Using Simple PID Controllers to Prevent and Mitigate Faults in Scientific Workflows

Rafael Ferreira da Silva¹, Rosa Filgueira², Ewa Deelman¹, Erola Pairo-Castineira¹, Ian Michael Overton¹, Malcolm Atkinson⁵

¹University of Southern California, Information Sciences Institute, Marina Del Rey, CA, USA
²British Geological Survey, Lyell Centre, Edinburgh EH14 4AP
³MRC Institute of Genetics and Molecular Medicine, University of Edinburgh, Edinburgh, UK
⁴Usher Institute of Population Health Sciences and Informatics, University of Edinburgh, Edinburgh, UK
⁵School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK
{rafsila,deelman}@isi.edu, rosa@bgs.ac.uk, Erola.Pairo-Castineira@igmm.ed.ac.uk, {ian.overton,malcolm.atkinson}@ed.ac.uk,

ABSTRACT

Scientific workflows have become mainstream for conducting large-scale scientific research. As a result, many workflow applications and Workflow Management Systems (WMSs) have been developed as part of the cyberinfrastructure to allow scientists to execute their applications seamlessly on a range of distributed platforms. In spite of many success stories, a key challenge for running workflows in distributed systems is failure prediction, detection, and recovery. In this paper, we propose an approach to use control theory developed as part of autonomic computing to predict failures before they happen, and mitigated them when possible. The proposed approach applying the proportional-integral-derivative controller (PID controller) control loop mechanism, which is widely used in industrial control systems, to mitigate faults by adjusting the inputs of the controller. The PID controller aims at detecting the possibility of a fault far enough in advance so that an action can be performed to prevent it from happening. To demonstrate the feasibility of the approach, we tackle two common execution faults of the Big Data era—data storage overload and memory overflow. We define, implement, and evaluate simple PID controllers to autonomously manage data and memory usage of a bioinformatics workflow that consumes/produces over 4.4TB of data, and requires over 24TB of memory to run all tasks concurrently. Experimental results indicate that workflow executions may significantly benefit from PID controllers, in particular under online and unknown conditions. Simulation results show that nearly-optimal executions (slowdown of 1.01) can be attained when using our proposed method, and faults are detected and mitigated far in advance of their occurrence.

Keywords

Scientific workflows, Fault detection and handling, Autonomic computing

1. INTRODUCTION

Scientists want to extract the maximum information out of their data—which are often obtained from scientific instruments and processed in large-scale distributed systems. Scientific workflows are a mainstream solution to process large-scale scientific computations in distributed systems, and have supported traditional and breakthrough researches across several domains. In spite of impressive achievements today, failure prediction, detection, and recovery are still a major challenge in workload management in distributed system, both at the application and resource levels. Failures affect the turnaround time of the applications, and that of the umbrella analysis and therefore the productivity of the scientists that depend on the power of distributed computing to do their work.

In this work, we investigate how the proportional-integral-derivative controller (PID controller) control loop mechanism, which is widely used in industrial systems, can be applied to predict and prevent failures in end-to-end workflow executions across distributed, heterogeneous computational environments. The basic idea behind a PID controller is to read data from a sensor, then compute the desired actuator output by calculating proportional (P), integral (I), and derivative (D) responses and summing those three components to compute the output. Each of the components can often be interpreted as the present error (P), the accumulation of past errors (I), and a prediction of future errors (D), based on current rate of change. The main advantage of using a PID controller is that the control loop mechanism progressively monitors the evolution of the workflow execution, detecting possible faults before they occur, and when needed performs actions that lead the execution to a steady-state.

The main contributions of this paper include:
1. The evaluation of PID controllers to prevent and mitigate two major problems of the Big Data era: data storage overload and memory overflow;
2. The characterization of a bioinformatics workflow, which consumes/produces over 4.4TB of data, and requires over 24TB of memory;
3. An experimental evaluation via simulation to demonstrate the feasibility of the proposed approach using simple PID controllers; and
4. A performance optimization study to tune the parameters of the control loop to provide nearly-optimal workflow executions, where faults are detected and handled

Copyright held by the author(s).
2. RELATED WORK

Several offline strategies and techniques were developed to detect and handle failures during scientific workflow executions [3, 5, 21, 27, 28, 36]. Autonomic online methods were also proposed to cope with workflow failures at runtime, for example by providing checkpointing [20, 25, 30], provenance [13, 25], task resubmission [10, 31], and task replication [5, 8], among others. However, these systems do not aim to prevent faults, but mitigate them, and although task replication may increase the probability of having a successful execution in another computing resource, it should be used sparingly to avoid overloading the execution platform [11]. These system also make strong assumptions about resource and application characteristics. Although several works address task requirement estimations based on provenance data [11, 15, 22, 29], accurate estimations are still challenging, and may be specific to a certain type of application.

