The Knowledge Engineering Review, Vol. 34, e9, 1 of 20. © Cambridge University Press, 2019
doi:10.1017/S0269888919000067

Time-sensitive resource re-allocation strategy for interdependent continuous tasks

Valeriia Haberland[1], Simon Miles[2] and Michael Luck[2]

[1]*MRC Integrative Epidemiology Unit, Population Health Sciences, Bristol Medical School, University of Bristol, Bristol BS8 2BN, UK*
e-mail: valeriia.haberland@bristol.ac.uk
[2]*Department of Informatics, King's College London, London WC2R 2LS, UK*
e-mail: simon.miles@kcl.ac.uk; michael.luck@kcl.ac.uk

Abstract

An increase in volumes of data and a shift towards live data enabled a stronger focus on resource-intensive tasks which run continuously over long periods. A Grid has potential to offer the required resources for these tasks, while considering a fair and balanced allocation of resources among multiple client agents. Taking this into account, a Grid might be unwilling to allocate its resources for long time, leading to task interruptions. This problem becomes even more serious if an interruption of one task may lead to the interruption of dependent tasks. Here, we discuss a new strategy for resource re-allocation which is utilized by a client with the aim to prevent too long interruptions by re-allocating resources between its own tasks. Those re-allocations are suggested by a client agent, but only a Grid can re-allocate resources if agreed. Our strategy was tested under the different Grid settings, accounting for the adjusted coefficients, and demonstrated noticeable improvements in client utilities as compared to when it is not considered. Our experiment was also extended to tests with environmental modelling and realistic Grid resource simulation, grounded in real-life Grid studies. These tests have also shown a useful application of our strategy.

1 Introduction

As our day-to-day life gradually becomes more automated, research has substantially focused on smart systems which, for example, can control climate parameters inside a building, or monitor the level of pollution in the environment (Ghanem *et al.*, 2004). Sensor networks, which produce huge amounts of data continuously, are applied in healthcare, weather monitoring, smart city concepts, etc. (Llanes *et al.*, 2016). In many cases, it is essential for these systems to respond in a real-time manner (Llanes *et al.*, 2016) and therefore, obtain and process data continuously over time. Hence, the tasks which are assigned to process these data streams also have to run continuously and potentially for an undefined or substantially long period of time (Haberland *et al.*, 2014; Haberland *et al.*, 2017*a*, 2017*b*). Ideally, these systems would aim to run these tasks without interruptions, but if durations are short enough relative to the nature of a task, they may not affect a monitored physical parameter (e.g. temperature levels) to large extent. A Grid (Foster *et al.*, 2001) has a large amount of distributed resources as well as an established infrastructure, and therefore could be a suitable solution for this type of task, which may also be geographically distributed. We focus only on the non-commercial solutions, where 'payment' is measured in terms of the allocated time. As these tasks are generally unbounded in terms of time, a Grid might be unable or unwilling to commit to the unlimited (long) periods of time in advance for various reasons, for example, a difficulty to predict the resource availability. As a result, each task could obtain only a time-limited resource allocation and it has to request resources again when this period of time elapses (Haberland *et al.*, 2017*a*, 2017*b*). A task might also be interrupted due to any type of failure.

These tasks might be independent but likely dependent or linked in terms of the data. For instance, data streams which are responsible for temperature and humidity monitoring can be linked as they indicate the airport's weather condition (Le-Phuoc *et al*., 2012). In another case, the vehicles' speed and location can be monitored (Sequeda and Corcho, 2009), for example, to identify traffic congestion. These examples show scenarios where several continuous tasks may stream their data simultaneously to be aggregated by another task. These scenarios have informed our model development, which focuses on continuous data flow from the bottom to the top of task hierarchy. As compared to common models of dependent tasks (often shown as *directed acyclic graphs (DAGs)*), our tasks do not have only direct dependence of sender–recipient but also a reverse dependence, meaning a failure of the recipient-task adversely affects its corresponding sender-tasks as well. This occurs as data are assumed to stream in real time and if cannot reach an intended recipient, then there is no utility gained by a sender-task for every missed data instance. Our model assumes a gradually increasing negative impact on the client utility during data exchange delays. If data are not received in time by a recipient, then the client's system will produce results with some degree of error, where longer delay leads to larger probability that the last received data instance substantially differs from the current would-be received data instance. A recipient has to stop at some point, avoiding to generate significantly deviated results. Existing work generally does not focus on how a client agent can avoid or reduce these delays if Grid resources are contested and/or limited and how delays may affect its system, where their effect also accumulates over time due to the continuous nature of these tasks.

The aim of our work is to develop a methodological approach which can be used to reasonably shorten the duration of interruptions for this type of task, reducing negative impact on the overall client utility. We describe a new strategy for resource re-allocation, *SimTask*[1], which allows a client agent to re-assign resources between its own tasks by negotiating these re-allocations with a *Grid Resource Allocator* (GRA). Resource re-allocation means that resources which have been allocated to a task are re-assigned to another task, and this can be implemented by the GRA with an appropriate intelligence capacity and policy. An essential assumption in our work is that a pseudo-periodicity exists in fluctuations of the resource availability in a Grid as discussed in many studies (Andrzejak and Ceyran, 2005; Iosup *et al.*, 2008; Kondo *et al.*, 2004), and this allows us to utilize some level of its predictability with the best outcome for a client (Haberland *et al.*, 2017*a*, 2017*b*). Existing work considers resource re-allocation mainly for the purpose of load balancing, for example (Meriem and Belabbas, 2010), (Yan *et al.*, 2007), while we aim to avoid an accumulation of long interruptions which would substantially affect a client's system (e.g. if there are regular and significant drops in temperature). This strategy consists of a decision-making mechanism for a client agent (referred as a client) to initiate resource re-allocation, choose the most suitable task for resource donation and update the execution states of its tasks. The abilities of agents (Wooldridge and Jennings, 1995) to decide autonomously, respond actively to any changes and meaningfully communicate are important for this strategy, considering the large number of negotiating agents.

The paper is structured as follows. Section 2 discusses related work in respect of the interdependent tasks. Then, Section 3 describes the formal model, while Section 4 presents our SimTask re-allocation strategy for a client and the algorithms which demonstrate how this strategy works in full such as when multiple tasks are interrupted. The evaluation results, including our case study with a realistic simulation of Grid resources and environment, are discussed in Section 5, and Section 6 concludes the paper.

2 Related work

In this paper, we focus on tasks with specific time-related constraints such as they are expected to run near-continuously (Haberland *et al.*, 2014; Haberland *et al.*, 2017*a*, 2017*b*), producing data in real time.

[1] This paper is an extended version of the paper presented at the joint European Conference on Multi-Agent Systems and Agreement Technologies by Haberland *et al.* (2017*b*). It extends a previously described strategy with detailed algorithms, an illustrative example, and an extended evaluation using a realistic Grid resource simulator and environment models, while also clarifying some model choices.

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

The tasks also depend on the data produced by other tasks. That is, a task sends a data instance to another task, while processing new input data. These tasks are running simultaneously in terms of processing each input data instance as soon as it has arrived, repeating these actions over time. In this section, we discuss different solutions and models, suggested by other researchers, which consider task interdependency in terms of data. The interdependent tasks can be found in many different research areas, and our focus in this review is primarily on dependent but near-continuous tasks at the same time.

Conventionally, the dependencies among tasks in Grid systems are depicted as a *DAG*, for example (Jin *et al.*, 2005; Lee *et al.*, 2009; Meriem and Belabbas, 2010; Zhao and Sakellariou, 2004). Dynamic allocation of such tasks has also been discussed such as by Meriem and Belabbas (2010). Meriem and Belabbas (2010) allocate dynamically tasks to resources as they arrive continuously over time. Here, dynamic allocation is intended to account for any resource availability fluctuations in a Grid, and resolves the problem of load balancing at runtime. Although this work addresses task dependencies in their resource allocation solutions, tasks are not meant to be repeated continuously in real time. However, this research points out important concepts which can be applied to near-continuous tasks such as *spare time* (Zhao and Sakellariou, 2004), which defines the longest time of task execution before it has an effect on the execution plan of dependent tasks. Some work such as Sandnes and Sinnen (2005), Sardinha *et al.* (2012) and Yang and Fu (1997) also considers a model of interdependencies among tasks where tasks are run repeatedly and each task is both a recipient and sender in a cycle. These dependencies are represented as a *cyclic task graph*, where cyclic dependencies may refer to data, instructions or something else.

There are other examples of dependent tasks discussed in the literature, for example, Decker and Lesser (1992), Lesser *et al.* (2004), Lesser (1991), Jin *et al.* (2005) and Lee *et al.* (2009). For example, Lesser *et al.* (2004) focuses on achieving a higher-level task by completing other *time-constrained and possibly interdependent tasks*. Here, the tasks might have a hierarchical structure; for example, the sub-tasks can be to check the websites *A* and *B* for a price on product *P* and the higher-level task can be to compare those prices. Each task might have several sub-tasks as well as several higher-level tasks, which directly or indirectly depend on success in completing this task. Lesser *et al.* (2004) offer an approach in planning mechanisms, where a task can be completed with some degree of quality rather than just completed or not by a deadline. Some tasks can be executed simultaneously and some may need to wait for results from other tasks and, as a result, their execution might be delayed.

