HPC Network Simulation Tuning via Automatic Extraction of Hardware Parameters

Joshua Suetterlein, Stephen J. Young, Jesun Firoz, Joseph Manzano,
Nathan Tallent, Ryan D. Friese, Kevin Barker, Timothy Stavenger
Pacific Northwest National Laboratory, USA
{{first name}.{last name}}@pnnl.gov

***Abstract*—Popular HPC network interconnection simulators such as `SST/macro` provide a variety of configurable parameters to explore the design space of hardware components such as network interface cards (NIC), switches, and links among them. While such knobs provide flexibility to explore design trade-offs for novel hardware, manually configuring simulations for matching configurations of the existing hardware to focus on topology exploration can be cumbersome and error-prone, leading to widely inaccurate simulations. This challenge is compounded when specifications of various (proprietary) technologies are not readily available or intentionally omitted.**

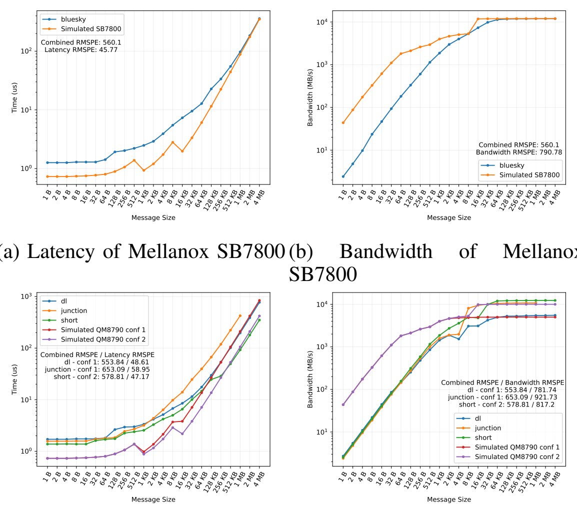
In this work, we propose a framework to autotune the multiple network models' simulation configurations within `SST/macro` using Tree-structured Parzen Estimator-based Bayesian optimization to observe the effect on simulation accuracy across different message regimes. These regimes consist of small to large message sizes and latency to bandwidth-bound messages. We provide a detailed analysis of the simulation error for four representative HPC systems. Our Bayesian optimization-based autotuning framework for network models achieves a maximum of 5x improvement in accuracy over best-effort manual configurations based on available hardware specifications.

I. Introduction

Researchers heavily rely on advanced simulation frameworks to develop, test, and validate new designs and complex technologies before making procurement or design decisions. Some notable simulators include Structural Simulation Toolkit Macroscale Element Library (`SST/macro`) [2], Co-Design of Multi-layer Exascale Storage Architecture (CODES) [3], BookSim [4], and others. While the flexibility of these simulators permits diverse and rapid prototyping, the fine-grain parameterization of the simulator's different models and components can result in an unwieldy configuration space. Such space is particularly overwhelming for research focused on novel network topologies, as a configuration that accurately represents the performance of the hardware/software stack is essential to guarantee correct performance attribution for **all message regimes** under novel technologies.

Finding an optimal target configuration for simulation can be challenging, even with expert knowledge. Specifically, practitioners face this challenge in two ways. First, finding the *correct parameters for existing hardware is difficult*. This challenge is partly due to a mismatch in the fine-grain level of a simulator's configuration and the potentially incomplete (or unavailable) hardware specification. Second, understanding how to meaningfully adjust the *multiple performance models* within a simulator is challenging, even with the hardware specifications.

[*]A preliminary version of this work, titled "Automatic Extraction of Network Configurations for Realistic Simulation and Validation", was presented as a poster at 2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) [1].



(a) Latency of Mellanox SB7800 (b) Bandwidth of Mellanox SB7800

(c) Latency of Mellanox QM8790 (d) Bandwidth of Mellanox QM8790

Fig. 1: `SST/macro` simulated versus real hardware latency and bandwidth between two adjacent nodes. RMSPE is reported for the nearest simulated configuration.

For example, consider Figure 1, which shows, for different message sizes, the performance reported for the Ohio State University Microbenchmark suite (OSU) point-to-point latency and bandwidth benchmarks [5] for two adjacent nodes communicating through a single switch for four representative HPC systems (namely Bluesky, dl, Junction and Short). Using the manufacturer's hardware description for each system, we populate the `SST/macro` parameters and simulate both benchmarks. We report the simulation's Root Mean Squared Percent Error (RMSPE) relative to the actual run on the system for the benchmarks individually and together. Our simulation demonstrated combined errors ranging from 560% to 650% for our best-effort configurations. Notice that even similar hardware can exhibit very different performance profiles (e.g., Figures 1c and 1d). This behavior can explain the omission of a single one-size-fits-all configuration in `SST/macro` and highlights the tuning challenge.

From our example, we also see that reducing simulation error extends beyond correctly transcribing hardware environment, as a workflow can fall into different *message regimes* that affect the simulation accuracy, especially when dealing with novel technologies. Figure 1 shows that simulation environments may treat message regimes with different predictive models, further expanding the parametric search space[1]. Traditionally, HPC network simulators have concentrated on sizeable bandwidth-bound message regimes (e.g., in [6] validation starts at 8K message sizes), which are common in traditional scientific workflows. Simulation of small messages is usually relegated to analytical models, which are faster but less accurate (c.f. Section V for empirical confirmation.) This decision might arise from the prevalence of bulk synchronous processing paradigm in scientific computing, which prefers large messages at specific computational boundaries [7].

However, the recent trend towards applying HPC resources to workflows like machine learning and graph analytics, which are more latency-bound, increases the need to tune simulations for accuracy across different message regimes. For instance, fine-grained small messages (2kB-4KB) can be found as part of distributed coordination protocols and data-driven dynamic applications, among many others. Hence, to explore hardware capabilities and protocols [8] for supporting fine-grained data sharing in contemporary applications, it is imperative to consider different message regimes for faithful simulation.

In this paper, to address the modern simulator's parametric explosion, we propose an auto-tuning framework based on Bayesian optimization to find message regime-aware simulation configurations without the need for extensive knowledge in *both* the hardware/software stack and the simulator. We demonstrate the effectiveness of our tuning method for multiple models within `SST/macro` and show how they can be used together to reduce errors across messaging regimes. Lastly, while Bayesian optimization has been applied to hardware simulation [6], critical details of the process have been omitted (such as validation for smaller size message operation), which this work attempts to rectify.

The contributions of this paper are as follows:

- We propose a framework to *automatically* tune the simulation configuration and the multiple models found within `SST/macro` providing a complete model (i.e., appropriate across message regimes) that achieves a 5x improvement in accuracy over best-effort configurations.
- We explore how fitting benchmarks with multiple metrics can provide a configuration appropriate for *both* latency and bandwidth-bound applications.
- We analyze the effects of various loss functions for Bayesian optimization in the context and their appropriateness in this application.
- Within the optimization context, we provide an in-depth analysis of the simulation error for `SST/macro`.

