Automating Data-Throttling Analysis for Data-Intensive Workflows

Ricardo J. Rodríguez, Rafael Tolosana-Calasanz
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, María de Luna 1, 50018 - Zaragoza, Spain
Email: {rjrodriguez,rafaelt}@unizar.es

Omer F. Rana
School of Computer Science & Informatics
Cardiff University - Cardiff, United Kingdom
Email: o.f.rana@cs.cardiff.ac.uk

***Abstract*—Data movement between tasks in scientific workflows has received limited attention compared to task execution. Often the staging of data between tasks is either assumed or the time delay in data transfer is considered to be negligible (compared to task execution). Where data consists of files, such file transfers are accomplished as fast as the network links allow, and once transferred, the files are buffered/stored at their destination. Where a task requires multiple files to execute (from different tasks), it must, however, remain idle until all files are available. Hence, network bandwidth and buffer/storage within a workflow are often not used effectively. We propose an automated workflow structural analysis method for Directed Acyclic Graphs (DAGs) which utilises information from previous workflow executions. The method obtains data-throttling values for the data transfer to enable network bandwidth and buffer/storage capacity to be managed more efficiently. We convert a DAG representation into a Petri net model and analyse the resulting graph using an iterative method to compute data-throttling values. Our approach is demonstrated using the Montage workflow.**

***Index Terms*—Performance analysis, System performance, Petri nets, Scientific computing**

I. Introduction

Using workflow techniques, scientists can specify their computational experiments by means of a control/data flow graph, consisting of a set of tasks and the dependencies between them. Workflow enactors can subsequently interpret these specifications, enabling tasks in the graph to be mapped onto distributed resources. There are, however, several efficiency limitations of the workflow system in performance and resource usage [16]. Such limitations may be due to limited parallelism within the application, or due to the workflow enactment engine. In data-intensive workflows, the enactors must be efficient in both mapping tasks to resources and in transferring large data files between tasks. Previous approaches exploit data location and link bandwidth information to minimise data movement or move data via higher capacity links whenever possible. Such an approach of moving the largest files via the highest-capacity links can result in sub-optimal workflow execution [16].

This data movement policy generally involves moving the output data of a task to its successor node immediately after completing its execution. However, if a task needs multiple files to be made available before it can begin execution, it will remain idle until all the required data files from other predecessor nodes have been delivered. It is therefore not how fast each file can be moved to the task, but the interval from the delivery of the first file to the last one that is most significant. Even if the first file is delivered quickly, the task must still remain idle until others are also available. In consequence, the effective use of network bandwidth and the buffer/storage at the receiving task is not made. If one file arrives too early taking up all of the network bandwidth for one task, it may be at a determent to other tasks (which may have to wait for their data to be delivered) – even though the receiving task still has to wait for other files. Similarly, if there is limited buffer capacity at the receiving task and the buffer needs to be shared between tasks, a quick delivery of one file (while waiting for other files to be delivered) for a task excludes other tasks from using the same buffer. In general therefore, the current practise of moving data from one location to another as early as possible is often either: (i) unnecessary when viewed in isolation (both in terms of networking and buffering): any data file that arrives much before the last needed data file or ii) harmful when viewed in-the-large: there is only finite capacity on each link and a limited buffer capacity – multiple concurrent data movement operations can significantly slow each other down, or an inappropriate buffer usage may lead to a buffer overflow. Park & Humphrey analysed this problem in [16] and proposed a data-throttling framework that allows a workflow programmer/engine to describe the requirements on the data movement delay. This technique can be utilised to balance the execution time of workflow branches and eliminate unnecessary bandwidth usage, resulting in more efficient buffer and network usage. However, they do not propose any mechanism for analysing and automatically deriving the values needed to throttle data exchange between nodes involved in the transfer.

We propose an automated analysis method for Directed Acyclic Graphs (DAGs) that may be used to derive data-throttling values. The method utilises Petri nets for the workflow specification and combines the abstract representation with performance information instrumented from past executions, obtaining a performance model of the workflow: an iterative method is used to compute data-throttling values associated with different links within the workflow. Subsequently, a performance analysis is conducted using the throttling values and the result is compared with the performance achieved without throttling. We analyse the impact of data throttling on performance and consider the speedup or degradation caused by this approach. As discussed in [16], data throttling has an impact on the overall workflow performance depending on the ratio between computational and transmission tasks. We therefore believe that a performance analysis is mandatory after the regulation of data transfer so that the trade-off (between performance and storage) can be identified more explicitly. The remainder of this paper is organised as follows. In Section II, we provide a background overview on Petri nets, and the Petri net-based analysis technique we are based on. Section III includes a review of related work. In Section IV, we describe our Petri net-based automated analysis method. In Section V, we conduct a number of experiments with the Montage workflow (an I/O intensive workflow used to create image mosaics within the astrophysics domain) – which has often been used in a number of other studies and forms a useful basis for comparison. Section VI provides conclusions and future work.

II. Background

In this section, we introduce the Petri net formalism used to represent our workflow models and the concept of *slack* – a key concept used here (described in section II-A). Although our analysis has been carried out using the Petri net formalism, we are also able to take scientific workflows represented using the DAX (Directed Acyclic graph in XML) format (used by the Pegasus and also Triana workflow systems) and automatically convert them into a Petri net representation (using the Petri Net Markup Language (PNML) representation).

A *Petri net* [13] (PN) is a mathematical formalism used for the modelling of concurrent and distributed systems. A PN is a bipartite graph, which consists of two set of nodes, places P, transitions T and directed arcs. An arc connects a place with a transition, or vice versa, but it never connects a place with a place or a transition with a transition. A place p may have initially a natural number of tokens, called *initial marking* of place p, denoted as $\mathbf{m_0}(p)$. A distribution of tokens over the places is then called a *marking* $\mathbf{m}$. PNs have a graphical notation where places are represented by circles, transitions by bars, tokens by black dots and directed arcs by arrows.

