A Model-Based Scheduling Framework for Enhancing Robustness

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

Abstract— *In previous work the performance of scheduling algorithms for dynamic scheduling of tasks in a distributed system were evaluated for their robustness to error in the model of tasks' information. There it was found that incorporating task completion timings from the actual distributed system into the algorithms' model of the system was crucial for achieving robustness. In this 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' 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.

Conversely, this paper addresses dynamic scheduling in which the schedule for tasks is determined online in real-time with the execution of those tasks on a distributed system. Unlike the static scheduling problem, 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.

Such algorithms for dynamic scheduling may be based on heuristics for selecting which tasks to prioritize execution of and determining when to begin their execution and on which machine. Other algorithms may actively 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 such 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 the amount of 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 simulated error. Additionally, the use of task completion time 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 to even large amounts of simulated error.

The remainder of the paper is organized in the following manner. Section 2 describes the problem domain and the simulation software's use of modeling the distributed system where errors in task requirements may be inherent. Section 3 presents an approach to counteracting model error to prevent scheduling algorithms from over-committing system resources to executing too many tasks concurrently. Section 4 presents results of simulated case studies within that software simulator. 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 the same 24-hour simulated period of arriving workflows ranging in size from 5 tasks up to 800. Each workflow had a known, predetermined deadline and scheduling algorithms were evaluated based both on the number of workflows completed before their deadline and the distribution of workflow counts completed at various

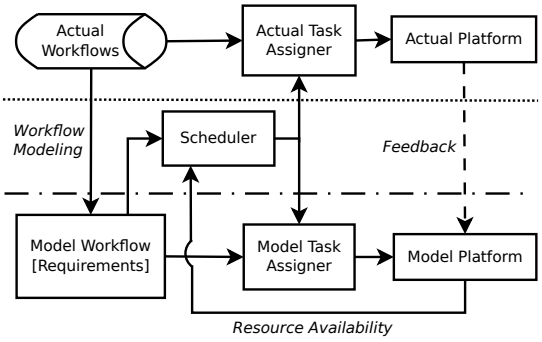

Fig. 1: Block diagram illustrating model-based framework from [3].

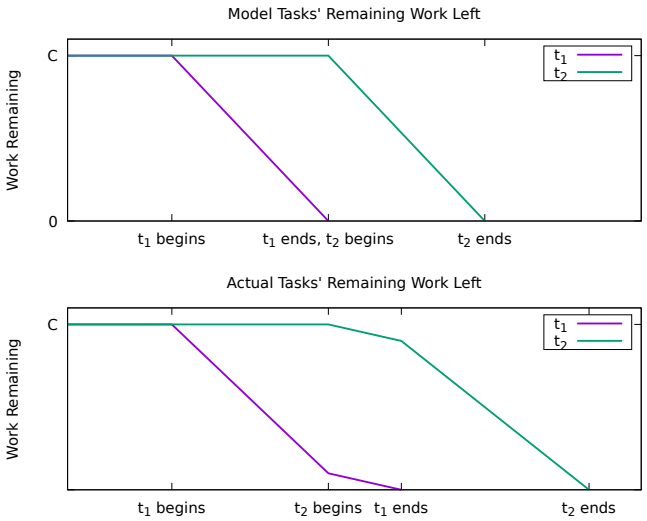

Fig. 2: Illustrating the disconnect due to error such that model task requirements underestimate actual task requirements and the model task finishes ahead of the actual task.

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 executing 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. Whereas the modeled tasks' requirements (CPU load, required number of CPU cycles, and memory load) may contain errors relative to the true values of those tasks, the model platform may diverge from the actual platform in terms of which tasks are completed and which are still executing.

All four studied scheduling algorithms were shown to be sensitive to even small amounts of error in modeled tasks. 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.

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 for even errors up to 90% (the highest amount studied). Figure 2 illustrates why the model platform underestimating 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. In the top half is 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, because 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 because of the unintentionally-overallocated 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 delayed by concurrent execution with t_1 and further delayed by concurrent execution with t_3.

