Towards Portable Online Prediction of Network Utilization using MPI-level Monitoring

Shu-Mei Tseng, Bogdan Nicolae, George Bosilca, Emmanuel Jeannot, Aparna Chandramowlishwaran, Franck Cappello

Towards Portable Online Prediction of Network Utilization using MPI-level Monitoring

Shu-Mei Tseng[1], Bogdan Nicolae[2], George Bosilca[3], Emmanuel Jeannot[4], Aparna Chandramowlishwaran[1], and Franck Cappello[2]

[1] University of California Irvine, Irvine, USA
{shumeit,amowli}@uci.edu
[2] Argonne National Laboratory, Lemont, USA
{bnicolae,cappello}@anl.gov
[3] University of Tennessee Knoxville, Knoxville, USA
bosilca@icl.utk.edu
[4] INRIA Bordeaux, France
emmanuel.jeannot@inria.fr

Abstract. Stealing network bandwidth helps a variety of HPC runtimes and services to run additional operations in the background without negatively affecting the applications. A key ingredient to make this possible is an accurate prediction of the future network utilization, enabling the runtime to plan the background operations in advance, such as to avoid competing with the application for network bandwidth. In this paper, we propose a portable deep learning predictor that only uses the information available through MPI introspection to construct a recurrent `sequence-to-sequence` neural network capable of forecasting network utilization. We leverage the fact that most HPC applications exhibit periodic behaviors to enable predictions far into the future (at least the length of a period). Our online approach does not have an initial training phase, it continuously improves itself during application execution without incurring significant computational overhead. Experimental results show better accuracy and lower computational overhead compared with the state-of-the-art on two representative applications.

1 Introduction

Network bandwidth is a precious resource on High Performance Computing (HPC) systems to the point where much of the performance of the applications depend on it [13]. However, HPC applications typically use the network bandwidth at full capacity only sporadically. This opens a *window of opportunity* for runtimes and services (that these applications depend upon) to seamlessly perform operations that require communication over the network in the background. For example, many applications need to periodically checkpoint to a parallel file system, which might be subject to I/O bottlenecks and therefore negatively impacts performance and scalability. To avoid this, runtimes stage the checkpoints first to a local storage and then flushes them in the background to the parallel file system, hiding the I/O overhead from the application.

In order to take advantage of this window of opportunity, the checkpointing runtime needs to clearly identify the time intervals when the network is underutilized by the

application. Failing to do so might lead to competition for network bandwidth and could cause undesired interference that slows down the application (e.g. the flushes to the parallel file system pushed by the checkpointing runtime compete with the applications' data exchanges over the same network interfaces).

Therefore, to avoid the competition for network bandwidth, it is necessary to schedule all background operations in such a way that they finish within the window of opportunity. Unfortunately, the background operations are often non-trivial and take time to complete and/or cannot be easily suspended and resumed later (e.g. writes to the parallel file system are outside the control of the checkpointing runtime). Thus, it is important to be able to predict the network utilization sufficiently early to precisely pinpoint when and for how long the network bandwidth will stay underutilized to actually create a usable window of opportunity.

The problem of predicting network utilization is non-trivial for several reasons. First, it is tedious or impossible to obtain system-level information about the network utilization because most platforms and vendors expose it through non-standardized performance counters and APIs or do not expose it at all. Second, network utilization is challenging to reason about in an offline fashion due to the application complexity and a large number of variables (e.g. platform, input data, system noise, global resources shared with other users such as a parallel file system, etc) that influence the network utilization at runtime.

To address these challenges, we propose a solution that combines portable MPI-level monitoring of network utilization with deep learning based time series forecasting. The key novelty of our approach is two-fold: (1) we devise a mechanism to approximate network utilization using only the information available at the MPI-level (which addresses the portability challenge); (2) we introduce a periodicity-aware deep learning approach that adapts sequence-to-sequence predictors based on recurrent neural networks for adaptive online learning. This approach is capable of maintaining high prediction accuracy with low computational overhead despite variations encountered during runtime. Although the focus of this work is the prediction of network utilization, it is important to note that the basic ideas can be easily extended to predict the utilization of other resources such as CPU, I/O bandwidth, etc.

We summarize our contributions as follows: (1) we present a series of general design principles that summarize the key ideas behind our approach (Section 3); (2) we show how to materialize these design principles in practice by introducing an MPI-based network monitoring infrastructure (Section 3.2) and a framework to leverage sequence-to-sequence predictors efficiently in an online fashion (Section 3.4); (3) we evaluate our approach for two representative HPC applications and show significantly better prediction accuracy and lower computational overhead compared with state-of-the-art approaches (Section 4).

