HAL
open science

From Simulation to Experiment: A Case Study on Multiprocessor Task Scheduling

Sascha Hunold, Henri Casanova, Frederic Suter

To cite this version:

Sascha Hunold, Henri Casanova, Frederic Suter. From Simulation to Experiment: A Case Study on Multiprocessor Task Scheduling. 13th Workshop on Advances in Parallel and Distributed Computational Models (APDCM), May 2011, Anchorage, Alaska, United States. pp.665-672. hal-00627842

HAL Id: hal-00627842

https://hal.science/hal-00627842v1

Submitted on 29 Sep 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

From Simulation to Experiment: A Case Study on Multiprocessor Task Scheduling

Sascha Hunold
CNRS / LIG Laboratory
Grenoble, France
sascha.hunold@imag.fr

Henri Casanova
Dept. of Information and Computer Sciences
University of Hawai'i at Mānoa, U.S.A.
henric@hawaii.edu

Frédéric Suter
IN2P3 Computing Center, CNRS/IN2P3
Lyon-Villeurbanne, France
Frederic.Suter@cc.in2p3.fr

***Abstract*—Simulation is a popular approach for empirically evaluating the performance of algorithms and applications in the parallel computing domain. Most published works present results without quantifying simulation error. In this work we investigate accuracy issues when simulating the execution of parallel applications. This is a broad question, and we focus on a relevant case study: the evaluation of scheduling algorithms for executing mixed-parallel applications on clusters. Most such scheduling algorithms have been evaluated in simulation only. We compare simulations to real-world experiments in a view to identify which features of a simulator are most critical for simulation accuracy. Our first finding is that simple yet popular analytical simulation models lead to simulation results that cannot be used for soundly comparing scheduling algorithms. We then show that, by contrast, simulation models instantiated based on brute-force measurements of the target execution environment lead to usable results. Finally, we develop empirical simulation models that provide a reasonable compromise between the two previous approaches.**

I. Introduction

Experimentation plays an important role in computer science, and in particular in parallel and distributed computing. They are used to prove or disprove conjectures, validate a model, or quantify the performance of a particular design under realistic conditions. Methodologies range from executing a real-world application on a real-world platform to modeling the application behavior in a simulator [1]. Simulations are not as realistic as real-world experiments, which we simply term "experiments" in this work. Simulations are attractive because they allow reproducibility of results, provide an objective basis for application comparison, and afford the capacity to explore a broad range of scenarios (some of them not tractable in experiments) in a reasonable amount of time.

Parallel and distributed applications consist of computation, communication, and possibly I/O activities. Consequently, all three must be accurately modeled by a simulator. Simulation models used in practice range from *analytical* models to *operational* models. For instance, the execution of an application task can be simulated using a simple analytical model based on computational complexity, or via execution of actual application code on a cycle-accurate simulator. Similarly, network communication can be simulated using a simple latency/bandwidth affine model of data transfer time, or using a packet-level simulator. The motivation for analytical models is clear: they can be computed quickly and scalably (e.g., a few minutes of simulation on a single computer could suffice for simulating an application that runs for several hours on hundreds of cluster nodes). Consequently, the vast majority of published simulation results in the parallel application scheduling literature are obtained with analytical simulation models. In particular, simulation tools used routinely in the community use analytical, some of which have been validated against operational models [2].

The key question we investigate in this work is whether relying on analytical simulation models leads to scientifically valid conclusions. This is a somewhat disturbing question to ask given the sheer number of published results that rely on such models. It is also a broad question that does not have a single answer since the answer depends on the applications and platforms at hand. Consequently, we focus on a relevant case study: scheduling algorithms for executing mixed-parallel applications on clusters. Like most scientific workflows, mixed-parallel applications can be represented as task graphs, but these tasks are data-parallel computations. By combining the task-parallelism offered by the workflow structure and data-parallelism within each task, mixed parallelism increases potential parallelism and can thus lead to higher scalability and performance. Mixed parallelism can be implemented in many scientific applications [3]. In recent years several algorithms have been proposed for scheduling mixed-parallel applications on clusters [4], [5], [6], [7], and most of these algorithms have been evaluated via simulations. Several of these simulations were enabled by the SimGrid toolkit [8], a state-of-the-art tool for performing discrete event simulations of distributed parallel systems using analytical simulation models. There are several other toolkits for building discrete event simulator, e.g., GridSim [9]. However, it is not the objective of this work to compare several simulation toolkits for their simulation quality. As all simulators entail an abstraction error, we are rather interested in answering if simulation results in this domain of scheduling allow the scientist to transfer the solution into a real runtime environment.

In this work, we compare SimGrid simulations with experiments on a real-world cluster in the context of mixed-parallel application scheduling. By comparing simulations to experiments, we can quantify the simulation error and, more importantly, identify its root causes. We also investigate to which extent simulation models should capture application-

and platform-specific overheads in order to improve accuracy so that a valid comparison of the scheduling algorithms is possible. We refine our simulation model accordingly, first using a brute-force approach that measures all possible overheads and later using empirical models that use interpolation for estimating overheads. From these developments we then infer guidelines for sound simulation practices.

This paper is organized as follows. Section II describes our case study. Section III presents our application execution environment and Section IV presents our simulation environment. Section V compares simulations to experiments. Section VI shows how simulation accuracy can be improved so that simulations lead to meaningful results. Section VII gives a method for obtaining a simulation model that is less involved than that in Section VI, but still produces reasonable results. Related work is discussed in Section VIII and overall conclusions are drawn in Section IX.