Another example of dependent tasks can be found in Motwani *et al.* (2003), whose work focuses on *continuous queries* (e.g. continuous tasks) that are intended to process the streams of data from multiple sources. A continuous query is issued once for a specific type of data and then runs continuously, producing new results for a client (Babu and Widom, 2001; Terry *et al.*, 1992). In the work Motwani *et al.* (2003), one query may consist of a number of other queries (operators), where the outputs of these queries might be shared with some other queries. Although this work discusses the techniques to approximate the query outputs in the case of scarce resources, it does not focus on how data delays or failed queries might affect the results from other queries, or whether the latter query can even be performed.

Different platforms (engines), for example, *Apache Storm* (Apache, 2014), *Esper* (EsperTech, 2014), *StreamBase* (TIBCO, 2014) and *Cyclops* (Lim and Babu, 2013), attempt to solve the problems related to the performance, scalability and memory usage for data stream processing. However, the problem of data interdependencies among tasks as discussed above and how or whether they could possibly be executed without all or some of the required input data is not the focus of these engines. In the case of resource failure, it is assumed that tasks can be reassigned to another computational node (Apache, 2014), but the level of error (if applicable) because of interruption in data processing is not clear as well as whether such a node is always available. In open computational environments such as a Grid, where multiple clients request resources at the same time, it might be difficult to re-allocate a task without affecting the performance of other tasks. A client is generally not well-equipped in existing approaches in terms of the decision-making mechanisms to effectively influence an allocation and execution of this type of task.

3 Formal model

In this work, we consider that tasks are interdependent, where some of them send data, *sender-tasks*, while others receive data, *recipient-tasks*. The dependencies among tasks are presented as a *rooted tree Tr*,

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

where data are considered to flow from the bottom to the tree top (i.e. from leaf to root), following a scenario of data aggregation (e.g. by counting, adding, etc., Barbieri *et al.* (2009)) from multiple sources (e.g. sensors). Each node in this tree denotes a task i and each edge indicates an interdependence between sender $i \in \mathbb{N}$ and recipient $j \in \mathbb{N}$ with a weight $\alpha_{i,j} \in [0, 1]$. A weight denotes how important the data produced by a specific sender-task is for its intended recipient-task, that is each edge points from the lower-layer tasks (senders) towards the upper-layer tasks (recipients) of the tree. This way, some tasks in the middle of the tree are also senders and recipients at the same time. Here, our model has a simplifying assumption that each sender has only one recipient, but every recipient might have one or more senders. We also assume that the sum of weights for all senders which are connected directly to the same recipient is equal to 1.0, and smaller weights $\alpha_{i,j}$ indicate less impact of sender, i, on the work of its recipient, j. We also consider a *sub-tree* $sTr_k \in Tr$ with the root task $k \in \mathbb{N}$.

Here, we define an abstract *parameter* P which is estimated by client tasks. This parameter $P_{i,S_i}(t) \in \mathbb{R}$ for task i with the corresponding set of direct senders $S_i = \{m, \ldots, k\}$ is a real-life characteristic (e.g. temperature), which is continuous or can be presented as continuous over time $t \in \mathbb{R}$, considering $\left|\left(P_{i,S_i}(t + \Delta t) - P_{i,S_i}(t)\right) / P_{i,S_i}(t)\right| \ll 1$, where $\Delta t \in \mathbb{R}$ is an arbitrary small time step. The parameter $P_{i,S_i}(t)$ is estimated directly by task i, if this task belongs to the tree's lowest layer (e.g. reads data directly from a sensor), that is, $S_i \in \emptyset$. If task i belongs to any tree's upper layer, that is, $S_i \neq \emptyset$, then $P_{i,S_i}(t)$ is estimated as a linear combination of all parameters $P_{j \neq i,S_j}(t)$ sent by its sender-task(s)[2] $P_{i,S_i}(t) = \sum_{j \in S_i} \alpha_{j,i} \times P_{j \neq i,S_j}(t)$, $S_i \neq \emptyset$, $i, j \in \mathbb{N}$, where S_j can be empty set.

3.1 Status and layer

In our model, each task i has its *status* $Status_i(t)$ of execution at time t, which can be as follows:

- *'Interrupted'*: a task is not running as no resources are allocated to this task;
- *'Stopped'*: a task is not running as its recipient-task is interrupted/stopped, or it has not received the accurate data from at least one of its sender-tasks for too long, but resources are still allocated to this task by the GRA;
- *'Inaccurate'*: a task is running, but at least one of its sender-task(s) either does not send any data or sends inaccurate data;
- *'Accurate'*: a task is running and all its sender-tasks send accurate data.

A task produces inaccurate data when at least one of its senders has status other than 'accurate'.

Each task i also belongs to a particular layer $Layer_i$ in a tree, and sends its data, except the root task, to the corresponding recipient j which belongs to the nearest upper layer, that is, $Layer_j = Layer_i + 1$. Note that the corresponding lower-layer tasks are stopped when their recipient at the top of a sub-tree (or the root) is stopped or interrupted. This reflects their continuous and real-time nature, meaning data become obsolete if not processed in real time.

3.2 Damping and delay time

Following a notion of short interruption, we define a *damping time* which determines for how long a task can be interrupted or stopped without substantial negative impact on parameter estimation, for example, a large temperature drop. If any task has been interrupted or stopped, then it ceases to estimate this parameter. From this point, a parameter starts changing in a way which is out of control of a client. The longer this task is idling, the higher the chance this parameter will change in a way that will have a significant negative effect for a client. We consider that this effect occurs when the damping time $\tau_i^{dam}(t_d)$, counted from $t_d \in \mathbb{R}$, has elapsed for task i. That is, the absolute difference $\Delta P_{i,S_i}(t)$ between the last produced value of parameter $P_{i,S_i}(t_d)$ by task i before interruption and the linearly extrapolated

[2] A linear combination (instead of a possible variety of nonlinear solutions) is chosen for a greater clarity of presentation and evaluation of SimTask.

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

value of this parameter $P^{ex}_{i,S_i}(t)$ at time t becomes larger than the predefined threshold $\eta^{dam}_i \in \mathbb{R}$. This threshold is determined by the nature of this task such as if we consider an estimated parameter to be temperature, then the change in 3 °C would be noticeable in the room and therefore will inform the choice of η^{dam}_i.

A *delay time* determines for how long a task can be running, using inaccurate data, and a task stops as soon as this time elapses. A delay time $\tau^{del}_i(t_{dl}, t)$, which starts at $t_{dl} \in \mathbb{R}$, for recipient-task $i \in \mathbb{N}$ is the duration of time when this task is still running but it uses inaccurate input data due to the interruption of some sender(s) (at least one) in its sub-tree sTr_i. This time terminates when the absolute difference $\Delta P_{i,S_i}(t) = \sum_{j\in S_i} \alpha_{j,i} \Delta P_{j\neq i,S_j}(t)$, $S_i \neq \emptyset$, becomes larger than the predefined threshold $\eta^{del}_i \in \mathbb{R}$, where we consider that $\eta^{del}_i < \eta^{dam}_i$. Note that the difference $\Delta P_{i,S_i}(t)$ is a linear combination of such differences for the lower-layer tasks from a sub-tree sTr_i and have the execution statuses of 'inaccurate', 'stopped' or 'interrupted'. For the lowest layer tasks, that is, $S_j = \emptyset$, these differences at time t are calculated as $\Delta P_{j,S_j}(t) = P_{j,S_j}(t_{dl}) - P^{ex}_{j,S_j}(t)$, where $P_{j,S_j}(t_{dl})$ is the last value of parameter produced by task j before its interruption at time t_{dl}. The difference $\Delta P_{i,S_i}(t)$ may change dramatically if some senders switch to other execution statuses. The delay time for recipient i becomes longer (i.e. $\Delta P_{i,S_i}(t)$ becomes smaller), if at least one of its sender(s) switches to the 'accurate' status, and shorter if it switches the opposite way.

3.3 Client utility

In our model, each task i is *near-continuous* (Haberland *et al.*, 2014; Haberland *et al.*, 2017*a*,*b*), that is, it has periods of interruption $\tau^{int}_{i,l} \in \mathbb{R}$ and execution $\tau^{exe}_{i,l} \in \mathbb{R}$, and the pairs of consecutive interruption and execution periods $\left(\tau^{int}_i, \tau^{exe}_i\right)_l$ have a counter $l \in \mathbb{N}$ within a total duration of task execution $\tau^{tot} \in \mathbb{R}$. Each $\tau^{int}_{i,l}$ starts at $t^{end}_{i,l-1}$ and ends at $t^{str}_{i,l}$, while each $\tau^{exe}_{i,l}$ starts at $t^{str}_{i,l}$ and ends at $t^{end}_{i,l}$. τ^{tot} is a total duration of execution for all tasks (as they are simultaneous), starting at t^{str}_{tot} as soon as the first resource request was submitted by a client, and ending at t^{end}_{tot}. If one task stops running, this starts adversely affecting all lower- and upper-layer tasks from the same sub-tree or branch at once. Our model also accounts for a *cumulative duration of interruptions* $\tau^{cum}_{i,l} = \sum_{k=1}^{l} \tau^{int}_{i,k}$ for each task i which is a total duration of all interruptions including $\tau^{int}_{i,l}$. A cumulative duration is intended to show how successfully the task was executed in general.

Additionally to these interruptions (Haberland *et al.*, 2014; Haberland *et al.*, 2017*a*), our model of interdependent tasks (Haberland *et al.*, 2017*b*) also considers inaccurate processing of data as a factor which has an adverse effect on client utility. That is, the longer the task is running with inaccurate input data (or idling), the more substantial is the negative impact on client utility. The impact of any adverse factor (interruption or inaccurate data processing) is designed as the corresponding *damping* function (monotonic and nonlinear): (I) $S(\tau^{int}_{i,l})$ for a single interruption; (II) $C(\tau^{cum}_{i,l})$ for a cumulative interruption and (III) $I(\hat{\tau}^{del}_i(t_{dl}, t))$ for a duration of inaccurate data processing $\hat{\tau}^{del}_i(t_{dl}, t)$, starting at t_{dl}. $\hat{\tau}^{del}_i$ denotes a part of delay time τ^{del}_i which has passed up to t.

