Performance and Accuracy Trade-offs of HPC Application Modeling and Simulation

Zhou Tong, Xin Yuan
Department of Computer Science
Florida State University
Tallahassee, Florida, USA
{tong, xyuan}@cs.fsu.edu

Scott Pakin, Michael Lang
Computer, Computational, and Stat. Sci. Div.
Los Alamos National Laboratory
Los Alamos, NM, USA
{pakin,mlang}@lanl.gov

***Abstract*—High Performance Computing (HPC) applications and systems are often studied through modeling and simulation at various granularities. As the size of HPC systems and applications and the cost of high fidelity simulation continue to grow, a good understanding of the trade-offs of the complexity and accuracy of HPC application modeling and simulation schemes can help balance the competing goals of accuracy and time. In this work, we investigate the complexity and accuracy trade-off using an MPI application modeling tool and an MPI application simulation tool. The performance and accuracy results of modeling and simulation of a large spectrum of HPC applications on three supercomputers are measured and compared. The results show that although modeling is often one to two orders of magnitude faster than simulation, it achieves within 5% of predicted application time in comparison to simulation for 85% of cases in our data set. We further enhance the modeling tool with a statistical model to predict whether simulation can yield significantly different results than modeling. The enhanced tool achieves a very high successful prediction rate of 93.2% on our dataset and is thus effective in determining whether modeling or simulation should be used.**

I. Introduction

The performance of HPC applications on contemporary supercomputing platforms is often studied through modeling and simulation. To identify performance bottlenecks and answer what-if questions, it usually takes a large number of simulations to have a comprehensive understanding. Traditional simulation tools such as BigSim [28], ROSS [3], Kojak [19], Scalasca [7], Dimemas [8], and the Structured Simulation Toolkit (SST) [1] allow application performance simulation on a target architecture at great detail.

Predicting the performance of HPC application on a large-scale system is complex as it involves the dynamic interaction of various components including the parallel algorithm, its implementation, and the underlying run-time systems, operating system, node architecture, and interconnection network. Due to these complexities, the time needed for full-system simulation is generally considered prohibitive for application-level simulations. As a result, modeling and simulation tools are designed to focus on specific components of an HPC system, such as the processor, memory subsystem, or interconnection network. Modeling and simulation tools typically rely on abstract machine models that trade accuracy for performance and scalability.

In this paper we focus primarily on the network. When analyzing network performance, modeling tools characterize the performance of an application on complex HPC systems with a short list of parameters, such as network bandwidth, latency, overhead, number of processes, message size, etc. In contrast, simulation tools represent the behavior of each network component (links, interfaces, and switches) at a fine level of detail, sometimes down to the individual queues, crossbars, and flits. Simulation tools can thereby provide a more accurate estimate of performance, especially in light of resource contention, but are more costly to run in term of both the time and computing resources required.

Based on the scope and type of research questions to answer, either the modeling or the simulation approach may be more suitable. For instance, detailed network simulation may be overkill for analyzing an embarrassingly parallel application, while communication-bound applications with complex communication patterns may require fine-grained simulations for precise performance tuning and optimization.

As exascale computing is approaching, the time needed for traditional one-configuration-at-a-time simulation process can be prohibitive so effectively selecting performance modeling and simulation tools to study HPC applications on current and future systems becomes imperative. An understanding of the complexity and accuracy trade-offs among modeling and simulation schemes can guide the selection of appropriate schemes for the study of applications and systems.

In this work we investigate the complexity and accuracy trade-offs between HPC application modeling and simulation schemes using a trace-driven MPI application modeling tool (MFACT) [27] and a popular MPI application simulation framework, SST/Macro [1]. Performance and accuracy results of MPI application modeling and simulation at different granularities on a wide range of MPI benchmarks are measured and compared. The results from the study show that modeling is one to two orders of magnitude faster in performance and that it can produce within 5% of predicted application time in comparison to simulation for 85% of the cases in our data set. Hence, properly using performance modeling can significantly reduce the time requirement in the study of future large scale applications and systems

without losing substantial accuracy. We enhance MFACT with a statistical model for predicting the necessity of more detailed simulation based on the application's sensitivity to network speed and other application features. The enhanced tool achieves a successful prediction rate of 93.2% on our dataset, which indicates that the statistical model is effective in deciding whether detailed simulation of an application in a platform is necessary.

The rest of the paper is structured as follows. Section II discusses the background of simulation and modeling. Section III covers the related work. Section IV describes the key features of the modeling and simulation tools used in this study. Section V presents experimental results of the performance and accuracy trade-off study. Section VI proposes a predictive model. We draw conclusions from our work in Section VII.

II. Background

A. Simulation

Simulation techniques are usually built upon a discrete-event simulation engine that models the "events" that occurred at each discrete point in time and in each component of a parallel system. Such simulation techniques can be applied to study a specific component or the parallel system as a whole at different levels of granularities. Full-system simulation, which simulates the operation of a full-scale HPC system from the hardware through an executing application, requires enormous amounts of computing resources and simulation time, but it can provide valuable performance information at great detail. In contrast, network simulations, which focus specifically on the design of the network interconnect, indicate how routing, arbitration, and congestion control utilize the available network resources and to improve the performance of parallel applications and workloads.

Network simulations can be performed at various granularities of increasing complexity: flow-level, packet-level, flit-level, or cycle-accurate. Flit-level and cycle-accurate simulations can model specific component such as switch micro-architecture with high confidence, but they do not scale well and are computationally prohibitive for large-scale application simulation. In packet-level simulation, each message is modeled as a series of small packets that are routed individually through the interconnection network subject to routing, arbitration, and congestion-avoidance schemes. Because packet-level simulation is based on larger packets (hundreds of bytes to tens of kilobytes) instead of smaller flits (a few bytes) as in a real system, it overestimates the serialization latency; each packet requires the exclusive reservation of channel bandwidth, introducing a certain degree of inaccuracy in the modeling process.