II. Background

a) ***Eager vs Rendezvous MPI Protocols:*** The most widely used communication libraries (MPICH, MVAPICH, IBM Spectrum, Intel MPI etc.) in HPC applications are based on the Message Passing Interface (MPI) [9] specification. According to the MPI specification, the unit of communication is the message which transfers information between a sender and a receiver. The way that both actors prepare to receive or to send the message is usually divided into eager and rendezvous protocols [10]. In eager protocols, the sender assumes that the receiver can handle the incoming data. The protocol implies that the sender is "pushing" its message to the receiver without waiting for a signal that it is ready. This protocol reduces synchronization delays, but additional copies and buffers might be needed. On the other hand, in the rendezvous protocols, there is a signaling phase (in which header data is exchanged) to ensure that the destination can handle the message before the sender begins transmitting the payload. Although this method introduces the signaling phase, the savings of not having to delete extra copies and to allocate buffer space might amortize these extra costs. Usually, MPI implementations will switch between these protocols based on the message regimes that the application uses.

Method	Formulation	Consequence
Mean Absolute Error (MAE)	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - x_i\rvert$	Emphasizes BW accuracy for max BW message sizes
Mean Square Error (MSE)	$\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2$	Further emphasis on BW accuracy for max BW message sizes
Root Mean Square Error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2}$	Increased emphasis on sub-maximal BW messages & runtime for large messages
Root Mean Square Relative Error (RMSRE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - y_i}{y_i}\right)^2}$	Moderate emphasis on accuracy for BW and latency for small message sizes
Root Mean Square Percent Error (RMSPE)	RMSRE $*100\%$	Interpretable rescaling of RMSRE
Huber	$\sum_{i=1}^{n} L_\delta(\lvert x_i - y_i\rvert)$ $L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & a \le \delta, \\ a\delta - \frac{1}{2}\delta^2 & a > \delta \end{cases}$	Reduced penalty for large errors

TABLE I: Bayesian optimizer loss functions in terms of observation/prediction pairs (y_i, x_i). BW is bandwidth

b) ***`SST/macro` Network Simulator:*** `SST/macro` ([2], [6]) is a coarse-grained, open-source, discrete event simulator for the communication and computation taking place in HPC systems. Recently, `SST/macro` has been extensively used to evaluate hardware trade-offs for the design of exascale systems [11] as well as next-generation HPC topologies [12], [13], [14]. In order to simulate the communication behavior of hypothetical systems, `SST/macro` provides coarse-grained *simulation models* that intercept a standard library of MPI calls and simulate message flow through NICs, switches, and links. To control all the hardware's parameters (such as buffer size and minimum transmission units, among others), `SST/macro` provides an extensive interface through its configuration files.

Among the network traffic simulation models available in `SST/macro`, two of them are most relevant for our present study: an analytical model based on the LogP [15] model and a packet-based model that implements Quality-of-Service called

[1]The model change is evident at the knees in the simulation curve. For example, in Figure 1a, the evidence of three packet-size base models is visible between message sizes 1B to 512B, 512B to 8K, and 8K to 4MB

the Simulator Network for Adaptive Priority Packet Routing (SNAPPR) model [16]. During a simulation, based on the message regime (i.e., the size of a message), `SST/macro` switches between these two network models. The LogP-based analytical model calculates the latency of a message by considering latency per byte, overhead, gap or delay, and the number of parallel processors. Although this model is fast, it might not accurately capture the contention profile of the underlying network (`SST/macro` only applies a simple contention factor by adding random noise). On the other hand, the packet-based SNAPPR network model simulates the packet flow across each network component such as routers, network controllers, and links. This detailed simulation model can more accurately simulate the message propagation, resource allocation, and contention (i.e., in terms of message queue length) at the cost of more computational / simulation time.

*c) **Autotuning Technologies**: Optuna* [17] is an open-source [18], hyper-parameter tuning software which implements both independent variable Bayesian optimization approaches (such as Tree-structured Parzen Estimators (TPE) [19], [20]) and other Bayesian optimization approaches which attempt to leverage the relationship between variables (such as covariance matrix adaption evolutionary strategy [21]) to improve optimization performance. These Bayesian optimization approaches are combined with advanced pruning heuristics, such as Asynchronous Successive Halving, to provide a lightweight, distributed framework for optimizing complex parameter spaces. We use this framework to automate and manage the distributed fitting of `SST/macro` parameters to the observed performance of our chosen HPC systems using a variety of loss functions, see Table I. We apply the Tree-structured Parzen Estimators as the surrogate model for exploring the hyperparameter search space, as it has a proven track record of achieving award-winning accuracy in competitions such as Kaggle and AutoML. Moreover, it also has several additional settings that are helpful for our training task (e.g., support for multiobjective optimization).

*d) **Benchmarks considered**:* The OSU microbenchmark suite [5] is a popular communication benchmark suite for standard programming models focused on distributed computing (e.g., MPI, SHMEM, and UPC). The benchmarks are designed to test the readiness of HPC systems for inter-node communication and for comparing different distributed programming models. From the suite, we use the point-to-point MPI latency and bandwidth benchmarks. Each benchmark sends messages between two ranks for messages sizes 1B to 4MB (2^{22}B). The *latency benchmark* performs a ping-pong between the ranks and reports the one-way time in microseconds. The *bandwidth benchmark* launches concurrent asynchronous messages and reports the observed bandwidth in MB/s. These benchmarks provide a way to measure some of the network's most important performance characteristics.

III. Our Autotuning Framework for Simulation

As our first step to construct an accurate network model across different message regimes, we focus on configuring the minimal simulation unit, two nodes communicating through a single switch. We collect data on real HPC systems with the same setup for validation. The hardware simulated by `SST/macro` includes two nodes, NICs, and links connected by a single switch. We assume symmetric performance between the two nodes and define the best configuration of the simulator to be one that accurately models the performance of the OSU point-to-point microbenchmarks for the latency and bandwidth between two MPI ranks (one per node) for an *existing system*. Combining these two benchmarks covers multiple operational regimes, including small and large messages bound by latency and bandwidth.

Although we acknowledge the shortcomings of using these basic two nodes with a single switch setup for experiments (i.e., not factoring in the complex dynamics that may arise from multiple interacting units), understanding the implications of autotuning hardware parameters and network models for different message regimes for this setup is a significant step towards extending the methodology beyond two-node setup. Moreover, these initial results showcase the advantages and pitfalls of using these autotuning techniques for network simulations. Specifically, we empirically analyze different metrics and optimizers to guide the users in choosing the best combinations of simulation configurations in terms of accuracy.

Name (Sim. Model)	Range	Description
switch.mtu (S)	128B - 16MB	Packet size, units each message is broken into
switch.link.latency (S)	1ns - 1ms	The latency to traverse the link
switch.link.bandwidth (S)	1 - 50 GB/s	Arbitrator's BW
switch.link.credits (S)	128B - 16MB	Switch buffer size
node.nic.injection.mtu (S)	128B - 16MB	NIC injection unit size
node.nic.injection.latency (S)	1ns - 1ms	NIC injection latency
node.nic.injection.bw (S)	1 - 50 GB/s	NIC injection BW
node.nic.injection.credits (S)	1KB - 16MB	NIC injection buffer size
node.nic.ejection.latency (S)	1ns - 1ms	NIC ejection latency
switch.logp.hop_latency (L)	1ns - 1ms	Per hop latency for LogP-based switch
switch.logp.out_in_latency (L)	1ns - 1ms	Latency for processing (delay)
switch.logp.bandwidth (L)	1 - 50 GB/s	LogP-based switch's BW

TABLE II: `SST/macro` parameters used as the hyperparameters for the TPE-based Bayesian optimizer. (S) is defined as the SNAPPR simulator model and (L) is the LogP model.

