SAND2014-16867C

Tools for Enabling Automatic Validation of Large-scale Parallel Application Simulations

Deli Zhang
University of Central Florida
Orlando, FL, USA
de-li.zhang@knights.ucf.edu

Gilbert Hendry
Sandia National Laboratories
Livermore, CA, USA
ghendry@sandia.gov

Damian Dechev
University of Central Florida
Orlando, FL, USA
Sandia National Laboratories
Livermore, CA, USA
dechev@eecs.ucf.edu

***Abstract*—Validation is highly important in parallel application simulations with a large number of parameters, a process that can vary depending on the structure of the simulator and the granularity of the models used. Common practice involves calculating the percentage error between the projected and the real execution time of a benchmark program. However, this coarse-grained approach often suffers from a parameter insensitivity problem in regions of high-dimensional parameter space. In this work we demonstrate the use of our fine-grained validation toolset to capture and compare the statistical characteristics of a parallel application's execution. Our experimental evaluation shows that our validation approach offers a significant improvement in fidelity when compared to validation using total execution time.**

I. Introduction

Building a system simulator is a well-known approach to predict the performance of large-scale parallel applications*. Performance projections can be produced by executing the application in a simulated environment comprised of software models in place of hardware components. To fully capture the interactions between computation and communication in hardware and software components, system simulators usually evolve into large software packages comprised of several layers of modules with dozens of input parameters.

Such complexity makes the task of establishing a simulator's accuracy a challenging problem. Most simulation frameworks evaluate accuracy in terms of total execution time [4, 3, 1, 12], where the projected execution time is compared against the real execution time and the error of the simulation is denoted by a percentage ratio. While the total execution time is one of the most important artifacts in a simulation environment, it lacks the fidelity (the ability to identify execution details) and the coverage (the ability to encompass all aspects of execution) to serve as a metric for fine-grained simulation accuracy.

In this work we present a validation toolset † that aims to capture fine-grained execution details. It consists of a trace analysis tool that decomposes execution time into finer granularity, a trace comparison tool that quantizes the disparity between corespondent metrics of two executions, and a visualization tool that renders the analysis and comparison results in intuitive graphs. The analysis process takes into account five groups of statistical data profiled from program traces: *overall traffic and timing*, *per-node traffic and timing*, *MPI function histogram*, *collective synchronization* and *node-to-node communication*. Although a trace file, such as one collected with the DUMPI format [5], can itself provide thorough details about the execution, quantitatively comparing two traces is difficult because the relevant information is scattered among a large volume of data. Our statistical metrics aggregate the relevant information from the traces and provide a scalable and lightweight overview of the application's execution details. To verify our approach, we conduct a number of trace-driven hardware model validation experiments on the SST/macro simulation framework [5].

II. Background

Most papers related to MPI simulation contain a rigorous discussion on their evaluation methodology. To the best of our knowledge, there is no dedicated toolset addressing fine-grained simulation validation. Our validation toolset greatly enhances the reliability of the simulation results. In addition, the presented approach is tightly integrated with our MPI tracing and profiling tools, thus offering a practical and effective technique for performance analysis of MPI codes.

A. Accuracy Evaluation Approaches

Evaluating the accuracy of simulation, or simulation validation, normally involves the choice of an evaluation metric and an error measure‡, and benchmark programs.

An accuracy evaluation metric is a quantitative measurement that captures certain aspects of a program's execution. Total execution time is the most pervasive choice since it is directly linked to performance. Other metrics such as network latency and bandwidth are also used to validate network models.