A transition of a PN is said to be *enabled* when all its input places have *tokens*. Once enabled, it can be *fired* and when fired all these tokens are consumed and placed in the output places. The firing of transitions of a PN is non-deterministic, i.e. when multiple transitions are enabled at the same time any of them can be fired.

The PN formalism can be extended by associating transitions with a delay. The delay is characterised by a random variable and represents the elapsed time for firing the transition. Such an extension is called *Stochastic Petri nets* (SPNs) [2]. *Stochastic Marked Graphs* (SMGs) are a subset of SPNs. A SMG is a SPN where each place of the SPN has exactly one input and exactly one output arc.

In this paper, we make use of SMGs with exponential random distributions associated with transitions in order to model scientific workflows. In particular, we are interested in workflows expressed as Directed Acyclic Graphs (DAGs), where vertices represent tasks and edges represent data dependencies between them. Figure 1 depicts how we derive a PN model from a DAG. Figure 1(a) shows two workflow tasks of a DAG, $task_1$ and $task_2$ and a data-link dependence from $task_1$ to $task_2$, while Figure 1(b) illustrates the transformation to a PN. Note that such a PN model fulfils the definition of a SMG model: there are timed transitions and each place has exactly one input and exactly one output arc.

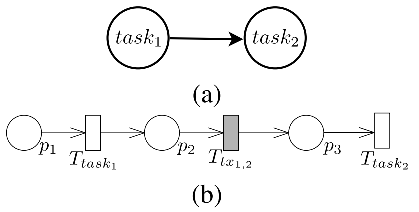


Figure 1. (a) Workflow tasks and (b) its transformation to PN.

Hence, each task of the workflow DAG is transformed to a place and a transition (represented by a white rectangle), joined by an arc. A task transmission is also transformed to a place and a transition (grey rectangle). For instance, $p_1 \rightarrow T_{task_1}$ represents $task_1$ of the workflow. Note that place p_1 models the input buffer of $task_1$. Finally, the data dependency between $task_1$ and $task_2$ is modelled by adding a place and a transition, p_2 and $T_{tx_{1,2}}$, respectively. Transition $T_{tx_{1,2}}$ represents the time spent in sending output data from $task_1$ to the input buffer of $task_2$ (place p_3).

A. Slack Calculation

Our PN model is used to calculate performance bounds, rather than provide an exact analytical performance model. This is due to the computational cost of deriving the reachability graph of a PN, needed for an exact performance analysis (an NP-hard problem requiring exponential space requirement [10]). We make use of the algorithm presented in [18] to compute upper performance bounds using an iterative strategy. Such a strategy needs, as an input, the SMG to be analysed and gives as an output an upper bound closer (i.e, more accurate) to real performance than other upper bound computational strategies. The core of the strategy is based on the concept of *slack*. Let us consider a transition t of a PN where several places $p_1, p_2 \dots p_n$ synchronise, i.e., all input places of transition t need to be marked (to have tokens) in order for t to be enabled. A slack at place p, represented as $\mu(p)$, identifies the difference in time between an input token being available and its subsequent use (i.e., when token is consumed). Hence, different values of slack measure the time required to store input data before it is consumed by a subsequent process.

As discussed in [18], the strategy makes use of linear programming techniques (LPP) for which polynomial complexity algorithms exist and it converges in a few iterations steps.

Slack values are obtained by computing a special distribution of tokens, called *tight marking*, $\tilde{\mathbf{m}}$, which can be achieved by solving the following LPP [5]:

$$\begin{aligned} & Maximize\ \Sigma\sigma : \\ & \qquad \delta(p^\bullet) \cdot \Theta \leq \tilde{\mathbf{m}}(p) \quad \text{for every } p \in P \\ & \qquad \tilde{\mathbf{m}} = \mathbf{m_0} + \mathbf{C} \cdot \sigma \\ & \qquad \sigma(t_p) = k \end{aligned} \tag{1}$$

where t_p is a transition that belongs to a critical cycle (given by the upper bound Θ of the system), $\mathbf{C}$ is the Petri net incidence matrix, σ is the vector of firing counts of transitions, $\delta(p^\bullet)$ is the average service time of output transition t of place p, i.e. $t = \{p^\bullet\}$, and k is any real constant number.

Once a tight marking $\tilde{\mathbf{m}}$ for an SMG is computed, slack values can be computed as follows:

$$\tilde{\mathbf{m}}(p) \geq \delta(p^\bullet) \cdot \Theta \rightarrow \mathbf{m}(p) = \delta(p^\bullet) \cdot \Theta + \mu(p)$$

where $\mu(p)$ is the *slack* of place p, $\forall p \in P$.

III. Related Work

Ordinary Petri nets and their extensions have been widely used for the specification, analysis and implementation of workflows [22]. In the scientific workflow community, they have also been utilized and GWorkflowDL [17], [23], Grid-Flow [11] and FlowManager [3] are representative examples of this. PNs have also been used for the specification of hierarchical scientific workflows, incorporating of exception handling and check-pointing mechanisms [12], [19], [20]. However, in this paper, we use them for deriving performance models of pure graph-based workflows.

On the other hand, an overview of the most significant systems was carried out by Yu and Buyya in [24], which also classifies existing automated data transfer strategies utilised among tasks in workflows, namely centralised, mediated, and peer-to-peer. A centralised approach utilises a central point for data transmission. This solution is not scalable, and occurs in systems where the time for data transfers is much smaller than computations. Taverna [15] typically utilises a centralised data transfer, due to the characteristics of the problems it tackles. In a mediated strategy the locations of the intermediate data are managed by a distributed data management system. Finally, a peer-to-peer approach transfers data directly between processing nodes. The direct transmissions of peer-to-peer approaches reduce both transmission time and the bottleneck problem caused by the centralised and mediated approaches. Thus, they are suitable for large-scale intermediate data transfer. The technique proposed in our paper is focused on data-intensive workflows whose data movement strategy is peer-to-peer based.