II. Case Study

A. Problem Statement and Algorithms

We target the scheduling of a mixed-parallel application onto a cluster with N identical compute nodes and a dedicated interconnect. The application consists of tasks with data dependencies, represented as a Directed Acyclic Graph (DAG). Each task is *moldable*, i.e., it can be executed with an arbitrary number of processors (within some bounds in practice). An MPI data-parallel application is an example of a moldable task. An example mixed-parallel application is multiple such applications in a scientific workflow.

Many algorithms to schedule mixed-parallel applications on homogeneous clusters to minimize application execution time, or makespan, have been proposed [5], [6], [7], [10], [11], [12], [13]. All these algorithms decompose the scheduling in two phases. In an *allocation* phase, they determine task allocations, i.e., the number of processors that should be used to execute each task. In a *mapping* phase, they select the respective processor set on which a task is executed. Previously published results show that the Critical Path and Area-based scheduling (CPA) algorithm [7] has a low computational complexity and leads to good results when compared to its competitors. Two extensions to CPA are used in this work: Heterogeneous CPA (HCPA) [12] and Modified CPA (MCPA) [5]. The original CPA algorithm produces task allocations that can become too large, thereby degrading overall performance. Both HCPA and MCPA remedy this problem using different solutions and thus leading to different schedules. See [5], [12] for full details on these algorithms.

B. Problem Instances

A problem instance for the scheduling consists of a platform and a mixed-parallel application. For all our experiments we use a cluster, located at the University of Bayreuth, Germany, which comprises $N = 32$ nodes (each with two 2 GHz AMD Opteron 246) interconnected with a Gigabit Ethernet switch. We had dedicated access to this cluster and could install our own application execution framework and scheduler. The vast majority of the production workflows today consist of non-moldable, purely sequential tasks. It would be possible to enhance such a workflow with mixed-parallelism, admittedly at the expense of development time and effort, and use this workflow as the target application for our case study. Instead, since our goal is to study simulation accuracy rather than obtaining novel results regarding scheduling algorithms themselves, we generate random DAGs, as described hereafter.

Table I
Parameters used for generating random DAGs.

parameter	values
number of tasks	10
number of input matrices (DAG width)	2, 4, 8
ratio addition / multiplication tasks	0.5, 0.75, 1.0
matrix size (# elements per dimension)	2,000, 3,000
number of samples	3
total DAG instances	54

In our random DAGs, tasks are either matrix additions or matrix multiplications. Both these computational kernels can be easily parallelized using standard parallel algorithms [14]. We use a vanilla 1D parallelization: if a $n \times n$ matrix of size n is mapped onto p processors, each processor holds n/p columns. For $n \times n$ square matrices, the time complexity of the addition is $\mathcal{O}(n^2)$ and that of the multiplication is $\mathcal{O}(n^3)$. We can thus easily generate applications with different computation to communication ratios (CCRs) by varying the fraction of tasks that implement matrix addition or multiplication. we have implemented matrix multiplication and addition in Java using the MPIJava (v1.2.5) library, using the `mpich2` (v1.0.8) native communication library.

Our DAG generator operates as follows. First, it randomly picks the number of entry tasks between 1 and $\log_2(v)$, where v denotes the number of input matrices. So as to generate DAGs of different *widths* (i.e., potential task parallelism), v is a parameter that we set to 2, 4, or 8. Each entry tasks operates on two matrices and produces a new matrix as output. The number of tasks in subsequent levels of the DAG is picked between one and the logarithm of the number of matrices so far, i.e., the input matrices and those produced by previous tasks. The DAG generation ends when a specified total number of tasks has been generated. We control the CCR by setting the ratio of matrix additions to matrix multiplications (e.g., a ratio of 0.2 for 10 tasks leads to 2 additions and 8 multiplications). We generate instances by setting this parameter to 0.5, 0.75, or 1.0. A fourth parameter specifies the size of the processed matrices and has an impact on the overall execution time. We consider $n \times n$ square matrices of double precision elements with $n = 2{,}000$ and $n = 3{,}000$, for 30 MB and 68 MB of data per matrix. The scheduling algorithms do not consider memory constraints, and these sizes ensure that all data will fit into the main memory of a processor. For each instantiation of the above parameters we generate three sample DAGs, for a total of 54 DAGs. Table I gives a summary of the DAG generator parameters and their values.

III. Execution Framework

We use the TGrid software for running experiments on our cluster [15]. TGrid is Java-based because initially designed for heterogeneous platforms. TGrid is ideally suited to our case study because it supports the execution of interdependent multiprocessor tasks, i.e., mixed-parallel applications. TGrid consists of a runtime environment and of a development library. The library makes it possible to implement parallel tasks with MPIJava and provides mechanisms for defining data dependencies between tasks. The runtime environment spawns and monitors tasks accordingly to a given schedule. We have implemented a module on top of TGrid to execute a mixed-parallel application according to a given schedule that specifies task execution order and which processors to use for each task.

In a mixed-parallel application, different parallel tasks may use different data distributions (e.g., 1-D, 2-D, block-cyclic). Data redistribution is thus often necessary when transferring output data from a task to its dependents. Implementing data redistribution is often difficult. Fortunately, TGrid provides a component for transparent data redistribution. This component determines the source and target processors for each data element, determines the messages that have to be transferred, and performs all necessary point-to-point data transfers.