A. Formulation of the Objective Function

For our current study, for a fixed loss function f with latency (L_b) and bandwidth (B_b) values from the OSU benchmark, we construct the following optimization problem:

$$\min_P \sum_{m=0}^{22} f(\{(\mathrm{SST}_L(P, 2^m), L_{2^m})\} \cup \{(\mathrm{SST}_B(P, 2^m), B_{2^m})\})$$

where P represents the `SST/macro` parameter settings, b is the total number of bytes sent, and $\mathrm{SST}_L(P, b)$ and $\mathrm{SST}_B(P, b)$ are the results from the `SST/macro` simulation of OSU micro-benchmarks for latency and bandwidth, respectively.

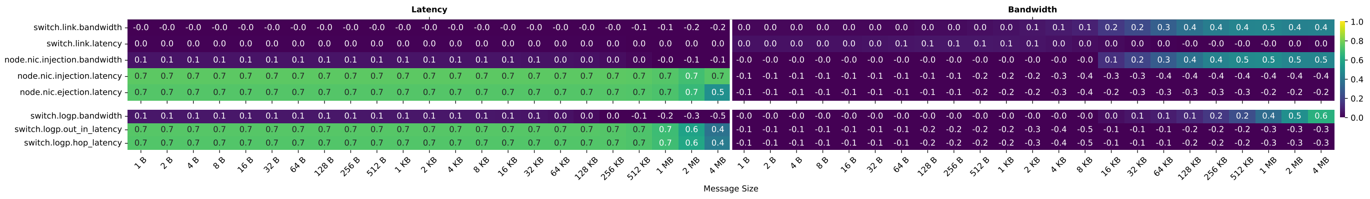


Fig. 2: A heatmap showing the correlation between latency/bandwidth and different hyperparameters for various message sizes.

B. The Role of Optuna's TPE

To solve the multi-objective optimization problem presented in Section III-A, our framework leverages the Tree-structured Parzen estimators (TPE)-based Bayesian Optimization, available in Optuna, to find the best possible configurations for the simulation parameters (i.e., the hyperparameters). TPE takes its name from its tree-structured search space expansion and Parzen-based Kernel Density Estimator. At a high level, the algorithm for TPE works as follows. In the first step, the algorithm initializes the hyperparameters with random values. Next, with the help of `SST/macro` , it calculates the scores (i.e., simulated time) for each of these configurations.

Then, it uses these scores to create an *observation space*, sorts the scores, and applies a kernel density estimator to obtain two density functions (one representing the "weighted distance" line to all the scores $l(x)$ and the ones with the best scores $g(x_{best})$). After, the estimator samples values for the hyperparameters from the all-score density $l(x)$ and find the set that minimizes the objective function when evaluated under the two density functions ($\frac{l(x)}{g(x_{best})}$). These scores are then added to the *observation space*. The procedure is repeated for a predetermined number of trials. At the end of the trials, the hyperparameter values are assumed to be the best configuration for the simulation involving a specific message size.

For evaluating the TPE's loss function, we use the ones described in Table I to empirically test which gives the best accuracy. For each loss function on each system, we explore three different fits based on latency, bandwidth, and a combination of both. The simulated performance loss is computed for each message size and averaged according to the metric. When fitting on both latency and bandwidth benchmarks, the metric's average is performed across the loss for each message size for each benchmark. We evaluate 2K trials/fit to ensure convergence. Invalid configurations (as determined by `SST/macro`) are penalized with an excessive score.

We evaluate the final configuration of each fit using the RMSPE method. This method normalizes the error for large and small message sizes across metrics (i.e., time and throughput) and presents the result as a percentage. We present the combined and individual RMSPE per fit to demonstrate the error attribution.

C. `SST/macro` Configuration

Our study focuses on the SNAPPR (Simulator Network for Adaptive Priority Packet Routing) and LogP packet models in `SST/macro` v11.1.0. By default, both models are used simultaneously, the LogP model [15] for small messages and SNAPPR for large messages. The switchover point between models is set to 512B by default but is configurable.

System	Net. Switch	Network Card	CPU
Bluesky	MX SB7800	IB HDR-100	Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz
Junction	MX QM8790	IB HDR-100	AMD EPYC 7543 32-Core Processor
Short	MX QM8790	IB HDR-100	AMD EPYC 7502 32-Core Processor
DL	MX QM8790	IB FDR-56	Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz

TABLE III: Hardware testbeds for our experiments. MX = Mellanox and IB = InfiniBand

Additionally, the SNAPPR model also considers the MPI's Eager and Rendezvous protocols [10]. We turned off the SNAPPR small message eager protocol to reduce hidden variables in our simulation and to be more representative of an OpenMPI implementation. Through trial and error, we found it most effective to fit a single model at a time for a fixed region of message sizes.

We use a fully connected switch for the topology with geometry (i.e., the topology shape as a list of dimensions) and concentration (i.e., number of computational nodes per switch) parameters set to one and two, respectively. We use the standard configuration given by the examples for the remaining parameters, such as a node's memory and processor. This setup is reasonable since our benchmarks are network-bound.

D. Feature Selection

Table II presents the configurable `SST/macro` parameters for the switch models and the node's NIC used as the hyperparameters for our TPE-based Bayesian optimizer. The parameters roughly fall into three categories: latency, bandwidth, and credit (indicating the capacity of a component). Figure 2 shows a heat map of the correlation matrix indicating the strength of the effect observed when changing a given parameter for the SNAPPR and LogP models, respectively. We run each model alone to demonstrate the impact of the configuration for message sizes from 1B to 4MB. The NIC injection and ejection latencies for the SNAPPR model are the most significant factors in determining the overall latency across all message sizes. Conversely, the switch's bandwidth and NIC effect are most prominent for messages greater than 8K (where the rendezvous protocol takes over). In the LogP model, the latency parameters have the most prevalent effect, which wanes as the message size increases. Further, in the bandwidth benchmark, we see the effect of the bandwidth parameter after message sizes of 8K.

We allow the optimizer to adjust all parameters for a given model in Table II within the bounds listed. The physical constraints of existing technology can more readily infer the acceptable values for the lower bounds. For the upper bounds, we do our best to provide sensible values based on existing technology and leave a more in-depth study to future work.

IV. Evaluation

A. Experimental Setup

To evaluate and validate our Bayesian optimization based autotuning approach for finding the best-matching hardware configurations and message regimes for simulation, we optimize the configurations of the four in-house representative HPC systems listed in Table III. Notice that the DL, Short, and Junction systems all use the same type of switch; however, DL uses an older network card, limiting its overall bandwidth. Figure 1 shows the achieved performance of the OSU benchmarks for each system, which will serve as our ground truth. When collecting the performance of each system, we ensured that two nodes directly communicate through a single switch.