The Pegasus workflow system [7] has also dealt with data-intensive workflows and has incorporated both mediated and peer-to-peer transfers. In the mediated approach, Pegasus utilises a data replica catalogue that stores the intermediate data generated, so data can be subsequently retrieved rather than recomputed again. Workflow performance speedup has also been a matter of study in Pegasus, and abstract workflow specifications go through a process of clustering (grouping of small tasks) and partitioning which helps the meta-scheduler to optimise the execution time. A similar approach was followed in [9] with further optimisation at runtime by adapting to the dynamically changing state of underlying resources. However, none of these approaches makes an effective usage of both network bandwidth and buffer/storage of files.

An analysis of the overhead for scientific workflows in Grid environments was given by Nerieri et al. in [14]. The analysis includes both load imbalance and data movement, with these being identified as the most significant sources of overhead. As discussed in this paper, Park & Humphrey [16] already analysed the problem of load imbalance and data throttling for scientific workflows. They proposed a process envelope based framework for throttling data transfers. Nonetheless, they do not provide any analysis method in order to automatically obtain such data-throttling values. In this paper, we describe a method that can automatically derive (sub-optimal) values for them.

Performance analysis of scientific workflows has also been studied in [8], [21]. The performance method by Duan et al. [8] is based on a hybrid Bayesian-neural network for predicting the execution time of workflow tasks. Bayesian network is used for a high level representation of activities performance probability distribution against different factors affecting the performance. The important attributes are dynamically selected by the Bayesian network and fed into a radial basis function neural network to make further predictions. The performance analysis approach used in the current work is based on [21], a method that provides a parameterised Petri net-based graphical model of the overall workflow structure. This graphical view is of particular importance for studying workflow performance, and the characterisation of tasks proposed in [8] could be utilised as an input for the parameterised model in [21].

Finally, Van der Aalst and Van Hee give in [1] a Petri net structural analysis which has been undertaken for business workflows. They utilised a specific class of Petri nets, WF-nets, tailored towards workflow analysis. WF-nets can model workflows with different kind of control operations such as sequence, choice, synchroniser, fork or merge. The types of structural analysis that can be undertaken includes correctness, deadlock analysis or liveness.

IV. Automating Data-Throttling Analysis

In this section we describe our analysis method, which consists of a sequence of four steps that are illustrated in Figure 2. As an input, the method receives a PN-based DAG workflow model (as described in Section II), and performance information obtained from past executions (if any), or estimations provided by the user (i.e. annotations given in a DAX file). As an output, it generates throttling values for network and buffer usage as well as the impact of these values have on the overall workflow performance.

The first step merges the PN model and the performance information (execution times for computational tasks and transfer times for transmission tasks) and builds an SMG

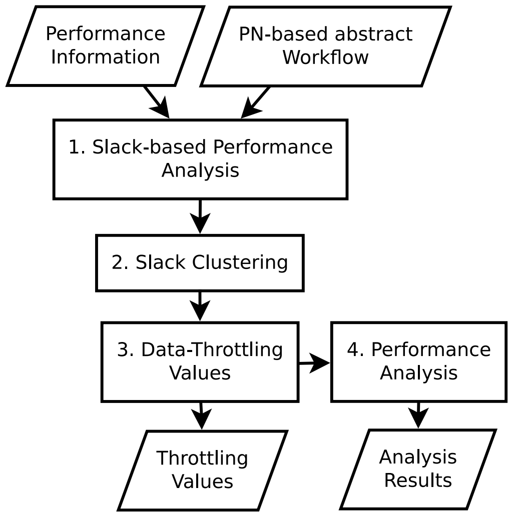


Figure 2. Automated Data-Throttling Analysis Flowchart.

model. Then, it performs the *Slack-based Performance Analysis* step described in Section II. The critical path (i.e. the path with the longest delay) is obtained and slack values derived for each input link of a task (primarily for those tasks which have multiple input links) in the workflow.

As we parse the workflow graph, calculation of a data-throttling value at one data transmission may impact a calculation of a data-throttling at other data transmissions. Hence, if we start at the output node of a graph and work towards the input nodes, the slack value (used for computing data-throttling values) previously calculated along links close to the output node would be influenced by those closer to the input node. For instance, in the example DAG illustrated in 3, calculating data-throttling values for all incoming links at $task_6$ first and then moving on to $task_4$ would imply that if incoming links at $task_4$ would be delayed, then $task_4 \rightarrow task_6$ might be implicitly delayed and the previous adjustment done at incoming links at $task_6$ is no longer valid.

In order to minimise the complexity of this process, our approach classifies the data transfers in a workflow into groups of elements that are mutually independent and not affecting one another. We refer to this as *Slack Clustering* in Figure 2. In the *Data-Throttling Values* step, the method chooses one of the clusters and derives throttling values from the calculated slacks. Afterwards, the throttling values are used in the SMG model and the remaining slacks in the workflow are re-calculated accordingly. The process is repeated until there are no more clusters to be regulated. For the computation of the throttling values, the third step considers a point-to-point network topology between tasks, i.e., all tasks are interconnected through dedicated links. Finally, the *Performance Analysis* step is carried out with (and without) data-throttling values to determine the impact of data throttling on the workflow makespan.

