Enhancing Scheduling Robustness with Partial Task Completion Feedback and Resource Requirement Biasing

Nicolas Grounds and John K. Antonio
School of Computer Science, University of Oklahoma, Norman, Oklahoma, United States of America
Research Paper

Abstract— *Performance and robustness of dynamic scheduling algorithms are evaluated in the presence of errors in the tasks' resource requirements. Previous work found that incorporating task completion events from the actual distributed system into the algorithms' model of the system was crucial for achieving robustness. In the present paper, various degrees of feedback, rather than simply all-or-none, are evaluated using the same simulated studies as in previous work and a proposed strategy for biasing model tasks' resource requirement information is proposed in order to counteract the most egregious effects of model error on performance.*

Keywords: distributed system, scheduling, performance, robustness, biasing

1. Introduction and Background

Scheduling computational tasks to machines so as to improve specified metrics of performance has been the topic of a plethora of good work produced over the past several decades [1]. The underlying assumptions and objectives of this body of work varies along several dimensions. First, some work assumes all tasks are independent whereas other work, as in this paper, allows for tasks to have dependency or precedence relationships with other tasks (for which interrelated tasks are typically represented in a directed acyclic graph, or DAG). Additionally, there are static formulations to scheduling in which a desired schedule is detemined offline based on assumed knowledge related to the machines' available resources and, correspondingly, the resource requirements of the computational tasks.

This paper addresses dynamic scheduling in which the schedule for tasks is determined online in real-time with the execution of those tasks performed on a distributed system. Unlike static scheduling, dynamic scheduling does not require upfront knowledge of the arrival of future tasks into the system for scheduling. Algorithms for dynamic scheduling make use of knowledge about the tasks which are ready for scheduling and their resource requirements such as CPU and memory load as well as the resource capacities of the machines in the distributed system.

Another dimension in the taxonomy of scheduling algorithms deals with whether tasks are executed 'one at a time' on the machine to which they are assigned. In the present work, multiple tasks may be executed concurrently on a single machine. This adds complexity in modeling the machines' performance because the machines' resources must be shared across multiple tasks assigned to the machine.

Some algorithms for dynamic scheduling are based on heuristics for selecting which tasks to prioritize and determining when to begin their execution and on which machine. Other algorithms attempt to optimize scheduling decisions with respect to a desired outcome based on a user-defined objective. Both types of algorithms are generally measured and compared to one another against such objectives as minimizing makespan (time required for completed execution of all tasks) [2], or, as in this paper, the degree to which all tasks of a DAG are completed by a DAG-associated deadline.

Scheduling algorithms may orthogonally be evaluated based on their robustness, for example, how well the same objective is achieved when information provided to the scheduling algorithm contains errors such as inaccuracies in resource requirements of the tasks. In previous work [3] four dynamic scheduling algorithms' robustness to error with respect to performance against an objective of completing DAGs before their deadline was presented, showing how some algorithms were not robust to even the smallest amount of error. The use of task completion event feedback from the actual distributed system back into the modeled system used by the scheduling algorithms was found to substantially improve robustness of all four scheduling algorithms even with large amounts of error. In the present paper we generalize this approach by investigating the utility of using only partial feedback of task completion events.

The remainder of the paper is organized in the following manner. Section 2 describes the problem domain and the simulation software's modeling of a distributed system in which errors in task requirements may be present. Section 3 presents a requirements biasing approach to counteracting model error to prevent scheduling algorithms from overcommitting system resources, i.e., executing too many tasks concurrently. Section 4 presents results of simulated case studies. Finally, Section 5 summarizes the findings from these simulations and presents the conclusions of our work.

2. Problem Domain and Simulation Environment

The present work is an extension to previous work [3] in which a model-based approach to dynamically scheduling tasks from DAGs (called workflows) was introduced. Four scheduling algorithms were tested in a simulation environment [4] given a 24-hour simulated period of arriving workflows ranging in size from 5 tasks up to 800. Each workflow has a known, predetermined deadline and scheduling algorithms were evaluated in [3] based both on the number of workflows completed before their deadline and the distribution of workflow counts completed at various normalized proportions past their deadline (e.g., workflows up to 100% late relative to the amount of time between their arrival and deadline).