The previous example illustrates why allowing the model

to 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. 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 times 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, in Section 3 that follows a specific approach to counteracting that problem by biasing model task requirements is proposed. 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 completion. 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 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 model task requirements are proposed.

The simplest form of biasing would be 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 but bound the worst overestimate to the smallest amount possible for such a constant value biasing strategy. Practically, the maximum error magnitude is unlikely to be known, though 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.

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

A more practical approach 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 $P\%$ of the actual task requirement. Although that means $\hat{X}$ may under- or overestimate 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.

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

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 increases the maximum overestimate of actual task requirement, X inherent in $\hat{X}$ 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 4 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 proposed strategy, known as the simple bias strategy, is given in Eq. 3. It fails to eliminate the possibility of underestimating though it does reduce its propability substantially, but also prevents wildly overestimating task requirements by decreasing the amount of 'stretch' in the biased term's distribution. Figure **??** illustrates the effect of the simple bias strategy on the distribution of the biased term.

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

4. Results

All results were collected using simulations performed using simulator software developed for previous research [6] and [3] 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 90% taken from a uniform distribution. For all numeric results for performance the value presented is an averaged value across ten simulations where the workflows and tasks were identical but a 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 this figure

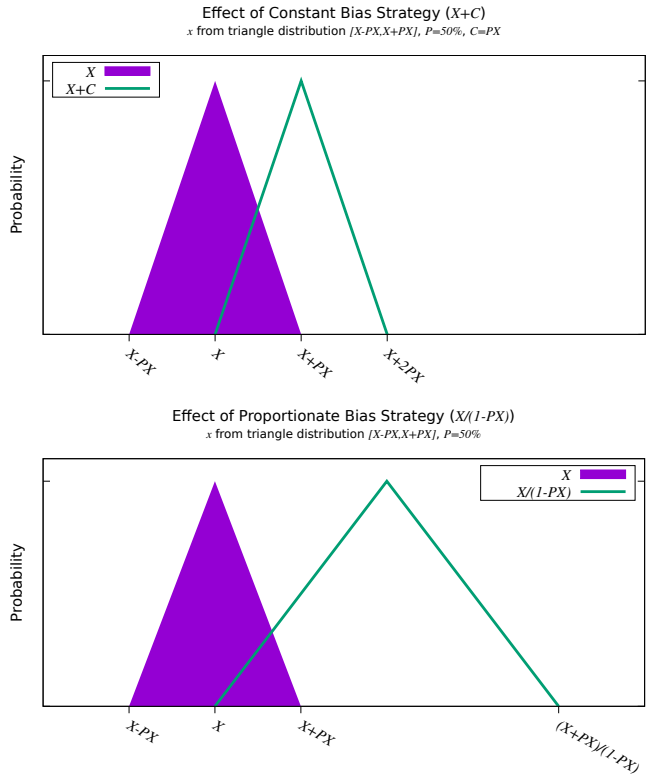

Fig. 3: Illustration of probabilities for a model task requirement with bound error as a percentage, P, of the actual task requirement, X, and the effect of constant and proportionate bias strategies transforming the distribution into one that never underestimates the actual term, X, given an ideal value for C and P. Distribution is assumed to be zero-mean and triangular.

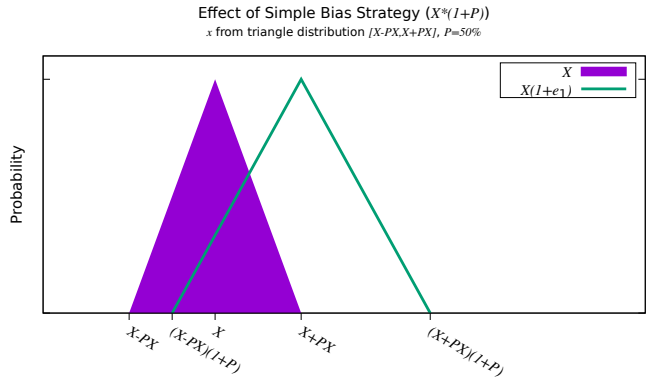

Fig. 4: Illustration of probabilities for a model task requirement with bound error as a percentage, P, of the actual task requirement, X, and the effect of the simple bias strategy transforming the distribution, given an ideal value for P. Distribution is assumed to be zero-mean and triangular.