IV. Simulation Framework

Our simulator is based on SimGrid [8], a framework specifically designed for the evaluation of scheduling algorithms. The simulator takes as input a platform specification, an application specification and a scheduling algorithm, and outputs an application execution trace.

The platform specification for our 32-node cluster is constructed as follows. Since our application execution framework is Java-based and that our application tasks are simple linear algebra kernels, we have quantified the compute speed of the cluster by benchmarking a matrix multiplication running on the JVM on our cluster. Accordingly, the compute speed of each node is set to 250 MFlop/s. The interconnection network is represented by four parameters: the bandwidths and latencies of the the cluster's switch and those of the private links connecting each node to the switch. Since our cluster uses a Gigabit Ethernet interconnect, these bandwidths are set to 1 Gb/s and the latencies to $100\,\mu$s. Note that SimGrid simulates contention between network communications that share a network link [2].

SimGrid provides a model (called *Ptask_L07*) to simulate the execution of parallel tasks. In this model, a user-provided array a describes the number of floating point operations that each processor has to execute and a user-provided matrix B describes the communication pattern (i.e., the number of bytes exchanged between each pair of processors). This model makes it possible to simulate fully parallel tasks ($a \neq 0$, $B = 0$), data redistribution tasks ($a = 0$, $B \neq 0$), and parallel tasks with communication ($a \neq 0$, $B \neq 0$). We use this model to instantiate simulation models representative of those used in the scheduling literature, as explained hereafter.

1) Modeling Task Execution Time: The traditional approach is to rely on analytical models of application execution times, which we can implement in SimGrid. Recall that our implementations of parallel tasks use 1D data distributions. Let n denote the dimension of the matrices, and p the number of processors on which a parallel task is executed. For the parallel matrix multiplication, each processor executes $2n^3/p$ floating point operations and sends n^2/p data elements per communication step. The number of operations to be performed for a parallel matrix addition is n^2/p, and no communication is needed. We use these quantities to instantiate the computation array and communication of the *Ptask_L07* parallel task model in SimGrid. Our initial experiments showed that the time to perform matrix additions is negligible. The $\mathcal{O}(n^2)$ complexity does imply a lower execution time than matrix multiplication, and in practice this time is so small that it has no effect on the overall schedules. For this reason we artificially increase the complexity of matrix additions by repeating them several times. More precisely, each matrix addition is executed $n/4$ times, leading to a total of $\frac{n}{4} \cdot \frac{n^2}{p}$ operations. Even after this adjustment, there is still a factor 8 between the number of floating point operations requires for matrix multiplications and matrix additions. Consequently, our additions and multiplication tasks still have significantly different CCRs. All results hereafter are obtained with this adjustment.

2) Modeling Data Redistribution Time: To model the data redistribution time between subsequent tasks, we create data distribution tasks defined in the *Ptask_L07* model by a communication matrix that contains the number of bytes exchanged by each processor pair. Our use of 1D data distributions for all our parallel tasks allows us to determine exactly the content of this communication matrix by computing the overlapping intervals between two matrix distributions.

V. Simulations vs. Experiments

A. Methodology

As described in Section II-B, we generate 54 random application DAGs of matrix addition and multiplication parallel tasks. Each such DAG is passed to our simulator, along with a scheduling algorithm and the platform specification. The simulator outputs the computed schedule and the (simulated) application makespan. The computed schedule specifies the order in which the tasks must be executed as well as the processors used for each task. We then execute the application on the physical cluster following this schedule, and measure the achieved makespan. We can then compare the simulated makespan to the experimental makespan. We present results of this comparison for all random DAGS, both for the HCPA and MCPA scheduling algorithms, and attempt to answer the question: can simulation results be used to predict real trends regarding which algorithm is preferable?

B. Results

Figure 1 shows results for all DAGs with input matrices of size $n = 2{,}000$. Each point on the x-axis corresponds to one specific DAG. The y-axis shows the makespan achieved by

HCPA relative to that achieved by MCPA, in percentage (a negative value means that HCPA leads to a shorter makespan than MCPA). For each DAG, two bars are shown. The black bar is for simulation results and the white bar is for experimental results. We sorted the DAGs on the x-axis by increasing relative simulated makespan to make the figure easier to read.

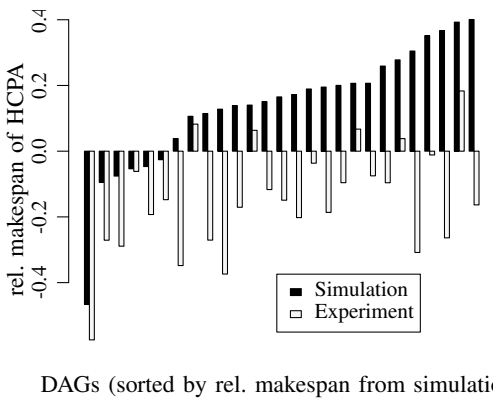

Figure 1. HPCA makespan relative to that of MCPA using analytical models ($n = 2{,}000$).

Simulation results differ from experimental results significantly, to the point that simulation results simply cannot be used to predict the relative performance of the two scheduling algorithms. For 16 out of the 27 DAGs shown in the figure, or 60%, relying on simulations to compare HCPA and MCPA lead to a result that is the opposite of the experimental result. For larger matrices, n =3,000, simulations lead to erroneous comparisons between HCPA and MCPA in 7 out of the 27 DAGs, or 26% (results not included here). We conclude that our simulator simply does not produce meaningful results.