Scheduling algorithms are used to determine — from a given queue of tasks ready to begin executing (i.e., tasks of arrived workflows that have no precedence constraints or whose precedent tasks have all completed) — both when the task should begin executing and on which machine of the platform. Unlike some scheduling research, tasks are permitted to executed concurrently with other tasks on the same machine, which increases machines' overall load on resources such as CPU and memory and thereby slows the rate of work on each executing task. Original work in [5] details this non-linear degradation of rate of work (efficiency) due to concurrent task execution.

Figure 1 illustrates the various components and general flow of information within the model-based approach to executing and evaluating scheduling algorithms. The modeled tasks' requirements (CPU load, required number of CPU cycles, and memory load) may contain errors relative to the true values; these errors then cause the model platform to diverge from the actual platform in terms of which tasks are completed and which are still executing.

Four scheduling algorithms were studied in this and previous work. The first is First-Come, First-Served (FCFS) which prioritizes scheduling tasks from workflows that arrived earlier over tasks from workflows that arrived later. The second is Proportional-Least Laxity First which projects a finish time of a task's overarching workflow based on the rate of completion of its tasks in the past and then prioritizes tasks from workflows projected to be completed most tardy proportional to the overall size of the workflow. The final two algorithms are variants of the Cost-Minimization Scheduling Algorithm (CMSA) [6], which projects a finish time of a task's workflow (as with PLLF) and uses a cost function to assign a cost to the projected tardiness. CMSA is used with two cost functions: a quadratic cost function and sigmoid cost function.

All four studied scheduling algorithms were previously shown to be sensitive to even small amounts of error in modeled tasks [3]. Each scheduling algorithm exhibited a decreased percent of workflow completed ahead of their deadline with the smallest amount of error studied. For three of the scheduling algorithms the decrease was substantial, but relatively equal, regardless of the amount of error. For the fourth scheduling algorithm, the decrease was less severe overall and was propotionate to the amount of error. The fourth algorithm was thus declared to be somewhat robust to small amounts of error (less than 0.5%) but ultimately, like all three others, was not robust for errors of 1% or greater.

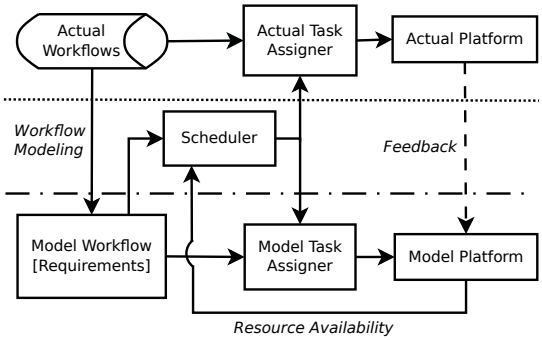

Fig. 1: Block diagram illustrating model-based framework from [3].

The main result of [3] showed that incorporating feedback of task completions from the actual platform to the model platform (thereby preventing the model platform from modeling a task completing before the actual task completed) dramatically increased robustness of all scheduling algorithms, even for errors up to 90% (the highest amount studied) in the modeled tasks' requirement for amount of work, measured in CPU cycles. Figure 2 illustrates why the model platform's underestimation of the requirements of a task (and thereby modeling it as completed ahead of the actual task) can be so detrimental.

In Figure 2 two tasks, t_1 and t_2, both have the same amount of CPU cycles, C, required for their completion. The upper chart illustrates the modeled platform's view of time in which task t_1 begins executing first and when it completed, t_2 is scheduled to immediately begin executing. However, assuming the modeled requirements of t_1 were an underestimate of the actual t_1's requirements, when the model of t_1 finishes and t_2 is scheduled to begin, the actual platform is not yet finished with t_1 and thus must work concurrently on t_1 and t_2 for some time until the actual t_1 task does complete. This causes both t_1 and t_2 to have actual completion times later than the modeled completion times