2 Related Work

MPI monitoring. There are many different ways to monitor the network utilization of an MPI application. The most common and generic way relies on intercepting MPI API calls of interest and delivering aggregated information. PMPI is a high-level customiz-

able profiling layer allowing tools to intercept MPI calls. Communication monitoring can be achieved by intercepting all MPI communications routines, including point-to-point, one-sided, and collectives. When such a communication routine is called, the processes involved (source and destination) as well as the amount of data involved in the transfer needs to be recorded. In addition to the overheads necessary to get information about the amount of data involved in communications, this approach cannot differentiate between point-to-point and collective data, as it is impossible to determine how the collective calls are implemented using point-to-point communications. One of the major advantages of PMPI is the existence of many stand-alone monitoring libraries, such as mpiP [27], Score-P [19], and DUMPI [3].

At a different level in the software stack, PERUSE [9, 18] allows the application to register callbacks that will be triggered at critical moments in the point-to-point request lifetime. This method provides opportunities to gather low-level information about MPI messages, including the number of unexpected messages, matching cost, payload type (i.e. point-to-point or collectives), etc. Unfortunately, this technique has failed to attract support from the MPI standardization body, and, as a result, support in widely used MPI implementations is almost non-existent.

Time Series Analysis/Forecasting. Traditional statistical models like Autoregressive Integrated Moving Average (ARIMA) have been widely used for the purpose of time series forecasting in the context of HPC applications. Prior work such as [24, 25] introduce a framework for online modeling and prediction of I/O operations to enable prefetching. Other efforts use ARIMA-based models to forecast CPU, memory, and network utilization to facilitate better resource allocation and load balancing [20].

Due to the successful application of deep learning techniques in various domains such as natural language processing [12] and language translation [17, 23] that require predictions of what elements are likely to follow in a sequence, such techniques are increasingly being considered in the context of time series forecasting. Sequence-to-sequence (seq2seq) models are particularly popular in this context. Kuznetsov and Mariet [21] provide a theoretical analysis and compare seq2seq with other classical time series models. Moreover, they also provide some quantitative guidance on how to choose different modeling approaches. One of the limitations of seq2seq is the predetermined output sequence length. Harmon and Klabjan [15] address this problem by making the network predict the dynamic length of the outputs. However, none of these approaches address the problem of efficient online learning. To the best of our knowledge, this paper is the first to address the problem of portable prediction of network utilization using online deep learning specifically tailored to the requirements of HPC applications.

3 System Design

In this section, we introduce the high-level design principles of our proposed approach, discuss the methodology, and provide a detailed description of the experimental prototype implemented to illustrate the benefits of our design on real applications.

3.1 Design Principles

Our system design is based on the following three principles.

Portable MPI-level monitoring. To solve the problem of portable monitoring, we propose to capture the network utilization directly from the communication library, in this context MPI. While we could get more information about the internal state of data transfers (one-sided and two-sided messages, as well as the state of all non-blocking communications) we restraint ourselves to the smallest subset of pertinent information: the number of bytes sent by each rank using MPI messages (which is of interest for HPC applications because it represents the majority of network traffic). We use this information to estimate the global network utilization imposed by the target application. However, having an accurate counting of the number of bytes sent is currently non-trivial to capture for two main reasons. (1) Messages exchanged by ranks do not necessarily go over the network (e.g., ranks co-located on the same node use shared memory). (2) Messages are not only generated by direct point-to-point communication initiated by the application but they are also generated by one-sided communications or collective operations (which often leads to complex patterns deep in the MPI library implementation). In addition, the number of bytes sent by MPI is a lower bound on the network utilization, as the network interface introduces additional overhead (headers, etc.). Section 3.2 details how we address these challenges.

Low-overhead online learning. Based on the network utilization estimates obtained at the MPI-level, we propose an adaptive online learning approach that continuously refines the quality of the predictions as more monitoring information becomes available. This approach has two advantages. (1) There is no need to perform separate training offline based on the network utilization observed in the previous runs (which may be difficult to obtain and/or unavailable if it is the first run). Therefore, it is robust to inaccurate predictions due to variations in the application configuration or input data used in subsequent runs which have a significant impact on the communication pattern. (2) It facilitates more accurate predictions by dynamically adapting to the variations that are naturally occurring during the same run (e.g. system noise, shared resources with other users such as a parallel file system, etc.).

Periodicity-aware forecasting. A large majority of HPC applications exhibit a repetitive communication pattern, which implicitly leads to a repetitive pattern of network utilization. Given the need to predict network utilization as far as possible in the future to enable background services to schedule their operations in advance, we argue that the most useful prediction needs to cover at least the duration of one period. To this end, we propose a periodicity-aware approach that employs a recurrent neural network specifically designed for online sequence to sequence forecasting. We discuss the proposed solution in detail in Section 3.4.