In flow-level simulations, messages traverse the network as a fluid, sharing link bandwidth among competing flows. In the absence of network congestion, a flow requires tracking only two events—start and finish. With congestion, each flow must be updated whenever network load changes. It is therefore subject to the "ripple effect" as flow updates propagate throughout the system [16]. Because packet-level and flow-level simulations are more scalable solutions with moderately accurate prediction, they are commonly used for application performance studies. In addition, hybrid packet-flow simulations enable channel multiplexing on fixed-sized, coarse-grained packets, thus avoiding the overestimation issues as in packet-level simulations and flow-update propagation issues as in flow-level simulations.

Performance analysis and simulation of message-passing applications are studied through on-line and off-line simulation techniques. Depending on the scope and the target of the analysis, communication events, computation events, or both can be considered. On-line simulation usually relies on application or messaging-library instrumentation so that the application can be simulated as if executed on a target system. Off-line simulation, on the other hand, requires only the collection of communication traces from an application's execution. The traces are then replayed to simulate the communication. Common trace files consist of the physical execution time and metadata of each communication and/or computation event plus overall application characteristics. For example, in DUMPI format [23], the entry and exit time of each MPI communication event along with its message size, data type, source and destination process ID(s) are recorded.

B. Modeling

Modeling captures the characteristics of parallel execution using a set of parameters such as data size, communication pattern, memory bandwidth, number of processes, message size, network bandwidth, latency, overhead, etc. Modeling techniques can offer quick performance evaluation of different parallel algorithms, but cannot represent the complex interactions among different modules in HPC systems or competing resource requests, unlike simulation techniques. Simulation is therefore the more appropriate tool for precise hardware analysis and application performance tuning but at the cost of substantially greater execution times. In short, both simulation and modeling serve a role in evaluating application, architecture, and hardware designs.

C. Practical considerations

This work focuses on understanding the trade-offs of modeling and simulation with respect to predicting HPC application performance. The results must thus be interpreted within the bigger picture of performance analysis with modeling and simulation in general. Under practical considerations, simulation and modeling each has its own strengths as mentioned earlier. For example, to study detailed hardware features such as systems with different injection capability, simulation is a better approach as it

does not require the understanding of the impact of the features. Similarly, to study the scenarios that models are difficult to develop such as the inter-job interference in a multi-job environment, simulation is a better choice (unless new interference models can be developed). On the other hand, modeling can be effective when new systems can be reasonably modeled. For example, to explore disruptive or significant different systems such as a cluster with a 10x faster network and 100x faster compute speed, modeling can give the prediction results for the large design space quickly as demonstrated in [27]. Finally, there is a question whether performance analysis of current applications is sufficient for evaluating future systems. We note that workload generation is a separate issue from performance analysis. If techniques for generating workload for future applications are available, the results from this paper can be extended to the future workloads.

III. Related Work

Due to the importance of performance analysis for HPC applications, various tools have been developed to collect traces and to model or simulate the applications. Example tracing tools include DUMPI [23], Intel Trace Analyzer and Collector [11], and ScalaTrace [20]. Example simulation and modeling tools include DIMEMAS [8], [9], Kojak [19], Scalasca [7], SST [12], eBigSim [28], PSINS [25], SIMGrid [5], and MFACT [27].

The accuracy of modeling and simulation of parallel applications and algorithms has been compared in different settings. Martinez et el. [17] developed automatic model selection techniques to predict the execution time of collective algorithms for a particular system with an average error of 5.0% as compared to 15.0%-20.0% for the theoretical LogGP-based models [6]. Tikir et al. [25] evaluated three benchmarks HYCOMM, AVUS, and ICEPIC using the modeling tool PSINS with the average absolute prediction errors of 7.4% and 11.6% on two target systems respectively. In addition, Casanova et al. [4] developed a time-independent trace replay framework that models application performance solely by the volumes of computation and communication using flow-level network models in SIMGrid [5]. The prediction errors for NAS's EP, DT, LU and CG benchmarks are 1.4%, 6.3%, 10.2%. 30.0% respectively. Nunez et el. [21] showed that their simulation tool achieved an average of prediction error within 6% for BIPS3d with up to 512 nodes. These existing studies mostly focus on the modeling and simulation accuracy for some particular HPC applications. While similar accuracy has been observed in our study, this work focuses on quantifying the accuracy and complexity trade-offs in modeling and simulation techniques for a broader range of HPC applications.

IV. Modeling and Simulation Tools

To study the complexity and accuracy trade-offs between application modeling and simulation, we consider a wide range of MPI applications. We evaluate the performance and accuracy of an MPI application modeling tool, MFACT [27], and an MPI application simulation tool, SST/Macro [1]. Both tools are trace-driven and replay the traces of the application. MFACT models the interconnect using Hockney's model [10] while SST/Macro simulates the network at flow, packet, and packet-flow levels.

A. MFACT

MFACT (MPI Fast Application Classification Tool) [27] models application performance based on DUMPI [23] traces. It classifies an MPI application as computation-bound, load-imbalance-bound, bandwidth-bound, latency-bound, or communication-bound for many network configurations from a single replay of the corresponding DUMPI trace. The classification is based on observing the performance sensitivity of the application to the changes in bandwidth and/or latency over a range of network configurations.

MFACT is implemented using MPI with a 1:1 ratio of MFACT processes to application processes represented by the trace. MFACT employs an extension of Lamport's logical-clock scheme [13], augmented with non-unit computation and communication times to track application progress consistently with real time. In the trace replay, timestamps are transmitted instead of real messages. This enables the trace replay to honor the logical happened-before relationships among communication operations while modeling both computation and communication events.