Moreover, to reduce the effects of the software stack, we set the MPI eager thresholds of the actual and simulated systems to match at 8KB. The variations in performance for a single type of switch highlight the pitfalls of just plugging hardware parameters into the `SST/macro` simulation configuration.

In the following discussion, we explore how to first optimize the SNAPPR model alone for messages sizes 1B to 4MB. Once we establish the best practices for optimization, we perform the same procedure for the default `SST/macro` model for different message regimes (LogP and SNAPPR models) and demonstrate the results of bolting two tuned models together in Section V.

B. Finding the Optimal Simulation Configuration for Representative Systems with different Loss Functions

Figure 3 shows the convergence of the optimizer for the various metrics across the individual systems. Each row and column corresponds to a specific fit and loss function. Each trial sweeps over the total number of messages for each case. In the case of the joint fit, the two sets are put together and swept accordingly. The loss plotted is the error used by the optimizer. Most loss functions converge well before 2K trials, indicating the optimizer has found a configuration to minimize the loss function. The different scales per y-axis highlight the effective range of each metric. RMSRE achieves the lowest final values (smaller being better).

C. Evaluation of the Optimal Configurations for different Loss Functions After Autotuning

Figures 4 and 5 present the error of the best configuration found after optimization. The three fits (combined latency and bandwidth, bandwidth only, and latency only, are given by a row, while a column corresponds to a particular system (e.g., Bluesky).

Figure 4 presents the percent error for the latency benchmark. This figure highlights the error for small messages, some of which are quite large. SNAPPR is not typically used for small messages (by default, the LogP model is used for messages less than 512B); however, messages close to 512B remain inaccurate across systems for several metrics. Across metrics, we see a similar shape (albeit at different magnitudes) for a given system, indicating a consistent behavior in the SNAPPR model (inaccurate modeling of small messages that gets progressively better as messages get larger).

Figure 5 presents the percent error for the bandwidth benchmark. While the error shown in Figure 5 is significantly lower than the latency benchmark, we still observe that the SNAPPR model better predicts the bandwidth for larger message sizes. In addition, note the change in the curve at 8KB corresponding to the change in MPI protocols. Such discontinuities can be used to infer the internal models being used.

Figure 6 shows the latency for each experiment. From the performance profiles, we see that the final configurations for each experiment result in different configurations. For the combined and bandwidth fits, we observe that the latency of small messages is noticeably higher than the ground truth. For all systems other than Junction and the combined fit of Huber for Short, the optimizer reasonably matches the latency of larger message sizes. When comparing the ground truth of Junction to short (a similar system) shown in Figure 1, we see a steeper latency curve beginning sooner. The bandwidth curve, however, remains similar to short. Moreover, when fitting for latency alone, most loss functions can better model the latency. This result is evidence of the competing trend in tuning for latency and bandwidth together.

Figure 7 shows the bandwidth achieved for each experiment. The achieved performance for the resulting configurations for this benchmark mostly straddles the actual bandwidth. The bandwidth fit performs the best prediction, highlighting the competing nature of matching latency and bandwidth performance.

D. Identification of the Best Loss Function for Evaluating Model Accuracy When Considering Multiple Competing Objectives

Figure 8 shows the RMSPE of predicted latency/bandwidth for each of the loss functions used in the Bayesian optimizer (cf. Table I) for each tested machine, as well as the combined latency and bandwidth performance relative to the ground truth for each fit. We also show the combined error for each fit/metric across all systems in Figure 10.

Both figures show that the SNAPPR model can be tuned to within 100% error despite its trouble with small latency-bound messages. **RMSRE stands out as the best metric for tuning consistently**, performing well across systems since it normalizes the error relative to the ground truth. The other metrics deal with the absolute error. In the case of competing objectives (such as optimizing for small versus large messages or latency versus bandwidth), the optimizer will be biased to fix the more significant error. Normalization significantly reduces this behavior and improves the autotuning performance.

Figures 9, 11, 15 and 16 show the attribution of error. We see RMSRE for the combined fit (Figures 15 and 16) balanced modeling latency and bandwidth with errors of 110.7% and 109.3%, respectively. Conversely, the combined fits for MAE, MSE, RMSE, and Huber have a much better fit for bandwidth, with Huber reaching 33%, but are significantly inaccurate in terms of latency (e.g., Huber at 973%). Further, tuning for an individual benchmark highlights the inverse relationship between latency and bandwidth in the SNAPPR model.

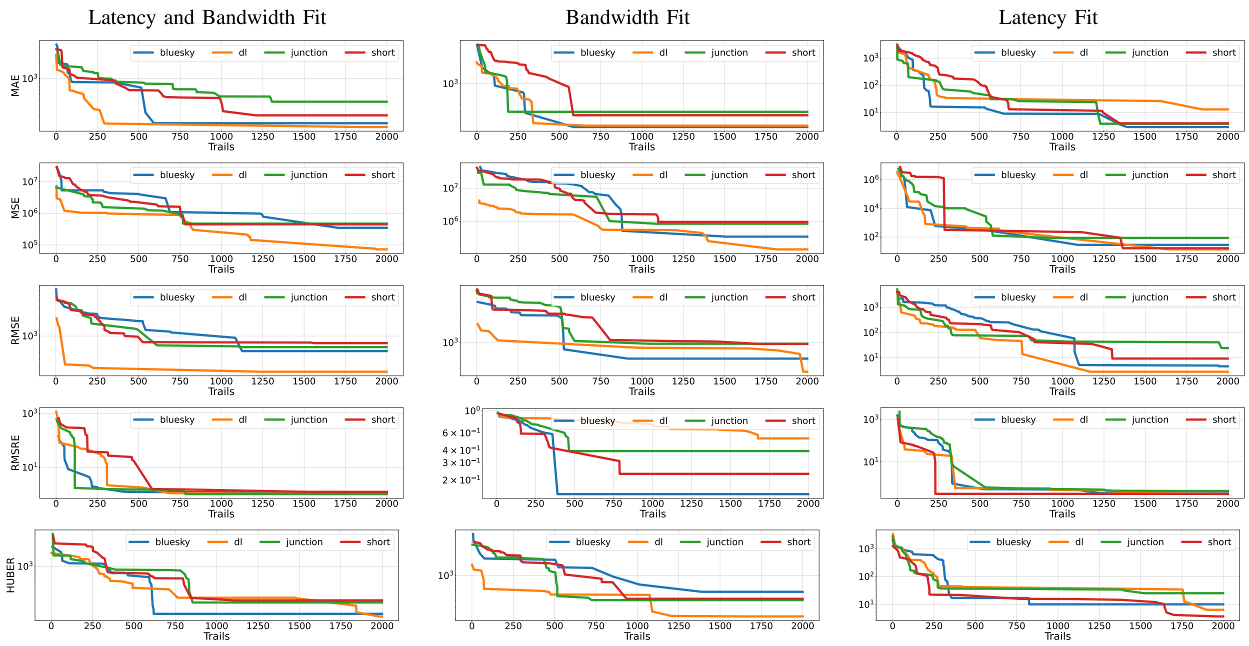


Fig. 3: The convergence profile of the Bayesian optimizer for the various loss functions across various machines. The horizontal axis represents the number of trials and the vertical axis represents error of the estimated model.