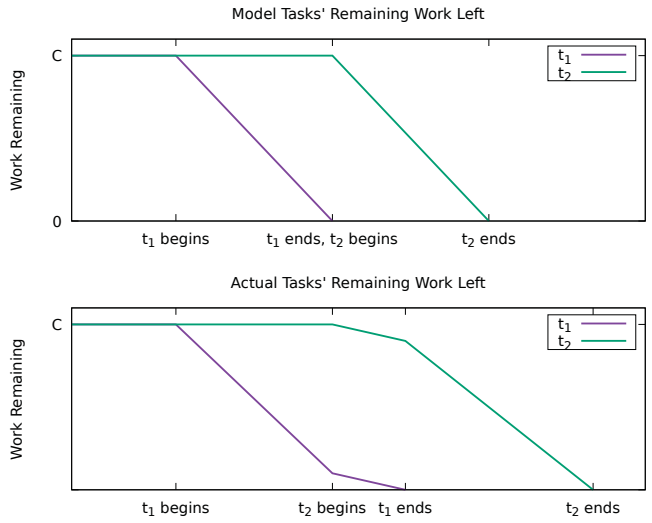

Fig. 2: Illustrating the disconnect due to error, assuming model task requirements underestimate actual task requirements and the model task finishes ahead of the actual task.

because of the unintentional over allocation of resources of the machine executing the tasks. By extension, if another task, say t_3, were to be scheduled to begin after t_2 finishes this problem could compound because t_2's actual completion would be further delayed by the additional concurrent execution with t_3.

The previous example illustrates why allowing the model platform to inform scheduling algorithms when tasks are complete and machines are idle (or less loaded) in the presense of error (which may underestimate tasks' true requirements) can be so detrimental. With feedback of every task's completion from the actual platform, the modeled completion times of tasks may be safely ignored, in favor of relying on task completion notifications from the actual platform. However, complete feedback of all tasks' completion may be impractical in a live system, or may simply be cost-prohibitive. This paper thus seeks to address the question of whether partial feedback of tasks' completion events may be sufficient to achieve some level of robustness to model error. In addition, based specifically on the knowledge of the underestimating problem illustrated prior, Section 3 proposes a specific approach to counteracting that problem by biasing model task requirements. Results of both partial feedback and biasing are presented in Section 4.

3. Biasing Model Requirements

As illustrated in Figure 2, if the model task requirements are an underestimate of the actual task's requirements then scheduling algorithms may schedule future tasks to begin executing unintentionally-concurrent with other tasks, delaying their completions. However, if the model is known to have error which may underestimate task requirements, then the model task requirements could simply be biased by increasing its assumed (provided) value in order to reduce (or ideally, eliminate) the probability that it underestimates the actual task requirements. In this section three proposed strategies for biasing task requirements are proposed.

The simplest form of biasing is to add a constant value to each task requirement. If the maximum magnitude of error for which a model task requirement may underestimate the actual task requirement is known, then that value would be the ideal bias constant value because it would eliminate the model from ever underestimating task requirements while minimizing the amount of overestimation. Practically, the maximum error magnitude is unlikely to be known. However, because it is the most ideal circumstance for biasing it is included here and in simulated results of Section 4. The equation for a simple constant bias value, C, addition to each model task requirement, $\hat{X}$, to yield a biased model task requirement, $\hat{X^b}$, is given in Eq. 1.

$$\hat{X^b} \xleftarrow{c} \hat{X} + C \tag{1}$$

A more realistic approach to biasing model task requirements is to assume a bound, not on the magnitude of error, but on the fraction (or percentage) of error. It may be possible, for example, through analysis of past executions of tasks and workflows, to estimate the maximum percentage of error for each model task requirement, $\hat{X}$, relative to the actual task requirement, X. In other words it may be practical to estimate that each model task requirement is within a factor of P of the actual task requirement. Although that implies $\hat{X}$ may under- or over-estimate X by as much as P, in order to prevent underestimating we can adjust $\hat{X}$ as in Eq. 2, hereafter referred to as the proportionate bias strategy.

$$\hat{X^b} \xleftarrow{p} \hat{X}/(1-P) \tag{2}$$

Although the proportionate bias strategy can effectively eliminate underestimated model task requirements with a judiciously chosen P which may require less knowledge about the nature of the error than choosing a proper magnitude for the constant bias strategy value, C, this relaxed requirement comes at a price. Specifically, where the constant bias strategy shifts the actual task requirement by C, the proportionate bias strategy 'stretches' the distribution of $\hat{X^b}$ and yields a much larger range of overestimates. This is illustrated in Figure 3 where the distribution of an example model value is adjusted according to the constant and proportionate bias strategies. For both adjustments, an ideal value of C and P are shown, though practically an ideal value wouldn't be known and have to be estimated itself.