In MFACT, the communication subsystem is characterized by two parameters: latency and bandwidth. MFACT reads the timestamps and communication metadata from input traces and models communication with Hockney's model [10] for point-to-point communication and Thakur and Gropp's models [24] for collectives. Computation events are based on measurements from the input traces but can be scaled arbitrarily to model systems of various processing speeds. One unique feature of MFACT is that it is designed to predict application performance on numerous network configurations from a single trace replay. For each target network configuration, one logical clock is maintained. During trace replay, logical clocks for different network configurations are maintained concurrently.

MFACT maintains four logical time counters (*wait*, *bandwidth*, *latency*, and *computation*) for each network configuration. It captures an application's performance characteristics by observing how these four counters react to the speed-up and slow-down of latency, bandwidth, and computation. Based on the performance trend over a range of network configurations, MFACT predicts the performance of the application on each configuration and gauges the potential

benefits of various networking options and predicts potential application performance bottlenecks.

B. SST/Macro

SST/Macro [1], the macro-scale component of SST, adopts a conservative PDES (Parallel Discrete Event Simulation) engine and supports various abstract machine models and interconnect topologies such as tori, dragonflies, and fat trees. SST/Macro 3.0 supports packet-level and flow-level simulations, while SST/Macro 6.1 supports a hybrid packet-flow model. Unlike traditional packet-level simulations, the packet-flow model supports channel multiplexing, thus avoiding the overestimation stemming from serialization latency and exclusive channel reservation. When multiple packets compete for a network channel, each packet "samples" the congestion and estimates the expected network delay. The SST/Macro developers recommend a packet between 1KB and 8KB, increasing scalability at a minor cost in simulation accuracy. Computation events can be modeled by heuristic models or scaling of the observed timestamps in the traces. Thus, the execution time of SST/Macro's packet-flow model is proportional to the number of packets generated and delivered through the network.

Unlike MFACT, SST/Macro can model network contention. SST/Macro performs routing, arbitration and congestion control on coarse-grained packet or message flow, not as on flit-by-flit basis as in a real network. Thus, SST/Macro is more accurate than MFACT. In this work, one of our goals is to understand the performance and accuracy trade-offs between application modeling and simulation techniques.

V. Performance and Accuracy Trade-offs

In this section, we discuss the application traces and machine configurations we use in our study. We then report our results and discuss the performance and accuracy trade-offs between the modeling and simulation tools.

A. Experimental setup

We utilize communication traces from a diverse set of MPI-based parallel applications running on various network topologies and technologies. The programs include extracted kernels, mini-apps, and full-sized applications from DOE's DesignForward project.[1] The extracted kernels are Big FFT and Crystal Router (CR); the mini-apps are AMG and MiniFE; and the full-sized applications are MultiGrid (MG) and FillBoundary (FB). We also include four mini-apps—LULESH, CNS, CMC, and Nekbone—from DOE's EXMATEX[2], CESAR[3] and EXACT[4] co-design centers. Additionally, we collected DUMPI traces from eight NAS Parallel Benchmarks(NPB)[5] on Cielito, a 64-node (1024-core) Cray XE6 system and on Mustang at Los Alamos National Laboratory (LANL). These programs with different problem sizes and different number of ranks result in a total of 235 sets of application traces that are used in this study.

Table I: Characteristics of the traces

a Number of ranks

Ranks	Traces
64	72
65–128	18
129–256	80
257–512	12
513–1024	37
1025–1728	16
Total	235

b Communication time

Comm. time (%)	Traces
≤5	26
5–10	30
10–20	55
20–40	54
40–60	30
>60	40
Total	235

Table I summarizes the characteristics of the traces. The distribution of the number of ranks, which ranges from 64 to 1728, is presented in Table Ia. The distribution of communication intensity—the fraction of total time spent in communication—is presented in Table Ib. As the table indicates, the set of traces represents a wide range of applications, from computation-bound applications to applications whose execution time is dominated by communication. Note that we do not cherry-pick traces for this study; the traces were collected and submitted by other researchers in different DOE laboratories to be used in the design and evaluation of future supercomputers. This set of traces represents a broad range of HPC applications, in particular, the applications used in DOE laboratories.

SST/Macro's packet, flow, and packet-flow models are able to successfully complete 216, 162, and 235 of the traces, respectively, while MFACT completes all 235 traces. SST/Macro's packet and flow models cannot process some traces since SST/Macro 3.0 is unable to handle complex MPI grouping operations and MPI multi-threading. In the study of the accuracy of the tools in Section V-C, all successful simulation and modeling results are considered. While in the study of the execution time of simulation and modeling in Section V-B, we only consider the traces when all four schemes are successful. In addition, we exclude traces with very small simulation times (less than 1 second) such as EP and DT in the NPB suite. As a result, a total number of 126 traces are used in the execution time study in Section V-B. When predicting the need for simulation in Section VI, all 235 packet-flow simulation and MFACT modeling results are used.

The traces were collected on three parallel computers: Cielito, Edison, and Hopper. The machine configurations and network latency and bandwidth values are taken from publicly available data. The specific network bandwidth and

[1] http://portal.nersc.gov/project/CAL/designforward.htm
[2] https://portal.nersc.gov/project/CAL/exmatex.htm
[3] https://portal.nersc.gov/project/CAL/cesar.htm
[4] https://portal.nersc.gov/project/CAL/exact.htm
[5] https://www.nas.nasa.gov/publications/npb.html

latency settings are {10Gbps, 2,500ns} for Cielito [26], {35Gbps, 2,575ns} for Hopper [15], and {24Gbps, 1,300ns} for Edison [14]. For SST/Macro, the machine configurations and hostmap of Hopper and Edison are included in SST/Macro releases. The machine configuration for Cielito is derived from the machine performance benchmarking data for Cielo (a much larger version of Cielito) published online [26]. All simulation results are collected on Jungla, a 64-core machine with AMD Opteron Processor 6272 @ 2.10GHz. Both SST and MFACT can utilize all cores in the system. The reported simulation times are the average of 10 simulation runs.