The steps of the method can be better illustrated using the workflow example of Figure 3. The derived Petri net model

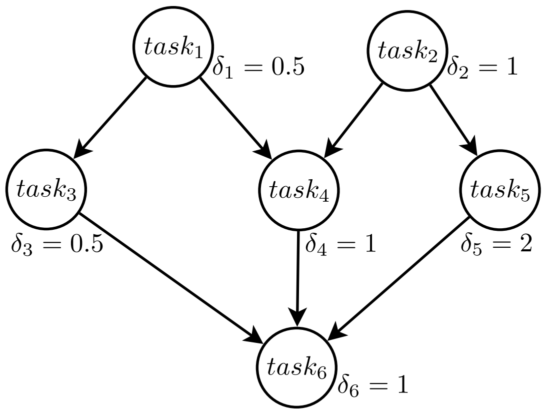


Figure 3. A workflow with 6 task and multiple inter-tasks dependencies.

is shown in Figure 4. As for the performance information, for each computational task in Figure 3, δ_i represents the average execution delay for $task_i$. We assume that each task is executed on a different host (resource), therefore there is no *workflow imbalance*. For data transfers, we assume a dedicated network topology, i.e., each host is inter-connected to the others.The available bandwidth is considered to be 100Mbps with a latency of $1e^{-4}$s and the initial transfer policy is to transmit as fast as possible. For the sake of simplicity in the example, we also assume that all datasets are of equal size, 10MB.

The first step of the method, *Slack-based Performance Analysis*, obtains the workflow's slowest path: $task_2 \rightarrow task_5 \rightarrow task_6$ in Figure 3. Such a workflow has three different inputs where some slack will appear, namely $task_1 \rightarrow task_4$ (because $task_4$ needs data from $task_2$, and $task_2$ takes longer than $task_1$ to execute), $task_3 \rightarrow task_6$, and $task_4 \rightarrow task_6$ (in both latter cases, the slacks happen because of the delay of the input $task_5 \rightarrow task_6$). Using the SMG model in Figure 4, the following slack values are derived: $\mu(p_9) = 0.089392$, $\mu(p_{15}) = 0.178451$ and $\mu(p_{16}) = 0.357129$.

Let us consider that we avoid the *Slack Clustering* step and start the regulation by throttling the input $task_1 \rightarrow task_4$: the regulation would result in a decrease in transfer time for $task_1 \rightarrow task_4$, so that data elements from $task_1$, and $task_2$ arrive to $task_4$ simultaneously. At $task_4 \rightarrow task_6$: an effective regulation would involve increasing the transfer time from $task_2 \rightarrow task_5$, while decreasing the transfer from $task_2 \rightarrow task_4$. Due to this regulation, the transfer time $task_2 \rightarrow task_4$ is modified, but the throttling value obtained in the the first regulation was obtained with the initial regulation, where the transfer $task_2 \rightarrow task_4$ was actually faster.

In order to avoid such an effect, we propose the *Slack Clustering* step, where inputs with slack that are not affecting each other are grouped together. For instance, in Figure 3, the input $task_1 \rightarrow task_4$ is at slack level 2 (because it contains the input $task_4 \rightarrow task_6$, which has some slack), while $task_3 \rightarrow task_6$ and $task_4 \rightarrow task_6$ are at slack level 1. In terms of the SMG model in Figure 4, the *Slack Clustering* step groups p_{15}, p_{16} in slack level 1, while p_9 is grouped in slack level 2. Such a slack clustering is therefore determined by the number of slack values in each workflow path. For instance, the number of slack places in the path $task_1 \rightarrow task_4 \rightarrow task_6$ is 2, i.e., $\mu(p_9), \mu(p_{15}) > 0$, where p_9, p_{15} belong to path $task_1 \rightarrow task_4 \rightarrow task_6$.

Updated data-throttling values are obtained in step *Data Throttling Values*, with all slack values being re-calculated after each adjustment of a cluster. The adjustment is done in increasing order of slack level (cluster), starting from level 1. Thus, in the example, we would increase the transmission bandwidth for $task_2 \rightarrow task_5$ (because it is the slowest path and if there is available bandwidth in the network link), and we would decrease the transmission bandwidth for $task_2 \rightarrow task_4$. As a result of conducting this step for the model in Figure 4, the following values are obtained: transmission between $task_1$ and $task_3$ (i.e., $task_1 \rightarrow task_3$) must be adjusted to a 28.57% of the total bandwidth (that is, to 28.57Mbps), while $task_1 \rightarrow task_4$ must be adjusted to 35.15Mbps, and $task_2 \rightarrow task_4$ to 44.55Mbps.

To summarise, data-throttling values are derived from the slack values as follows. A slack value $\mu(p)$ indicates that a transmission with duration tx_{old} must be delayed, i.e., $tx_{new} = tx_{old} + \alpha$. The value of α depends on the PN structure and relates to $\mu(p)$ and Θ (the upper performance bound of the PN model): $\alpha = \mu(p)/\Theta$. The bandwidth BW of a network link depends on the transmission time tx, $tx = d/BW$, where d is the size of the data sent through the network link. Hence, the new bandwidth BW_{new} can be computed as:

$$BW_{new} = \frac{1}{\dfrac{1}{BW_{old}} + \dfrac{\mu(p)}{\Theta \cdot d}} \tag{2}$$

where BW_{old} is the current bandwidth assigned to that transmission.

Finally, *Performance Analysis* must be conducted with and without the obtained data-throttling values, so that the impact on performance can be determined. We utilise SimGrid in this paper, though other techniques are briefly discussed in Section III. Figure 5 shows three execution timelines of the previous workflow with different network topologies. Figure 5 (a) involves a single output link per node, Figure 5 b) corresponds to a point-to-point network topology (i.e., dedicated links), and Figure 5 c) is the same latter scenario but where data regulation has been undertaken. $Task_i$ represents the duration of each task i (white boxes), and $tx_{i,j}$ represents the time for sending data from $task_i$ to $task_j$ (grey boxes). The symbol $\Delta_{i,j}$ means the waiting time for the dataset sent from $task_i$ in the input buffer of $task_j$, until $task_j$ begins execution. Although data throttling does not significantly impact makespan (5.6779 seconds versus 5.7750 seconds), it can be noticed that it does impact the input buffer waiting times for $task_4$ and $task_6$.