Error measure is a binary function that computes the disparity between two input arguments. The most common choice

*Since Message Passing Interface (MPI) is the dominating parallel programing model in high performance computing, we use the term large-scale parallel application and MPI application interchangeably

†Both the toolset and demonstration video can be downloaded from http://cse.eecs.ucf.edu/download.php?approval=417740585

‡We use the term error measure to refer to the function that calculates the error between two values

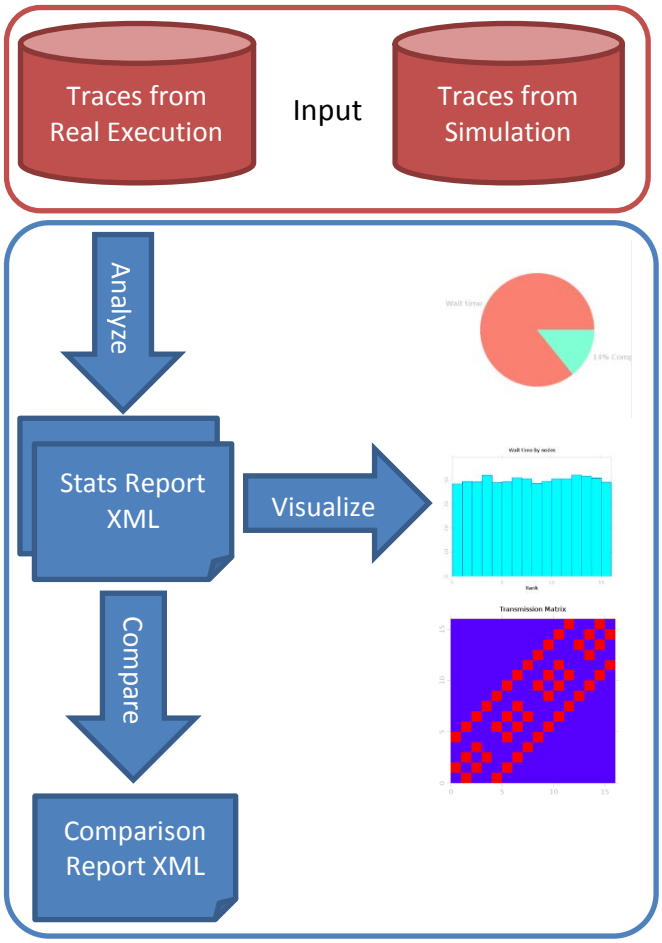

Figure 1: Data flow of the validation toolset. The visualization graphs are (from top to bottom): break down of computation and communication time, per-node communication time breakdown, and communication traffic matrix.

is the percentage error or relative error:

$$Err = \frac{X - R}{R} \tag{1}$$

where X denotes the experimental value and R denotes the reference value. However, relative error is not symmetrical, meaning that if the experiment's value and reference value are swapped, the result is not the same. To overcome the biases introduced by the relative error metric, logarithmic error was proposed in the accuracy study of SimGrid [9]. Using the logarithmic error provides a symmetrical function which also allows for computing the maximum, the mean and the variance.

B. Tracing and Profiling

Tracing is the postmortem approach for analyzing the performance of MPI applications. It collects detailed event history data including function name, call stack, timing, and payload, which could be visualized on a timeline display. Vampir [6], Scalasca [2], and TAU [7] are some of the well known comprehensive tracing tools. However, tracing tools place noticeable overhead on the application and produce large volumes of trace data (up to hundreds of Gigabytes), which makes working with them a laborious task.

Profiling, on the other hand, is the low overhead alternative to providing scalable, lightweight overview of the applications communication activity. Tools like IPM [11] and mpiP [10] have been extensively used for application optimization. Unlike a full-fledged tracing toolset that records the event history, a profiling tool only aggregates and reports runtime statistics.

Our toolset bears close resemblance to existing profiling tools in that they both capture execution statistics. Moreover, our toolset support quantitative comparison of two traces, which is essential in simulation validation tasks. This functionality is not present in existing tracing or profiling tools.