B. Simulation and modeling time

For each application, we record the time for SST/Macro's packet, flow, and packet-flow simulations and MFACT's modeling. Then, we rank the four times for each application and determine which approach requires the least time. MFACT's modeling technique ranks the first place for all cases. Flow and packet-flow (P-flow) models claim the second place for roughly 41% and 59% of cases respectively, while packet, flow and packet-flow models rank the third for 11%, 48% and 41% of application runs. The packet model unsurprisingly requires the longest simulation time for 89% of cases. The results show that, within SST/Macro, the hybrid packet-flow model requires less simulation time than packet and flow models and that modeling is more efficient than simulation.

Figure 1 shows the simulation time of SST/Macro's packet, flow and packet-flow models as multiples of MFACT's modeling time. The figure depicts the percentage of applications whose simulation time is no more than 10 times the modeling time, no more than 100 times the modeling time, no more than 1000 times the modeling time, and more than 1000 times more than the modeling. Form the figure, we can see that SST/Macro's packet, flow and packet-flow simulations is no more than 10 times the MFACT time for 21%, 33% and 28% respectively of the applications, and is no more than 100 times the MFACT time

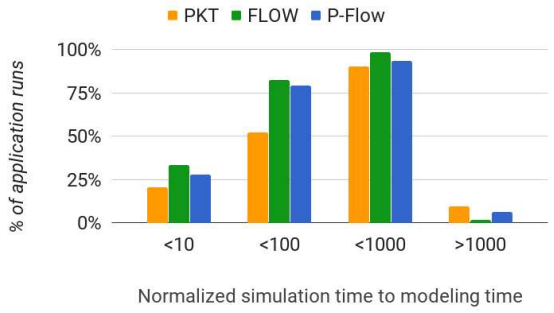


Figure 1: Execution time of SST/Macro's packet, flow, packet-flow simulations as multiples of MFACT's time

Table II: Execution time in seconds

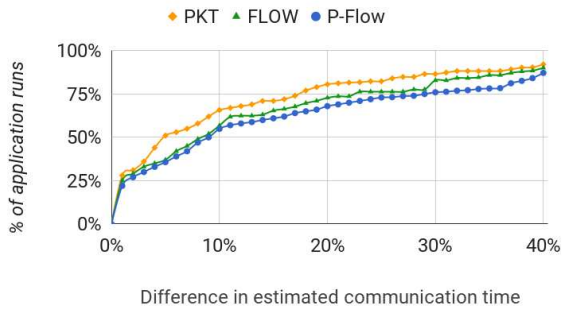
	SST/Macro.			MFACT
	Pkt	Flow	Pkt-flow	
CMC(1024)	172.17	22.45	25.94	1.26
LULESH(512)	941.77	208.63	110.27	3.02
MiniFE(1152)	1608.57	929.37	367.08	35.15

for 52%, 83%, and 79% respectively of the applications, and is no more than 1000 times the MFACT time for 90%, 98% and 94% respectively of the applications. In other words, in comparison to the packet-level simulation, modeling with MFACT is at least 10 times faster for 79% of the applications, at least 100 times faster for 48% of the applications, and at least 1000 times faster for 10% of the applications. In comparison to SST's most efficient packet-flow model, modeling with MFACT is at least 10 times faster for 72% of the applications, at least 100 times faster for 21% of the applications, and more than 1000 times faster for about 6% of the applications. These results indicate that in terms of execution time, modeling is often orders of magnitude faster than simulation. Table II lists some actual execution times of the tools for three DOE applications CMC, LULESH, and MiniFE, which depicts the typical relative executive times of the tools.

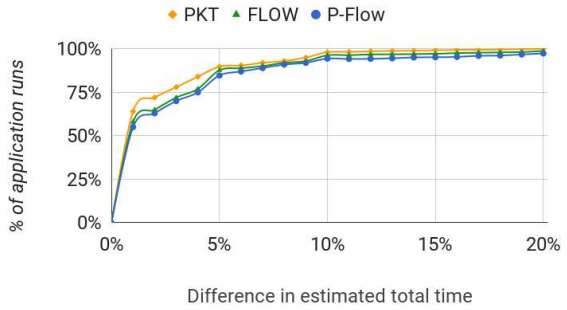
C. Accuracy of the tools

We compare the prediction results of SST/Macro with MFACT using the same task-mapping as the original application execution where the DUMPI traces are collected. The cumulative distribution plots in Figure 2 represent the difference in estimated communication and total time of the three simulation models and MFACT's modeling results. All reported simulation results in the rest of this study are normalized to the modeling result of MFACT unless noted otherwise.

In Figure 2(a), the estimated communication times for the three simulation models in SST are relatively close to one another. In approximately 90% of cases, SST's estimated communication time is within 40% of MFACT's modeled communication time. More interesting is the prediction of the overall application time, which is shown in Figure 2(b). The packet-flow model's estimated total time is within 5% of MFACT's prediction for 85% of cases. Furthermore, the difference is within 10% for 94% of application runs. The trends for packet and flow models are similar, but the values are slightly different. For example, the percentage of traces whose predicted total application time is within 10% that of MFACT are 96% and 98% for packet and flow models respectively. In the majority of the cases, the accuracy trade-off between the simulation and modeling tools is well within 20% in term of the estimated total time; and there is no significant difference in overall prediction power among three models within SST/Macro.