Each damping function produces values from the interval [0, 1], where 1 denotes no impact. These functions reflect our assumption that the adverse effect on client utility might not be significant as long as the influence of adverse factors is reasonably short. For example, temperature levels may not change noticeably in a few minutes. Note that only execution periods are counted for utility gains, and the amount of such gain is affected negatively by these damping functions. We describe them first in the abstract way, assuming $F(x)$ is the damping function and x is some adverse factor (e.g. the duration of interruption, $\tau^{int}_{i,l}$). Then, the general equation for all of them would be

$$F(x) = \frac{1}{e^{(x-x^{max})/\epsilon} + 1} \tag{1}$$

where x^{max} denotes the critical value of the adverse factor x, while ϵ reflects the speed of decrease of $F(x)$ around x^{max}. Now, the functions $S(\tau^{int}_{i,l})$, $C(\tau^{cum}_{i,l})$ and $I(\hat{\tau}^{del}_i(t_{dl}, t))$ have their critical values (inflection points) set to $\tau^{max}_{int[i]}(t_d)$, $\tau^{max}_{cum[i]}(t_d)$ and $\tau^{max}_{del[i]}(t_{dl}, t)$, which denote the critical length of time for the client utility as if passed the utility would be noticeably affected by the corresponding time-related factor, and

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

their $\epsilon_{int[i]}(t_d)$, $\epsilon_{cum[i]}(t_d)$ and $\epsilon_{del[i]}(t_{dl}, t)$ are calculated in proportion to the corresponding critical values. As $S(\tau_{i,l}^{int})$ and $C(\tau_{i,l}^{cum})$ show the impact of interruptions on the client utility, their critical values can be calculated proportionally to the corresponding damping time $\tau_i^{dam}(t_d)$. In this work, $\tau_{int[i]}^{max}(t_d) = \tau_i^{dam}(t_d)$ as the damping time determines how fast an interruption would have a substantial effect on the utility in terms of the unestimated changes in the parameter. Considering $I(\hat{\tau}_i^{del}(t_{dl}, t))$ shows an impact of inaccurate data processing on the client utility, its critical value is set to the delay time $\tau_i^{del}(t_{dl}, t)$.

In our work, the *effectiveness function* $E\ (t)$ (Haberland *et al.*, 2014; Haberland *et al.*, 2017*a*, 2017*b*) demonstrates the success of task execution over time t. This function changes only during execution periods, and it can be reduced if multiplied by the values of damping functions. If not affected by the adverse factors, it should increase linearly during task execution, starting from the same value at which it ended before interruption. Hence, an estimate $Es\left(t, E\left(t_{i,l-1}^{end}\right)\right)$ is also linearly increasing during an execution period:

$$Es\left(t, E\left(t_{i,l-1}^{end}\right)\right) = \frac{\left(1 - E\left(t_{i,l-1}^{end}\right)\right)t + E\left(t_{i,l-1}^{end}\right)t_{tot}^{end} - t_{i,l}^{str}}{t_{tot}^{end} - t_{i,l}^{str}} \tag{2}$$

This estimate $Es\ (\cdot)$ starts increasing from the value of effectiveness function $E\left(t_{i,l-1}^{end}\right)$ at the end of previous execution period $\tau_{i,l-1}^{exe}$ towards the desirable end of execution at t_{tot}^{end} when the value of effectiveness function is equal to 1. The full effectiveness function, affected by the adverse factors, is presented below (we use simplified notation to highlight the differences with our previous $E(t)$):

$$E(t) = \begin{cases} Es(t) \times S \times C \times I(t), & if\ \tau_{i,l}^{exe} \neq 0\ and\ \tau_i^{del}(t_{dl}, t) \neq 0 \\ Es(t) \times S \times C, & if\ \tau_{i,l}^{exe} \neq 0\ and\ \tau_i^{del}(t_{dl}, t) = 0 \\ E(t = t_{i,l-1}^{end}), & if\ \tau_{i,l}^{exe} = 0 \end{cases} \tag{3}$$

Note that the values of $S(\tau_{i,l}^{int})$ and $C(\tau_{i,l}^{cum})$ are constants within an execution period $\tau_{i,l}^{exe}$ (i.e. $\tau_{i,l}^{exe} \neq 0$), while the values of $I(\hat{\tau}_i^{del}(t_{dl}, t))$ decrease within this period. $I(\hat{\tau}_i^{del}(t_{dl}, t))$ affects the effectiveness of task execution only if task i uses inaccurate input data (i.e. $\tau_i^{del}(t_{dl}, t) \neq 0$) from its sender(s).

In an ideal scenario of no interruptions as described previously, $E(t)$ would be a line between 0 and 1 at t_{tot}^{end}, forming a triangular shape. Due to the interruptions, this shape will not be completely filled though. Therefore, we calculate a total square under the broken curve of $E(t)$ for each task i, counting only execution periods. The utility U_i for i is then

$$U_i = 1/Sq_{max} \sum_{l=1}^{L_i} \int_{t_{i,l}^{str}}^{t_{i,l}^{end}} E(t)dt \tag{4}$$

Here, $Sq_{max} = \tau^{tot}/2$ is the largest possible square under $E(t)$ if task i has no failures till t_{tot}^{end} (i.e. a completely filled triangular shape) and L_i is the number of execution periods within $[t_{tot}^{str}, t_{tot}^{end}]$.

The total client utility U_{total} is calculated as a sum of all U_i with the respective weights ϖ_i which denote how relevant this task is for the client:

$$U_{total} = \sum_{i=1}^{N} \varpi_i \times U_i \tag{5}$$

where N denotes the total number of client tasks. The sum of ϖ_i over all client tasks is equal to 1.0, where the total sum of all ϖ_i from the same tree layer is equal for each layer. In this way, the upper-layer tasks influence the client utility more substantially as their performance is crucial for the lower-layer tasks.

4 Resource re-allocation strategy

Here, we describe our re-allocation strategy, *SimTask*, which is designed to help a client decide on re-allocation of resources among its own tasks by negotiating such re-allocation with the GRA. As a result, some tasks would be able to resume their execution (unless other conditions apply) once resources have been allocated, and the tasks which donated their resources, that is *donor-tasks*, would be interrupted. A

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

client can only ask the GRA to re-allocate resources between its own tasks, but not from other clients. The aim of this internal resource re-allocation is to avoid too long interruptions which might lead to substantial utility loss. Considering the length of time to be a resource only negotiated with the GRA, a task cannot share this resource with another task and can only donate it in full. The GRA is modelled as an abstract actor whose behaviour is reflective of the resource availability and demand (and also covertly indicates a success of clients in resource negotiation), while our strategy focuses on ensuring as seamless as possible execution of the discussed type of task. The scalability of this approach can potentially be improved if tasks are clustered into the local data-independent groups, following their own tree structure and coordinated by their respective agents, which will likely be our future work.

A problem following from resource re-allocation does not only constitute a decision with regard to which task should donate resources with a possibly smaller loss in utility, but also whether these actions and to which degree are tolerated by the GRA. Generally, the GRA is assumed to allow resource re-allocations among tasks of the same client, but only with a penalty due to its own resource cost, that is the GRA has to use its time and physical resources in order to make a decision and then transfer a task in question to another resource if applicable. As a result, a task might be allocated a much shorter remaining execution period of its donor instead of the whole available one at the time of agreement.

4.1 Condition of application

A client decides whether a task was idle for too long and, therefore, if it should be donated a resource from another task. The resource re-allocation from one task to another one might not always be beneficial for a client, because a donor-task has to be interrupted instead of the currently idle one and the re-allocated remaining execution period can be shortened by the GRA. However, if any task is idling for so long that its damping time $\tau_i^{dam}(t_d)$ is exceeded, then the client's utility will be substantially decreased. For example, the prolonged interruption of the sender-task might affect its recipient which could be stopped due to the large error in its estimations. As a result, the interrupted task has to be allocated resources before its damping time has elapsed. Then, the condition for resource re-allocation is

$$\hat{\tau}_{i,l}^{int}(t) > k_i^{dam} \times \tau_i^{dam}(t_d) \tag{6}$$

where $\hat{\tau}_{i,l}^{int}(t)$ is the duration of interruption at t and $k_i^{dam} \in [0, 1]$ determines to which extent a client agent is willing to take a risk of exceeding the damping time. If an interruption becomes longer than the specified part of the damping time, a client initiates negotiation with the GRA to re-allocate resources among its tasks.

4.2 Criteria of decision-making mechanism

If a client decides that the interrupted task was idling for too long, it then needs to choose a donating task which remaining execution period (or a part of it) might be re-allocated. Here, a client aims to choose such donor which would have the smallest impact on the client utility, if it stops running. We present two criteria to choose the best donor-task, where the first one reflects the duration of time to be re-allocated to the idling task and the second one shows the effect of donor-task's interruption on a task tree (ideally, resources should be re-allocated from less to more relevant tasks if needed).