V. Experiments and Results

Our strategy has been evaluated using the Montage [4] workflow (see Figure 6) – a real application that has been used to create image mosaics in the astrophysics domain. A description of the abstract workflow and performance information were collected from DAX files generated with the Pegasus workflow system [1]. We have performed two different experiments. The first one determines the impact of bandwidth throttling on the workflow makespan, while the second one shows how buffer and network bandwidth utilizations change.

Network topology	**Network bandwidth**		
	10Mbps	**100Mbps**	**1Gbps**
Single output	$193.20s$	$61.18s$	$47.98s$
PP without BW throttling	$153.15s$	$57.17s$	$47.58s$
PP with BW throttling	$153.32s$	$57.21s$	$47.58s$

Table I
Makespan (in seconds) for Montage workflow with 5 input files, with different network topology and bandwidth.

Our strategy has been implemented in Matlab, whilst simulation of workflow execution has been carried out through SimGrid [6]. SimGrid is a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. We utilised SimGrid as it provides direct support for DAX files. For the experiments, we have used the Montage workflow for 5 input files, composed of 25 tasks. Each task is considered to be executed on separate hosts. Moreover, we have assumed two different network topologies between hosts on the experiments: (1) each host has just one output data-link; or (2) each host is connected point-to-point (PP) to other hosts. Bandwidth throttling (BW throttling) has been considered only for topology (2). We have considered three different data-link bandwidths: 10Mbps, 100Mbps and 1Gbps. The experiments have been executed on an Intel Pentium IV 3.6GHz with 2GiB RAM DDR2 533MHz host machine.

A. Impact on the Workflow Makespan

Table I summarises experimental results showing the impact on makespan. The first column indicates the network topology, the others show the different network bandwidth used in the experiment: 10Mbps, 100Mbps and 1Gbps. The network latency is assumed to be constant and equal to 10^{-4} seconds for all network topologies. The workflow makespan (in seconds) is calculated based on the simulation time from SimGrid.

A point-to-point (PP) connection *without* throttling is a 26.15% faster than a single output data-link connection in the first case (10Mbps), and only 0.84% in the last case (1Gbps). Additionally, in all cases a PP connection *with* throttling is slower than without throttling. As the bandwidth increases, the impact on the workflow makespan becomes smaller. Park & Humphrey [16] also indicated a correlation between $\frac{data\ transmission}{computation}$ ratio and makespan, which our results verify.

B. Input Buffers and Network Bandwidth Usage

In this section we measure the impact of data throttling on the input buffer occupancy and network bandwidth. In order

[1]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

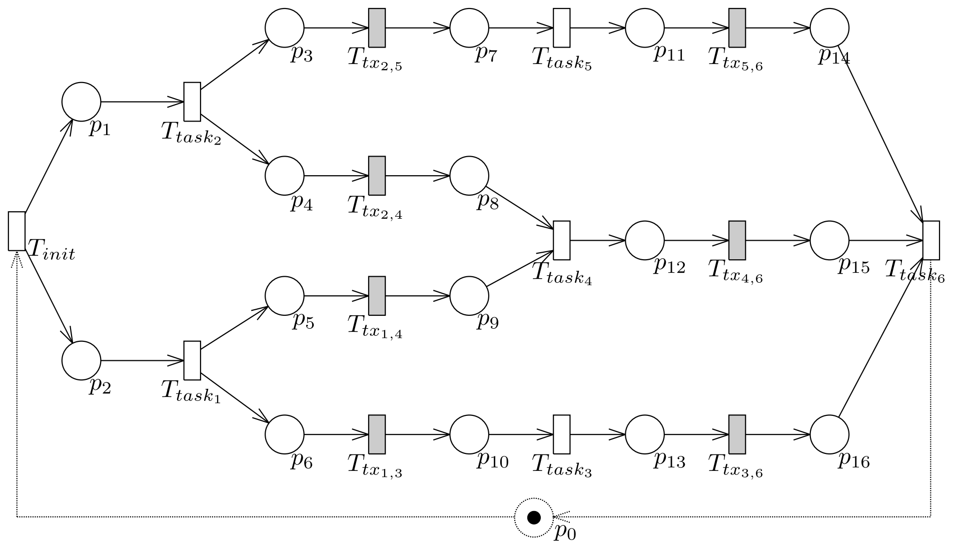
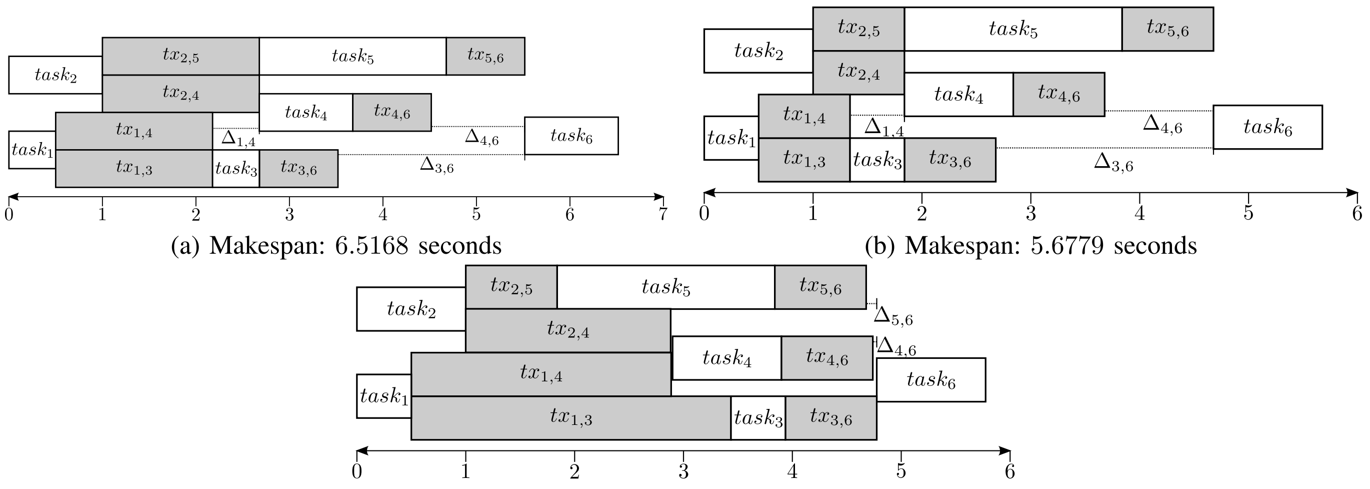


Figure 4. PN-based abstract workflow with explicit data-transfer transitions.