3.2 Portable MPI-level Monitoring

Our MPI-level monitoring is based on previous work to design a portable monitoring interface in OpenMPI [7]. We take advantage of the modular implementation of

OpenMPI [5], to add support for a dynamically activated communication monitoring module. This module can be activated at runtime and distinguishes between several types of MPI traffic such as point-to-point, one-sided, and collectives and creates a global heatmap by recording, for each rank, the number of bytes and the number of messages sent to any other rank. Note that the recording is done after the collectives have been decomposed into point-to-point messages, providing a more precise picture of overall transfers. Therefore, the monitoring sees the impact of the algorithm implementing the collective.

We design a high-level abstraction called *monitoring session* that integrates the capability of the new MPI_T tools support in MPI. Once created, a monitoring session can be started, stopped, resumed, and reset. Several sessions can simultaneously coexist, allowing for independent monitoring of different parts of the code. For the purpose of our monitoring needs, and in order to guarantee timely monitoring information each MPI rank launches, on initialization, a separate background thread that starts a monitoring session. At regular intervals (e.g., every second), this thread stops the monitoring session, reads the number of bytes sent by all ranks during the previous interval and then resets and resumes the session. Using this approach, we can obtain a history of the number of bytes sent per time unit for each rank.

However, to estimate the network utilization of a node, it is not enough to count the traffic for each process located on the node but instead we need to aggregate the number of bytes sent by each rank to other ranks that are not co-located on the same node. This is necessary because, at least in OpenMPI, co-located ranks use a different low-level communication substrate, i.e. shared memory for communication. To efficiently perform this aggregation, we create a 2-level hierarchy with local and remote peers and a designated leader on each node to aggregate the information from the other co-located ranks. This is done by creating a local MPI communicator on each node that includes all ranks sharing the node. The MPI process with the lowest rank in the local communicator become the leader, and is in charge of collecting the monitoring data. An MPI reduce operation collects the information from the other co-located ranks on the leader. Then, the leader sums up the number of bytes sent to all the ranks that do not have a corresponding rank in the local communicator. This way, only the bytes that need to pass over the network (and could therefore interfere with other operations that generate network traffic) are counted. This is an approximation of the network utilization per time unit which we subsequently use for forecasting.

3.3 Sequence-to-Sequence Recurrent Neural Networks (Seq2Seq)

Recurrent neural networks (RNNs) are a type of neural networks that contain loops. Unlike convolutional neural networks (CNNs), which are feed-forward (i.e., the information only passes through the network in one direction), these loops enable RNNs to capture sequence dependencies. However, conventional RNNs [6, 16] have a major limitation in the form of exploding/vanishing gradient in the training stage, which makes them unable to handle long-term dependencies accurately. To address this issue, long short term memory networks (LSTMs) [16] and gated recurrent unit networks (GRUs) [11] have been proposed. They are special types of RNNs that solve this issue

by controlling what information is propagated into its internal state and what information is forgotten.

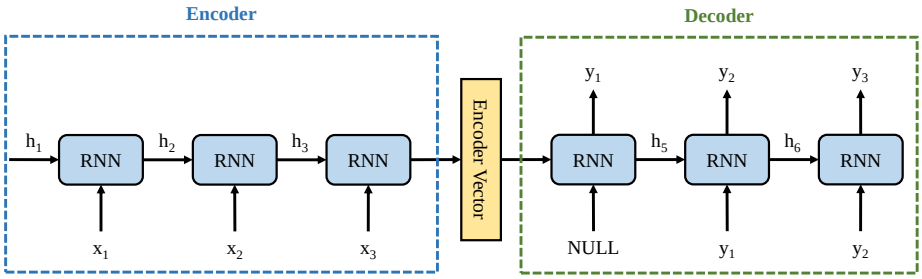


Fig. 1. Encoder-decoder diagram of the seq2seq model.

The sequence-to-sequence model (seq2seq) [23] is a particular instance of RNNs which can make use of LSTMs and GRUs as its recurrent units. Initially proposed in the context of natural language processing [23], the seq2seq model is now being used for a variety of other applications (e.g., speech recognition [10] and video captioning [26]). The model is composed of two components: an encoder and a decoder, as illustrated in Figure 1. The input sequence is fed into the encoder one element at a time (e.g., x_1, x_2, x_3). Each recurrent unit in the encoder is a typical recurrent neural network which computes its hidden state using the hidden state of its predecessor and the current input (e.g., h_3 is computed using h_2 and x_2). The last hidden state of the input sequence is known as a *thought vector* and is used as the initial state of the decoder. It aims to encapsulate all the information from the input sequence to make the prediction of the decoder more accurate. Unlike the recurrent units in the encoder, the recurrent units of the decoder use both the previous hidden state and the last predicted output to obtain the new hidden state (e.g., h_6 is computed using h_5 and y_1).