(a) Estimated communication time

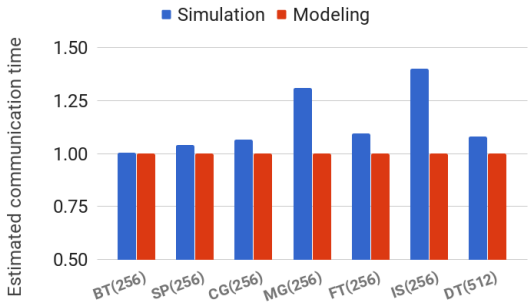


(b) Estimated total time

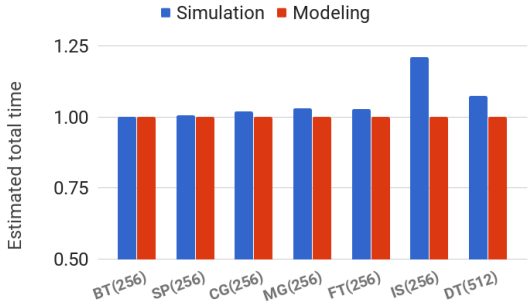
Figure 2: Simulation result of SST/Macro's packet, flow and packet-flow models

Figures 3 and 4 present the simulation accuracy trade-offs between the maximum estimated communication and total time produced by SST/Macro's packet, flow, and packet-flow simulations and MFACT's modeling results. In Figure 3(a), we observe that SST/Macro's estimated communication time is at most 10% higher for the select NAS benchmarks except that FT and IS have a difference of more than 25%. In terms of estimated total time, only IS results in a difference of more than 20% shown in Figure 3(b). There is roughly 7% difference in DT, and the rest are under 3%. In Figure 3(c), the estimated total time by the simulation and modeling tools that are normalized to the measured application time observed in the traces. From the figure, both SST/Macro and MFACT's predicted application execution time are lower than the measured time. SST/Macro has more accurately predictions. Overall, SST/Macro's estimations on average are 10.86% below the measured total time and MFACT's modeled modeling results are slightly lower of 14.83%, mostly due to IS and DT.

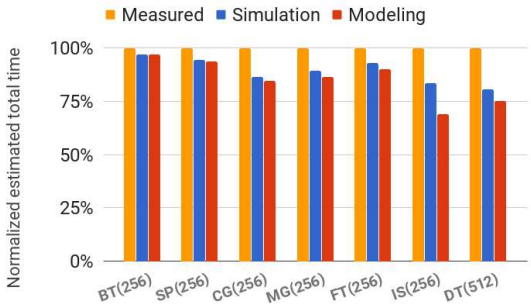
We show the simulation accuracy trade-offs of DOE's mini-apps, extracted kernels and production applications in Figure 4. Figure 4(a) demonstrates that the difference in estimated communication time is within 10% except for CR and FB due to their irregular and intensive communication patterns. In Figure 4(b), the difference in total time is within 1% for MiniFE, CMC, AMG, and LULESH and it is under



(a) Estimated communication time for NAS benchmarks



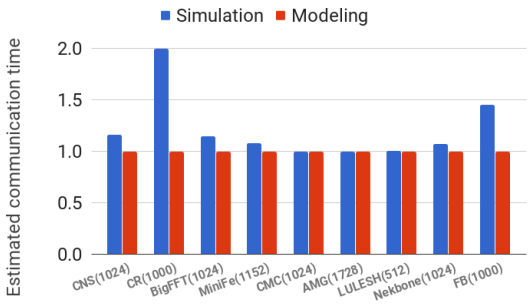
(b) Estimated total time for NAS benchmarks



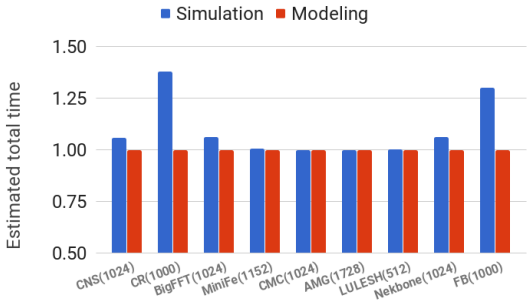
(c) Normalized estimated total time for NAS benchmarks

Figure 3: Measured, modeling and simulation results for NAS benchmarks

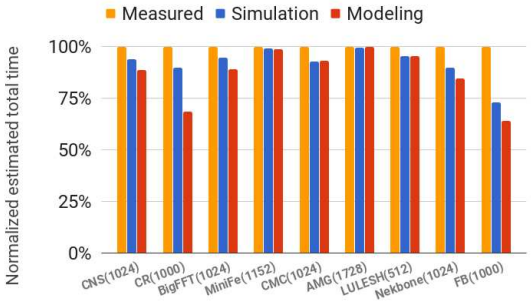
6% for CNS, BigFFT, and Nekbone. CR and FB yield larger differences of more than 20% which indicate that pursuing more detailed simulation in SST/Macro would be beneficial. However, for applications that have prediction accuracy gain of less than 1% in total time, the incentive for finer-grained simulations is very limited. In Figure 4(c), the estimated total time by the simulation and modeling tools that are normalized to the measured application time observed in the trace are presented for DOE applications. Similar to results for NAS benchmarks, SST/Macro's estimated total time is slightly closer to the measured time achieving higher accuracy. For the select DOE applications, SST/Macro's estimations on average are 7.95% below the measured total time and MFACT's modeled modeling results are slightly lower of 13.10%, mostly due to CR and FB.