(a) Makespan: 6.5168 seconds

(b) Makespan: 5.6779 seconds

(c) Makespan: 5.7750 seconds

Figure 5. Makespan (in seconds) of workflow depicted in 3 with different network topologies.

Montage task	Network topology		
	Single output	PP w/o. BW throttling	PP w. BW throttling
mDiffFit$_1$	$20.49s$	$0.45s$	$0.22s$
mDiffFit$_2$	$20.49s$	$0.45s$	$0.21s$
mDiffFit$_3$	$20.24s$	$0.22s$	$0.14s$
mDiffFit$_4$	$0.09s$	$0.05s$	$0.23s$
mDiffFit$_5$	$20.59s$	$0.49s$	$0.24s$
mDiffFit$_7$	$20.24s$	$0.23s$	$0.14s$
mDiffFit$_8$	$26.84s$	$0.08s$	$0.09s$
mDiffFit$_9$	$6.60s$	$0.15s$	$0.08s$
mConcatFit	$124.76s$	$4.20s$	$1.57s$
mBackground$_1$	$33.84s$	$13.80s$	$7.02s$
mBackground$_2$	$13.35s$	$13.35s$	$6.61s$
mBackground$_3$	$33.94s$	$13.84s$	$6.55s$
mBackground$_4$	$33.59s$	$13.57s$	$6.31s$
mBackground$_5$	$40.19s$	$13.43s$	$6.45s$
mImgTbl	$1.63s$	$1.63s$	$0.75s$

(a) 10Mbps

Montage task	Network topology		
	Single output	PP w/o. BW throttling	PP w. BW throttling
mDiffFit$_1$	$2.44s$	$0.44s$	$0.21s$
mDiffFit$_2$	$2.44s$	$0.44s$	$0.21s$
mDiffFit$_3$	$2.23s$	$0.23s$	$0.12s$
mDiffFit$_4$	$0.04s$	$0.03s$	$0.02s$
mDiffFit$_5$	$2.48s$	$0.47s$	$0.23s$
mDiffFit$_7$	$2.23s$	$0.23s$	$0.11s$
mDiffFit$_8$	$2.73s$	$0.05s$	$0.03s$
mDiffFit$_9$	$0.50s$	$0.18s$	$0.09s$
mConcatFit	$15.96s$	$4.06s$	$1.96s$
mBackground$_1$	$15.50s$	$13.49s$	$6.91s$
mBackground$_2$	$13.05s$	$13.05s$	$6.45s$
mBackground$_3$	$15.53s$	$13.52s$	$6.48s$
mBackground$_4$	$15.28s$	$13.28s$	$6.30s$
mBackground$_5$	$15.78s$	$13.11s$	$6.31s$
mImgTbl	$1.60s$	$1.60s$	$0.32s$

(b) 100Mbps

Montage task	Network topology		
	Single output	PP w/o. BW throttling	PP w. BW throttling
mDiffFit$_1$	$0.64s$	$0.44s$	$0.24s$
mDiffFit$_2$	$0.64s$	$0.44s$	$0.24s$
mDiffFit$_3$	$0.43s$	$0.23s$	$0.11s$
mDiffFit$_4$	$0.03s$	$0.03s$	$0.01s$
mDiffFit$_5$	$0.67s$	$0.47s$	$0.25s$
mDiffFit$_7$	$0.43s$	$0.23s$	$0.14s$
mDiffFit$_8$	$0.32s$	$0.05s$	$0.05s$
mDiffFit$_9$	$0.11s$	$0.18s$	$0.11s$
mConcatFit	$5.19s$	$4.05s$	$1.95s$
mBackground$_1$	$13.66s$	$13.46s$	$6.73s$
mBackground$_2$	$13.02s$	$13.02s$	$6.42s$
mBackground$_3$	$13.69s$	$13.49s$	$6.61s$
mBackground$_4$	$13.45s$	$13.25s$	$6.31s$
mBackground$_5$	$13.34s$	$13.07s$	$6.44s$
mImgTbl	$1.60s$	$1.60s$	$0.21s$

(c) 1Gbps

Table II
BUFFER WAITING TIME (IN SECONDS) OF MONTAGE TASKS WITH DIFFERENT NETWORK TOPOLOGY AND BANDWIDTH.

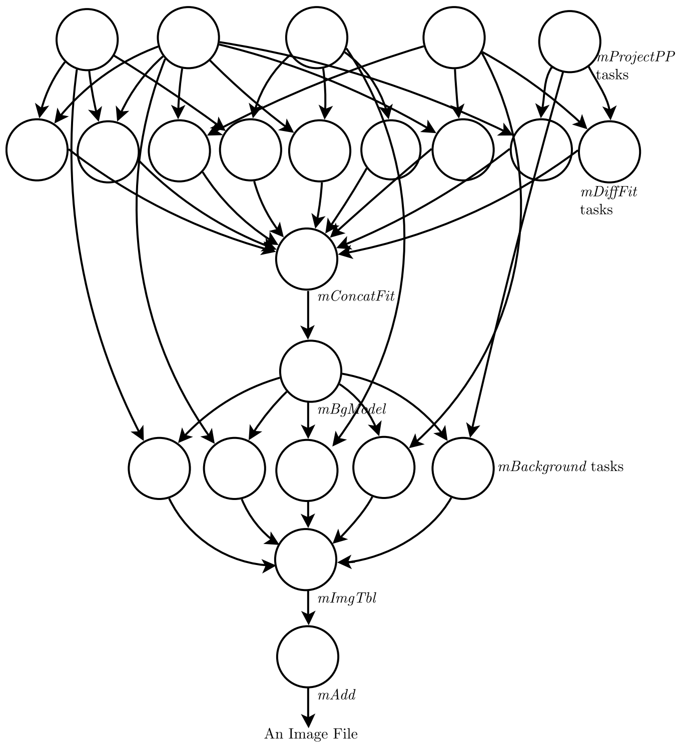


Figure 6. Montage workflow for 5 input files.