C. Analysis

When analyzing the simulated and real-world schedules, we found that some tasks have negligible execution time in simulations but not in experiments, meaning that simulated execution times are often grossly underestimated. Furthermore, data redistributions take significantly longer in experiments. We have isolated three causes for these discrepancies:

a) Task execution time: Our task implementations do not use highly-optimized kernels, e.g., assembly or finely tuned BLAS implementations. This contributes to less predictable/-modelable task execution time, while optimized BLAS libraries often perform close to peak performance. Our Java code is often far from peak performance and turns out to be sensitive to number of processors and the size of the matrices, which is not capture by the simulated analytical models.

b) Task startup overhead: Starting a task in TGrid is expensive as it entails starting a Java Virtual Machine (JVM) on each processor via SSH. This overhead is not captured in the simulated analytical models.

c) Data redistribution overhead: In TGrid, for a process to communicate with a process from a different context (i.e., another MPI task) it must first register to a local subnet manager component and then retrieve information regarding other processes from that component. There is a single subnet manager and the time to retrieve information increases with the number of processes assigned to a task. Again, this overhead is not captured by the simulated analytical models.

The three culprits above pertain to our execution environment. Another environment would possibly obviate some of these factors or reveal new ones. Regardless, the overall message here is that, unsurprisingly, to be meaningful a simulator must account for specifics of the environment. Yet, simulators used in the scheduling literature are typically generic and can thus easily lead researchers to draw erroneous conclusions.

VI. Refining the Simulation Models

A. Task Execution Times Based on Profiles

One of the reasons why our simulator leads to inaccurate results is because it uses flawed analytical models of (parallel) task execution times. The graph on the left-hand side of Figure 2 shows the relative error of the analytical performance model for matrix multiplication when compared to our Java implementation for $n = 2{,}000$ and $n = 3{,}000$. The x-axis shows the number of processors allocated to the task. We see that the error fluctuates without clear patterns up to 60%. One may argue that this negative result is due to our use of Java. To counter this argument, we have also conducted experiments with PDGEMM, a parallel matrix multiplication implementation from the LibSci scientific library. The graph on the right-hand side of Figure 2 shows similar results obtained on a Cray XT4 (Franklin, LBNL) for three matrix sizes (1024, 2048, and 4096). The analytical model is $\frac{2n^3}{p} \cdot \frac{1}{FLOPS}$, where $FLOPS$ is the flop rate measured on the machine, which is at 4165.3 MFLOPS. The average prediction error oscillates at about 10% and goes up to 20%.

We conclude that, even for a task as simple as parallel matrix multiplication, task execution time is not easily captured by a simple analytical model. Instead, one should use "black box" simulation models that are built directly from experimental measurements. Accordingly, we modify our simulator using a brute-force approach by which we simply profile each task on our cluster for all possible allocations ($p = 1, \ldots, 32$) and matrix sizes ($n = 2{,}000, 3{,}000$). The simulator can then simulate task execution times by looking up a table of profiled execution times.

B. Task Startup Overhead

The TGrid framework spawns an MPI process by connecting to remote hosts via SSH, starting on that host a JVM and a task container, which then registers with the local TGrid server. This server sends byte code to the container, which executes it. We measured the task startup overhead for parallel tasks as the execution time of an application that consists of $p = 1, \ldots, 32$ no-op processes. Since the byte code of a real process would be larger than that for a no-op process, our measured task startup overheads are underestimations.

Figure 3 shows the startup overhead for $p = 1, \ldots, 32$, averaged over 20 trials. Surprisingly, the average startup time

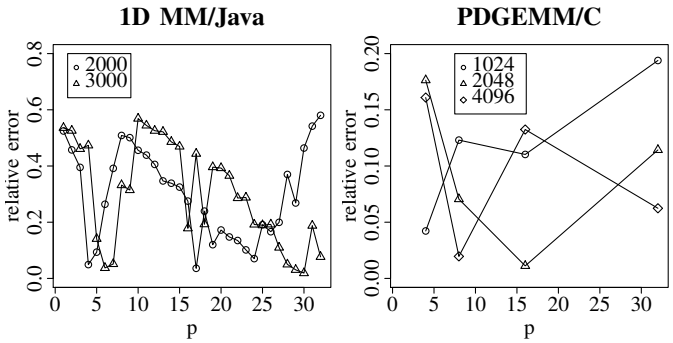

Figure 2. Relative runtime prediction errors when relying on an analytical model: 1D matrix multiplication in Java (on the left-hand side) and PDGEMM matrix multiplication in C (on the right-hand side).

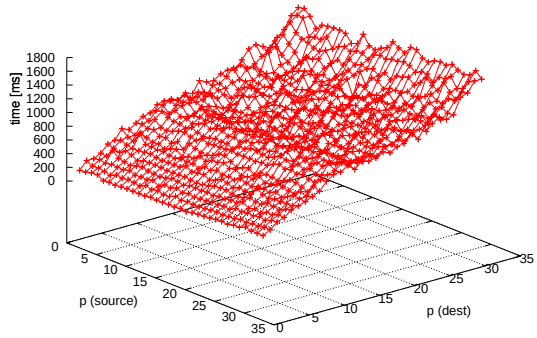

Figure 4. Data redistribution overhead. *p(src)*: number of source processors. *p(dst)*: number of destination processors.