In [4], a prediction algorithm based on machine learning (Naïve Bayes classifier) is proposed to identify faults before they occur, and to apply preventive actions to mitigate the faults. Experimental results show that faults can be predicted up to 94% of accuracy, however the approach is tied to a small set of applications, and it is assumed that the application requirements do not change over time. In previous works, we proposed an autonomic method described as a MAPE-K loop to cope with online non-clairvoyant workflow executions faults on grids [13, 17], where unpredictability is addressed by using a priori knowledge extracted from execution traces to identify severity levels of faults, and apply a specific set of actions. Although this is the first work on self-healing of workflow executions in online and unknown conditions, experimental results on a real platform show an important improvement of the QoS delivered by the system. However, the method does not prevent faults from happening (actions are performed once faults are detected). In [19], a machine learning approach based in inductive logic programming is proposed for fault prediction and diagnosis in grids. This approach is limited to small scale applications and a few parameters—the number of rules may exponentially increase as the number of tasks in a workflow or the accounted parameters increase.

To the best of our knowledge, this is the first work that uses PID controllers to mitigate faults in scientific workflow executions under online and unknown conditions.

3. PID CONTROLLERS

The keystone component of the proposed process is the proportional-integral-derivative controller (PID controller) [34], control loop mechanism, which is widely used in industrial control systems, to mitigate faults by adjusting the process control inputs. Examples of such systems are the ones where the temperature, pressure, or the flow rate, need to be controlled. In such scenarios, the PID controller aims at detecting the possibility of a fault far enough in advance so that an action can be performed to prevent it from happening.

Figure 1 shows the general PID control system loop. The setpoint is the desired or command value for the process variable. The control system algorithm uses the difference between the output (process variable) and the setpoint to determine the desired actuator input to drive the system.

The control system performance is measured through a step function as a setpoint command variable, and the response of the process variable. The response is quantified by measuring defined waveform characteristics as shown in Figure 2. Raise time is the amount of time the system takes to go from about 10% to 90% of the steady-state, or final, value. Percent overshoot is the amount that the process variable surpasses the final value, expressed as a percentage of the final value. Settling time is the time required for the process variable to settle to within a certain percentage (commonly 5%) of the final value. Steady-state error is the final difference between the process variable and the setpoint. Dead time is a delay between when a process variable changes, and when that change can be observed.

Process variables (output) are determined by fault-specific metrics quantified online. The setpoint is constant and defined as 1. The output of the PID controller is an input value for a Curative Agent, which determines whether an action should be performed (Figure 3). Negative input values mean the control system is raising too fast and may tend to the overshoot state (i.e., a faulty state), therefore preventive or corrective actions should be performed. Actions may include task pre-emption, task resubmission, task clustering, task cleanup, storage management, etc. In contrast, positive input values mean that the control system is smoothly rising to the steady state. The control signal $u(t)$ (output) is defined as follows:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt},$$

where $K_p$ is the proportional gain constant, $K_i$ is the integral gain constant, $K_d$ is the derivative gain constant, and $e$ is the error defined as the difference between the setpoint and the process variable value.
4. DEFINING PID CONTROLLERS

In our proposed approach, a PID controller is defined and used for each possible-future fault identified from workload traces (historical data). In some cases, a particular type of faults cannot be modeled as a full PID controller. For example, there are faults that cannot be predicted far in advance (e.g., unavailability of resources due to a power cut). In this case, a PI (proportional-integral) controller can be defined and deployed. In production systems, a large number of controllers may be defined and used to control, for example, CPU utilization, network bandwidth, etc. In this paper, we demonstrate the feasibility of the use of PID controllers by tackling two common issues of workflow executions: data and memory overflow.

4.1 Workflow Data Footprint and Management

In the era of Big Data Science, applications are producing and consuming ever-growing data sets. A run of scientific workflows that manipulate these data sets may lead the system to an out of disk space fault if no mechanisms are in place to control how the available storage is used. To prevent this, data cleanup tasks are often automatically inserted into the workflow by the workflow management system (e.g., the number of concurrent task executions is limited to prevent data usage overflow. Cleanup tasks remove data sets that are no longer needed by downstream tasks, but nevertheless they add an important overhead to the workflow execution [1].