(a) Estimated communication time for DOE applications



(b) Estimated total time for DOE applications



(c) Normalized estimated total time for DOE benchmarks

Figure 4: Measured, modeling and simulation results for DOE applications

Overall, modeling achieves within 5% of predicted application time in comparison to simulation for 85% of cases in our data set, but with one or two orders of magnitude less execution time. Within SST/Macro, different network models seem to yield similar prediction results. The packet-flow model has demonstrated its advantage in simulation scalability and it also allows application-level simulation with small loss in accuracy compared to packet or flow models.

VI. Predicting the Need for Simulation

Since the packet-flow model is the most robust and its prediction results are similar to the packet and flow models, we will use the packet-flow model to represent the simulation approach in this section. As shown in Figure 2(b), In 63% of the test cases, the simulation results differ from the modeling results by less than 2% in terms of total application time; in 85% of the cases by less than 5%. For the applications in which simulation yields similar results to modeling, modeling is a more effective way to study performance because it is one to two orders of magnitude faster. Characterizing such applications can significantly reduce the time to study the performance of the applications on a particular system. We will use the term $DIFF_{total} = |\frac{\text{estimated total time with SST}}{\text{estimated total time with MFACT}} - 1|$ to denote the difference between the estimated total time with simulation (SST/Macro) and modeling (MFACT). We define an application with $DIFF_{total} \leq 2\%$ to be the application that does not require simulation. Otherwise, the application requires simulation. We enhance MFACT with a statistical model to predict the necessity for application simulation. With our dataset, the enhanced MFACT has a 93.2% prediction successful rate, and thus, is an effective tool in separating MPI applications that can be sufficiently modeled by the enhanced MFACT from those that require more detailed simulation.

A. Enhanced MFACT

As discussed earlier, MFACT can classify MPI applications into computation-bound, load-imbalance-bound, bandwidth-bound, latency-bound, and communication-bound for many network configurations by replaying the DUMPI traces once; and the classification is based on observing the performance sensitivity of the application to changes in bandwidth and/or latency by examining the performance trend over a range of network configurations. We leverage MFACT's ability to characterize an MPI application's type.

MFACT's classification results strongly correlate to the type of applications. For example, applications with a high $DIFF_{total}$ tend to be sensitive to network bandwidth and latency. Applications that are classified as computation-bound or load-imbalance-bound in general have lower $DIFF_{total}$ as they are less likely being affected by network contention. The proposed statistical model exploits the intelligence of the classification results and other performance counters (some are in the original MFACT; others are added in the enhanced MFACT) so that it can automatically identify the type of MPI applications with high confidence.

To develop the statistical model, we examine MFACT classification results of 235 application runs and sort out the applications that are sensitive to network communication in general but not to load-imbalance and computation. In this study, we take a conservative approach and consider applications as *communication-sensitive* if the estimated total time increases by more than 5% as the bandwidth decreases by a factor of 8 [27]. Note that latency is not considered here as we find that very few applications show sensitivity to latency in the applications in our dataset; and communication-sensitive applications are sensitive to the

changes in network bandwidth. As a result, we group applications into communication-sensitive, computation-bound and load-imbalance-bound based on their performance predictions.

Out of the 235 applications, 70 are classified as computation-bound, 63 are load-imbalanced-bound, and 102 are considered communication-sensitive. Figure 5 shows the absolute $DIFF_{total}$ of three classification groups. In Figure 5(a), almost all computation-bound applications have small $DIFF_{total}$ within 2%. In Figure 5(b), 79% load-imbalance-bound applications have $DIFF_{total}$ within 1%. In Figure 5(c), it is also clear that communication-sensitive applications have higher $DIFF_{total}$ with a maximum value of 26.97% and more than 90% of the cases are within 10%.

Based on these statistics, a naive heuristic is to recommend only communication-sensitive applications (classified by MFACT) for simulation, but not the load-imbalanced and computation-bound cases. By relying on MFACT, this naive approach achieves an overall successful prediction rate of 73.4% in this data set. This motivates us to develop a more sophisticated statistical model, which improves the successful prediction rate to 93.2%.

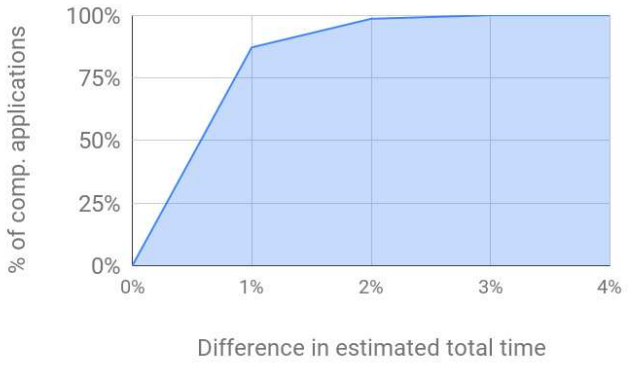
B. Statistical model

1) Candidate features for the statistical model: Table III lists 35 candidate features that we considered representative to the performance characteristics, communication workload and implementation of the application. Among these features, "CL" is defined based on the classification results of whether an application is sensitive to the speedup and slowdown of network bandwidth and communication in general. It has two levels, "cs" and "ncs": "cs" are communication-sensitive cases, "ncs" include both load-imbalance-bound and computation-bound cases.