to fulfill this goal, we consider the inter-arrival time between data items (e.g. files) to the same task. The results of this experiment with three different network bandwidths (10Mbps, 100Mbps and 1Gbps) are shown in Table II((a), (b) and (c), respectively). The first column indicates Montage tasks which need more than one input to execute. The following columns show the waiting time of input data in the buffer until the task begins its execution with different network topologies: one single output data-link per host, a PP connection between hosts without (and with) BW throttling.

Hence, bandwidth throttling has a great impact on the buffer waiting time of the tasks. Bandwidth throttling outperforms both single out and PP in all cases, with *mDiffFit*$_4$ for 10Mbps network being an anomaly. Note that even the bandwidth throttling is not enhancing the buffer waiting time for *mDiffFit*$_4$, such buffer waiting time is strongly improved in the case of the task *mConcatFit* (the next task in the workflow to execute after *mDiffFit*$_4$): it is reduced by 98.74% with respect to the single output data-link topology and 62.61% with respect to PP without bandwidth throttling.

When the network bandwidth is 1Gbps, the results of single output data-link and PP without bandwidth throttling are quite similar, whilst applying the bandwidth throttling still improves the buffer waiting time of the workflow tasks.

We have plotted the results of tasks with the greatest number of input dependencies in the Montage workflow (in this case, *mConcatFit* and *mImgTbl*) in Figure 7((a) and (b), respectively). As it can be observed, performing a bandwidth throttling outperforms both other network topologies in the experiment.

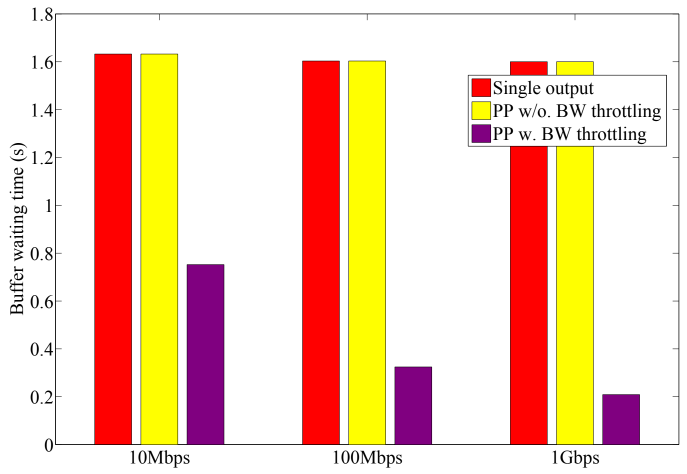
We can summarise our experimental findings as: i) there exists an intrinsic relationship between the transmitted data size and workflow task execution time which can be useful for deciding when bandwidth throttling would be most appropriate; ii) applying data throttling in a PP network topology does not make a significant change to the overall workflow makespan (compared to other topologies); but iii) performing data throttling has a great impact on the input buffer and network bandwidth usage, as outlined in our results.

VI. Conclusions

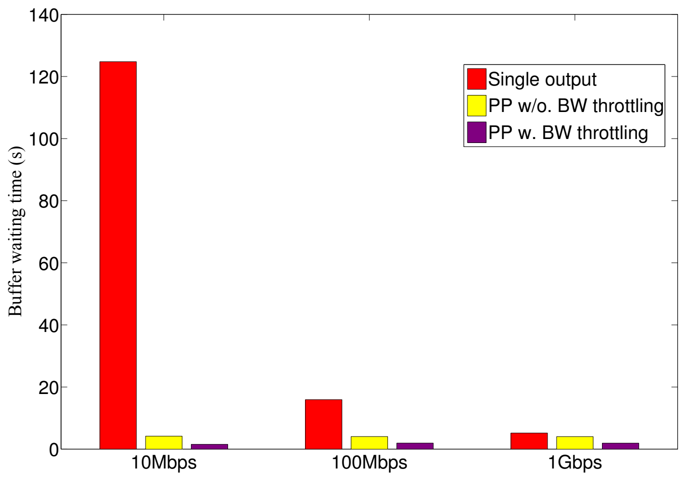
Current scientific problems usually demand a large amount of data transmissions and complex data analysis. Scientific workflows have become the computational technology of choice for solving such problems. However, scientific workflows can have data dependencies which neglect their intrinsic parallel behaviour. Previous approaches attempt to either avoid data movement completely (i.e. by co-locating tasks or moving them closer to the data source), or move data as fast as possible (based on the network link capacity). This last approach, however, can lead to some workflow tasks waiting for incoming data, while blocking use of shared buffers that could be better utilised.

In this paper, we study a data-throttling strategy for improving the use of bandwidth and input buffers, so that tasks have all their inputs arriving at the same time (buffer usage minimised). In order to achieved this, some transmissions have their speed modified (either reduced or increased), making a better usage of bandwidth. The strategy takes as an input a Petri net-based model of a workflow. Our approach makes use of linear programming techniques for which polynomial algorithms exist. As an output, the method obtains the throttling data values, but it also analyses the affection that data-throttling can have on workflow performance. The affection depends on the workflow structure, and on the ratio between computational and transmission and tasks. By means of this analysis, the user can establish a trade-off between throttling (and possible degrading the performance) or not. We have tested our approach with Montage, a real scientific workflow from astrophysics, supposing different data-link speeds. The results are very promising, and show us how our approach enables a more effective use of bandwidth, and reduces the waiting time of data in input buffers.