4.2.1 Remaining execution period

In general, a client prefers to allocate a longer execution period to the idling task, and this period should preferably terminate in the proximity of a peak of resource availability (Haberland *et al.*, 2014; Haberland *et al.*, 2017*a*). Within the scope of this paper, we only highlight the importance of starting negotiation when resources are less scarce, which is a major decision component of our previously developed negotiation strategy, ConTask. It is desirable therefore for the donor's remaining execution period to cover at least one peak of resource availability. A client also has to account for the fact that the re-allocated remaining execution period $\tau_{j,l}^{rem}(t)$ can substantially be shortened by the GRA.

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

Accounting for the mentioned above considerations, a client is designed to find donor j with a remaining execution period $\tau_{j,l}^{rem}(t)$ closer to an arithmetical average $\tau_{av}^{rem}(t)$ of the minimum acceptable $\tau_{min}^{rem}(t)$ and maximum available $\tau_{max}^{rem}(t)$ remaining periods among all client tasks which have resources at t. The maximum available remaining period $\tau_{max}^{rem}(t)$ is the longest remaining execution time available among client tasks, naturally excluding those tasks which are interrupted. Ideally, the minimum acceptable donor's remaining execution period $\tau_{min}^{rem}(t)$ should allow the recipient of its resources to terminate around the closest peak of resource availability from the current point in time. However, if all available remaining execution periods have no peaks of resource availability, the minimum acceptable remaining period is calculated proportionally to the maximum available remaining execution time, that is, $\tau_{min}^{rem}(t) = k^{rem} \times \tau_{max}^{rem}(t)$ with $k^{rem} \in [0, 1]$.

Then, the first criterion, $Rem_{j,l}(t)$, is a function which formalizes the client's preference with regard to the length of the remaining execution time and produces values from 0 to 1, that is from the worst to the best donor candidate. If the remaining execution period is shorter than the minimum acceptable one, this function will return a negative number, which is algorithmically replaced with zero. Although those tasks are not excluded entirely as possible donors, they are unlikely to be chosen. This function for the donor candidate, j, at time t is presented as:

$$Rem_{j,l}(t) = \frac{\left(\tau_{j,l}^{rem}(t) - \tau_{min}^{rem}(t)\right) \times \left(\tau_{max}^{rem}(t) - \tau_{j,l}^{rem}(t)\right)}{\left(\tau_{av}^{rem}(t) - \tau_{min}^{rem}(t)\right) \times \left(\tau_{max}^{rem}(t) - \tau_{av}^{rem}(t)\right)} \tag{7}$$

4.2.2 Dependencies of the donor-task

A client aims to avoid a substantial negative impact on its utility when a chosen donor loses its resource. Assume that a client has a list of donor candidates and each of them has some remaining execution time. However, these candidates have different levels of importance to their corresponding recipient(s) and those recipients may have different execution statuses. If the recipient of a potential donor is running, then a client has to estimate when this recipient stops due to inaccurate input data, if its sender is chosen to be a donor. If this potential donor is of less importance to its recipient, then the delay time for this recipient will be longer, meaning it will be running longer (even with inaccurate data, it will still contribute something to the client utility). If the recipient i of a potential donor j is not running (i.e. 'stopped' or 'interrupted'), the preferable donor still has to be one of less importance to its recipient but defined by its weight $\alpha_{j,i}$.

Finally, a client prefers that donor candidate at t which has the smallest level of importance for its recipient. This consideration means that the most preferable donor should ideally have the longest remaining delay time $\check{\tau}_i^{del}(t_{dl}, t) = \tau_i^{del}(t_{dl}, t) - \hat{\tau}_i^{del}(t_{dl}, t)$ for its recipient i when interrupted if its recipient is running, or the smallest level of importance $\alpha_{j,i}$ if its recipient is not running as compared to all other donor candidates. Considering this intuition, we define a time-dependent variable, $Imp_j^i(t_{dl}, t)$, for each potential donor j, which produces values from 0 to 1, meaning from the least to the most desirable donor (this applies to other variables below).

$$Imp_j^i(t_{dl}, t) = \begin{cases} \dfrac{\check{\tau}_i^{del}(t_{dl}, t) - \check{\tau}_{min}^{del}(t)}{\check{\tau}_{max}^{del}(t) - \check{\tau}_{min}^{del}(t)}, & \textit{when recipient i is running} \\[2ex] \dfrac{\alpha_{max} - \alpha_{j,i}}{\alpha_{max} - \alpha_{min}}, & \textit{when recipient i is not running} \end{cases} \tag{8}$$

where $\check{\tau}_{max}^{del}(t)$ and $\check{\tau}_{min}^{del}(t)$ are the longest and shortest remaining delay times at t among all running recipients of potential donors, estimated under a presumption as if those candidates donated their resources. Then, α_{max} and α_{min} are the largest and smallest levels of importance among all client tasks, respectively (not only donor candidates).

Another consideration is that potential donors from the bottom of a tree are less relevant to the client than from the top as their interruption will cause less disruption for the client's system, and therefore less utility loss. The upper-layer tasks are also assumed to perform calculations related to the larger part of the system, for example, a floor monitoring compared to a room in the building. The root task indicates the

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

highest layer $N_{lay}-1$, while the lowest layer of a tree is considered to be zero layer. Finally, a variable

$$Lay_j = 1 - (Layer_j/(N_{lay}-1)), \quad Lay_j \in [0, 1] \tag{9}$$

is defined, which value also lies between 0 and 1 the same as $Imp_j^i\,(t_{dl}, t)$.

A client also accounts for an execution status $Status_j\,(t)$ (see Section 3.1) of a donor candidate j at t, where 'interrupted' candidates are naturally not considered. The 'stopped' donor candidates are considered as the most desirable for a client with regard to their utility impact, because they are not running anyway. If a potential donor has the status 'inaccurate' or 'accurate' and it is interrupted, then this will affect an execution of all dependent tasks (both at the lower and upper tree layers). Hence, these statuses are regarded as equally non-preferable. Finally, we introduce a variable, $Stat_j\,(t)$, that reflects the execution status of a potential donor j as

$$Stat_j\,(t) = \begin{cases} 0, \ if \ Status_j\,(t) = \text{`}interrupted' \\ \lambda, \ if \ Status_j\,(t) = \text{`}inaccurate' \vee \text{`}accurate' \\ 1, \ if \ Status_j\,(t) = \text{`}stopped' \end{cases} \tag{10}$$

where $\lambda \in\,]0, 1[$. Now, we summarize the second criterion for a client to choose the best donor as a function $Dep_j^i\,(t_{dl}, t)$, $j \in S_i$, $j \neq i$, which produces the values from 0 to 1, aggregating all components mentioned above.

$$Dep_j^i\,(t_{dl}, t) = Imp_j^i\,(t_{dl}, t) \times Lay_j \times Stat_j\,(t) \tag{11}$$

A final decision is produced by function $Donor_{j,l}^i\,(t_{dl}, t)$ for each donor candidate j. This function combines both client criteria, $Rem_{j,l}\,(t)$ and $Dep_j^i\,(t_{dl}, t)$, and calculates a score between 0 (worst) and 1 (best) for each potential donor:

$$Donor_{j,l}^i\,(t_{dl}, t) = W_{rem} \times Rem_{j,l}\,(t) + W_{dep} \times Dep_j^i\,(t_{dl}, t) \tag{12}$$

where the weights W_{rem} and $W_{dep} \in [0, 1]$ and their sum equals to 1. Those weights set priorities for client decisions between the candidate's remaining execution period and its impact on a task tree. A client chooses a donor j with the largest score at time t.

4.3 Algorithm description

This section discusses the full algorithm of choosing a donor-task as well as the consequences of its resource re-allocation for the whole tree of tasks. As above, a client decides when it is necessary to donate a resource to an interrupted task from the chosen donor-task and how to choose the best donor-task. A client also negotiates its decision to re-allocate a resource from one task to another one with the GRA, because only the GRA has an authority and knowledge to re-allocate resources. If the demand on resources is high in a Grid, then the GRA might become greedy in negotiation and this may lead to a failure of negotiation or allocation of the shorter execution time than the remaining execution period of the chosen donor-task.

The expected and unexpected interruptions are both considered for each client's task, where the unexpected interruptions (such as resource failure) may occur randomly r with some probability $Prob$, and the expected ones occur when the execution period $\tau_{i,l}^{exe}$ terminates as described in Algorithm 1. This algorithm shows how the statuses of all dependent tasks (upper- or lower-layer) change, if some of the client's tasks are interrupted. Consequently, if task i is interrupted, then all lower-layer tasks j which belong to its sub-tree $j \in sTr_i$ are 'stopped'. In this case, all upper-layer tasks which depend on the data from task i change their statuses of execution to 'inaccurate' if they were running with 'accurate' input data until now. Otherwise, the tasks do not change their execution statuses. The change of statuses also means a calculation of damping and delay times, according to those statuses, as described in Section 3.2.

The execution statuses of dependent tasks (upper- or lower-layer) may also change when one of the interrupted tasks obtains resources. That is, if task i obtains resources through negotiation with the GRA, its execution status initially switches from 'interrupted' to 'accurate'. However, the 'accurate' status is assigned only nominally, and it can be re-assigned to 'inaccurate' or 'stopped' if task i cannot be

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

Algorithm 1 The change in tasks' statuses when task i is interrupted

```
1:  for Each task $i \in Tr$ do
2:    if $(r < Prob)$ OR $(\tau_{i,l}^{exe}\ ends)$ then
3:      Change $Status_i(t)$ to 'interrupted'
4:      for Each task $j \in Tr,\ \ j \neq i$ do
5:        if $(Status_j(t) \neq$ '$interrupted$'$)$ AND $(j \in sTr_i)$ then
6:          Change $Status_j(t)$ to 'stopped'
7:        end if
8:        if $(Status_j(t) =$ '$accurate$'$)$ AND $(i \in sTr_j)$ then
9:          Change $Status_j(t)$ to 'inaccurate'
10:       end if
11:     end for
12:   end if
13: end for
```

run with accurate input data or cannot be run at all due to the interruption of the dependent lower- or upper-layer tasks. Algorithm 2 shows how the execution statuses of the tasks are re-assigned, if one task obtains resources. Initially, the execution statuses of all non-interrupted tasks are nominally switched to 'accurate'. Then, the execution statuses are changed again, considering all interrupted tasks as in Algorithm 1.