For the purpose of our work, we leverage the same idea for network utilization prediction. We train the model with recent utilization patterns that are represented as time series. The encoder is fed one part of the time series, while the decoder is fed the other part. After training the model with multiple such time series, it learns to "translate" from a recent history of observations into a likely future evolution.

3.4 Online Periodicity-Aware Forecasting using seq2seq RNNs

The key novelty of this paper is to adapt seq2seq for use as an online learning tool. This is a difficult problem because our model does not have a separate training phase (as is the case with traditional machine learning) and needs to learn on-the-fly as the application is progressing. This also places a strict requirement to be capable of continuously updating the model with low computational overhead.

To address this issue, we introduce the following approach, which is illustrated in Figure 2. A history of the network utilization that is large enough to cover the most recent h repetitive patterns is kept, where h is the history size. We call the time series corresponding to a repetitive pattern an *epoch*. Using h epoch as training input helps the

learning process account for potential variations between the most recent epochs. We assume the periodicity of the network utilization (and therefore the length of an epoch) is either known in advance or can be determined using an FFT-based approach (applied at key points during the application runtime when sufficient monitoring information is available, e.g., after the first checkpoint request).

The model is valid within the scope of a specific application run and starts with no initial history. After the history has accumulated two epochs, the initial training is performed by feeding the first epoch to the encoder and the second to the decoder. We perform this initial training over multiple iterations to reinforce this first pattern. At this point, we can make the first prediction of the third epoch.

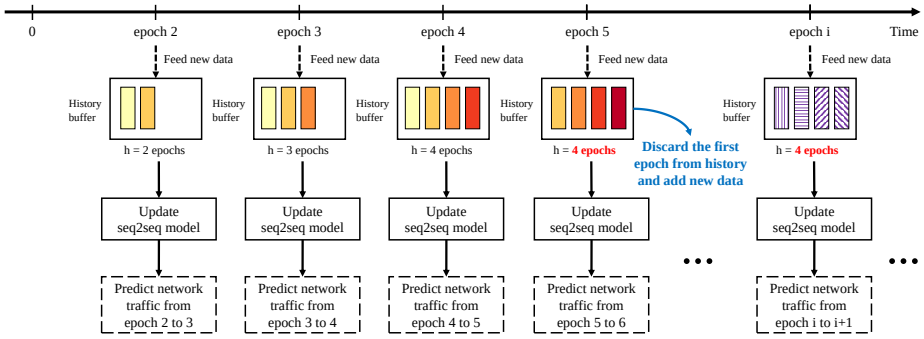


Fig. 2. Evolution of the online seq2seq predictor with the history size, $h = 4 \cdot epochs$.

Then, we wait until a new epoch is available and append this epoch to the history (which is truncated to keep only the last h epochs), as shown in Figure 2. As the application is progressing, the model is retrained using a sliding window learning approach. Specifically, we pass over the new history in increments of one time step using a window size equal to two epochs. For each such window, the first and second epoch are fed to the encoder and decoder, respectively, in a similar way as the initial training is performed. Again, to reinforce the pattern, we repetitively pass over the history for k iterations. k is determined in an adaptive fashion based on two criteria: (1) the loss of the current iteration is smaller than the loss of each of the last p iterations (to avoid oscillation around a local optimum and to avoid unnecessary computational overhead when the loss is small); (2) a predefined number of iterations q is reached (to avoid too much overhead when convergence is slow). The entire process is then repeated whenever a new epoch is available. Independent of online learning, the model can be used at any moment during the runtime to predict the next epoch.

3.5 Implementation Details

We implemented our approach on top of OpenMPI version 4.1.0a1, which includes support for low-level monitoring of bytes sent from one rank to every other rank. We implemented the monitoring session as a library that exposes a convenient high-level API. This is then used by a separate thread spawned in each MPI rank. To create a local communicator that includes all ranks co-located on the same node, we use

MPI_Comm_split_type using the MPI_COMM_TYPE_SHARED flag. To perform the aggregation on the leader, we use an in-place reduce operation on the local communicator. To find out what ranks are remote, the leader uses MPI_Group_translate_ranks. The online predictor described in Section 3.4 is implemented in Python and it uses TensorFlow 1.0 as the backend.

4 Experimental Results

4.1 Experimental Setup

We ran our experiments on the Grid'5000 testbed. For this paper, we use 16 nodes of the parapluie cluster. Each node is equipped with an AMD Opteron 6164 HE CPU (12 cores), 48 GB RAM, and two network interfaces: Intel 82576 1 Gbps Ethernet and Mellanox MT25418 20 Gbps Infiniband. We use the Infiniband network interface since it's a common high-end networking technology adopted on many supercomputing machines.