In order to achieve a less 'stretched' distribution for the biased model task requirement than the proportionate bias strategy while still maintaining the need only for an estimated maximum percentage, not magnitude, of error, the third

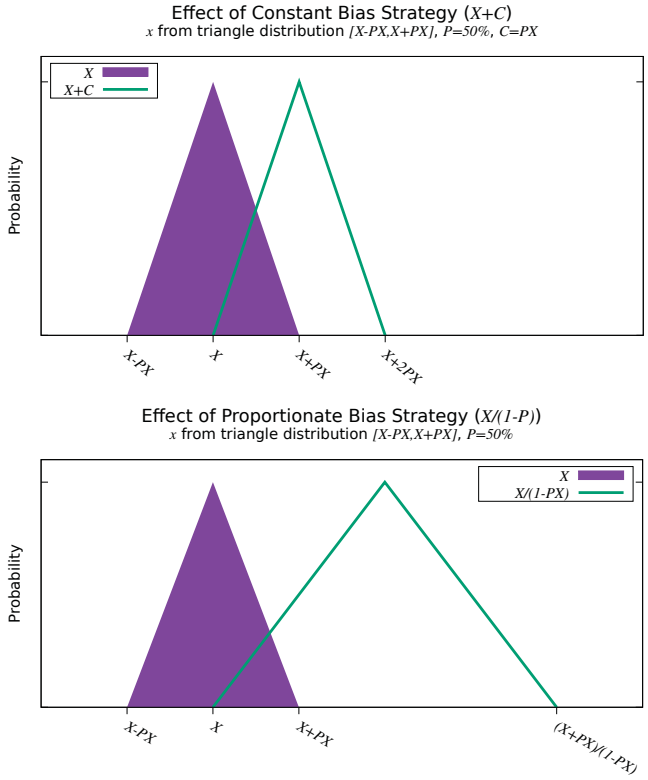

Fig. 3: Illustration of probabilities for a model task requirement with bound error as a fraction, P, of the actual task requirement, X, and the effect of constant and proportionate bias strategies transforming the distribution into one that never underestimates the actual term, X, given an ideal value for C and using the exact value of P (the bounded error) in the proportionate bias strategy's Eq. 2. Distribution is assumed to be zero-mean and triangular.

proposed strategy, known as the simple bias strategy, is given in Eq. 3. This third strategy fails to eliminate the possibility of underestimating though it can reduce its propability substantially, but also prevents wildly overestimating task requirements by decreasing the amount of 'stretch' in the biased term's distribution. Figure 4 illustrates the effect of the simple bias strategy on the distribution of the biased term.

$$\hat{X}^b \xleftarrow{s} \hat{X}(1+P) \tag{3}$$

4. Results

All results were collected using simulations performed using simulator software developed for previous research [3] and [6] and made publically available as open source [4]. Workflows and task requirements are the same as those in [3] as are simulated error amounts which ranged from 0.1% up to 50% taken from a uniform distribution. For all numeric

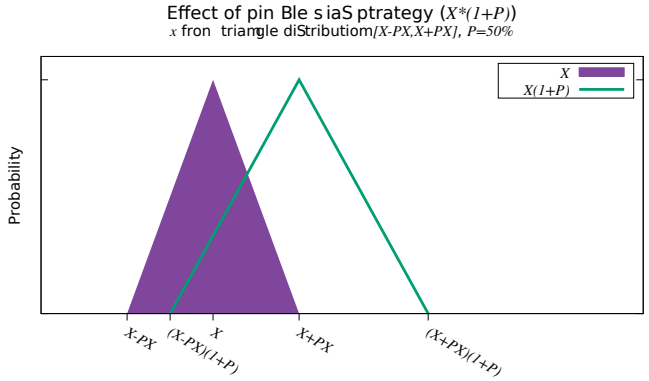

Fig. 4: Illustration of probabilities for a model task requirement with bound error as a fraction, P, of the actual task requirement, X, and the effect of the simple bias strategy transforming the distribution, given an ideal value for P. Distribution is assumed to be zero-mean and triangular.

results for performance the value presented is an averaged value across ten simulations where the workflows and tasks were identical but an error term applied to model tasks' requirements were unique.