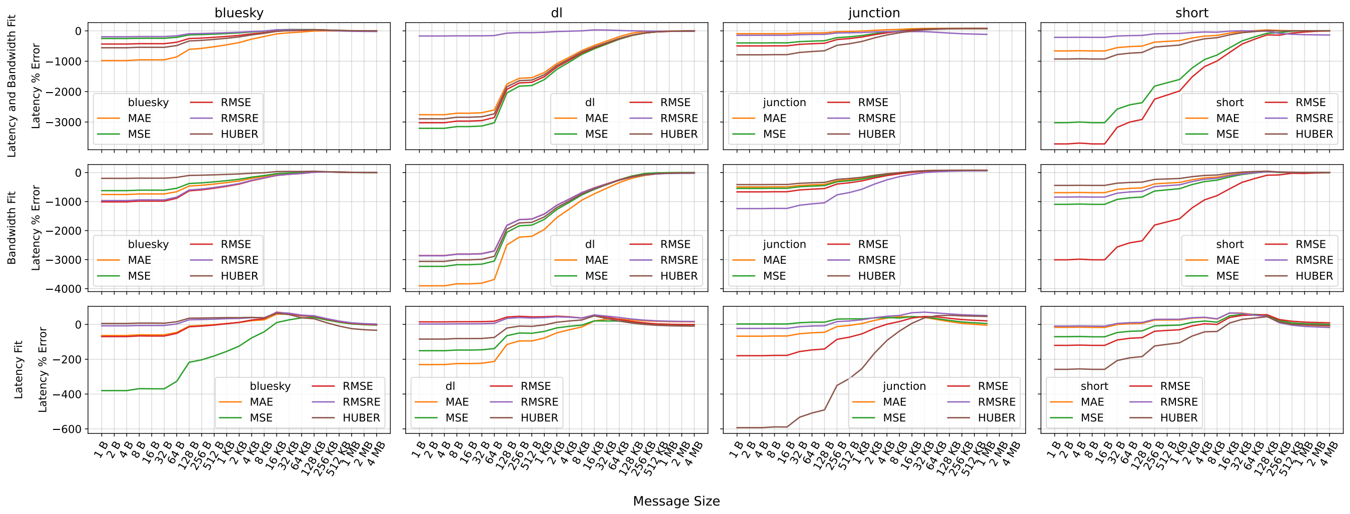


Fig. 4: Percent error of simulated OSU latency benchmark w.r.t the ground truth per loss function per system.

From the perspective of metric selection, it is easy to (over)fit the simulation when only one benchmark is used. For example, when tuned for bandwidth on Bluesky, all metrics reach between 11.2 to 19.7% error. While scale can be an issue for MAE, metrics such as MSE, RMSE, and Huber attempt to mitigate the bias against small changes. In the context of a single benchmark fit, we still see issues of scale in the small message errors (Figures 4 and 5), but overall the model performs well for the fitted benchmarks. The inability of MAE, MSE, RMSE, and Huber to achieve such good results when fitting for multiple benchmarks highlights the need for normalization.

From these results, we see that the SNAPPR model can be configured to achieve an accurate simulation (30%) for either latency or bandwidth messaging regimes with a bias in accuracy toward large messages. For applications with latency and bandwidth-bound phases, we have achieved an accuracy of approximately 100% with and bias toward large messages. The bias toward large messages gives rise to the need for an additional small message model.

V. Combining Models

To model small messages, `SST/macro` uses the LogP model with the SNAPPR model used after a specific threshold. In this section, we take the default `SST/macro` model (LogP for small messages, SNAPPR for large ones), a tuned SNAPPR model for all message regimes (no separate training for small and large messages), and two SNAPPR models bolted together, one trained on small message regime only, and one trained on large message regime, being switched according to message size. Moreover, we specified the threshold between large and small messages at 512 bytes.

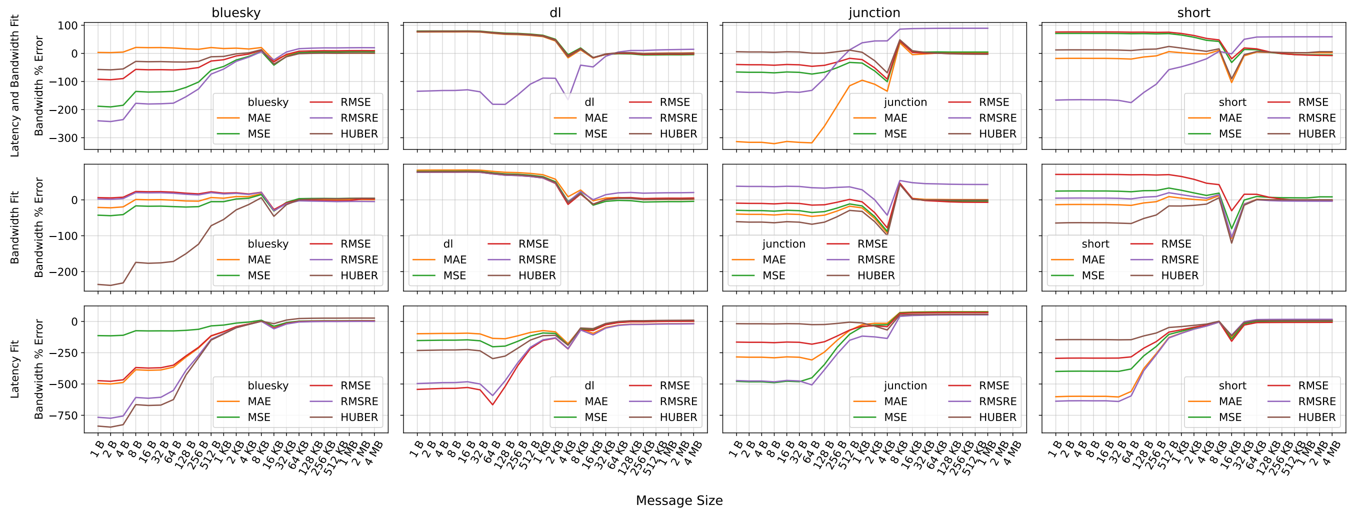


Fig. 5: Percent error of simulated OSU bandwidth benchmark w.r.t the ground truth per loss function per system.