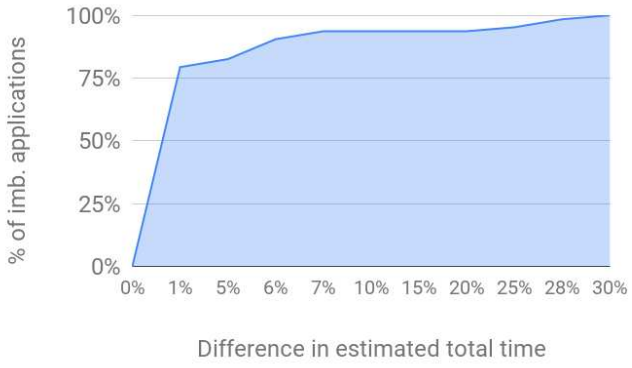
2) Statistical method for training: Because 235 observations represents a rather small data set for model training and evaluation, we select the parametric logistic regression [18] model for this study. In order to reduce the risk of over-fitting and evaluate the generality of any predictive model, we build the model on a sub-partition of the original data as training set and evaluate the error on the rest as testing set.

We use Monte Carlo cross-validation [22] to generate 100 sets of training and testing partitions, each time sampling 80% of the observations as training data without replacement. Test errors are aggregated on the 100 test sets by looking at the trimmed mean, standard deviation, and distribution in general.

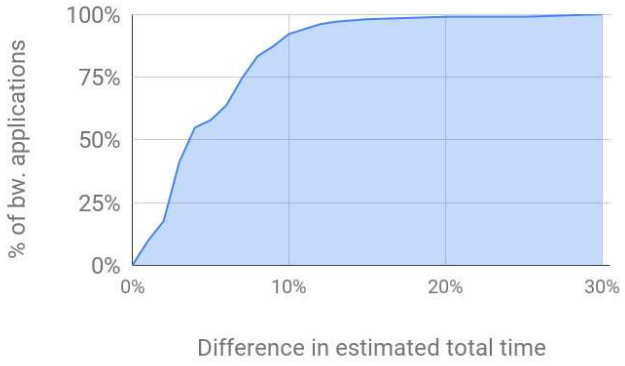
To avoid over-fitting and multi-collinearity, we limit the maximal number of variables to five while using a step-wise forward selection scheme. The step-wise approach evaluates a pool of available variables and select the variable that can bring the most improvement at each step in terms of the Akaike information criterion [2]. Table IV lists the top ten variables selected by the step-wise selection method. The



(a) Computation-bound applications



(b) Load-imbalanced applications



(c) Communication-sensitive applications

Figure 5: Absolute difference in estimated total time for communication-sensitive, computation-bound and load-imbalanced applications

coefficients are the average values of the 100 cross-validated partitions. Positive coefficient indicates positive correlation to giving a recommendation, negative otherwise.

In Table IV, "CL{ncs}" is the strongest predictor, and it has been selected every time. The negative coefficient of CL{ncs} implies that if an application is insensitive to the change of network communication, then it would be

Table III: Candidate features

Group	Variable	Description
Application	R	Number of ranks
	RN	Ranks per node
	N	Number of nodes deployed
Execution	T	Total execution time
	Tcp	Computation time
	PoCP	% of computation time
	Tc	Communication time
	PoC	% of communication time
Collective	Tbr	Barrier time
	PoBR	% of barrier time
	Tfbr	First barrier time
	PoFBR	% of first barrier time
	Tcoll	Collective time
	PoCOLL	% of collective time
	Tfcoll	First all-to-all collective time
	PoFCOLL	% of Tfcoll
Point-to-point	Tp2p	Point-to-point time
	PoTp2p	% of peer-to-peer time
	Tsyn	Synchronous peer-to-peer time
	PoSYN	% of synchronous peer-to-peer time
	Tasyn	Asynchronous peer-to-peer time
	PoASYN	% of asynchronous peer-to-peer time
Message	TB	Total bytes sent
	NoM	Number of messages sent
	TBp2p	Total peer-to-peer bytes sent
	CR	Number of destination ranks per source
	CRComm	Average peer-to-peer comm. per dest.
MPI	NoCALL	Number of MPI calls
	NoS	Number of blocking sends
	NoIS	Number of non-blocking sends
	NoR	Number of blocking receives
	NoIR	Number of non-blocking receives
	NoB	Number of barriers
	NoC	Number of collectives
Classification	CL	Sensitivity to communication

unnecessary to pursue detailed simulations. This is consistent with our observations that computation-bound and load-imbalanced applications can be modeled by the modeling tools with good accuracy. In the final model, we pick the top five variables from the list and compute coefficients to derive a predictive model. The source code used to train our model is available online.[6] Users can collect data and build their own predictive models.

3) Results: The misclassification rate (MR), false negative (FN) rate and false positive (FP) rate are computed over 100 test runs. The FN rate is defined as the number of FN cases divided by the sum of the FN and true positive cases. The FP rate is defined as the number of FP cases divided by the sum of the FP and true negative cases. We report the trimmed mean that discards the top and bottom 2% of the 100 test results. The misclassification rate has a trimmed mean of 6.8% which indicates that the enhanced MFACT can effectively separate MPI applications for modeling and simulation with success rate of 93.2% on our dataset. The trimmed means for the FN rate and the FP rate are 6.2% and 6.7%, respectively.

[6]https://github.com/ztong87/MFACT/blob/master/model.R

Table IV: Variables selected in step-wise selection

Rank	Variable	% Selected	Coefficient
1	CL{ncs}	100%	-1.68E+03
2	PoSYN	97%	-3.73E-02
3	R	74%	3.04E-01
4	Tasyn	63%	-3.34E-09
5	CRComm	44%	-8.90E-02
6	NoB	32%	-2.56E-01
7	N	24%	7.25E-02
8	Tfbr	16%	1.48E-08
9	RN	15%	2.47E-01
10	PoCOLL	7%	4.48E-02

4) Discussion: We examine the mis-classified applications in our tests. Computation-bound applications were never mis-classified, which is understandable as communication time has a minor impact on such applications. Both load-imbalanced applications such as IS, MG, and FT with a large number of ranks as well as communication sensitive applications such as CR and Nekbone account for significant numbers of mis-classifications. Two main factors contribute to the mis-classifications: (1) the limited number of traces, which limit the coverage of the training set, and (2) the existence of applications whose DIFF values are close to the 2% threshold. We believe that the impacts of both factors can be alleviated when more training data are available.