Figure 5 shows the performance of the four scheduling algorithms and the significant performance impact that the smallest amount of error tested has even when complete feedback is available from the actual platform but the model platform is still allowed to model tasks as completing early due to underestimates of task requirements. (In other words, the model platform only utilizes actual completion events when they occur before modeled completion events.) In this figure as with prior research, performance is depicted visually as a histogram of the number of workflows completed in intervals based on their normalized tardiness (the time difference between the completion and the target deadline normalized by amount of time available to execute the workflow, i.e., the difference of deadline and arrival time of the workflow). In this representation a normalized tardiness of 0 represents a workflow that completed exactly at its deadline, negative values represent workflows completed before their deadline, and positive values those completed late.

The histogram bars of Figure 5 represent the performance of scheduling algorithms under the ideal circumstance of no error in the model (i.e., the model platform perfectly predicts and represents the resources required by a task and its execution completion time). These histogram bar results demonstrate how CMSA with either of the two cost functions (Sigmoid or Quadratic) completes the largest majorities of workflows ahead of their deadline. The PLLF (proportional least laxity first) algorithm completes workflows up to 4 times later (as a proportion of the ideal finish time) than the deadline. The FCFS (first-come, first-serve) algorithm

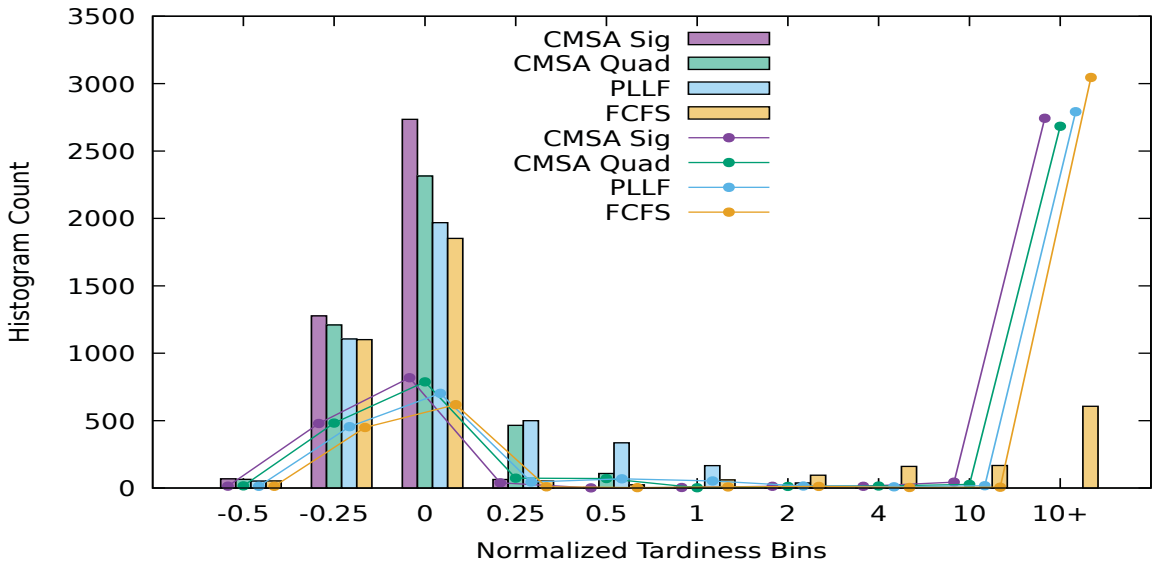

Fig. 5: Histogram of workflows by normalized tardiness comparing relative performance of scheduling algorithms with no error (vertical bars) and with 0.1% error applied to the model of the workflow requirements (line graphs). Complete feedback of task completion from the actual platform was used, but model platform was, due to underestimated task requirements, allowed to model tasks as completing and, as a result, scheduling algorithms were allowed to schedule additional tasks to the machine.

performs relatively poorly with workflows completing far later than their deadline because the algorithm doesn't use deadline information in making its scheduling decisions.