**PID Controller.** The controller process variable (output) is defined as the ratio of the estimated disk space required by current tasks in execution, and the actual available disk space. The system is in a non-steady state if the total amount of disk space consumed is above (overshoot) a predefined threshold (setpoint), or the amount of used disk space is below the setpoint. The proportional (P) response is computed as the difference between the current and the previous memory usage errors; and the derivative (D) response is computed from the cumulative proportional responses (previous memory usage errors); and the integral (I) response is computed as the difference between the current and the previous memory overflow (or underutilization) error values.

**Corrective Actions.** Negative values for the control signal indicate that the ensemble of running tasks are leading the system to an overflow state, thus some tasks should be preempted to prevent the system to run out of memory. For positive u(t) values, the memory consumption of current running tasks is below a predefined memory consumption setpoint. Therefore, the workflow management system may spawn additional tasks for concurrent execution.

5. EXPERIMENTAL EVALUATION

5.1 Scientific Workflow Application

The 1000 genomes project provides a reference for human variation, having reconstructed the genomes of 2,504 individuals across 26 different populations [12]. The test case used in this work identifies mutational overlaps using data from the 1000 genomes project in order to provide a null distribution for rigorous statistical evaluation of potential disease-related mutations. This test case (Figure 1) has been implemented as a Pegasus workflow, and is composed of five different tasks:

**Individuals.** This task fetches and parses the Phase 3 data [12] from the 1000 genomes project per chromosome. These files list all of Single nucleotide polymorphisms (SNPs) variants in that chromosome and which individuals have
The goal of this experiment is to ensure that correctly defined executions complete, that performance is acceptable, and that possible-future faults are quickly detected and automatically addressed. To achieve this, we implemented mechanisms to handle interruptions and recover from faults. For instance, if a task is interrupted due to a disk I/O error, it can be resumed from the last checkpoint. Similarly, if a task fails due to a memory overflow, it can be restarted with the necessary memory allocation. These mechanisms help in ensuring the reliability of the workflow.

### 5.3 Experiment Conditions

The experiments use trace-based simulation. Since most workflow simulators are event-based, we developed an activity-based simulator to simulate every time slice of the PID controllers behaviors (which is available online [32]). The simulator provides support for task scheduling and resource provisioning at the workflow level. The simulated computing environment represents the three nodes from the Eddie Mark 3 cluster described in Section 5.2 (total 80 CPU cores). Additionally, we assume a shared network file system among the nodes with total capacity of 500GB.

We use an FCFS policy with task preemption and backfill for task scheduling—tasks submitted at the same time are randomly chosen, and preempted tasks return to the top of the queue. To avoid unrecoverable faults due to run out of disk space, we implemented a data cleanup mechanism to remove data that are no longer required by downstream tasks [33]. Data cleanup tasks are only triggered if the maximum storage capacity is reached. In this case, all running tasks are preempted, and the data cleanup task is executed, and the workflow resumes its execution. Note that this mechanism may add a significant overhead to the workflow execution.

The goal of this experiment is to ensure that correctly defined executions complete, that performance is acceptable, and that possible-future faults are quickly detected and automatically addressed.
types of controllers: (P) proportional, (PI) proportional-integral, and (PID) proportional-integral-derivative. Reference denotes the makespan of a reference workflow execution computed offline and under known conditions.

Overall makespan evaluation. Table 2 shows the average makespan (in hours) for the three configurations of the controller and the reference workflow execution. The degradation of the makespan is expected due to the online and unknown conditions (no information about the tasks is available in advance). In spite of the fact that the mean does not provide accurate estimates, the use of a control loop mechanism diminishes this effect. The use of controllers may also degrade the makespan due to task preemption. However, if tasks were scheduled only using the estimates from the mean, the workflow would not complete its execution due to lack of disk space or memory overflows.

Task characteristics estimation is beyond the scope of this work, and sophisticated methods to provide accurate estimates can be found in [1][18][22][29]. However, this work intends to demonstrate that even using inaccurate estimation methods, PID controllers yield good results.