4.2 Methodology

To measure the effectiveness of our approach, we perform the following steps. First, we instrument an HPC application to monitor the network utilization at the MPI-level, using the approach described in Section 3.2.

Second, we run the application on all 16 nodes with a representative use case that generates an inter-node communication pattern specific to the application. We log the network utilization (expressed in MB/s) at the granularity of one second, creating a time series that includes both the value reported by the MPI-level monitoring, as well as the corresponding value reported by the performance counters available through the sys\class operating system interface (henceforth referred to as system-level).

Third, we take a representative log file from one of the nodes (all nodes exhibit similar behavior for the applications we study, which are detailed below) and simulate online learning based on it. We focus on three aspects: (1) the accuracy of the MPI-level monitoring vs. system-level monitoring; (2) the accuracy of the predictions that are made by online learning using MPI-level monitoring vs. actually observed system-level values; (3) computational overhead of online learning.

This process is illustrated in Figure 3. The log file contains the timestamp of the network utilization data, the node id, the number of bytes reported by the MPI-level monitoring approach, and the number of bytes obtained from system-level monitoring.

The accuracy is measured using two representative metrics widely used in time series analytic: *mean squared error* (MSE) and *dynamic time warping* (DTW). Both metrics quantify the distance between two time series, which in our case is the prediction vs. the actual system-level time series. For MSE, we use a standard implementation (available in the numpy library). For DTW, we use an optimized implementation (FastDTW) based on a linear algorithm [22].

We use two representative applications in our experiments: (1) HACC [14], a complex framework that simulates the mass evolution of the universe using particle-mesh

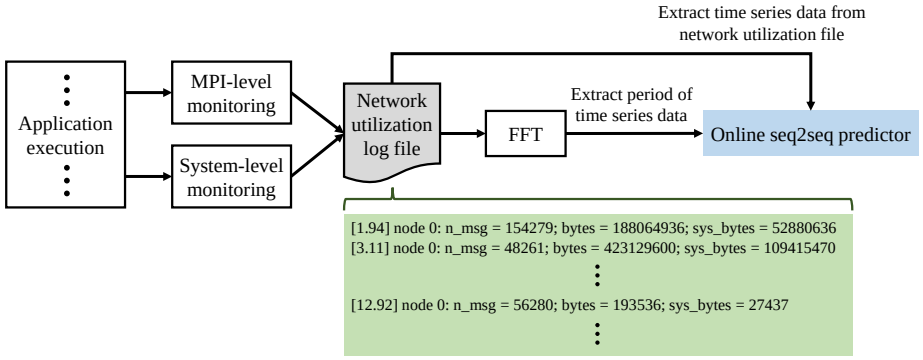


Fig. 3. Experimental methodology.

techniques. HACC splits the force calculation into a specially designed grid-based long/medium range spectral particle-mesh component that is common to all architectures, and an architecture-specific short-range solver. HACC generates a regular communication pattern, which is typical of a large class of HPC applications. (2) AMG [4], a parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids. It is derived directly from the BoomerAMG solver in the hypre library, a large linear solver library that is being developed in the Center for Applied Scientific Computing (CASC) at LLNL. AMG is part of the ECP proxy application suite [2] and exhibits a highly dynamic communication pattern that is difficult to predict. For more details, please consult the artifact that accompanies this paper [1].

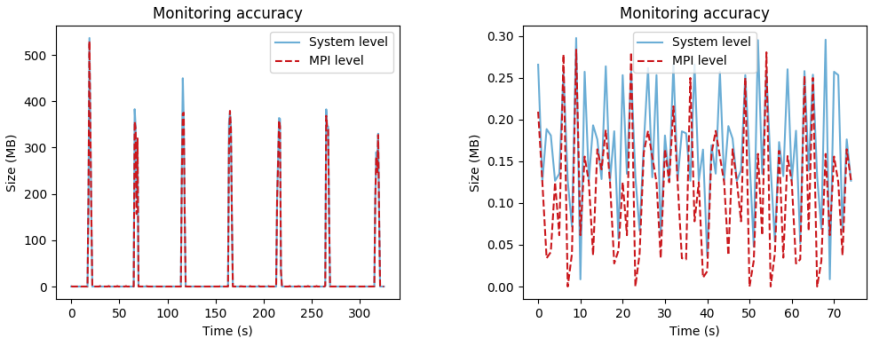
4.3 Monitoring Accuracy

Before being able to confidently use the data reported by the monitoring to train the RNN we need to quantify how accurate our MPI-level monitoring solution is compared to a system-level solution, in order to understand what trade-off is necessary to achieve the much desired portability that enables users to avoid implementing a custom monitoring solution specific for each platform.