The lines graphed in Figure 5 represent the same algorithms' histogram of workflow completion in the presence of 0.1% error in the model platform. Due even to this small amount of error, the problem of underestimating task requirements and overallocating resources, and the nature of this issue compounding results in all four scheduling algorithms exhibiting substantially worse performance. This is illustrated by the reduction to less than half as many workflow completed before their deadlines (normalized tardinesses less than 0) compared to the no-error case. It is also demonstrated by the large increase of number of workflows completed with a normalized tardiness of 10 or higher compared to the no-error case.

Therefore, with any level of partial feedback available none of these scheduling algorithms would perform better than the case of complete feedback being available but still allowing modeling of early task completions due to underestimated task requirements. However, the use of model task requirement biasing showed promising results at restoring performance of the algorithms by eliminating or reducing the likelihood of the underestimating problem.

Figure 6 depicts the effect of the constant bias strategy in the present of the highest (50%) error tested with no feedback (the lines of the graph) against the no-error ideal case (the bars of the graph). Key results are that for constant bias strategy is of no help to the FCFS algorithm which completes the majority of workflows with a normalized tardiness of 10 or above. For the PLLF algorithm, nearly as many workflows are no longer completed on time as for FCFS but the maximum normalized tardiness of workflows remains bounded at about 4. Both variants of CMSA seem to similarly complete fewer overall workflows on time through for the Quadratic cost function variant those workflows seem to be completed mostly at 0.5 normalized tardiness and below while for the Sigmoid cost function variant they are spread from normalized tardinesses under 0.25 up to 10+. Although not depicted, for smaller bounded errors, the constant bias strategy does expectedly better, achieving results nearly as good as the no-error ideal case for errors up to about 5%.

Neither the proportionate nor simple bias strategies achieve acceptible results with errors as high as 50%. In Figures 7 and 8 the effect of proportionate and simple bias strategies are shown, respectively, in the case of 5% error with no feedback and ideal bias value $P = 5\%$. For the proportionate bias strategy, all four scheduling algorithms complete about one third fewer workflows at a normalized tardiness of -0.25 and below although most of those workflows are completed at normalized tardiness up to zero, which is still before their deadline. CMSA with both cost functions completes a few more workflows mildly late (normalized tardiness 0.25 and below), PLLF completes more workflows at normalized tardinesses up to 1, and FCFS a few more at 4+. Overall, the proportionate bias strategy fairly effectively makes all

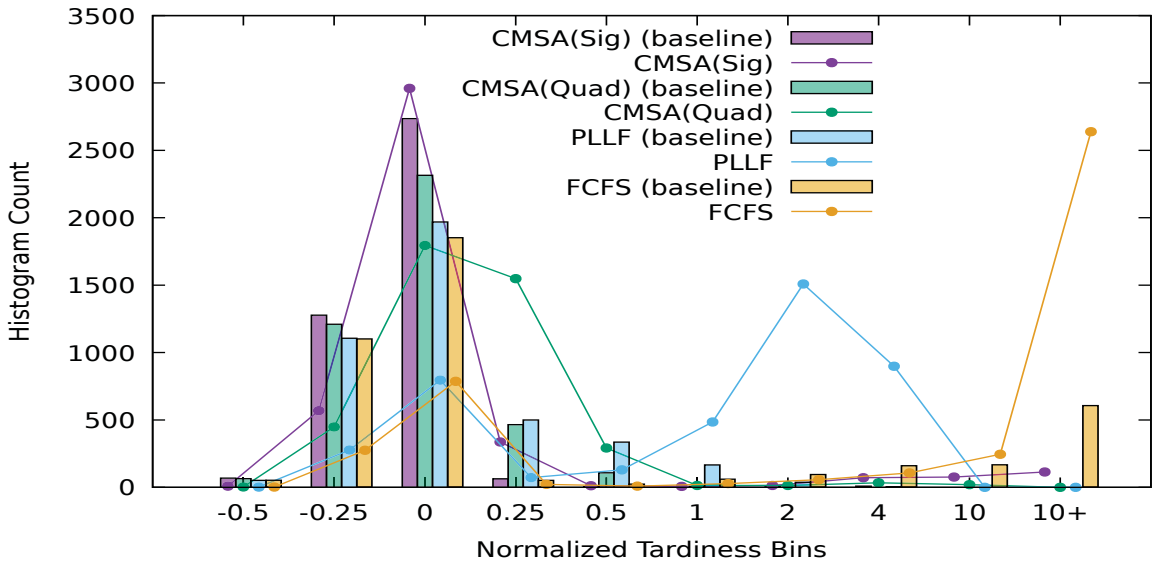

Fig. 6: Histogram of workflows by normalized tardiness comparing relative performance of scheduling algorithms in ideal case with no error (vertical bars) and with 50% error applied to the model of the workflow requirements (line graphs) given no feedback of task completion times and using the constant bias strategy with ideal constant to prevent underestimating task requirements in even the worst instance of error.