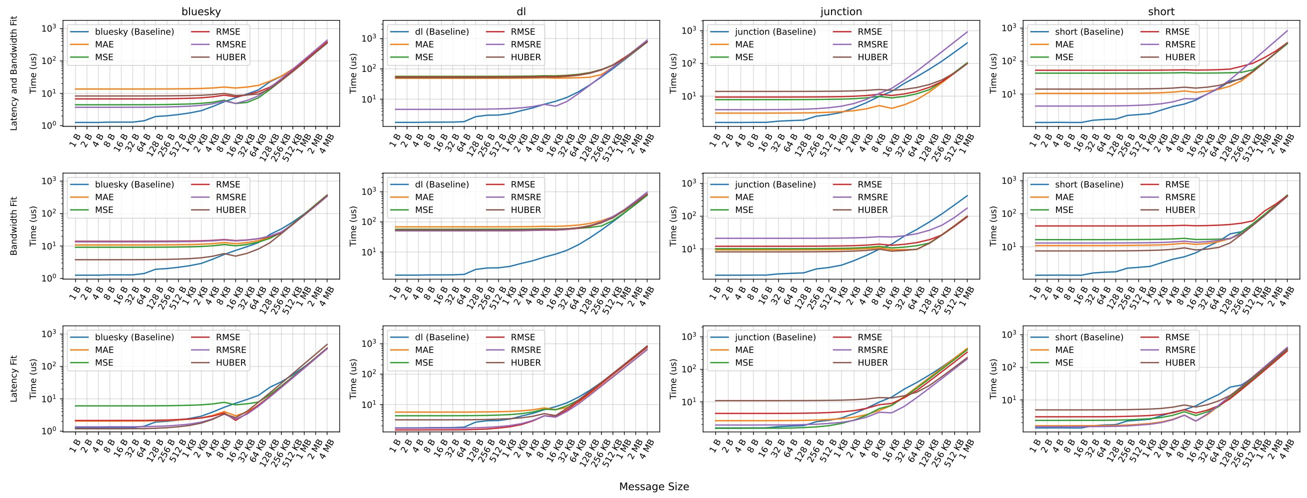


Fig. 6: Simulated versus real latency from OSU benchmark per loss function per system.

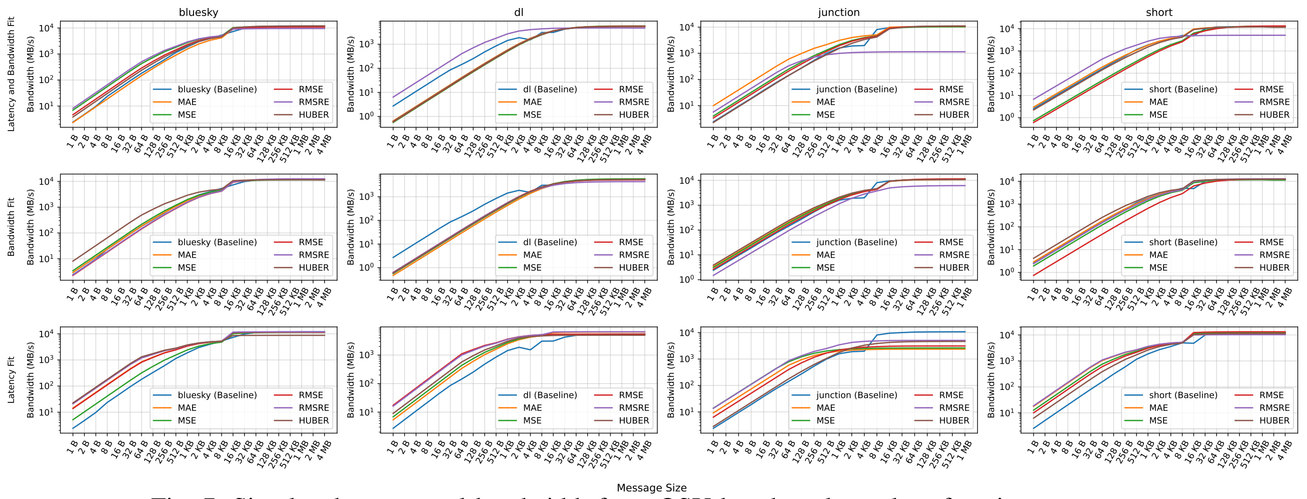


Fig. 7: Simulated versus real bandwidth from OSU benchmark per loss function per system.

In Figure 17, we present our results of this model listed as the LogP Low/SNAPPR High model, a.k.a. SST default, and

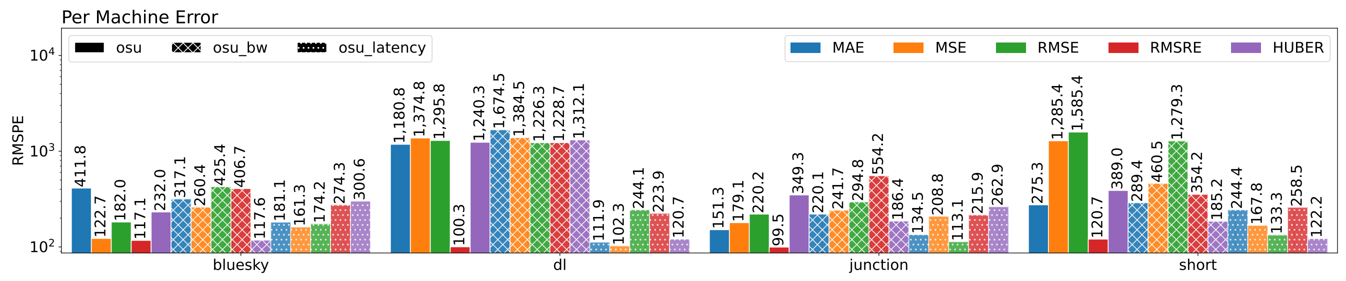


Fig. 8: Per Machine observed latency and bandwidth RMSRE for the tested loss functions across systems. The error is relative to both the latency and bandwidth benchmarks.

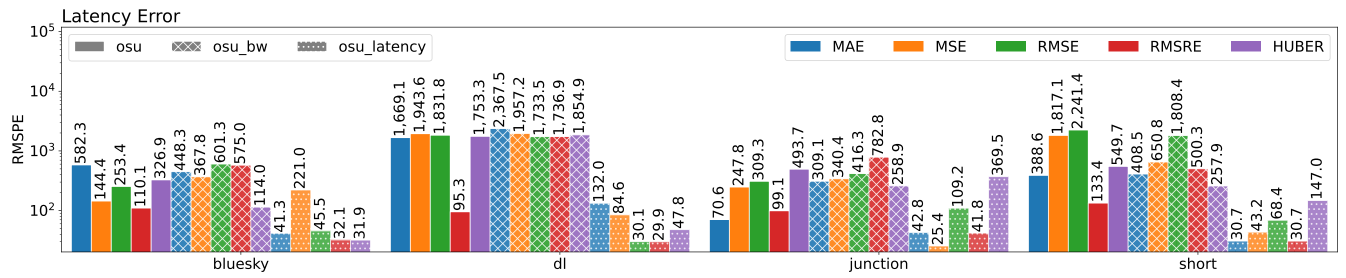


Fig. 9: Observed latency RMSPE for the tested loss functions per system. The error is relative to the latency benchmark.

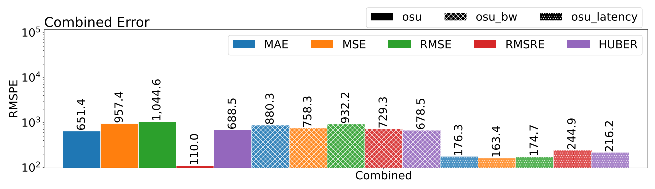


Fig. 10: Combined observed latency and bandwidth RMSRE for the tested loss functions across systems. The error is relative to both the latency and bandwidth benchmarks.

our optimized SNAPPR model (for both message regimes) as a reference. The tuned SNAPPR model outperforms the SST default model for all systems except for the Short system. We individually compare the error for small and large messages to shed light on this result. SNAPPR tuned for large messages is marginally better than our baseline. Conversely, the LogP model performs worse than our baseline SNAPPR model for small messages.