As future work, we plan to test data-throttling values obtained from the approach in more realistic environments. Our approach can also be used for any workflow encoded in DAX-based representation used within the Pegasus workflow system. We are aware of a number of other workflow examples – such as Cybershake, Epigenomics and Inspiral Search and CIVET (brain imaging) – which all have DAX-based workflow graph representations, involve synchronisation points and can benefit from our approach.



(a) Workflow task *mImgTbl*



(b) Workflow task *mConcatFit*

Figure 7. Buffer waiting time in task *mImgTbl* (a) and *mConcatFit* (b) for different network topologies and bandwidths.

ACKNOWLEDGEMENTS

This work was supported by the EU FP7 Project DISC (INFSO-ICT-224498), Spanish Projects CICYT DPI2010-20413, TIN2010-17905 and by Fundación Aragón I+D. Mr Rodríguez's work was supported by grants from DGA (CONAID) and CAI, "Programa Europa XXI de Estancias de Investigación" ref. n. IT 5/11, and Ministry of Education, Culture and Sport, ref n. TME2011-00255.

REFERENCES

[1] W. M. P. v. d. Aalst, A. Hirnschall, and H. M. W. E. Verbeek. An Alternative Way to Analyze Workflow Graphs. In *Proceedings of CAiSE'02*, pages 535–552, London, UK, UK, 2002. Springer-Verlag.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons, 1995.

[3] L. Aversano, A. Cimitile, P. Gallucci, and M. Villani. FlowManager: a workflow management system based on Petri nets. In *Proceedings of COMPSAC'02*, pages 1054–1059, 2002.

[4] G. B. Berriman, E. Deelman, J. Good, J. C. Jacob, D. S. Katz, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su. Generating Complex Astronomy Workflows. In *Workflows for e-Science*, pages 19–38. Springer London, 2007.

[5] J. Carmona, J. Júlvez, J. Cortadella, and M. Kishinevsky. Scheduling Synchronous Elastic Designs. In *Proceedings of ACSD 2009*, Augsburg, Germany, July 2009.

[6] H. Casanova, A. Legrand, and M. Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE Int. Conf. on Computer Modeling and Simulation*, March 2008.

[7] E. Deelman, G. Mehta, G. Singh, M. Su, and K. Vahi. *Workflows for eScience*, chapter Pegasus: Mapping Large-Scale Workflows to Distributed Resources, pages 376–394. Springer, 2007.

[8] R. Duan, F. Nadeem, J. Wang, Y. Zhang, R. Prodan, and T. Fahringer. A Hybrid Intelligent Method for Performance Modeling and Prediction of Workflow Activities in Grids. In *CCGrid 2009*, pages 339–347, Shanghai, China, 18–21 May 2009. IEEE Computer Society.

[9] R. Duan, R. Prodan, and T. Fahringer. Run-time optimisation of grid workflow applications. In *ICGRID'06*, pages 33–40, Washington, DC, USA, 2006. IEEE Computer Society.

[10] J. Esparza and M. Nielsen. Decidability Issues for Petri Nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.

[11] Z. Guan, F. Hernandez, P. Bangalore, J. Gray, A. Skjellum, V. Velusamy, and Y. Liu. Grid-Flow: A Grid-Enabled Scientific Workflow System with a Petri-Net-Based Interface: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18:1115–1140, August 2006.

[12] A. Hoheisel. User tools and languages for graph-based Grid workflows: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1101–1113, 2006.

[13] T. Murata. Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, April 1989.

[14] F. Nerieri, R. Prodan, T. Fahringer, and H.-L. Truong. Overhead analysis of grid workflow applications. In *ICGRID'06*, pages 17–24, 2006.

[15] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1067–1100, 2006.

[16] S.-M. Park and M. Humphrey. Data Throttling for Data-Intensive Workflows. In *IEEE IPDPS*, pages 1–11, April 2008.

[17] S. Pellegrini, A. Hoheisel, F. Giacomini, and A. Ghiselli. Using GWorkflowDL for Middleware-Independent Modeling and Enactment of Workflows. In *Proceedings of the CoreGRID Integration Workshop 2008*, Crete, Greece, 2008.

[18] R. J. Rodríguez and J. Júlvez. Accurate Performance Estimation for Stochastic Marked Graphs by Bottleneck Regrowing. In *Proceedings of EPEW'10*, volume 6342 of *LNCS*, pages 175–190. Springer, 2010.

[19] R. Tolosana-Calasanz, J. A. Bañares, P. Álvarez, J. Ezpeleta, and O. F. Rana. An Uncoordinated Asynchronous Checkpointing Model for Hierarchical Scientific Workflows. *Journal of Computer and System Sciences*, 76(6):403–415, September 2010.

[20] R. Tolosana-Calasanz, J. A. Bañares, O. F. Rana, P. Álvarez, J. Ezpeleta, and A. Hoheisel. Adaptive Exception Handling for Scientific Workflows. *Concurr. Comput. : Pract. Exper.*, 22(5):617–642, April 2010.

[21] R. Tolosana-Calasanz, O. Rana, and J. Bañares. Automating Performance Analysis from Taverna Workflows. In *CBSE*, volume 5282 of *LNCS*, pages 1–15. Springer Berlin / Heidelberg, 2008.

[22] W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*, volume 1 of *MIT Press Books*. The MIT Press, 2004.

[23] M. Vossberg, A. Hoheisel, T. Tolxdorff, and D. Krefting. A Reliable DICOM Transfer Grid Service Based on Petri Net Workflows. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 441–448, Washington, DC, USA, 2008. IEEE Computer Society.

[24] J. Yu and R. Buyya. A taxonomy of workflow management systems for Grid computing. *CoRR*, abs/cs/0503025, 2005.