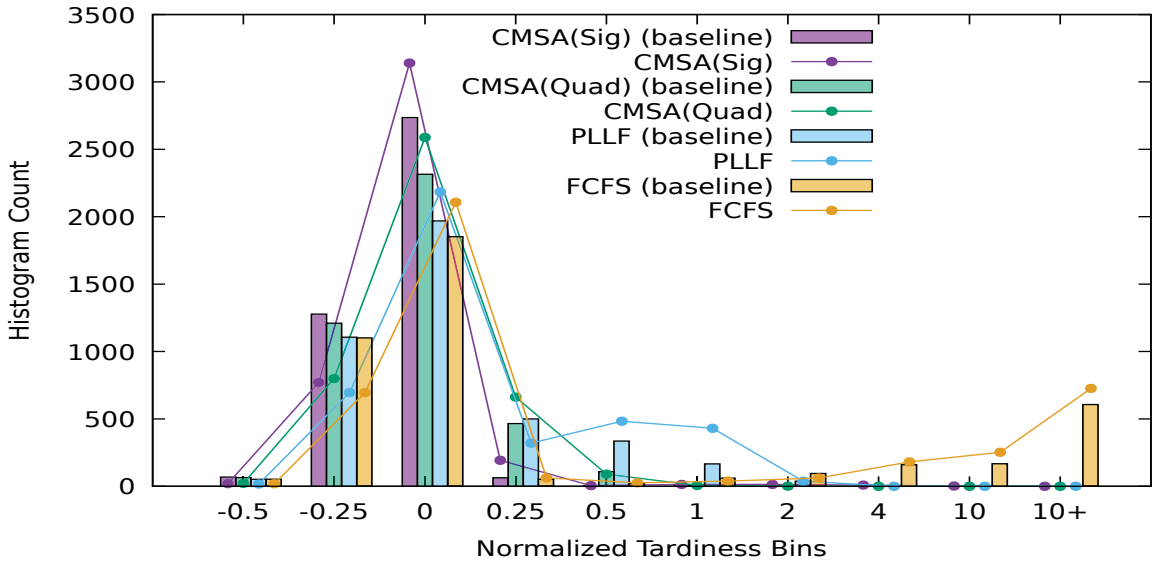

Fig. 7: Histogram of workflows by normalized tardiness comparing relative performance of scheduling algorithms in ideal case with no error (vertical bars) and with 5% error applied to the model of the workflow requirements (line graphs) given no feedback of task completion times and using the proportionate bias strategy with ideal $P = 5\%$ to prevent underestimating task requirements in even the worst instance of error.