In contrast to the default SST model, we present the error if we bolt two tuned SNAPPR models together, trained for small and large messages, respectively. Listed as SNAPPR Low/SNAPPR High in Figure 17, we see it outperforms the baseline (the default SST model) on the Bluesky system (86.8% to 117.1%) and Short (161.3% to 151%). Note that we calculate this error from the relative error per message size. Unlike the SST default model, `SST/macro` does not currently support concurrent SNAPPR models.

VI. Related Work

The original paper introducing the preliminary implementation of the `SST/macro` [6] simulator applied uncertainty quantification (UQ) techniques to compare the predicted performance of various workloads running within the simulator and the actual execution time on a HPC system, Hopper. In particular, the authors employed Bayesian inference to quantify the simulation model error distribution for different MPI collective operations. Our work differs by exploring Bayesian optimization for multiple benchmarks at the same time. In addition, we provide an in-depth analysis of loss functions and benchmark selection. Another well-known HPC interconnect and storage simulator is the Co-Design of Multi-layer Exascale Storage Architecture (CODES) simulator [3]. Compared to the Booksim simulator [4], flit-level CODES delivers significantly faster simulation time while achieving comparable fidelity. To test and validate the simulation results for network interconnects, several guidelines have been suggested in [22], including a selection of different communication patterns (latency vs. bandwidth bound, adversarial), taking into consideration the effect of job placement, network interference, MPI eager and rendezvous protocols, MPI overheads etc..

In a preliminary work by Najafi et al. [23], the authors recently proposed to train a transformer-based model [24] with simulation input parameters and to use the trained model to predict the simulation time of computer systems, to avoid requiring full-fledged simulations during parameter sweeps. To improve the network traffic forecasting capability of Parallel Discrete Event Simulator (PDES), authors in [25] proposed two surrogate time series methods: the Auto-regressive Integrated Moving Average (ARIMA) and the Adaptive Long Short-Term Memory (ADP-LSTM).

Machine learning models have been also applied to analyze [26] and predict the performance of an application. For example, authors of [27] used supervised algorithms, specifically forests of randomized trees and gradient boosted regression trees to predict workload execution time under network congestion. In addition, machine learning based anomaly detection has been applied to diagnose performance variability [28].

VII. Conclusion and Future Work

This paper presented an auto-tuning framework for selecting the best matching configurations for complex network simulators modeling existing hardware and explored the construction

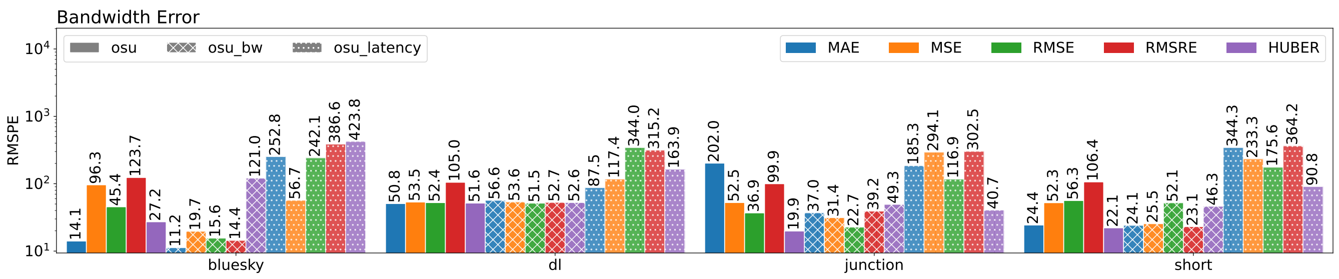


Fig. 11: Observed bandwidth RMSPE for the tested loss functions per system. The error is relative to the bandwidth benchmark.

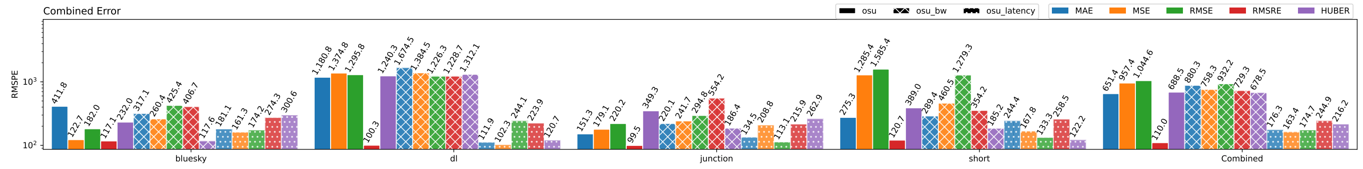


Fig. 12: Combined observed latency and bandwidth RMSRE for tested loss functions across systems. The error is relative to both the latency and bandwidth benchmarks.

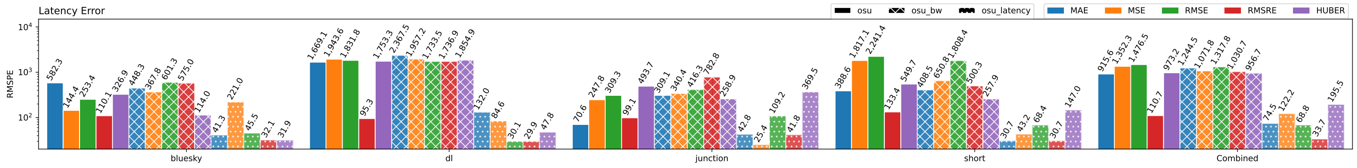


Fig. 13: Observed latency RMSPE for tested loss functions across systems. The error is relative to the latency benchmark.

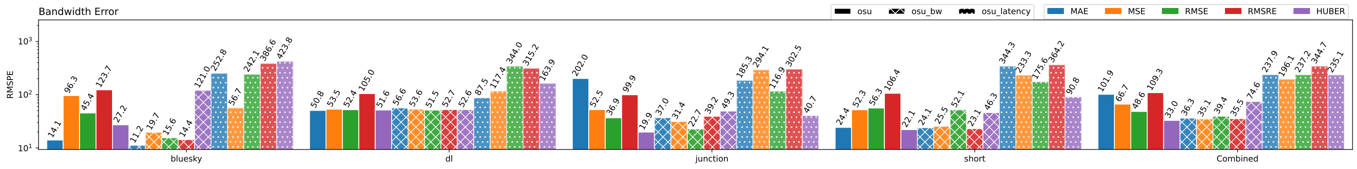


Fig. 14: Observed bandwidth RMSPE for tested loss functions across systems. The error is relative to the bandwidth benchmark.