III. Fine-grained Validation Toolset

The core validation toolset consists of two loosely coupled component: a Trace Analysis Tool (TAT) and a Trace Comparison Tool (TCT). The validation process takes as input two traces and quantizes the error between them. In the case of hardware model validation, these two traces come from executing the benchmark on the real machine and the simulator respectively. An additional Trace Visualization Tool (TVT) helps render the analysis and comparison results in graphs. Figure 1 shows the data flow among the tools in a typical validation scenario. The action arrows represents the data transformation process done by the three tools. The tools are only coupled together by a common XML format, facilitating both standalone invocation and scripted automation.

A. Trace Analysis Tool

TAT extracts five groups of statistics from each input trace, which are used as the fine-grained evaluation metrics. It transforms the trace files (in binary format) into statistics report files in XML format as shown in Figure 2. Each statistics group is a modular entry in the report file. The advantage of this structured file format is that new entries can be added without interfering with the existing ones. This allows us to tailor the analysis process according to the simulation framework used.

The analysis process aggregates timing and traffic information from MPI function calls within trace files. The core idea is to decompose traffic and timing information along different dimensions to obtain zoom-in views of the same data. At the topmost level, we simply separate the total computation and communication time (see Figure 1). As we progress, the granularity of the decomposition increases. Namely, we break down the traffic and timing on per-node, per-function, per-collective-phase and node-to-node bases. This process also strikes a balance between granularity and noise. While trace files provide finest execution details, isolated function logs are noisy. By aggregating the information, we quash the noise and distill statistically significant characteristics.

The first metric is the *overall traffic and timing*, which lists the respective total of communication traffic, wait time and compute time. The traffic measures the communication data flow. In our case we count the total number of data bytes sent out by each node. Additionally, we obtain bandwidth via

```
<?xml version="1.0" ?>
<DUMPI_Trace_Report>
    <Meta fileprefix="dumpi-2013.01.17.16.33.05"
        numprocs="64" startime="1358469185">
    </Meta>
    <Statistics count="5">
        <Entry name="Overall">
            <![CDATA[Actual Data]]>
        </Entry>
        <Entry name="PerNode">
            <![CDATA[Actual Data]]>
        </Entry>
        <Entry name="PerFunction">
            <![CDATA[Actual Data]]>
        </Entry>
        <Entry name="PerCollectivePhase">
            <![CDATA[Actual Data]]>
        </Entry>
        <Entry name="NodeToNode">
            <![CDATA[Actual Data]]>
        </Entry>
    </Statistics>
</DUMPI_Trace_Report>
```

Figure 2: The structure of statistics report file

dividing traffic by communication time, which is a necessary measurement to identify the network bottlenecks. Wait time is the amount of time spent on blocking communication calls. Comparing with the total execution time, listing both wait time and compute time breaks down timing granularity, allowing us to capture the communication and computation performance separately. The second metric, *per-node traffic and timing*, decomposes the traffic and timing by node. These breakdown statistics reflect the load balancing among the nodes, because we expect an accurate simulation to also reproduce the deviation among the nodes. The third metric is the *MPI function histogram*. For each profiled MPI function call, we record the total block time on each node. We also collect the average and the standard deviation of block time for each MPI function across all nodes. The fourth metric is the *collective synchronization*. We define a collective phase as an interval of program execution between two consecutive collective MPI calls. Unlike point-to-point functions, collective functions operate on every node within a communicator, and they effectively act as the barriers that synchronize program execution across all nodes. The sequence of collective function calls between `MPI_Init` and `MPI_Finalize` divide the whole execution into numbers of synchronization phases. The time interval of these phases on all nodes also forms a matrix. We then compute the average and the standard deviation. The last metric is the *node-to-node communication*, which quantitatively shows the communication traffic and timing among all pairs of nodes.