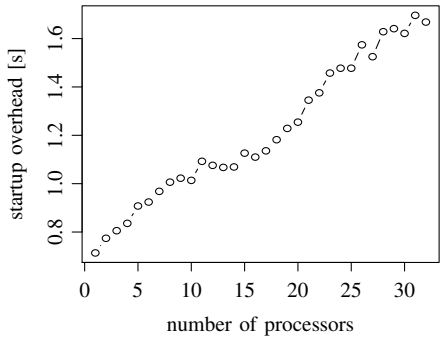

Figure 3. Task startup overhead for allocations from $p = 1$ to $p = 32$.

is not monotonically increasing with the number of processors. The simulator simulates task startup overhead by looking up a table that records these average values.

C. Redistribution Overhead

In TGrid, source and destination tasks do not know each other prior to the start of the redistribution. As explained earlier, a subnet manager component is used to establish communication between all relevant tasks, which causes overhead not captured in the original analytical model. We measure this overhead as the time to perform a data redistribution for a mostly empty matrix so that the overall data transfer times are negligible, but designed so that each processor must send at least one byte of data. This ensures that the maximum number of protocol messages will be transferred. Figure 4 shows the overhead versus the number of sending (*p(src)*) and receiving (*p(dst)*) processes, averaged over 3 trials. We see that the overhead depends mostly on *p(dst)*. We thus compute this overhead for a given *p(dst)* value averaged over all *p(src)* values. The simulator simulates data redistribution overhead by looking up a table that records these average values.

D. Experimental Evaluation

We ran all simulations for the generated DAGs with a new simulator that uses the refinements to the simulation model described in the previous three sections. The simulator now uses task profiles for simulating task execution times, with an added startup overhead depending on the size of the allocation. The time for redistributing data is still based on the SimGrid simulation, but an extra redistribution overhead is added.

Results are presented in Figure 5, which is similar to Figure 1. The chart on the left-hand side if for all DAGs with $n = 2{,}000$ increasing relative makespans obtained in simulation. The simulation outcome (i.e., whether HCPA outperforms MCPA or not) is erroneous for only two cases, and in these cases the difference between the relative makespans is well bellow 10%. For results for $n = 3{,}000$, shown on the right-hand side, the simulation outcome is erroneous in only three cases. The simulation and experimental results are in general agreement and happen to show that HCPA produces shorter schedules than MCPA for $n = 2{,}000$, while no algorithm is a clear winner for $n = 3{,}000$. We conclude that the refined simulator makes it possible to draw scientifically sound conclusions regarding the relative performance of the scheduling algorithms.

VII. Deriving an Empirical Model

The downside of our refined simulator is that it relies on extensive (and thus time-consuming) measurements of the target platform. The measurements of the task startup overheads and of the data redistribution overheads are independent on the nature of the tasks. They can thus be obtained once and for all on a given platform (and possibly scaled to simulate hypothetical target platforms). Unfortunately the measurements of task execution times are application and problem size specific. Even if, as in our case, only two computation kernels and two problem sizes are used, obtaining the complete profiles for all numbers of processors is time consuming, expensive, and ultimately not practical. In this section we attempt to derive empirical models of task execution times and of the overheads that (i) rely only on a few measurements for construction a regression; and (ii) lead to simulation results accurate enough that they can be used to draw valid conclusions.

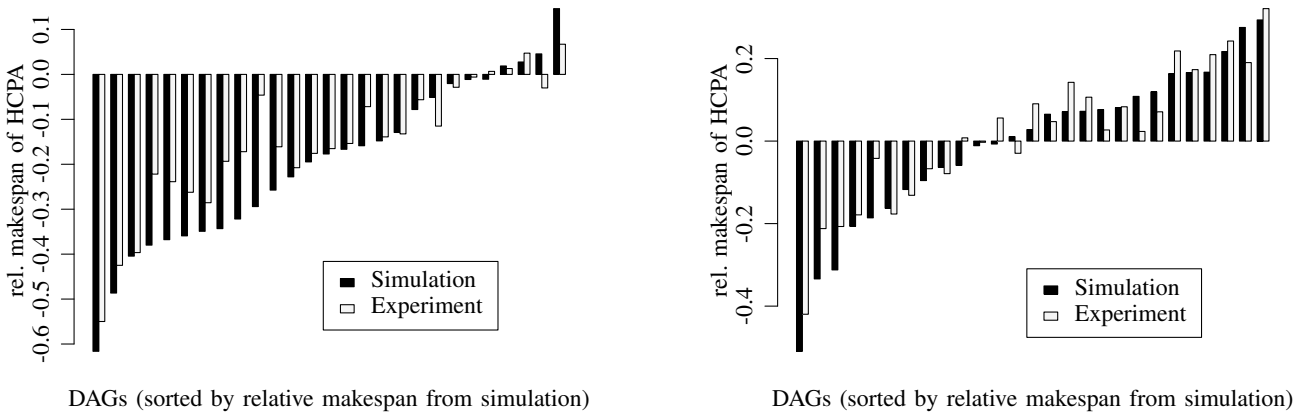

Figure 5. HPCA makespan relative to that of MCPA using full profiles. Left-hand side: $n = 2{,}000$. Right-hand side: $n = 3{,}000$.