To this end, we compare the time series from the log files in Figures 4(a) and 4(b). As we can observe visually, for HACC (Figure 4(a)) the difference between MPI-level and system-level is negligible. On the other hand, for AMG (Figure 4(b)) there are slight discrepancies introduced by delays between the moment when MPI queues messages to be sent to the network interface and the moment when the network interface actually sends them. Given the high dynamicity of the communication pattern, this is expected.

Table 1. Mean squared error and fast dynamic time warping of MPI- vs. system-level network utilization (lower is better). Normalized version included for easier comparison (lower is better).

*A*pplication	MSE	FastDTW	Norm-MSE	Norm-FastDTW
HACC	36.35	376.75	0.0001	0.59
AMG	0.0074	4.57	0.07	14.23



(a) HACC (b) AMG

Fig. 4. Monitoring accuracy: MPI- vs. system-level network utilization measured every second.

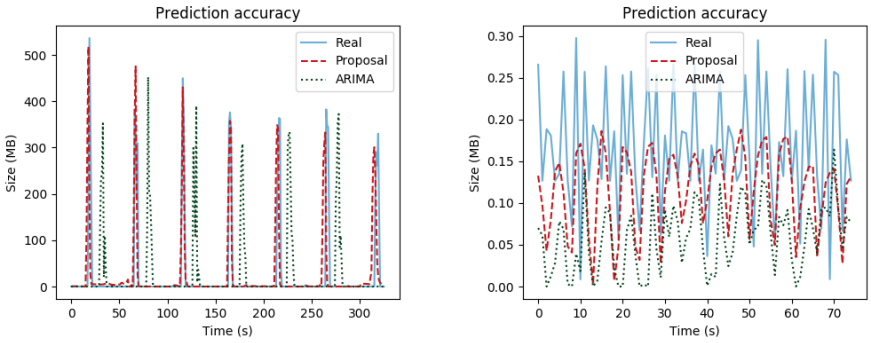
Quantitatively, Table 1 details the MSE and FastDWT for both applications, both in raw and normalized form. The normalized form is calculated by scaling the values of the time series to the interval $[0,1]$. As expected, it reveals a much better accuracy for HACC than for AMG. The raw form is interesting to note for subsequent comparison with the accuracy of the prediction, which is based on the MPI-level monitoring and thus subject to the errors introduced by it.

4.4 Prediction Accuracy

Based on the accuracy of the collected monitoring data, we can study how accurate the predictions of our proposed approach (henceforth referred to as OnlineS2S) is compared with the actual values reported at the system-level. To this end, we simulate online learning as follows. First, we determine the periodicity of the communication pattern (as discussed in Section 3.4). For HACC, the periodicity is 60 seconds, while for AMG, the periodicity is 20 seconds. Then, we set the $epoch$ for training of our model to be equal to the periodicity. Our goal is to successfully predict one epoch in advance at every moment during the application runtime. To achieve this, we adopt the following approach: for each timestamp t in the time series, we predict the network utilization at $t + epoch$, then update the history and the model as detailed in Section 3.4. Then, we plot the resulting time series together with its system-level counterpart. We fix $p = 5$, $q = 100$, and $h = 5 \cdot epochs$.

We compare our approach against ARIMA [8], a popular method used in time series forecasting that combines an autoregressive (AR) with a moving average (MA) model. We also adopt the sliding window approach for ARIMA, updating the model and history as t increases. We use a standard implementation of ARIMA that is available as part of the statsmodel Python package.

The results are shown in Figures 5(a) and 5(b) where the superior quality of the prediction of OnlineS2S vs. ARIMA is clearly visible. In the case of HACC (Figure 5(a)), the spikes are accurately predicted by our approach both in terms of time and amplitude. On the other hand, ARIMA exhibits a delay in the prediction of the spikes, which means a background service relying on such predictions will incorrectly assume the



(a) HACC (b) AMG

Fig. 5. Prediction accuracy: Estimated network utilization one epoch in the future (OnlineS2S vs. ARIMA) compared with actual system-level utilization measured at the same moment.

application will not communicate when it actually does, potentially scheduling its own network I/O at the same time and therefore causing interference. Also, the amplitude of the predictions exhibits noticeable inaccuracies. In the case of AMG (Figure 5(b)), both predictions show a visible under-estimation of the network utilization. However, in the case of ARIMA, the under-estimation is significantly larger.

Table 2. Mean squared error and fast dynamic time warping of OnlineS2S and ARIMA predicted network utilization vs. actual system-level utilization (lower is better). Relative improvement of OnlineS2S vs. ARIMA included for easier comparison (higher is better).

