple servers as is shown by **Method 2** and **Method 4** arriving at the same value and the optimisation choosing to not utilise the subtask restart.

7 CONCLUSIONS

This paper introduced three metrics for measuring the performance of a split–merge system and four techniques that can be used to improve the trade-off of those systems.

From the results in Sec. 6.1 it can be seen that dynamic subtask dispersion control is better than the regular subtask dispersion technique of [9]. Also in the same section it can be observed that the state-dependent delays introduced in Sec. 3 decrease the trade-off cost when compared with a single delay model. The state-dependent delays result in particular is promising as it can be applied in many contexts when optimising between two or more metrics.

From the results in Sec. 6.2 it can be seen that using multiple servers and subtask service reduction are both able to reduce the cost of the trade-off metric defined in Sec. 4.4. However their combination is not able to further reduce the cost, as the combination of the techniques failed to deliver an improvement when compared to subtask service replication alone.

An interesting avenue of further research will be to investigate extending our techniques from split–merge queueing systems to fork–join queueing systems. In fork–join systems tasks are queued at the subtask level, which improves task response time, as subtasks from the next task can begin service before all the sibling subtasks of the current task have finished. However, their analysis is generally acknowledged as being considerably harder on account of the lack of a synchronisation point at the start of every task.

REFERENCES

[1] Hossein Abouee-Mehrizi and Opher Baron. 2016. State-dependent M/G/1 queueing systems. *Queueing Systems* 82, 1-2 (2016), 121–148.

[2] The GPyOpt authors. 2016. GPyOpt: A Bayesian Optimization framework in python. http://github.com/SheffieldML/GPyOpt. (2016).

[3] Ana Paula Couto Da Silva, Michela Meo, and Marco Ajmone Marsan. 2012. Energy-performance trade-off in dense WLANs: A queuing study. *Computer Networks* 56, 10 (2012), 2522–2537.

[4] Anshul Gandhi, Varun Gupta, Mor Harchol-Balter, and Michael A Kozuch. 2010. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation* 67, 11 (2010), 1155–1171.

[5] Carl M. Harris. 1967. Queues with State-Dependent Stochastic Service Rates. *Operations Research* 15, 1 (1967), 117–130. http://www.jstor.org/stable/168515

[6] Andrea Marin and Sabina Rossi. 2016. *Dynamic Control of the Join-Queue Lengths in Saturated Fork-Join Stations.* Springer International Publishing, Cham, 123–138. https://doi.org/10.1007/978-3-319-43425-4_8

[7] Tommi Pesu and William J. Knottenbelt. 2015. Dynamic Subtask Dispersion Reduction in Heterogeneous Parallel Queueing Systems. *Electronic Notes in Theoretical Computer Science* 318 (2015), 129 – 142. https://doi.org/10.1016/j.entcs.2015.10.023

[8] J. G. Shanthikumar. 1979. On a single-server queue with state-dependent service. *Naval Research Logistics Quarterly* 26, 2 (1979), 305–309. https://doi.org/10.1002/nav.3800260210

[9] Iryna Tsimashenka and William J. Knottenbelt. 2013. Trading Off Subtask Dispersion and Response Time in Split-Merge Systems. In *Proc. ASMTA 2013.* Ghent, Belgium, 431–442.

[10] Iryna Tsimashenka, William J Knottenbelt, and Peter G Harrison. 2012. Controlling Variability in Split-Merge Systems.. In *ASMTA*. Springer, 165–177.

[11] Aad PA Van Moorsel and Katinka Wolter. 2006. Analysis of restart mechanisms in software systems. *IEEE Transactions on Software Engineering* 32, 8 (2006), 547–558.

[12] Sebastian Zander, Ian Leeder, and Grenville Armitage. 2005. Achieving Fairness in Multiplayer Network Games Through Automated Latency Balancing. In *Proc. 2005 ACM SIGCHI ACE.* 117–124. https://doi.org/10.1145/1178477.1178493

A RESPONSE TIME, SUBTASK DISPERSION AND ENERGY CONSUMPTION TRADE-OFF MINIMUM PROBLEMS

This section shows that optimal RE and DE sometimes have trivial solutions indicating an infinite number of servers. As a consequence RDE^2 will also suffer from the same problem. Therefore our experiments use the product RDE^3. The following split–merge example is used: $X_1 = \text{Exp}(\mu_1 = 1), X_2 = \text{Exp}(\mu_2 = 2)$

A.1 Task Response Time and Energy Consumption Trade-Off as n approaches ∞

The minimum of two exponentially distributed random variables is exponentially distributed with the μ_s given below:

$$\mu_{\min} = \mu_1 + \mu_2 \tag{23}$$

Duplicating service server n times results in a service rate of $n\mu$. Given that the variance of an exponential distribution is μ^{-2}, the variance with n duplicated servers is $(n\mu)^{-2}$.

Substituting $n\mu$ for μ into the Pollaczek–Khinchine formula results in the following Task Response Time and Energy trade-off:

$$\text{Resp}(\lambda) = \frac{\lambda/(n\mu) + n\mu\lambda(n\mu)^{-2}}{2(n\mu - \lambda)} + (n\mu)^{-1} \tag{24}$$

$$\text{Energy}(n) = kn \tag{25}$$

Multiplying the two metrics and simplifying we obtain:

$$\text{Resp}(\lambda)\text{Energy}(n) = \frac{k\lambda\mu^{-1}}{(n\mu - \lambda)} + \frac{k}{\mu} \tag{26}$$

From this it is apparent that when the number of servers $n \to \infty$ the Trade–Off approaches k/μ, which is the optimal.

A.2 Subtask Dispersion and Energy Consumption Trade-Off as n approaches ∞

When using dispersion technique from [7] the optimal strategy is the have the slower subtask server finish service first and only then begin work on the second subtask. Assuming we duplicate the faster server n times results in the following subtask dispersion and energy usage:

$$\text{Disp}(n) = \frac{1}{n\mu} \tag{27}$$

$$\text{Energy}(n) = 1 + n \tag{28}$$

Multiplying the two metrics and simplifying we obtain:

$$\text{Disp}(n)\text{Energy}(n) = \frac{1+n}{n\mu} = \frac{1}{n\mu} + \frac{1}{\mu} \tag{29}$$

From this it is visible that when the number of servers $n \to \infty$ the Trade–Off approaches μ^{-1}, which is the optimal.