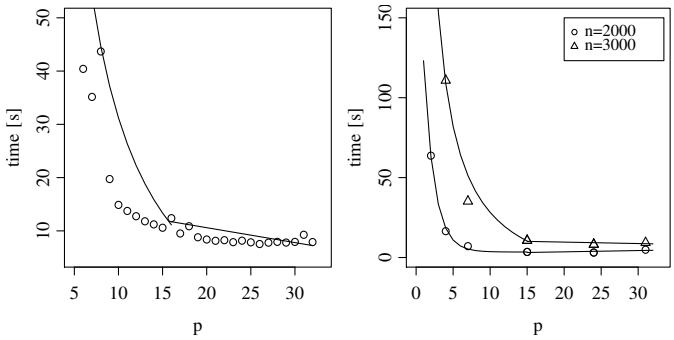

Figure 6. Fitting Results. Left: Regression model with outliers at $p = 8$ and $p = 16$ and $n = 3{,}000$. Right: Final regression model without outliers for $n = 2{,}000$ and $n = 3{,}000$.

A. Regression Models

Our goal is to obtain an regressive models of task execution times, both to matrix multiplication and addition, given the number of processors p. We initially based our regression on measured execution times for $p = \{1, 2, 4, 8, 16, 32\}$ because such powers-of-two values are commonly used in performance analysis. A single regression model does not suffice because overhead start dominating task execution times when $p \geq 16$. Consequently, we use two models: a non-linear $a \cdot 1/p + b$ model for $p \leq 16$, and a linear $a \cdot p + b$ model for $p > 16$. Unfortunately, even with these two models, the fit to experimental data is of poor quality. Consider the results shown on the left-hand side of Figure 6 for matrix multiplication and $n = 3{,}000$. The poor quality of the fit comes from unexpected "outliers" at $p = 8$ and $p = 16$. For these numbers of processors, our Java implementation suffers from abnormally long execution times. For $p = 8$, although the parallel execution is load-balanced, the computation of the local matrix updates for the multiplication are simply slower. We have observed this phenomenon repeatedly for many trials. We conclude that the likely causes are memory hierarchy effects, which are notoriously difficult to model, and especially with the JVM. For $p = 16$, the computation is not well balanced. This is due to our vanilla implementation of the 1D matrix multiplication, which leads to noticeable load imbalance for $n = 3000$ and $p = 16$ processors (the last processor is simply allocated two many matrix rows/columns). This outlier is thus less "interesting" and would not be present with a better matrix multiplication implementation. Both these outliers highlight the difficulty of deriving high-quality models of parallel task execution times in practice.

To side-step the outlier problem for now, we have used different data points for building our regression model (replacing 8 and 16 by 7 and 15). In practice, one could address this problem by obtaining a larger number of measurements for the regression, and/or possibly identify outliers, still without requiring a full profile. The modified fit is shown on the right-hand side of Figure 6 both for $n = 2{,}000$ and $n = 3{,}000$. This regressive model is based on only 6 measurements as opposed to 32 measurements for the simulator in Section VI. For larger clusters one would likely need to perform more measurements in order to derive a robust model. A similar regression analysis for HPC applications on large clusters has shown that 15 to 20 measurements are needed to get robust fits [16]. Note also that for practical uses one would have to include the matrix size into the model as an independent variable, which we did not do in this case study.

Straightforward regressions can be used to model matrix additions (a single $a \cdot 1/p + b$ model), task startup overheads, and redistribution overheads. Table II provides a summary of all these empirical models.

B. Experimental Evaluation

Results with our empirical simulation models are shown in Figure 7. Simulation results match well with experimental ones, especially for $n = 2{,}000$ in which case errors on task execution time have less impact on the overall makespan. For $n = 2{,}000$, the simulation result regarding which of HCPA or MCPA achieves the lowest makespan is incorrect in only one case out of 27. For $n = 3{,}000$, the conclusion is incorrect in 6 cases. This is twice as many incorrect results as when using the brute-force simulation approach in Section VI, but still well below the 60% error rate with the purely analytical simulation models in Section IV. When analyzing schedules

Table II
REGRESSION MODELS (INCLUDING p VALUES FOR INSTANTIATION AND REGRESSION COEFFICIENTS).

time to model	functions		regression values
execution time (multiplication)	$p = \{2, 4, 7, 15\}$	$p = \{15, 24, 31\}$	
$n = 2000$	$a \cdot \frac{1}{2^p} + b$	$c \cdot p + d$	$(a, b, c, d) = (239.44, 3.43, 0.08, 1.93)$
$n = 3000$	$a \cdot \frac{1}{p} + b$	$c \cdot p + d$	$(a, b, c, d) = (537.91, -25.55, -0.09, 11.47)$
execution time (addition)	$p = \{2, 4, 7, 15, 24, 31\}$		
$n = 2000$	$a \cdot \frac{1}{p} + b$		$(a, b) = (22.99, 0.03)$
$n = 3000$	$a \cdot \frac{1}{p} + b$		$(a, b) = (73.59, 0.38)$
	$p = \{1, 16, 32\}$		
redistribution startup	$a \cdot p + b$		$(a, b) = (7.88, 108.58)$
task startup time	$a \cdot p + b$		$(a, b) = (0.03, 0.65)$