*H*ACC	MSE	FastDTW
OnlineS2S	6194	2737
ARIMA	14433	4344
Relative	2.3×	1.6×

*A*MG	MSE	FastDTW
OnlineS2S	0.00797	4.77
ARIMA	0.0168	7.14
Relative	2.11×	1.5×

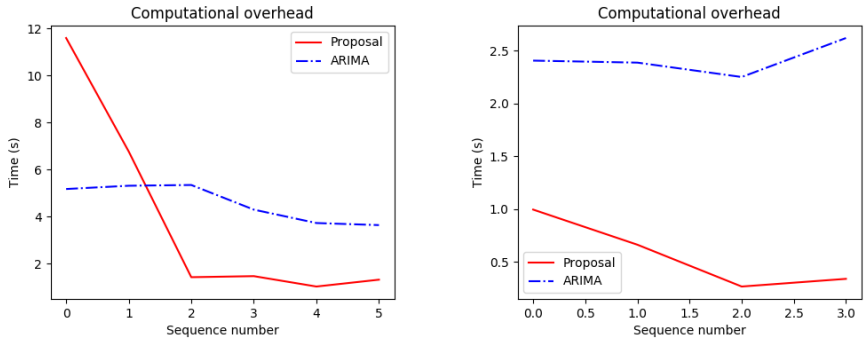
Table 2 shows the MSE and FastDTW for both applications. In addition to the raw values, we calculate the relative improvement (values for ARIMA divided by values for OnlineS2S) for easier comparison. As we can observe, OnlineS2S has more than 2× smaller MSE and 1.5× smaller FastDTW. Thus, our approach consistently outperforms ARIMA in both typical and highly dynamic HPC network utilization scenarios.

4.5 Computational Overhead

Our last study focuses on the computational overhead required to perform the online learning during the application runtime. This is an important aspect, because online learning may cause interference with the CPU utilization of the application.

To estimate the severity of the interference, we record the time required to update the model as we pass from one epoch to another (which we refer to as sequence number). In the worst case scenario, the application will use the CPUs at 100% for the entire

duration of the epoch. Assuming that the update of the model will also use the CPUs at 100%, the worst case overhead is the time required for the update divided by the length of the epoch.



(a) HACC (b) AMG

Fig. 6. Computational overhead: Time required to process an epoch (lower is better).

The time needed for each epoch are shown in Figures 6(a) and 6(b). For HACC, OnlineS2S has a higher initial overhead but quickly stabilizes after two epochs and is consistently 2.5× faster than ARIMA. Since the epoch is 60 s in this case, this means OnlineS2S can achieve a worst-case overhead of less than 3%, whereas ARIMA is closer to 7%. In the case of AMG, OnlineS2S is much faster from the beginning and stabilizes at a point where it is at least 5× faster than ARIMA. Since the epoch is 20 s in this case, the worst-case overhead for OnlineS2S is 2.5% and more than 10% for ARIMA. With such high worst-case overhead, we conclude that ARIMA may be unfeasible to adopt for online prediction, especially for applications that exhibit small epochs.

5 Conclusions

This paper introduced an online prediction approach for network utilization specifically designed for HPC applications that exhibit periodic communication behavior. It is based on the idea of combining a mechanism to approximate network utilization at the MPI-level in a portable fashion with a deep learning approach that adapts sequence-to-sequence predictors based on recurrent neural networks for adaptive online learning.

We evaluated the accuracy and computational overhead of our approach experimentally on two representative HPC applications. We show that our approach is consistently, at least twice as accurate and at least twice as fast compared with state-of-the-art prediction approaches based on traditional time series analysis.

Encouraged by these results, we plan to broaden the scope of our work in future efforts. Specifically, there are several promising directions. First, we will run new experiments to measure the actual computational overhead of online learning when integrated with the HPC applications (as opposed to the worst case scenario we studied

in this paper). Second, we will evaluate the actual benefits of leveraging predictions of network utilization to improve asynchronous checkpointing.

Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This material was based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357, and by the National Science Foundation under Grant No. #1664142. The experiments presented in this paper were carried out using the Grid'5000/ALADDIN-G5K experimental testbed, an initiative of the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see http://www.grid5000.fr/).