Reference Workflow Execution. In order to measure the efficiency of our online method under online and unknown conditions, we compare the workflow execution performance (in terms of the turnaround time to execute all tasks) to a reference workflow—computed offline under known conditions, i.e., all requirements (e.g., runtime, disk, memory) are accurate and known in advance. We performed several runs for the reference workflow, which yielded an averaged makespan of 382,887.7s (~106h, standard deviation ≤ 5%).

5.4 Experimental Results and Discussion

We have conducted workflow runs with three different types of controllers: (P) only the proportional component is evaluated: \( K_p = 1 \), and \( K_i = K_d = 0 \); (PI) the proportional and integral components are enabled: \( K_p = K_i = 1 \), and \( K_d = 0 \); and (PID) all components are activated: \( K_p = K_i = K_d = 1 \). The reference workflow execution is reported as Reference. We have performed several runs of each configuration to produce results with statistical significance (errors below 5%).
are represented as negative values (red bars), while positive values (blue bars) indicate the number of tasks scheduled at an instant of time. Additionally, the right $y$-axis shows the step response of the controller input value (black line) for disk usage during the workflow execution. Recall that positive input values ($u(t) > 0$, Equation 1) trigger task scheduling, while negative input values ($u(t) < 0$) trigger task preemption.

The proportional controller ($P$, Figure 5a) is limited to the current error, i.e., the amount of disk space that is over/underutilized. Since the controller input value is strictly proportional to the error, there is a burst on the number of tasks to be scheduled at the beginning of the execution. This bursty pattern and the nearly constant variation of the input value lead the system to an inconsistent state, where the remaining tasks to be scheduled cannot lead the controller within the steady-state. Consequently, tasks are constantly scheduled and then preempted. In the example scenario shown in Figure 5a, this process occurs at about 4h, and performs more than 6,000 preemptions. Table 3 shows the average number of preemptions and cleanup tasks occurrences per workflow execution. On average, proportional controllers produced more than 7,000 preemptions, but no cleanup tasks. The lack of cleanup tasks indicate that the number of concurrent executions is very low (mostly influenced by the number of task preemptions), which is observed from the high average application slowdown of 1.30.

The proportional-integral controller ($PI$, Figure 5b) aggregates the cumulative error when computing the response of the controller. As a result, the bursty pattern is smoothed along the execution, and task concurrency is increased. The cumulative error tends to increase the response of the $PI$ controller at each iteration (both positively or negatively). Thus, task preemption occurs earlier during execution. On the other hand, this behavior mitigates the vicious cycle present in the $P$ controllers, and consequently the average number of preempted tasks is substantially reduced to 168 (Table 3). A drawback of using a $PI$ controller, is the presence of cleanup tasks, which is due to the higher level of concurrency among task executions.

The proportional-integral-derivative controller ($PID$, Fig-
ure.\(^2\)) gives importance to the previous response produced by the controller (the last computed error). The derivative component drives the controller to trigger actions once the current error follows (or increases) the previous error trend. In this case, the control loop only performs actions when disk usage is moving towards an overflow or underutilization state. Note that the number of actions (scheduling/preemption) triggered in Figure 5 is much less than the number triggered by the PI controller: the average number of preempted tasks is 73, and only 4 cleanup tasks on average are spawned (Table 3).

**Memory Usage.** Figure 6 shows the time series of the number of tasks scheduled or preempted during the workflow executions for the memory controllers. The right y-axis shows the step response of the controller input value (black line) for memory usage during the workflow execution. We present the response function of a controller attached to a standard cluster (32 cores, 64GB RAM, Section 5.2), which runs the population, pair_overlap_mutations, and frequency_overlap_mutations tasks. The total memory allocations required to run all these tasks is over 4TB, which might lead the system to memory overflow states.

When using the proportional controller (P, Figure 4), most of the actions are triggered by the data footprint controller (Figure 4a). As aforementioned, memory does not become an issue when only the proportional error is taken into account, since task execution is nearly sequential (low level of concurrency). As a result, only a few tasks (on average less than 5) are preempted due to memory overflow. Note that the process of constant task scheduling (∼50h of execution) is strongly influenced by the memory controller. The step response shown in Figure 4b highlights that most of the task preemptions occur in the standard cluster. This result suggests that actions performed by the global data footprint controller is affected by actions triggered by the local memory controller. The analysis of the influence of multiple concurrent controllers is out of the scope of this paper; however this result demonstrates that controllers should be used sparingly, and actions triggered by controllers should be performed by priority or the controller hierarchical level.