VII. Conclusions

We compared the performance and accuracy of a modeling tool (MFACT) and a simulation tool (SST/Macro). The comparison shows that detailed simulations do not necessarily produce more accurate application performance predictions. In particular, for almost all computation-bound applications, most load-imbalance-bound applications, and some communication-sensitive applications, faster modeling approaches perform just as well as slower simulation approaches. Finally, we demonstrated that it is possible to enhance a modeling tool with sufficient introspection to identify its own shortcomings and the ensuing need for more detailed simulation.

Acknowledgments

We gratefully acknowledge the support of the U.S. Department of Energy through grant DE-SC0016039 for this work. Los Alamos National Laboratory is operated by Los Alamos National Security LLC for the U.S. Department of Energy under contract DE-AC52-06NA25396.

References

[1] H. Adalsteinsson, S. Cranford, D. A. Evensky, J. P. Kenny, J. Mayo, A. Pinar, and C. L. Janssen. A simulator for large-scale parallel computer architectures. *Int. J. Distrib. Syst. Technol.*, 1(2):57–73, Apr. 2010.

[2] H. Akaike. *Information Theory and an Extension of the Maximum Likelihood Principle*, pages 199–213. Springer New York, New York, NY, 1998.

[3] C. Carothers, D. Bauer, and S. Pearce. ROSS: A high-performance, low memory, modular time warp system. In *Proceedings of the Furteenth Workshop on Parallel and Distributed Simulation (PADS 2000)*, pages 53–60, 2000.

[4] H. Casanova, F. Desprez, G. S. Markomanolis, and F. Suter. Simulation of mpi applications with time-independent traces. *Concurr. Comput. : Pract. Exper.*, 27(5):1145–1168, Apr. 2015.

[5] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.

[6] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM.

[7] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr. The Scalasca performance toolset architecture. *Concurr. Comput.: Pract. Exper.*, 22(6):702–719, Apr. 2010.

[8] S. Girona and J. Labarta. Sensitivity of performance prediction of message passing programs. In *in Proc. International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 620–626, 1999.

[9] S. Girona, J. Labarta, and R. M. Badia. Validation of Dimemas communication model for MPI collective operations. In *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 39–46, London, UK, UK, 2000. Springer-Verlag.

[10] R. W. Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Comput.*, 20(3):389–398, Mar. 1994.

[11] Intel Corporation. Introduction to Ethernet latency. 2014. [Online; accessed 14-AUG-2015].

[12] C. L. Janssen, H. Adalsteinsson, and J. P. Kenny. Using simulation to design extreme-scale applications and architectures: Programming model exploration. *SIGMETRICS Perform. Eval. Rev.*, 38(4):4–8, Mar. 2011.

[13] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[14] Lawrence Berkeley National Lab. Edison system interconnect. 2016. [Online; accessed 6-Aug-2016].

[15] Lawrence Berkeley National Lab. Hopper system interconnect. 2016. [Online; accessed 6-Aug-2016].

[16] B. Liu, D. R. Figueiredo, Y. Guo, J. Kurose, and D. Towsley. A study of networks simulation efficiency: fluid simulation vs. packet-level simulation. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1244–1253 vol.3, 2001.

[17] D. R. Martinez, V. Blanco, J. C. Cabaleiro, T. F. Pena, and F. F. Rivera. Modeling the performance of parallel applications using model selection techniques. *Concurrency and Computation: Practice and Experience*, 26(2):586–599, 2014.

[18] P. McCullagh. Generalized linear models. *European Journal of Operational Research*, 16(3):285–292, 1984.

[19] B. Mohr and F. Wolf. KOJAK—a tool set for automatic performance analysis of parallel applications. In *Euro-Par*, 2003.

[20] M. Noeth, P. Ratn, F. Mueller, M. Schulz, and B. R. de Supinski. ScalaTrace: Scalable compression and replay of communication traces for high-performance computing. *J. Parallel Distrib. Comput.*, 69(8):696–710, Aug. 2009.

[21] A. Núñez, J. Fernández, J. D. Garcia, F. Garcia, and J. Carretero. New techniques for simulating high performance mpi applications on large storage networks. *J. Supercomput.*, 51(1):40–57, Jan. 2010.

[22] R. Remesan and J. Mathew. *Hydrological Data Driven Modelling: A Case Study Approach*. Springer, 2015.

[23] Sandia Corporation. SST: The structural simulation toolkit. 2014. [Online; accessed 14-DEC-2012].

[24] R. Thakur and W. D. Gropp. Improving the performance of MPI collective communication on switched networks. 11/2002 2002.

[25] M. M. Tikir, M. A. Laurenzano, L. Carrington, and A. Snavely. Psins: An open source event tracer and execution simulator. In *2009 DoD High Performance Computing Modernization Program Users Group Conference*, pages 444–449, June 2009.

[26] B. Tomlinson, J. Cerutti, R. A. Ballance, M. Vigil, J. Johnson, K. Haskel, and R. A. Ballance. Cielo usage model. July 2012. [Online; accessed 6-Aug-2016].

[27] Z. Tong, S. Pakin, M. Lang, and X. Yuan. Fast classification of MPI applications using Lamport's logical clocks. In *Parallel and Distributed Processing, 2016. IPDPS 2016. IEEE International Symposium on*, pages 1–8. IEEE, 2016.

[28] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. V. Kalé. Simulation-based performance prediction for large parallel machines. *Int. J. Parallel Program.*, 33(2):183–207, June 2005.