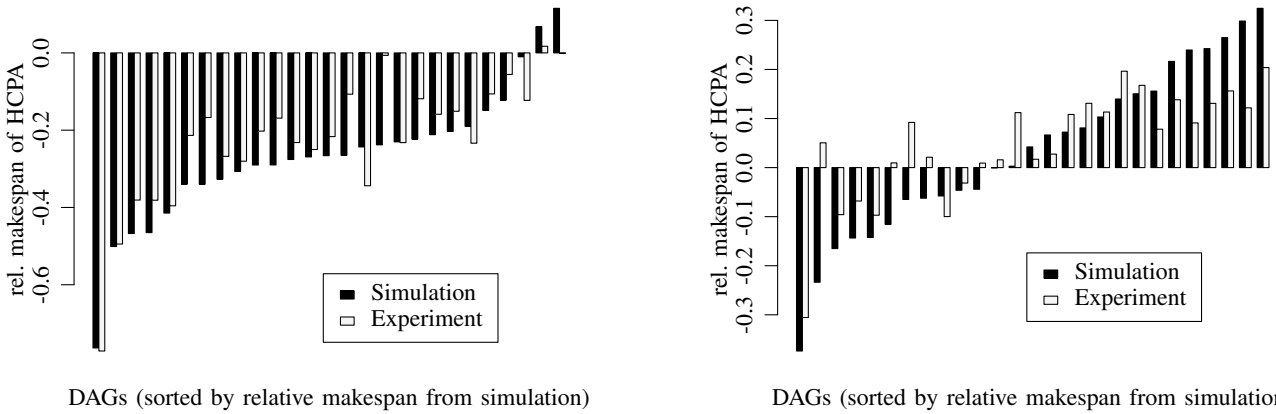

Figure 7. HPCA makespan relative to that of MCPA using empirical models. Left-hand side: $n = 2{,}000$. Right-hand side: $n = 3{,}000$.

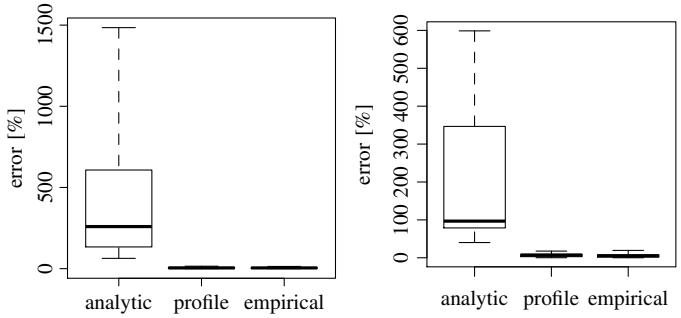

Figure 8. Makespan simulation error for the three different simulation models. Left-hand side: HCPA results. Right-hand side: MCPA results.

for $n = 3{,}000$, we found that the largest discrepancies seen on the right-hand side of Figure 7 are for cases in which a scheduling algorithms allocates $p = 16$ processors to several tasks. For this number of processors, recall that we have found our implementation of matrix multiplication to lead to unexpectedly poor performance (the outliers in Figure 6). Our regressive task execution time model in this case is a poor fit to experimental results, which explains the discrepancies.

To provide a global view of simulation errors for the three versions of our simulator (analytical, profile-based, and empirical), Figure 8 shows error statistics over all experiments for both each scheduling algorithm in a box-and-whisker fashion. The purely analytical version leads to errors larger than the two other versions by orders of magnitude, while the empirical version provides a reasonable alternative to the profile-based version. We conclude that the empirical models provide a good trade-off between the time and effort needed to instantiate them and the provided simulation accuracy. In the scope of our case study, simulation based on these models is accurate enough to draw valid conclusions regarding the HCPA and MCPA algorithms. Such conclusions would simply not have been achievable with purely analytical models.

VIII. RELATED WORK

There is a large literature on simulation techniques and models for (predicting the performance of) parallel applications. Not surprisingly, many authors have proposed simulators for MPI applications ranging from analytical simulators (e.g., [17]) to operational simulators (e.g., [18]). An overarching conclusion from such works is that reasonable simulation accuracy requires extensive and possibly application-specific measurements on a representative real-world execution environment. Yet, most published results in the scheduling literature are obtained with purely analytical simulation models. In this work we used a case study to quantify how such models should be evolved to achieve scientifically sound results. We have proposed empirical models that rely on simple regressions, which has also been done by other authors, and recently in [16]. Beyond those authors that have proposed a

simulator, some have performed studies to compare simulations to experiments. For instance, the work in [19], which also uses SimGrid, compares simulations and experiments for a heat propagation application. However, the impact of the execution environment on the simulation errors is not studied.

IX. Conclusions

Simulation is an attractive proposition for studying the performance of parallel applications, and particular popular for studying scheduling algorithms. However, since simulation errors exist and are typically not quantified, it is not clear whether simulation results lead to scientifically valid conclusions. In a case study, we have compared two state-of-the-art algorithms for scheduling mixed-parallel applications, using both simulations and real-world experiments. We have created three versions of our simulator, which are all built on top of the SimGrid framework. Thus, simulators and case study are bound to the features provided by SimGrid. The first simulator relies on purely analytical models such as those commonly used in the scheduling literature. We found that this simulator simply does not produce scientifically meaningful results. Root causes of simulation errors were identified and boil down to a need to account for specifics of the target execution environment. We thus developed a second version of the simulator based on a brute-force approach: instantiate the simulation based on extensive and possibly application specific profiles measured on the execution environment. This simulator produces dramatically better accuracy (under 10% error on average) and can be used to draw valid conclusions regarding the two scheduling algorithms. Since obtaining full profiles may be impossible in practice, we have developed a third version of our simulator based on empirical (regression-based) models built from sparse profiles. This simulator provides a good compromise between the first two versions, leading to results dramatically better than those of the first version while still enabling meaningful scientific conclusions.