Algorithm 2 The change in tasks' statuses when task i has obtained a resource

```
1:  {Task $i$ has just obtained a resource, i.e. $Status_i\ (t) =$ '$accurate$'}
2:  for Each task $j \in Tr,\ \ j \neq i$ do
3:    if $Status_j\ (t) \neq$ '$interrupted$' then
4:      Change $Status_j\ (t)$ to 'accurate' {This status is assigned temporarily}
5:    end if
6:  end for
7:  for Each task $k \in Tr$ do
8:    if $Status_k\ (t) =$ '$interrupted$' then
9:      Repeat lines 4–11 of Algorithm 1 for each task $j \in Tr,\ \ j \neq k$
10:     {Here, $i$ from Algorithm 1 is substituted by $k$}
11:   end if
12: end for
```

The execution statuses of dependent tasks may change, if one of the corresponding tasks is stopped due to the large error in its parameter's estimation; that is, the error $\Delta P_{k,S_k}\ (t)$ exceeds the predefined threshold η_k^{del}, according to Section 3.2. Algorithm 3 demonstrates how the tasks' execution statuses change when one task is stopped. The stopped task contributes to the error of parameter's estimation in the same way as the interrupted task, because it is not running. The only difference between these execution statuses is that the stopped task can be re-launched immediately, when the source of error disappears; that is its recipient-task or sender-tasks obtain resources. The interrupted task should be allocated resources before it can be re-launched again. In Algorithm 3, a variable *Label* becomes 0 when all tasks have been checked with respect to the exceeded error and all dependent tasks have changed their execution statuses if applicable.

Finally, the whole algorithm of resource re-allocation from one client's task to another one is presented in Algorithm 4. Here, the statuses of all tasks are re-assigned if necessary, according to their execution statuses, every virtual time unit t as presented in lines 4–9, where a Boolean variable OBTAINED becomes true if at least one task has been allocated an execution period during previous time unit. Then, the algorithm calculates the values of $Donor_{j,l}^{i}(t_{dl}, t)$ for each task $j \in Tr$ which is not interrupted at time t, and the

Algorithm 3 The change in the tasks' statuses due to the large difference $\Delta P_{i,S_i}(t)$

```
repeat
  Label = 0
  for Each task $k \in Tr$ do
    if $\Delta P_{k,S_k}(t) > \eta_k^{del}$ then
      Change $Status_k(t)$ to 'stopped'
      Repeat lines 4–11 of Algorithm 1 for each task $j \in Tr$, $j \neq k$ {The status 'stopped' is considered as 'interrupted' in terms of the increase in $\Delta P_{k,S_k}(t)$}
      Label = 1
    end if
  end for
until Label = 0
```

Algorithm 4 Client's SimTask re-allocation strategy based on $Donor_{j,l}^{i}(t_{dl}, t)$

```
OBTAINED = FALSE
repeat
  Each virtual time unit $t$
  if OBTAINED = TRUE then
    Algorithm 2
    OBTAINED = FALSE
  end if
  Algorithm 1
  Algorithm 3
  $Donor_{max}(t_{dl}, t) = 0$
  for Each task $j \in Tr$ with $Status_j(t) \neq$ '$interrupted$' do
    Calculate $Donor_{j,l}^{i}(t_{dl}, t)$ (see Formula (12))
    {Here, task $i$ is the recipient-task in respect of task $j$}
    if $Donor_{j,l}^{i}(t_{dl}, t) > Donor_{max}(t_{dl}, t)$ then
      $Donor_{max}(t_{dl}, t) = Donor_{j,l}^{i}(t_{dl}, t)$
      $i_{best} = j$
    end if
  end for
  for Each task $k \in Tr$ with $Status_k(t) =$ '$interrupted$' do
    Exchange the proposals with the GRA in respect of $\hat{\tau}_{k,r_k}^{exe}(t)$
    if Agreement is reached then
      Change $Status_k(t)$ to 'accurate' {This status is assigned temporarily}
      OBTAINED = TRUE
    end if
    if $(\hat{\tau}_{k,l}^{int}(t) > k_k^{dam} * \tau_k^{dam}(t_d))$ AND $(Status_k(t) =$ '$interrupted$'$)$ AND $(Status_{i_{best}}(t) \neq$ '$interrupted$'$)$ then
      Exchange the proposals with the GRA in respect of $\tau_{i_{best},l}^{rem}(t)$
      if Agreement is reached then
        Change $Status_k(t)$ to 'accurate' {This status is assigned temporarily}
        OBTAINED = TRUE
        Change $Status_{i_{best}}(t)$ to 'interrupted'
      end if
    end if
  end for
until $t_{tot}^{end}$ is reached (see Section 3.3)
```

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

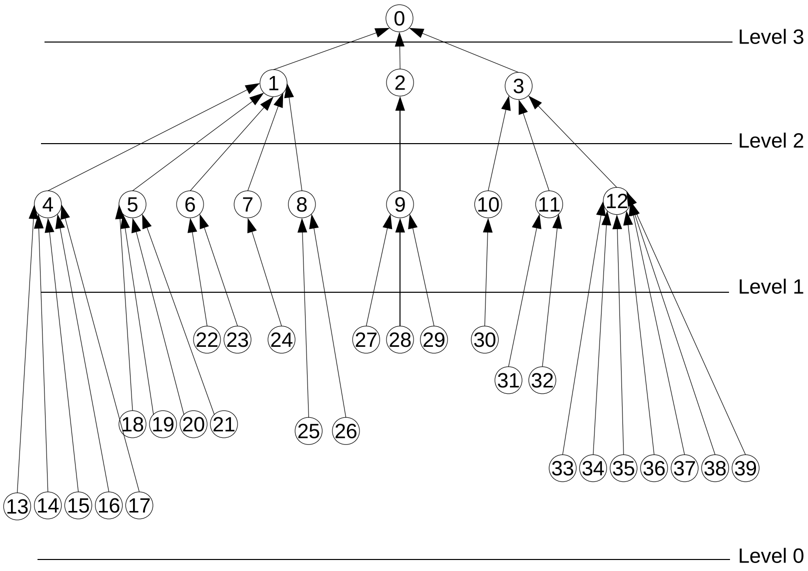

Figure 1 An example of a tree of tasks

maximal value $Donor_{max}(t_{dl}, t)$ is chosen, indicating the best donor candidate i_{best}. Next, a client starts or continues negotiating with the GRA with respect to the execution period $\hat{\tau}^{exe}_{k,r_k}(t)$ at time t in round r_k, which is counted separately for each interrupted task k. For example, the largest number of negotiation rounds for a single negotiation can be set to 100 rounds and every time it fails, it starts again from round 0.

In Algorithm 4, if a client and the GRA cannot reach an agreement with respect to the new resources and the condition presented in Formula (6) is satisfied for the interrupted task k, then resource re-allocation among client tasks can be considered as described in lines 25–32. However, the resource re-allocation among tasks also should be agreed with the GRA, and this negotiation may also fail. Negotiations are repeated until resources are allocated or t^{end}_{tot} is reached, which is a hypothetical long-term deadline for all client tasks. A client negotiates the remaining execution period $\tau^{rem}_{i_{best},l}(t)$ of a chosen donor-task i_{best} with the GRA for each interrupted task k at time t, and the first one which reaches an agreement with the GRA obtains the agreed portion of this remaining execution time. If the donor-task has lost its resources due to the resource re-allocation, then a new donor-task is chosen. At the same time, all remaining interrupted tasks continue to negotiate with the GRA with regard to available Grid resources every next time unit. Consequently, each interrupted task may obtain an execution period as a result of regular negotiation with the GRA or an agreed resource donation from another client's task at time t. This algorithm can be applied to facilitate a faster resource allocation when an ordinary negotiation (using any negotiation strategy) with the GRA keeps failing.

4.4 Illustrative example

This example (see Algorithm 4) shows how a donor-task is chosen by a client using the SimTask re-allocation strategy. Assume that we have 40 tasks, which are connected as presented in Figure 1, where the root task has index 0 and the lowest layer tasks belong to the layer zero. The lower-layer tasks send data to the upper-layer tasks as we discussed above. Then, assume that task 31 changes its status of execution to 'interrupted', then tasks 11, 3 and 0 change their statuses to 'inaccurate', while all other tasks have the status 'accurate'. Any task with resources can potentially be chosen as a donor-task for task 31.

First, the re-allocation algorithm assesses the remaining execution time for all tasks, where the maximum available remaining time is 338 290.0 (virtual seconds)[3] as shown in Table 1 and the minimum acceptable remaining time is 129 600.0 (virtual seconds). Here, the preference is given to the remaining

[3] These values were taken from our computer simulation.