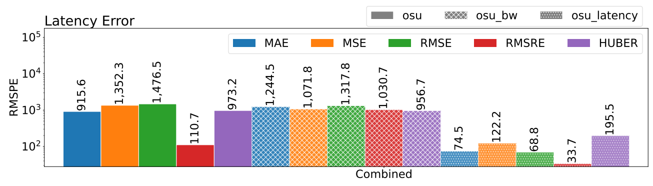


Fig. 15: Observed latency RMSPE for tested loss functions combined across multiple systems. The error is relative to the latency benchmark.

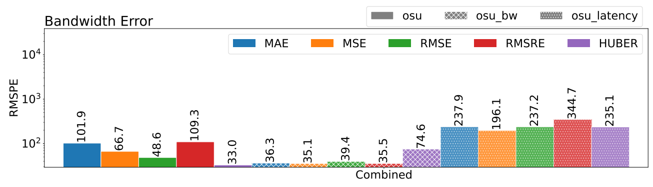


Fig. 16: Observed bandwidth RMSPE for tested loss functions combined across multiple systems. The error is relative to the bandwidth benchmark.

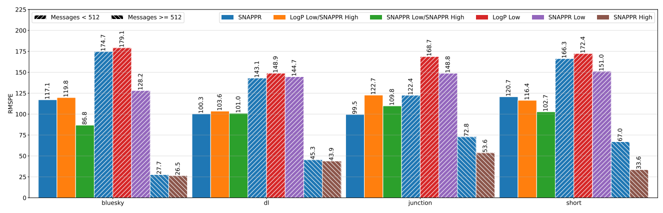


Fig. 17: The RMSPE for models bolted together.

of hybrid models. We demonstrate our approach's efficacy by improving the simulation's overall accuracy by more than **five** times based on RSMPE.

Our characterization of simulation error for multiple message regimes has highlighted the challenge of modeling *both* bandwidth and latency together. Traditionally, the community has relied on tools including `SST/macro` to explore large-message, bandwidth-bound applications where they have excelled in producing accurate simulations. However, the emergence of new domains and use cases pushes us to refine these models to include a broad spectrum of message regimes and adopt an automated framework to explore design spaces for the next generation of applications.

In the future, we envision the following next steps for this research: (1) providing better upper bounds for tuning current and experimental hardware, (2) exploring the propagation of error post-tuning from the minimal simulation unit to an entire topology, and (3) evaluating tuning for contention-based messaging regimes.

Acknowledgment

We thank the reviewers for their helpful feedback. This research used funding and resources supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation". The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

References

[1] J. Suetterlein, S. Young, J. Firoz, J. Manzano, R. Friese, N. Tallent, K. Barker, and T. Stavenger, "Automatic extraction of network configurations for realistic simulation and validation," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2024, pp. 310–312.

[2] J. Wilke *et al.*, "Structural Simulation Toolkit (SST) Macroscale Element Library," https://github.com/sstsimulator/sst-macro, 2023, accessed: 2023-01-05.

[3] M. Mubarak *et al.*, "Enabling parallel simulation of large-scale hpc network systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 87–100, 2016.

[4] N. Jiang *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*, IEEE. New Jersey, USA: Institute of Electrical and Electronics Engineers, 2013, pp. 86–96.

[5] D. K. Panda *et al.*, "MPI microbenchmarks from Ohio State University," https://mvapich.cse.ohio-state.edu/benchmarks/, 2023, accessed: 2023-01-05.

[6] J. J. Wilke *et al.*, "Validation and uncertainty assessment of extreme-scale hpc simulation through bayesian inference," in *Euro-Par 2013 Parallel Processing: 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings 19*, Springer. London, UK: Springer Nature, 2013, pp. 41–52.

[7] D. Krizanc and A. Saarimaki, "Bulk synchronous parallel: Practical experience with a model for parallel computing," in *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '96. USA: IEEE Computer Society, 1996, p. 208.

[8] D. D. Sharma, R. Blankenship, and D. S. Berger, "An introduction to the compute express link (cxl) interconnect," *arXiv preprint arXiv:2306.11227*, 2023.

[9] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 4.1*, Nov. 2023. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf

[10] W. Gropp, "Lecture 24: Buffering and Message Protocols ," https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture24.pdf, 2015.

[11] K. S. Hemmert *et al.*, "Evaluating trade-offs in potential exascale interconnect technologies," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States ..., Tech. Rep., 2020.

[12] S. Young, S. Aksoy, J. Firoz *et al.*, "Spectralfly: Ramanujan graphs as flexible and efficient interconnection networks," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE. New Jersey, USA: Institute of Electrical and Electronics Engineers, 2022, pp. 1040–1050.

[13] M. N. Newaz and M. A. Mollah, "Optimizing k-path selection for randomized interconnection networks," in *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, IEEE. New Jersey, USA: Institute of Electrical and Electronics Engineerings, 2021, pp. 222–231.

[14] K. Wen *et al.*, "Flexfly: Enabling a reconfigurable dragonfly through silicon photonics," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE. New Jersey, USA: Institute of Electrical and Electronics Engineers, 2016, pp. 166–177.

[15] 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," *SIGPLAN Not.*, vol. 28, no. 7, p. 1–12, jul 1993. [Online]. Available: https://doi.org/10.1145/173284.155333

[16] J. J. Wilke and J. P. Kenny, "Opportunities and limitations of quality-of-service in message passing applications on adaptively routed dragonfly and fat tree networks." 9 2020. [Online]. Available: https://www.osti.gov/biblio/1820287

[17] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 2623–2631.

[18] T. Yanase *et al.*, "Optuna: A hyperparameter optimization framework ," https://github.com/optuna/optuna, 2023, [Online; accessed 1-August-2023].

[19] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24. New York, USA: Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

[20] Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi, "Multiobjective tree-structured parzen estimator for computationally expensive optimization problems," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 533–541.

[21] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, p. 159–195, jun 2001.

[22] M. Mubarak *et al.*, "Toward reliable validation of hpc network simulation models," in *2017 Winter Simulation Conference (WSC)*, IEEE. New Jersey, USA: Institute of Electrical and Electronics Engineers, 2017, pp. 659–674.

[23] H. Najafi and X. Lu, "Deepsim: A transformer based model for fast simulation and exploring computer system design space," in *Proceedings of the 2023 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. New York, USA: Association of Computing Machinery, 2023, pp. 54–55.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[25] X. Xu *et al.*, "Machine learning for interconnect network traffic forecasting: Investigation and exploitation," in *Proceedings of the 2023 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. New York, USA: Association for Computing Machinery, 2023, pp. 133–137.

[26] J. J. Thiagarajan, R. Anirudh, B. Kailkhura, N. Jain, T. Islam, A. Bhatele, J.-S. Yeom, and T. Gamblin, "Paddle: Performance analysis using a data-driven learning environment," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 784–793.

[27] A. Bhatele, A. R. Titus, J. J. Thiagarajan, N. Jain, T. Gamblin, P.-T. Bremer, M. Schulz, and L. V. Kale, "Identifying the culprits behind network congestion," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2015, pp. 113–122.

[28] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Diagnosing performance variations in hpc applications using machine learning," in *High Performance Computing: 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18–22, 2017, Proceedings 32*. Springer, 2017, pp. 355–373.