The PI controller (Figure 5a) mitigates this effect, since the cumulative error prevents the controller from triggering repeated actions. Observing the step response of the PI memory controller and the PI data footprint controller (Figure 5a), we notice that most of the task preemptions are triggered by the memory controller, particularly in the first quarter of the execution. The average data footprint per task of the population, pair_overlap_mutations, and frequency_overlap_mutations tasks is 0.02GB, 1.85GB, and 1.83GB (Table 3), respectively. Thus, the data footprint controller tends to increase the number of concurrent tasks. In the absence of memory controllers, the workflow execution would tend to memory overflow, and thus lead to a failed state.

The derivative component of the PID controller (Figure 4) acts as a catalyst to improve memory usage: it decreases the overshoot and the settling time without affecting the steady-state error. As a result, the number of actions triggered by the PID memory controller is significantly reduced when compared to the PI or P controllers.

Table 4: Ziegler-Nichols tuning, using the oscillation method. These gain values are applied to the parallel form of the PID controller, which is the object of study in this paper. When applied to a standard PID form, the integral and derivative parameters are only dependent on the oscillation period $T_u$.

<table>
<thead>
<tr>
<th>Control Type</th>
<th>$K_p$</th>
<th>$K_i$</th>
<th>$K_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>0.50 $\cdot K_u$</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PI</td>
<td>0.45 $\cdot K_u$</td>
<td>1.2 $\cdot K_p/T_u$</td>
<td>–</td>
</tr>
<tr>
<td>PID</td>
<td>0.60 $\cdot K_u$</td>
<td>2 $\cdot K_p/T_u$</td>
<td>$K_p \cdot T_u/8$</td>
</tr>
</tbody>
</table>

**6. Tuning PID Controllers**

The goal of tuning a PID loop is to make it stable, responsive, and to minimize overshooting. However, there is no optimal way to achieve responsiveness without compromising overshooting, or vice-versa. Therefore, a plethora of methods have been developed for tuning PID control loops. In this paper, we use the Ziegler-Nichols method to tune the gain parameters of the data footprint and memory controllers. This is one of the most common heuristics that attempts to produce tuned values for the three PID gain parameters ($K_p$, $K_i$, and $K_d$) given two measured feedback loop parameters derived from the following measurements: (1) the period $T_u$ of the oscillation frequency at the stability limit, and (2) the gain margin $K_u$ for loop stability. In this method, the $K_i$ and $K_d$ gains are first set to zero. Then, the proportional gain $K_p$ is increased until it reaches the ultimate gain $K_u$, at which point the output of the loop starts to oscillate. $K_u$ and the oscillation period $T_u$ are then used to set the gains according to the values described in Table 4.

A detailed explanation of the method can be found in [37]. In this section, we will present how we determine the period $T_u$, and the gain margin $K_u$ for loop stability.

**6.1 Determining $T_u$ and $K_u$**

The Ziegler-Nichols oscillation method is based on experiments executed on an established closed loop. The overview of the tuning procedure is as follows [21]:

1. Turn the PID controller into a P controller by setting $K_i = K_d = 0$. Initially, $K_p$ is also set to zero;
2. Increase $K_u$ until there are sustained oscillations in the signal in the control system. This $K_p$ value is denoted the ultimate (or critical) gain, $K_u$;
3. Measure the ultimate (or critical) period $T_u$ of the sustained oscillations; and
4. Calculate the controller parameter values according to Table 4, and use these parameter values in the controller.

Since workflow executions are intrinsically dynamic (due to the arrival of new tasks at runtime), it is difficult to establish a sustained oscillation in the signal. Therefore, in this paper we measured sustained oscillation in the signal within the execution of long running tasks—in this case the individual...
Figure 6: Memory Usage: Number of tasks scheduled (blue bars for positive values) and preempted (red bars for negative values) during the lifespan of a workflow execution (left y-axis). The right y-axis represents the step response of the controller input value (black line) during the workflow execution. This figure shows the step response function of a controller attached to a standard cluster (32 cores, 64GB RAM), which has more potential to arise memory overflows.

Table 5: Tuned gain parameters \(K_p\), \(K_i\), and \(K_d\) for both the data footprint and memory usage PID controllers. \(K_u\) and \(T_u\) are computed using the Ziegler-Nichols method, and represent the ultimate period and critical gain, respectively.