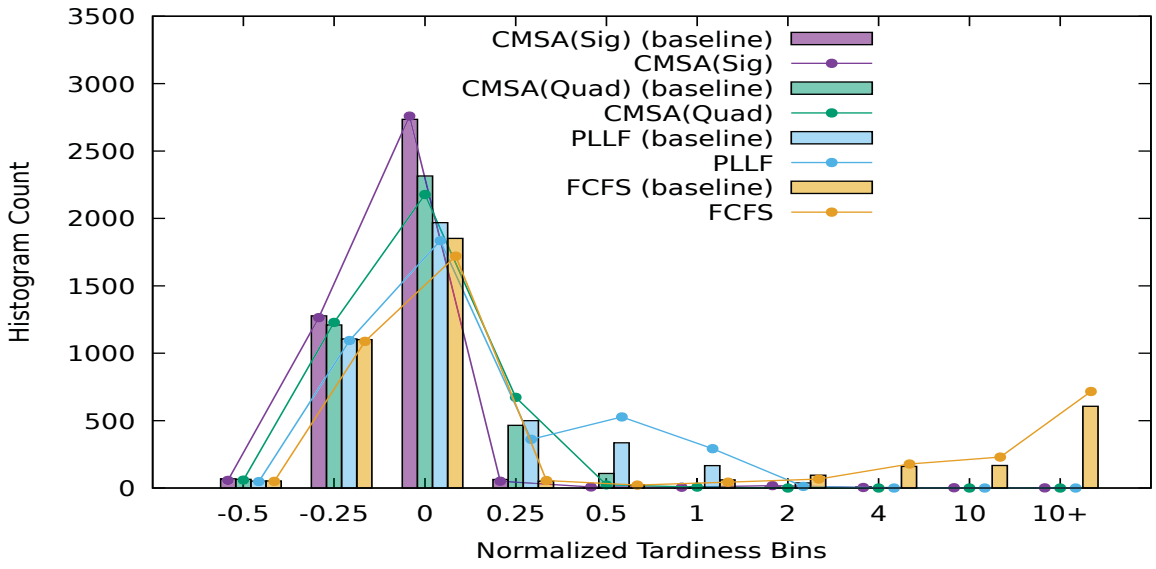

Fig. 8: Histogram of workflows by normalized tardiness comparing relative performance of scheduling algorithms in ideal case with no error (vertical bars) and with 5% error applied to the model of the workflow requirements (line graphs) given no feedback of task completion times and using the simple bias strategy with ideal $P = 5\%$ to nearly prevent underestimating task requirements in even the worst instance of error.

scheduling algorithms robust to errors up to 5% even without any actual platform feedback available.

The simple bias strategy achieves an even better result than the proportionate bias strategy for the 5% bounded error case, showing fewer workflows shifting from being completed early to completed late. In fact, the CMSA with Sigmoid cost function appears to achieve nearly identical results despite no feedback and up to 5% error when using the simple bias strategy. It therefore appears that the small chance for simple bias strategy to still underestimate task requirements is a better tradeoff than the exagerrated overestimates produced by the proportionate bias strategy.

5. Summary and Future Work

The case for why complete and reliable feedback of scheduled task completions from an actual platform to the model used for scheduling decision-making was found to be necessary to achieve robust performance of scheduling algorithms, as first described in previous work [3]. Furthermore we have demonstrated the use of partial feedback to be insufficient at achieving similar robustness, but have proposed the use of biasing the model parameters to prevent or reduce the likelihood of the model underestimating task requirements thereby modeling tasks as completing earlier than their actual completion. Three biasing strategies were proposed with unique trade-offs of assumed knowledge about error bounds and degree to which they decrease underestimation and increase overestimation of task requirements. Through simulated case studies we demonstrated that each biasing strategy is useful for achieving robustness of scheduling algorithm performance at different amounts of model error.

Future work may include combination of partial feedback with biasing strategies. Additionally, other biasing strategies or alternatives to biasing will be explored to increase robustness or address cases where the amount of error in the model is unknown.

References

[1] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Softw. Eng.*, vol. 14, no. 2, pp. 141–154, Feb. 1988. [Online]. Available: http://dx.doi.org/10.1109/32.4634

[2] M. A. Salehi, J. Smith, A. A. Maciejewski, H. J. Siegel, E. K. Chong, J. Apodaca, L. D. Briceno, T. Renner, V. Shestak, J. Ladd, *et al.*, "Stochastic-based robust dynamic resource allocation for independent tasks in a heterogeneous computing system," *Journal of Parallel and Distributed Computing*, vol. 97, pp. 96–111, 2016.

[3] N. Grounds and J. K. Antonio, "A model-based scheduling framework for enhancing robustness," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2018, pp. 10–15.

[4] N. Grounds, "SOASim: Simulator for distributed system scheduling," http://soasim.sourceforge.net/, 2010–2018.

[5] H. Shrestha, N. Grounds, J. Madden, M. Martin, J. Antonio, J. Sachs, J. Zuech, and C. Sanchez, "Scheduling workflows on a cluster of memory managed multicore machines," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2009.

[6] N. G. Grounds, J. K. Antonio, and J. Muehring, "Cost-minimizing scheduling of workflows on a cloud of memory managed multicore machines," in *Cloud Computing*, ser. Lecture Notes in Computer Science, M. Jaatun, G. Zhao, and C. Rong, Eds. Springer Berlin Heidelberg, 2009, vol. 5931, pp. 435–450. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10665-1_40