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

Table 1 This table shows the remaining execution period for tasks

Task	Remaining execution time (virtual seconds)
0	204 942.0
1	312 833.0
2	214 528.0
...	...
6	220 063.0
7	73 042.7
...	...
32	338 290.0
33	139 458.0
34	242 676.0
...	...
39	165 681.0

execution periods, which are closer to the average remaining execution period between the maximum available and the minimum acceptable one. The remaining execution periods for some tasks are presented in Table 1. Second, the re-allocation algorithm assesses which donor candidate's interruption will have the least negative effect on the tree of tasks. Here, the preference is given to the lowest layer tasks, the tasks with the status 'stopped' and the tasks with the longer possible delay times (or remaining delay times) for their corresponding recipient-tasks in the case if they were chosen to be a donor. In our example, the shortest delay time would be for task 2 if task 9 was chosen to be a donor and it equals to 67.3064 (virtual seconds). The longest delay time would be for task 12 if task 34 was chosen to be a donor and it equals to 730 720.0 (virtual seconds). The task 34 also belongs to the lowest layer of the tree, and its remaining execution period is close to the average remaining period as listed above.

In this example, the client prioritizes a donor candidate's impact on the tree of tasks, if it is chosen as a donor, over its remaining execution time, that is, $W_1 = 0.3$ and $W_2 = 0.7$ in Formula (12). Therefore, the task 34 is chosen as a donor.

5 Evaluation

We test our SimTask strategy in terms of the utility gain versus when it is not used within different Grid environments and with the different criteria weights which inform a choice of donor. *Reliability* of a Grid environment is modelled by different probabilities of unexpected task interruption, simulating resource failure or withdrawal. *Predictability* of a Grid environment is simulated by varying the precision with which a client is able to estimate the peaks of resource availability. The more precisely a client identifies peaks of resource availability, the better negotiation conditions are expected at the end of task execution period. Here, a task tree consists of 40 tasks, distributed across four layers, where each task has three senders (if applicable), respectively. The values of $\alpha_{i,j}$ are generated randomly for each test, where all tasks have to run together continuously for $\tau^{tot} = 300\,000$ virtual seconds. The period of change in resource availability is 3000 virtual seconds. The client utility is then averaged over 200 runs. Note that k_i^{dam} (see Equation (6)) and λ (see Equation (10)) are set to 0.5 and 0.6 for all tasks.

Negotiation is simulated abstractly in this evaluation to test a stand-alone contribution of the SimTask strategy. The probability of a task to obtain a longer execution period and succeed in negotiation is simulated, following a periodicity of resource availability, where larger availability increases this probability. We also assume that the GRA is more generous in the re-allocation negotiation than in the ordinary one, as it re-allocates resources which were previously granted to a client. As a result, the re-allocation negotiation has a higher probability of succeeding. In this part of evaluation (not applicable to Section 5.3), the negotiation agreements are modelled based on the above assumptions, when potentially any

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

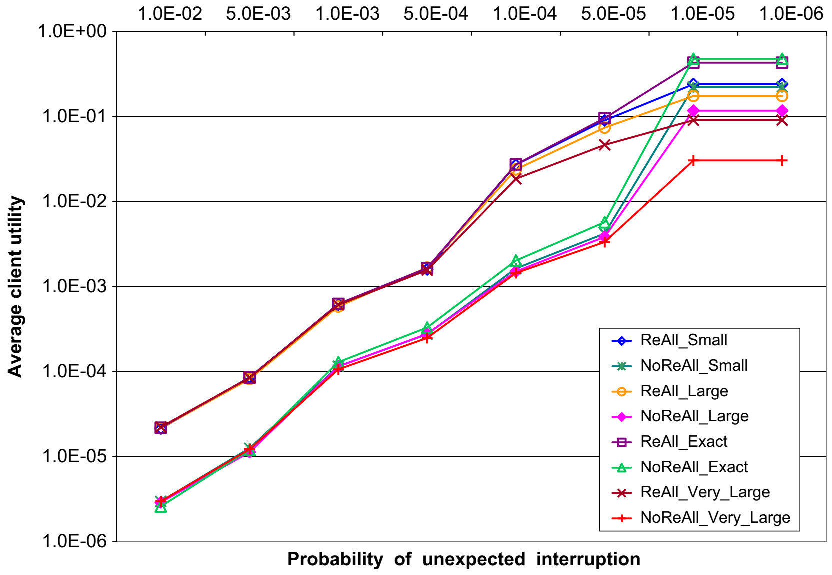

Figure 2 The changes in the client utility in the different Grid environments (Haberland *et al.*, 2017*b*)

negotiation strategy can be used by a client. For transparency, an estimated parameter's, $P_{i,S_i}(t)$, change over time is modelled as a periodic function (can be modelled with other functions but should follow the parameter's definition in Section 3). It is modelled differently in Section 5.3, following a realistic temperature monitoring scenario.

5.1 Grid environments

Here, we evaluate the change in the client utility within different Grid environments as discussed above, which might be less or more favourable for negotiation. The probabilities of unexpected task interruption are considered between 1.E-02 and 1.E-06, where probabilities larger than 1.E-02 are assumed to be highly unrealistic (all tasks would fail almost every simulated second) and therefore not accounted for. Figure 2 shows results in favour of this assumption where the client utility changes less substantially above the probability 5.E-04 and generally tends to zero. The difference in utilities for very small probabilities may not be significant too if the Grid environment is more predictable.

The varying precision with which a client can estimate the peaks of resource availability means that a task might stop running farther from a peak, which will result in more challenging negotiation and a likely shorter execution period for a client. Here, a client is more likely to use the SimTask strategy in order to re-start interrupted tasks. We consider four different levels of estimation precision, where a precise one is indicated as 'Exact', while an inaccurate estimation with a small deviation is denoted by 'Small', with a large deviation by 'Large', and with an extremely large deviation by 'Very_Large' in Figure 2. A small deviation (positive or negative) from the peak of resource availability is assumed to be up to 1% of the period duration (one virtual day) of the change in resource availability. Large and very large deviations are up to 2% and 4% of this duration, respectively. Finally, we compare the cases where a client uses the SimTask strategy, that is, 'ReAll', or does not use it, that is, 'NoReAll'. In the cases 'ReAll', the weights which inform the choice of a donor are $W_{rem} = 0.3$ and $W_{dep} = 0.7$ (see Equation (12)), and these weights are the most successful combination among all considered choices as discussed below.

Figure 2 shows the average utilities for a client with regard to the different probabilities of unexpected task interruption in a logarithmic scale. The SimTask strategy evidently improves the client utility in almost all tested cases, except for the two smallest probabilities and only when a client can precisely estimate the peaks of resource availability. In these cases, a Grid environment is the most favourable for negotiation as it is most reliable and predictable, where a resource re-allocation is not warranted. However, using the SimTask re-allocation strategy in the cases of small, large or very large estimation

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

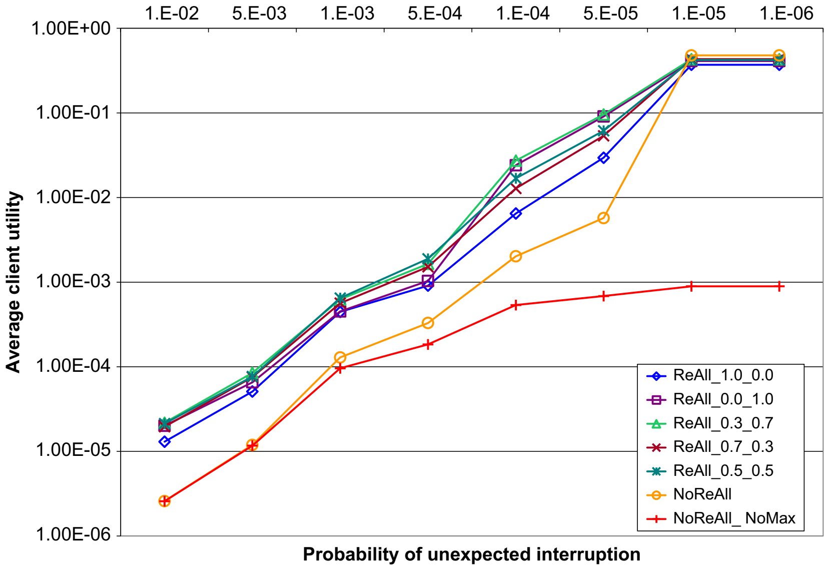

Figure 3 The changes in the client utility with the different preferences over criteria (Haberland *et al.*, 2017*b*)

deviations leads to noticeable utility improvements (especially, for the larger deviations) as compared to the cases when it is not used.

5.2 Client priorities

We evaluate how the choices of weights, which set the priorities for a client to choose a donor, might affect the utility within different Grid environments. Here, we consider the different values of W_{rem} and W_{dep} for the SimTask strategy, that is, 'ReAll', compared to the cases when it is not applied, that is, 'NoReAll' and 'NoReAll_NoMax'. The case 'NoReAll_NoMax' also assumes that a client is unable to estimate the peaks of resource availability, while other cases consider that a client can estimate them with the high precision. Figure 3 shows the average utilities for the different pairs of weights in less or more reliable Grid environments, where W_{rem} and W_{dep} are indicated on the labels 'ReAll'; for example, 'ReAll_1.0_0.0' denotes $W_{rem} = 1.0$ and $W_{dep} = 0.0$.