<table>
<thead>
<tr>
<th>Controller</th>
<th>(K_u)</th>
<th>(T_u)</th>
<th>(K_p)</th>
<th>(K_i)</th>
<th>(K_d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Footprint</td>
<td>0.58</td>
<td>3.18</td>
<td>0.35</td>
<td>0.22</td>
<td>0.14</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>0.53</td>
<td>12.8</td>
<td>0.32</td>
<td>0.05</td>
<td>0.51</td>
</tr>
</tbody>
</table>

6.2 Experimental Evaluation and Discussion

We have conducted runs with the tuned PID controllers for both the data footprint and memory usage. Figure 7 shows the time series of the number of tasks scheduled or preempted during the workflow executions, and the step response of the controller input value (right y-axis). The average workflow execution makespan is 386,561s, which yields a slowdown of 1.01. The average number of preempted tasks is around 18, and only a single cleanup task was used in each workflow execution. The controller step responses, for both the data footprint (Figure 7a) and the memory usage (Figure 7b), show lower peaks and troughs values during the workflow execution when compared to the PID controllers using equal weights for the gain parameters (Figures 5c and 6c, respectively). More specifically, the controller input value is reduced by 30% for the memory controller attached to a standard cluster. This behavior is attained through the ponderations provided by the tuned parameters. However, tuning the gain parameters cannot ensure that an optimal scheduling will be produced for workflow runs (mostly due to the dynamism inherent to workflow executions) as few preemptions are still triggered.

Although the Ziegler-Nichols method provides quasi-optimal workflow executions (for the workflow studied in this paper), the key factor of its success is due to the specialization of the controllers to a single application. In production systems, such methodology may not be realistic because of the variety of applications running by different users—deploying a PID controller per application and per component (e.g., disk, memory, network, etc.) may significantly increase the complexity of the system and the system’s requirements. On
Figure 7: Tuning PID Controllers: Number of tasks scheduled (blue bars for positive values) and preempted (red bars for negative values) during the lifespan of a workflow execution (left y-axis). The right y-axis represents the step response of the controller input value (black line) during the workflow execution. The bottom of the figure shows the step response function of a memory controller attached to a standard cluster (32 cores, 64GB RAM), which has more potential to arise memory overflows. The average workflow makespan is 386,561s, i.e. an average application slowdown of 1.01.

the other hand, controllers may be deployed in the user’s space (or per workflow engine) to manage a small number of workflow executions. In addition, the time required to process the current state of the system and decide whether to trigger an action is nearly instantaneous, what favors the use of PID controllers on online and real-time workflow systems. More sophisticated methods (e.g., using machine learning) may provide better approaches to tune the gain parameters. However, they may also add an important overhead.

7. CONCLUSION

In this paper, we have described, evaluated, and discussed the feasibility of using simple PID controllers to prevent and mitigate faults online and under unknown conditions in workflow executions. We have addressed two common faults of today’s science applications, data storage overload and memory overflow (main issues in data-intensive workflows), as use cases to demonstrate the feasibility of the proposed approach.

Experimental results using simple defined control loops (no tuning) show that faults are detected and prevented before their occur, leading workflow execution to its completion with acceptable performance (slowdown of 1.08). The experiments also demonstrated the importance of each component in a PID controller. We then used the Ziegler-Nichols method to tune the gain parameters of the controllers (both data footprint and memory usage). Experimental results show that the control loop system produced nearly optimal scheduling—slowdown of 1.01. Therefore, we claim that the preliminary results of this work open a new avenue of research in workflow management systems.

We acknowledge that PID controllers should be used sparingly, and metrics (and actions) should be defined in a way that they do not lead the system to an inconsistent state—as observed in this paper when only the proportional component was used. Therefore, we plan to investigate the simultaneous use of multiple control loops at the application and infrastructure levels, to determine to which extent this approach may negatively impact the system. We also plan to extend our synthetic workflow generator [16] (that can produce realistic synthetic workflows based on profiles extracted from execution traces) to generate estimates of data and memory usages based on the gathered measurements.

Acknowledgments. This work was funded by DOE contract number #DESC0012636, “Panorama—Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows”. This work was carried out when Rosa Filgueira worked for the University of Edinburgh, and was funded by the Postdoctoral and Early Career Researcher Exchanges (PECE) fellowship funded by the Scottish Informatics and Computer Science Alliance (SICSA) in 2016, and the Wellcome Trust-University of Edinburgh Institutional Strategic Support Fund.

8. REFERENCES