B. Trace Comparison Tool

TCT computes the element-wise percentage difference on a per-group basis between the extracted metrics. It produces trace comparison report in a structured format similar to that in Figure 2. From the timing data obtained from the trace analysis process, we first compute the element-wise percentage difference and then take the average as final error:

$$Err = \frac{1}{n} \sum_{1 \le i \le n} \frac{|x_i - r_i|}{\frac{x_i + r_i}{2}} \tag{2}$$

Unlike percentage error, percentage difference only measures magnitude and is completely symmetrical. This can be observed from $|x - r| = |r - x|$. The symmetric error function is important when comparing two simulated executions. We expect the same error regardless which is used as a reference.

We also consider the validation of software models, such as skeleton applications [8], which are used in on-line simulators as the surrogates of the original program. A skeleton is derived by removing computationally intensive program fragments, but retaining code that determines the execution behavior. The program's communication pattern has the most significant impact on the performance and scalability of large-scale applications. Validating a skeleton is similar to machine model validation except that now we compare the execution of two different programs (the original program and its skeleton) on the same simulated environment. Under such circumstances, we also compute an additional qualitative error that measures the structural difference between the statistics entries. We present this qualitative error alongside the regular quantitative error in our comparison report.

IV. Towards Automatic Simulation Tuning

Using our validation toolset, we conducted a preliminary evaluation of the simulation accuracy of the SST/macro simulation framework against a Cray XE6 cluster [§]. SST/macro is a coarse-grained system simulator that supports both on-line and off-line simulations. When it is driven by a trace file in off-line mode, the simulator accepts trace files in both the standard Open Trace Format (OTF) and its custom DUMPI [¶] format. We prioritize the support of the DUMPI trace format because of its performance advantages over OTF. Support for other formats is also possible since we isolate the trace file interface from the core analysis and comparison module.

In our tests we execute miniMD [‖] on the cluster using 64, 128, 256, and 512 nodes. We then simulate these executions with a nominal hardware model and four of its variations (by randomly changing parameters of the nominal model). We expect the nominal model to give the most accurate simulation results. However, all projected total execution time agrees with the real execution time within 5% error, making it impossible to judge the simulation accuracy based on this metric alone. By resorting to manual inspection of the TAT results, we see that the decomposition of the timings are actually different at finer granularity. For example, *MPI function histogram* shows that `MPI_Allreduce` contribute most the disparity between the real run and the simulations. We also observe that the projected time of variation 1 is the closest to the actual execution time, but it erroneously prolongs `MPI_Send` for more than 10%. On

[§] http://www.cray.com/Products/Computing/XE/XE6.aspx
[¶] http://sst.sandia.gov/using_dumpi.html
[‖] http://www.mantevo.org/packages.php