In general, the utilities appear to be larger for $W_{rem} = 0.3$ and $W_{dep} = 0.7$ than for all other pairs of weights. A strict prioritization of the remaining execution period of a potential donor as compared to its possible impact on the other tasks or vice verse, that is, 'ReAll_1.0_0.0' or 'ReAll_0.0_1.0', generally shows the smallest utilities among all weight pairs. However, the case 'ReAll_0.0_1.0' shows larger utilities for the smaller probabilities, that is, below 5.E-04, leading to conclude that a slight prioritization of the criterion $Dep_j^i\,(t_{dl}, t)$ over the criterion $Rem_{j,l}\,(t)$ would be beneficial for a client, that is, 'ReAll_0.3_0.7'. The difference in utilities appear to be not large for the cases where the weights are better balanced such as 'ReAll_0.3_0.7', 'ReAll_0.7_0.3' and 'ReAll_0.5_0.5', while 'ReAll_0.3_0.7' mostly demonstrates better utilities. However, the most appropriate choice of a pair of weights depends on a specific use case. Finally, the SimTask re-allocation strategy with any weight pair outperforms almost all cases 'NoReAll_'.

5.3 Case study

In this section, we also test our SimTask re-allocation strategy[4], but using the Grid resource simulator (Haberland *et al.*, 2017*a*), which models the fluctuations of resource availability based on the dependencies observed in the Grid-related studies, mostly in Iosup *et al.* (2008). The simulator is based on the system of non-stationary (time-dependent) differential balance equations which calculates the amount of

[4] This is a new experiment as compared to our evaluation in (Haberland *et al.*, 2017*b*)

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

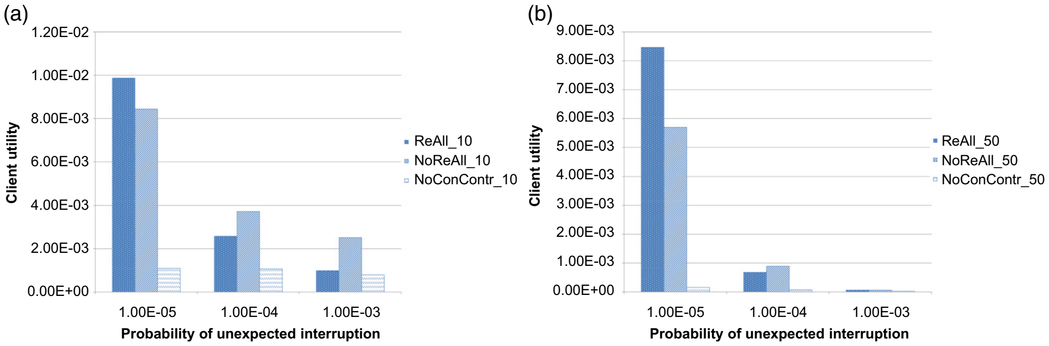

Figure 4 Client utilities with the different GRA's behaviour. (a) Client utilities with the fairly predictable GRA. (b) Client utilities with the highly unpredictable GRA

free, busy and demanded resources at any time point. The GRA's negotiation behaviour is then modelled based on the amount of free and demanded resources. For example, the resources might be available but highly contested such as the demand is higher than supply, which would lead to the more greedy GRA's behaviour. This simulator also accounts for the scenario when the total amount of resources in a Grid changes such as when some machines fail.

The other difference of this experiment as compared to the previous sections is that we use actual negotiation strategies rather than model the probability of agreement. That is, this case study intends to test a usefulness of the SimTask re-allocation strategy when the negotiation strategy suited to continuous tasks is used, ConTask (Haberland *et al.*, 2017*a*), while our previous evaluation was intended to measure a stand-alone contribution of the SimTask strategy regardless of the negotiation strategy used. The ConTask strategy was designed for the continuous long-term tasks, but did not account for the dependencies. As a result, we tested the cases where our resource re-allocation strategy is used 'ReAll_' or not used 'NoReAll_' and 'NoConContr_'. In the case 'ReAll_', we also employ the ConTask strategy for negotiation with the GRA, while in the cases 'NoReAll_' and 'NoConContr_', we only have an option to negotiate with the GRA in order to obtain new resources, employing the ConTask strategy in 'NoReAll_' and more general (not continuous task oriented) strategy in 'NoConContr_' (Haberland *et al.*, 2015). Our baseline in all tests is represented by 'NoConContr_'.

As we use the negotiation strategies in this evaluation, we also explicitly model the GRA's negotiation behaviour which can be more or less predictable by itself. As much as the GRA's proposals depend on the resource availability and demand, they also depend on the GRA's allocation strategy, which we simulate by allowing smaller or larger deviations of its negotiation parameters over time. The smaller deviations (up to 10% denoted as '..._10' in Figure 4) simulate the more consistent and moderate GRA's behaviour, while the larger deviations (up to 50% denoted as '..._50' in Figure 4) simulate the large level of unpredictability which might be advantageous for the client occasionally when the GRA becomes substantially generous, but more likely disadvantageous as the negotiation strategies will struggle to predict the GRA's behaviour.

We compare the obtained utilities for the different probabilities of the unexpected task interruption as in the previous sections. The larger number of the unexpected interruptions will lead to more tasks requiring resources at the same time. If the interruptions occur more frequently, they are also generally longer, because of the increasingly scarce resources. The probabilities for this experiment were chosen based on our simulation results in Sections 5.1 and 5.2 as very small probabilities have shown to be non-descriptive for our strategy, while some large probabilities are less realistic. Therefore, we consider three different probabilities of the unexpected task interruption, which are 1.E-05, 1.E-04 and 1.E-03. We have also chosen the best configuration for our SimTask re-allocation strategy based on the evaluation in the previous sections, that is, $W_{rem} = 0.3$ and $W_{dep} = 0.7$. In this experiment, we also model the change in the estimated parameter $P_{i,S_i}(t)$ (see Section 3), following the realistic scenario of temperature fluctuation, explained in the following section.

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

5.3.1 Temperature modelling

Here, we consider a tree of 40 tasks where each recipient task has an arbitrary number of the sender-tasks as compared to the previous experiment, and those tasks are assumed to linearly combine the levels of temperature from the lower-layer tasks. The lowest-layer tasks are considered to monitor the level of temperature which is modelled as discussed in this section. We also consider that each lowest-layer task is responsible for a particular room, which is then reflected in our model below.

Here, we assume that the level of temperature changes pseudo-periodically over time as demonstrated in the work of Lacroix *et al.* (2012, p. 7). Our assumptions were largely based on this work. In our simulation, the objective is to maintain the preferable temperature level T_{pref} in all rooms in a building. We model the changes in the temperature which are caused by the external factors (e.g. the impact of temperature from the outside of the building) as well as the internal heating.

The external factors may cause the non-stationary fluctuations of the temperature. In this case, the deviated temperature from the preferable one T_{pref} is modelled as the function $T_{ext_i}(t)$ over time t for task i. A deviation (decrease or increase) is modelled with the two complementary functions $f_{dec_i}(t)$ and $f_{inc_i}(t)$. The temperature can increase above the preferable temperature level; that is, it may decrease or increase up to the reasonable border temperature $T_{bor_i}(t)$, where $T_{bor_i}(t)$ can be larger or smaller than T_{pref}. The functions $f_{dec_i}(t)$ and $f_{inc_i}(t)$ can be generically presented as

$$f_{x_i}(t) = \frac{T_{pref} - T_{bor_i}(t)}{e^{(\pm t \mp t_{x_i}(t))/\tau} + 1} + T_{bor_i}(t) \tag{13}$$

where x_i denotes either decrease dec_i or increase inc_i, affecting consequently the signs in the denominator, and

$$T_{ext_i}(t) = 0.5 \times (T_{pref} + \frac{1}{T_{bor_i}(t)} \times f_{dec_i}(t) \times f_{inc_i}(t)) \tag{14}$$

We also model slight periodic temperature oscillations which depict the internal heating process around T_{pref} or $T_{ext_i}(t)$. The amplitude of these oscillations $T_{amp_i}(t)$ for each task i is calculated as a sum of, first, a task-independent temperature amplitude component T_A (e.g. $T_A = 2\,°\mathrm{C}$), second, a task-dependent i temperature amplitude component $T_{A_i} \in [-T_A, +T_A]$ and third, a time t and task-dependent i temperature amplitude component $T_{A_{i,t}} \in [-T_A/20, +T_A/20]$, and this sum is multiplied by the value of function $f_{A_i}(t)$, which can attenuate this amplitude in the case of significant non-stationary temperature changes. Here, the attenuation function $f_{A_i}(t)$ suppresses the temperature level oscillations, when the heater is not working or unable to significantly affect temperature in a room. We assume that the rooms in the building can be of the different sizes, thermal isolation, etc., and, therefore, they should have distinctive temperature deviations T_{A_i}, while those deviations $T_{A_{i,t}}$ should not normally be significant for a particular room over time. Consequently, the final amplitude of those small temperature oscillations is calculated as in the following equation:

$$T_{amp_i}(t) = 0.5 \times (T_A + T_{A_i} + T_{A_{i,t}}) \times f_{A_i}(t) \tag{15}$$

where the attenuation function is presented as

$$f_{A_i}(t) = e^{-k_A \times |1.0 - ((2 \times T_{ext_i}(t) - T_{pref})/T_{pref})|} \tag{16}$$

which is equal to one when there is no noticeable non-stationary effects, that is, $T_{ext_i}(t) = T_{pref}$, while k_A is an empirically chosen coefficient which is equal to 40 in our model. We then model the pseudo-periodicity in the temperature oscillations as a sine-type function with the period T^{tem}_{per} derived from the work Lacroix *et al.* (2012). Finally, the temperature level $T_i\,(t)$ estimated by the lowest-layer task $Layer_i = 0$ at time t is calculated as in the following equation:

$$T_i(t) = T_{ext_i}(t) - T_{amp_i}(t) \times \sin(2\pi(t - Ph_i)/T^{tem}_{per}) \tag{17}$$

5.3.2 *Results discussion*

Figure 4 shows the average client utilities obtained using our strategies for the interdependent tasks, where both the Grid resource availability and the estimated parameter were realistically modelled utilizing dependencies presented in other studies. Note that our ConTask strategy was tuned to use the most successful settings as described in Haberland *et al.* (2017*a*), meaning that the client would exhibit a moderately generous behaviour in negotiation. In this way, we tested the extent of applicability of our SimTask re-allocation strategy.