Although the vast majority of published results in the scheduling literature are obtained with analytical simulation models, our findings should provide a strong recommendation for the use of empirical simulation models that are connected to the target execution environment. Alternately, these models could be instantiated for an existing execution environment and scaled to simulate an hypothetical execution environment. In our case study, idiosyncrasies of our application's implementation made it difficult to build valid empirical models due to outliers. Arguably, our use of Java and of a vanilla application implementations was the root cause for these outliers, and our results could be improved with better implementations. Indeed, others have successfully applied regression techniques for parallel application performance modeling [16]. Nevertheless, we expect that many relevant real-world applications and platforms may suffer from outlier problems and that deriving reasonable empirical models from sparse performance profiles is challenging. Addressing this challenge is, however, necessary for running valid simulations without resorting to costly complete performance profiles.

Acknowledgments

We would like to thank the University of Bayreuth and Lawrence Berkeley Labs for providing access to their machines. This work is partially supported by the ANR project USS SimGrid (08-ANR-SEGI-022).

References

[1] J. Gustedt, E. Jeannot, and M. Quinson, "Experimental Methodologies for Large-Scale Systems: A Survey," *Parallel Processing Letters*, vol. 19, no. 3, pp. 399–418, Sep. 2009.

[2] P. Velho and A. Legrand, "Accuracy Study and Improvement of Network Simulation in the SimGrid Framework," in *2nd Int. Conf. on Simulation Tools and Techniques (SimuTools)*, Rome, Italy, March 2009.

[3] S. Chakrabarti, J. Demmel, and K. Yelick, "Modeling the Benefits of Mixed Data and Task Parallelism," in *Symposium on Parallel Algorithms and Architectures (SPAA'95)*, 1995, pp. 74–83.

[4] N. Vydyanathan, S. Krishnamoorthy, G. Sabin, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz, "An Integrated Approach for Processor Allocation and Scheduling of Mixed-Parallel Applications," in *Proc. of the 35th Int. Conf. on Parallel Processing*, Aug. 2006, pp. 443–450.

[5] S. Bansal, P. Kumar, and K. Singh, "An Improved Two-Step Algorithm for Task and Data Parallel Scheduling in Distributed Memory Machines," *Parallel Computing*, vol. 32, no. 10, pp. 759–774, 2006.

[6] S. Hunold, T. Rauber, and F. Suter, "Redistribution Aware Two-Step Scheduling for Mixed-Parallel Applications," in *Proc. of the IEEE Int. Conf. on Cluster Computing*, Tsukuba, Japan, Sep. 2008, pp. 50 – 58.

[7] A. Radulescu and A. van Gemund, "A Low-Cost Approach towards Mixed Task and Data Parallel Scheduling," in *Proc. of the 15th Int. Conf. on Parallel Processing*, Valencia, Spain, Sep. 2001.

[8] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a Generic Framework for Large-Scale Distributed Experiments," in *Proc. of the 10th IEEE Int. Conf. on Computer Modeling and Simulation*, Mar. 2008.

[9] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13, pp. 1175–1220, 2002.

[10] R. Lepère, D. Trystram, and G. Woeginger, "Approximation Algorithms for Scheduling Malleable Tasks Under Precedence Constraints," *Int. J. on Foundations of Computer Science*, vol. 13, no. 4, pp. 613–627, 2002.

[11] J. Turek, J. Wolf, and P. Yu, "Approximate Algorithms for Scheduling Parallelizable Tasks," in *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA, Jun. 1992, pp. 323–332.

[12] T. N'takpé, F. Suter, and H. Casanova, "A Comparison of Scheduling Approaches for Mixed-Parallel Applications on Heterogeneous Platforms," in *Proc of the 6th Int. Symp. on Parallel and Distributed Computing*, Hagenberg, Austria, Jul. 2007.

[13] S. Hunold, "Low-Cost Tuning of Two-Step Algorithms for Scheduling Mixed-Parallel Applications onto Homogeneous Clusters," in *Proc. of the 10th Int. Symp. on Cluster, Cloud and Grid Computing*, 2010.

[14] H. Casanova, A. Legrand, and Y. Robert, *Parallel Algorithms*. Chapman and Hall, 2008.

[15] S. Hunold, T. Rauber, and G. Rünger, "TGrid – Grid Runtime Support for Hierarchically Structured Task-parallel Programs," in *Proc. of the 5th Int. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (Heteropar'06)*, 2006.

[16] W. Pfeiffer and N. Wright, "Modeling and Predicting Application Performance on Parallel Computers Using HPC Challenge Benchmarks," in *Proc. of the Int. Parallel and Dist. Processing Symp.*, 2008, pp. 1–12.

[17] T. Hoefler, C. Siebert, and A. Lumsdaine, "LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model," in *Proc. of the ACM Workshop on Large-Scale System and Application Performance*, 2010.

[18] E. León, R. Riesen, and A. Maccabe, "Instruction-Level Simulation of a Cluster at Scale," in *Proc. of the Int. Conf. for High Performance Computing and Communications*, Nov. 2009.

[19] A. Guermouche and H. Renard, "A First Step to the Evaluation of SimGrid in the Context of a Complex Application," in *Proc. of 19th Int. Heterogeneity in Computing Workshop*, 2010.