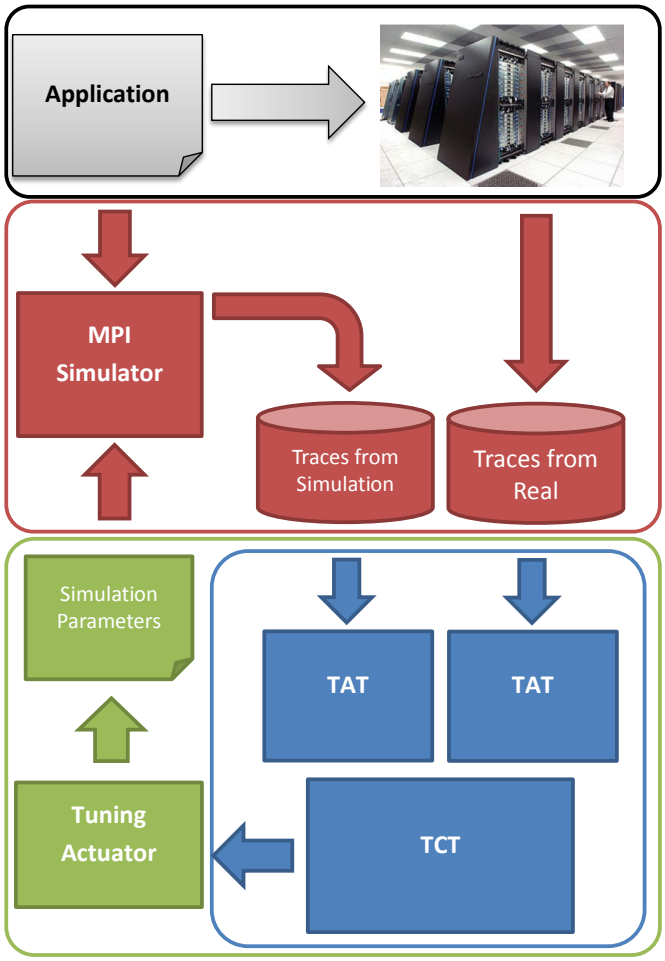

Figure 3: The automatic simulation tuning framework.

the other hand, the TCT reports reveal that the nominal model systematically produces the least amount of quantized error while the errors of other variations increase as the parameters drifting further away from the nominal values. The quantized error thus serves as a reliable criterion for identifying the optimal simulation parameters.

Moreover, the validation toolset fits into the automated simulation tuning process as shown in Figure 3. Based on the error generated by comparing two traces, the automatic simulation tuning framework finds optimal simulation parameters with little or no human intervention. The auto-tuning process would heuristically search through the valid simulation parameter space, and favor those which produce the most similar traces. This automation eases the repetitive simulation process in designing large-scale parallel machines and could potentially shorten the development cycle.

To conclude, we presented a validation toolset that uses statistical metric instead of total execution time to evaluate the accuracy of large-scale parallel simulation. The toolset is modular and can be used in isolation or within scripted automation frameworks. It provides improved fidelity over the current accuracy evaluation approaches and additional trace comparison functionality over existing profiling tools.

V. Acknowledgements

Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] W. Denzel, J. Li, P. Walker, and Y. Jin. A framework for end-to-end simulation of high-performance computing systems. *Simulation*, 86(5-6):331–350, 2010.

[2] M. Geimer, F. Wolf, B. Wylie, E. Ábrahám, D. Becker, and B. Mohr. The scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, 2010.

[3] S. Hammond, G. Mudalige, J. Smith, S. Jarvis, J. Herdman, and A. Vadgama. Warpp: a toolkit for simulating high-performance parallel scientific codes. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, page 19. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[4] T. Hoefler, T. Schneider, and A. Lumsdaine. Loggopsim-simulating large-scale applications in the loggops model. In *Proceedings of the 19th ACM international symposium on high performance distributed computing, HPDC*, volume 10, pages 597–604, 2010.

[5] C. Janssen, H. Adalsteinsson, S. Cranford, J. Kenny, A. Pinar, D. Evensky, and J. Mayo. A simulator for large-scale parallel computer architectures. *International Journal of Distributed Systems and Technologies (IJDST)*, 1(2):57–73, 2010.

[6] W. Nagel, A. Arnold, M. Weber, H. Hoppe, and K. Solchenbach. *VAMPIR: Visualization and analysis of MPI resources*. Citeseer, 1996.

[7] S. Shende and A. Malony. The tau parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[8] M. Sottile, A. Dakshinamurthy, G. Hendry, and D. Dechev. Automatic extraction of software skeletons for benchmarking large-scale parallel applications. In *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS)*, May 2013.

[9] P. Velho and A. Legrand. Accuracy study and improvement of network simulation in the simgrid framework. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, page 13. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[10] J. Vetter and C. Chambreau. mpip: Lightweight, scalable mpi profiling. *URL: http://www.llnl.gov/CASC/mpiP, Accessed 4/26/2013*, 2005.

[11] N. Wright, W. Pfeiffer, and A. Snavely. Characterizing parallel scaling of scientific applications using ipm. In *The 10th LCI International Conference on High-Performance Clustered Computing*, pages 10–12, 2009.

[12] J. Zhai, W. Chen, and W. Zheng. Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node. In *ACM Sigplan Notices*, volume 45, pages 305–314. ACM, 2010.