References

1. Accompanying artifact. https://doi.org/10.6084/m9.figshare.8491058
2. ECP proxy applications project. https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/
3. Adalsteinsson, H., Cranford, S., Evensky, D.A., Kenny, J.P., Mayo, J., Pinar, A., Janssen, C.L.: A simulator for large-scale parallel computer architectures. International Journal of Distributed Systems and Technologies 1(2), 57–73 (2010)
4. Baker, A.H., Falgout, R.D., Kolev, T.V., Yang, U.M.: Multigrid smoothers for ultraparallel computing. SIAM Journal on Scientific Computing 33(5), 2864–2887 (2011)
5. Barrett, B., Squyres, J.M., Lumsdaine, A., Graham, R.L., Bosilca, G.: Analysis of the component architecture overhead in Open MPI. In: EuroPVM/MPI'05: 12th European Parallel Virtual Machine and Message Passing Interface Users Group Meeting. pp. 175–182. Sorrento, Italy (2005)
6. Bengio, Y., Simard, P., Frasconi, P., et al.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks 5(2), 157–166 (1994)
7. Bosilca, G., Foyer, C., Jeannot, E., Mercier, G., Papauré, G.: Online dynamic monitoring of MPI communications. In: EuroPar'17: 23rd International European Conference on Parallel and Distributed Computing. pp. 49–62. Springer, Santiago de Compostella, Spain (2017)
8. Box, G.E., Jenkins, G.M., Reinsel, G.C., Ljung, G.M.: Time series analysis: forecasting and control. John Wiley & Sons (2015)
9. Brown, K.A., Domke, J., Matsuoka, S.: Tracing data movements within MPI collectives. In: EuroMPI'14: Proceedings of the 21st European MPI Users' Group Meeting. pp. 117:117–117:118. Kyoto, Japan (2014)
10. Chiu, C.C., Sainath, T.N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R.J., Rao, K., Gonina, E., et al.: State-of-the-art speech recognition with sequence-to-sequence models. In: ICASSP'18: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing. pp. 4774–4778. Calgary, AB, Canada (2018)
11. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: EMNLP'14: 2014 Conference on Empirical Methods in Natural Language Processing. pp. 1724–1734. Doha, Qatar (2014)

12. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. Journal of machine learning research 12(Aug), 2493–2537 (2011)
13. Gerber, R., Hack, J., Riley, K., Antypas, K., Coffey, R., Dart, E., Straatsma, T., Wells, J., Bard, D., Dosanjh, S., et al.: Crosscut report: Exascale requirements reviews, march 9–10, 2017–tysons corner, virginia. an office of science review sponsored by: Advanced scientific computing research, basic energy sciences, biological and environmental research, fusion energy sciences, high energy physics, nuclear physics. Tech. rep., Oak Ridge National Lab. (ORNL) (2018)
14. Habib, S., Morozov, V., Frontiere, N., Finkel, H., Pope, A., Heitmann, K.: HACC: Extreme scaling and performance across diverse architectures. In: SC'13: 2013 International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 1–10. Denver, USA (2013)
15. Harmon, M., Klabjan, D.: Dynamic prediction length for time series with sequence to sequence networks. arXiv preprint arXiv:1807.00425 (2018)
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
17. Jean, S., Cho, K., Memisevic, R., Bengio, Y.: On using very large target vocabulary for neural machine translation. In: ACL-IJCNLP'15: 53rd Annual Meeting of the Association for Computational Linguistics and 7th International Joint Conference on Natural Language Processing. pp. 1–10. Beijing, China (2015)
18. Keller, R., Bosilca, G., Fagg, G., Resch, M., Dongarra, J.J.: Implementation and usage of the PERUSE-interface in Open MPI. In: EuroPVM/MPI'06: 13th European Parallel Virtual Machine/Message Passing Interface Users Group Meeting. pp. 347–355. Bonn, Germany (2006)
19. Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., et al.: Score-p: A joint performance measurement runtime infrastructure for periscope, scalasca, TAU, and vampir". In: 5th International Workshop on Parallel Tools for High Performance Computing. pp. 9–91. Dresden, Germany (2012)
20. Kumar, A.S., Mazumdar, S.: Forecasting HPC workload using ARMA models and SSA. In: ICIT'16: 2016 International Conference on Information Technology. pp. 294–297. Bhubaneswar, India (2016)
21. Kuznetsov, V., Mariet, Z.: Foundations of sequence-to-sequence modeling for time series. arXiv preprint arXiv:1805.03714 (2018)
22. Salvador, S., Chan, P.: Toward accurate dynamic time warping in linear time and space. Intelligent Data Analysis 11(5), 561–580 (2007)
23. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: NIPS'14: 27th Annual Conference on Neural Information Processing Systems. pp. 3104–3112. Montreal, Quebec, Canada (2014)
24. Tran, N., Reed, D.A.: ARIMA time series modeling and forecasting for adaptive I/O prefetching. In: ICS'01: Proceedings of the 15th international conference on Supercomputing. pp. 473–485. Sorrento, Italy (2001)
25. Tran, N., Reed, D.A.: Automatic ARIMA time series modeling for adaptive I/O prefetching. IEEE Transactions on Parallel and Distributed Systems 15(4), 362–377 (2004)
26. Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., Saenko, K.: Sequence to sequence-video to text. In: ICCV'15: 2015 IEEE International Conference on Computer Vision. pp. 4534–4542. Santiago, Chile (2015)
27. Vetter, J.S., McCracken, M.O.: Statistical scalability analysis of communication operations in distributed applications. In: PPoPP'01: Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming. pp. 123–132. Snowbird, Utah, USA (2001)