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 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 it is unexpected that any 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.

5. Summary and Future Work

References

[1] Thomas L. Casavant and Jon G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2):141–154, February 1988.

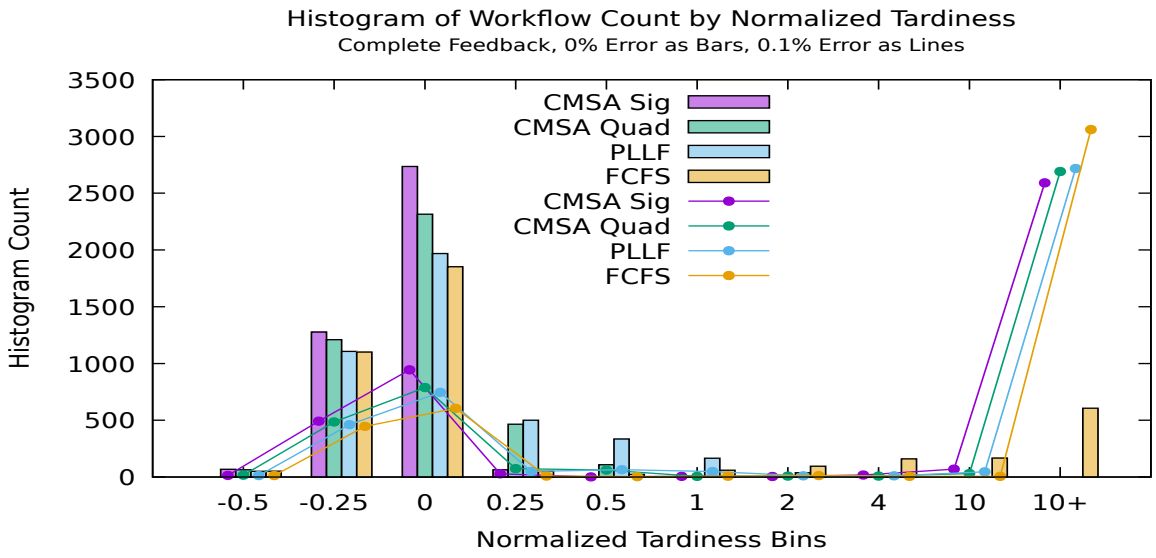

Fig. 5: Histogram of workflows by normalized tardiness comparing relative performance of scheduling algorithms with no error (vertical bars) and with 0.1% error applied to the model of the workflow requirements (line graphs). Complete feedback of task completion from the actual platform was used, but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

[2] Mohsen Salehi, Jay Smith, Anthony Maciejewski, Howard Jay Siegel, Edwin Chong, Jonathan Apodaca, Luis D. BriceÃśo, Timothy Renner, Vladimir Shestak, Joshua Ladd, Andrew Sutton, David Janovy, Sudha Govindasamy, Amin Alqudah, Rinku Dewri, and Puneet Prakash. Stochastic-based robust dynamic resource allocation for independent tasks in heterogeneous computing system. 97, 06 2016.

[3] Nicolas Grounds and John 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)*, pages 10–15. The Steering Committee of The World Congress in Computer Science, Computer âĂę, 2018.

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

[5] J. Madden M. Martin J.K. Antonio J. Sachs J. Zuech C. Sanchez H.K. Shrestha, N. Grounds. 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] Nicolas G. Grounds, John K. Antonio, and Jeff Muehring. Cost-minimizing scheduling of workflows on a cloud of memory managed multicore machines. In MartinGilje Jaatun, Gansen Zhao, and Chunming Rong, editors, *Cloud Computing*, volume 5931 of *Lecture Notes in Computer Science*, pages 435–450. Springer Berlin Heidelberg, 2009.