This evaluation to some extent confirms our previous conclusions that the SimTask re-allocation strategy is more useful in the cases, where the GRA is less predictable such as depicted in Figure 4(b) as compared to Figure 4(a). Therefore, if it is harder for a client to obtain resources through an ordinary negotiation with the GRA, and then the resource re-allocation would serve well as an alternative solution.

However, we also observed another pattern, which was not noticeable in our previous evaluation (as we did not use actual negotiation strategies), that our ConTask strategy by itself might be more helpful in the cases of the long resource scarcity periods (i.e. the larger probabilities of the unexpected task interruption). The more frequent interruptions would increase the resource scarcity as more tasks will be unallocated and their number will keep increasing, leading to the longer periods of resource scarcity in the Grid. The SimTask re-allocation strategy would be beneficial if a donor-task is able to obtain a resource from the GRA by itself as the subsequent re-allocations during the long period of resource scarcity might decrease the client utility. Nevertheless, it needs to be mentioned that 'ReAll_' is still better in the realistic cases than the ordinary negotiation 'NoConContr_' with the strategy which does not account for the near-continuity of the tasks, which was also observed previously as the case when our SimTask strategy was not used. It would also be worth to mention that the larger probabilities of the unexpected interruptions are rather less possible scenarios in a Grid.

6 Conclusions

Here, we discussed the formal model which depicts data interdependencies between near-continuous tasks, and how these dependencies might affect their execution and a whole tree of tasks if a data instance is not received in time. Not only we consider a one-way dependence between the data sender and recipient, but also the opposite dependence which means that the sender-task cannot be run if its recipient has been interrupted. This reflects the nature of the tasks which have to be run near-continuously in real time.

As a solution to this problem, we presented a resource re-allocation strategy, SimTask, which offers the decision-making mechanism for a client to re-allocate resources between its own tasks. This re-allocation still has to be negotiated with the Grid and can only be performed by the GRA. The SimTask strategy considers two criteria for a client to choose the donor-task, which take into account the remaining execution period and an impact of the donor-task's interruption on the rest of the tree. The detailed description of all algorithms which shade the light on how this whole strategy works in the case of multiple interruptions elsewhere in the tree, and an illustrative example were also presented. We had thoroughly evaluated this strategy under two different settings, one of which assumes a more abstract but extensive modelling of the Grid environment and tuning of the strategy, while the other applies actual negotiation strategies for resource allocation and uses a realistic Grid resource simulation as well as environmental modelling (i.e. the level of temperature). Both settings have shown the benefits of using the SimTask re-allocation strategy as well as its possible limitations.

Acknowledgements

Many thanks to King's College London for awarding Graduate School Studentship and King's Overseas Research Studentship to Valeriia Haberland in order to accomplish her PhD (Haberland, 2015).

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

References

Andrzejak, A. & Ceyran, M. 2005. Characterizing and predicting resource demand by periodicity mining. *Network and Systems Management* **13**(2), 175–196.

Apache. 2014. *Storm - distributed and fault-tolerant realtime computation*. http://storm.incubator.apache.org/

Babu, S. & Widom, J. 2001. Continuous queries over data streams. *SIGMOD Record* **30**(3), 109–120.

Barbieri, D. F., Braga, D., Ceri, S., Della Valle, E. & Grossniklaus, M. 2009. C-SPARQL: SPARQL for continuous querying. In *The 18th International Conference on World Wide Web*. ACM, 1061–1062.

Decker, K. S. & Lesser, V. R. 1992. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems* **1**, 319–346.

EsperTech. 2014. *Event Series Intelligence: Esper & NEsper*. http://esper.codehaus.org/

Foster, I., Kesselman, C. & Tuecke, S. 2001. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, **15**, 200–222. http://portal.acm.org/citation.cfm?id=1080644.1080667

Ghanem, M., Guo, Y., Hassard, J., Osmond, M. & Richards, M. 2004. Sensor Grids for Air Pollution Monitoring. In *The 3rd UK e-Science All Hands Meeting*.

Haberland, V. 2015. *Strategies for the Execution of Long-Term Continuous and Simultaneous Tasks in Grids*. PhD thesis NMS. King's College London.

Haberland, V., Miles, S. and Luck, M. 2014. Negotiation to Execute Continuous Long-Term Tasks. In *The 21st European Conference on Artificial Intelligence*, T. Schaub et al. (eds.) Vol. 263. *Frontiers in Artificial Intelligence and Applications*, 1019–1020.

Haberland, V., Miles, S. & Luck, M. 2015. Adjustable fuzzy inference for adaptive grid resource negotiation. In *Next Frontier in Agent-based Complex Automated Negotiation*. Vol. 596. *Studies of Computational Intelligence*. Springer, 37–57.

Haberland, V., Miles, S. & Luck, M. 2017*a*. Negotiation strategy for continuous long-term tasks in a grid environment. *Autonomous Agents and Multi-Agent Systems* **31**(1), 130–150.

Haberland, V., Miles, S. & Luck, M. 2017*b*. *Resource Re-allocation for Data Inter-dependent Continuous Tasks in Grids*. In Vol. 10207. *LNCS*. Springer International Publishing, 187–201.

Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L. & Epema, D. H. J. 2008. The grid workloads archive. *Future Generation Computer System* **24**(7), 672–686.

Jin, H., He, Y., Wen, W. & Liu, H. 2005. A run-time scheduling policy for dependent tasks in grid computing systems. In *The 6th International Conference on Parallel and Distributed Computing, Applications and Technologies*, 521–523.

Kondo, D., Taufer, M., Brooks, C., Casanova, H. & Chien, A. 2004. Characterizing and evaluating desktop grids: An empirical study. In *The 18th International Parallel and Distributed Processing Symposium*.

Lacroix, B., Paulus, C. & Mercier, D. 2012. Multi-agent control of thermal systems in buildings. In *Proceedings of the 3rd International workshop on Agent Technologies in Energy Systems*.

Le-Phuoc, D., Nguyen-Mau, H. Q., Parreira, J. X. & Hauswirth, M. 2012. A middleware framework for scalable management of linked streams. *Web Semantics: Science, Services and Agents on the World Wide Web* **16**(0), 42–51.

Lee, L.-T., Chen, C.-W., Chang, H.-Y., Tang, C.-C. & Pan, K.-C. 2009. A non-critical path earliest-finish algorithm for inter-dependent tasks in heterogeneous computing environments. In *The 11th IEEE International High Performance Computing and Communications*, 603–608.

Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., Nagendra Prasad, M., Raja, A., Vincent, R., Xuan, P. and Zhang, X. Q. 2004. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems* **9**(1–2), 87–143.

Lesser, V. R. 1991. A retrospective view of FA/C distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics* **21**(6), 1347–1362.

Lim, H. and Babu, S. 2013. Execution and Optimization of Continuous Queries with Cyclops. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 1069–1072.

Llanes, K. R., Casanova, M. A. & Lemus, N. M. 2016. From Sensor Data Streams to Linked Streaming Data: a survey of main approaches. *Journal of Information and Data Management* **7**(2), 130–140.

Meriem, M. and Belabbas, Y. 2010. Dynamic dependent tasks assignment for grid computing. In *Algorithms and Architectures for Parallel Processing*, C.-H. Hsu et al. (eds.) Vol. 6082. *LNCS*. Springer. 112–120.

Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J. & Varma, R. 2003. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In *The 1st Biennial Conference on Innovative Data Systems Research*, 245–256.

Sandnes, F. E. and Sinnen, O. 2005. Stochastic DFS for Multiprocessor Scheduling of Cyclic Taskgraphs. In *Parallel and Distributed Computing: Applications and Technologies*, K.-M. Liew et al. (eds.) Vol. 3320. *LNCS*. Springer, 354–362.

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press

Sardinha, A., Alves, T. A. O., Marzulo, L. A. J., Franca, F. M. G., Barbosa, V. C. & Costa, V. S. 2012. Scheduling cyclic task graphs with SCC-Map. In *The 3rd Workshop on Applications for Multi-Core Architectures*, 54–59.

Sequeda, J. F. and Corcho, O. 2009. Linked stream data: A position paper. In *The 2nd International Workshop on Semantic Sensor Networks*. Vol. 522, 148–157.

Terry, D., Goldberg, D., Nichols, D. & Oki, B. 1992. Continuous queries over append-only databases. *SIGMOD Rec.* **21**(2), 321–330.

TIBCO. 2014. http://www.streambase.com/

Wooldridge, M. and Jennings, N. R. 1995. Intelligent agents: Theory and practice. *The Knowledge Engineering Review* **10**, 115–152.

Yan, K. Q., S. C. Wang, C. P. Chang & J. S. Lin. 2007. A hybrid load balancing policy underlying grid computing environment. *Computer Standards & Interfaces* **29**(2), 161–173.

Yang, T. and Fu, C. 1997. Heuristic algorithms for scheduling iterative task computations on distributed memory machines. *IEEE Transactions on Parallel and Distributed Systems* **8**(6), 608–622.

Zhao, H. & Sakellariou, R. 2004. A low-cost rescheduling policy for dependent tasks on grid computing systems. In *The European Across Grids Conference*, 21–31.

https://doi.org/10.1017/S0269888919000067 Published online by Cambridge University Press