Optimizing Asynchronous Multi-Level Checkpoint/Restart Configurations with Machine Learning

Tonmoy Dey, Kento Sato, Bogdan Nicolae, Jian Guo, Jens Domke, Weikuan Yu, Franck Cappello, Kathryn Mohror

Optimizing Asynchronous Multi-Level Checkpoint/Restart Configurations with Machine Learning

Tonmoy Dey*, Kento Sato†, Bogdan Nicolae‡, Jian Guo†,
Jens Domke†, Weikuan Yu*, Franck Cappello‡, Kathryn Mohror§
*Florida State University, USA
†RIKEN Center for Computational Science, Japan
‡Argonne National Laboratory, USA
§Lawrence Livermore National Laboratory, USA

***Abstract*—With the emergence of versatile storage systems, multi-level checkpointing (MLC) has become a common approach to gain efficiency. However, multi-level checkpoint/restart can cause enormous I/O traffic on HPC systems. To use multi-level checkpointing efficiently, it is important to optimize checkpoint/restart configurations. Current approaches, namely modeling and simulation, are either inaccurate or slow in determining the optimal configuration for a large scale system. In this paper, we show that machine learning models can be used in combination with accurate simulation to determine the optimal checkpoint configurations. We also demonstrate that more advanced techniques such as neural networks can further improve the performance in optimizing checkpoint configurations.**

***Index Terms*—Machine Learning, Neural Network, Multi-Level Checkpointing (MLC)**

I. Introduction

Current petascale systems in High-performance computing (HPC) deal with enormous workloads. Such workloads require a massive number of components simultaneously. Though the HPC systems are built using highly reliable components, with the sheer number of components, there is an increasing frequency of component failures, resulting in degradation of system and application reliability. To reliably run applications on such large-scale systems, a common technique is checkpoint/restart (CR)[17], in which the system writes a snapshot of the application's state at fixed intervals to persistent storage, which is called a checkpoint. Application states can later be restored to the last saved checkpoint in case of a failure. Though CR is useful for large scale systems, its overhead can be an enormous challenge on large-scale ssystems.

One of approaches to reduce the overhead of CR is to determine the optimal checkpoint interval and checkpoint count. Poorly determined checkpoint interval makes system resilience worse. There are two approaches for obtaining the optimal checkpoint interval and checkpoint count values for any given configuration, namely the modeling approach [12] and the simulation approach [16]. The modeling approach mainly formulates an analytical solution to obtain optimal values, whereas the simulation approach runs the application across multiple failures to check different scenarios for obtaining the optimal values.

In simple checkpoint models [17] with synchronous checkpointing, where the checkpointing process mainly comprises of compute, checkpoint, and recovery state in a serial manner without other concurrent operations in the background. such modeling is beneficial in formulating an analytical solution for the optimal interval. However, with fast local storage becoming commercially available, asynchronous multi-level checkpointing (Async-MLC)[16] has become a common approach for efficient checkpointing. As shown in Figure 1, in Async-MLC, compute, checkpoint and recovery occur asynchronously from the application computation. This allows most of the checkpoint operations to happen in the background without delaying the critical compute operations in the applicaiton, thus minimzing the checkpoint overhead. This is particularly attractive when a parallel file system (PFS) is equipped with multi-level storage devices, which allows the checkpointing operations to happen on lower-level storage devices while the application I/O continues with upper-level storage devices.

Although introduction of multi-level checkpointing has greatly improved its performance over the simple model, however without the optimal interval and checkpoint count configuration, the performance of the system will still be degraded due to the checkpoint overhead. To obtain the optimal configuration of checkpoint interval and checkpoint count in MLC, the modeling approach is ineffective as it faces significant difficulty to formulate an analytical solution unless we simplify the model and/or make strong assumptions. Another approach is simulation. It is very effective in determining the optimal checkpoint configuration, however, it is very slow as it runs thousands of scenarios before obtaining the optimal values and this makes the simulation approach impractical for real world usage.

In this paper, we try to obtain the optimal checkpoint configuration for a given HPC system using the effectiveness and accuracy of the simulation approach while reducing the time taken by simulation to obtain the optimal result. We achieve this by combining machine learning methods with the

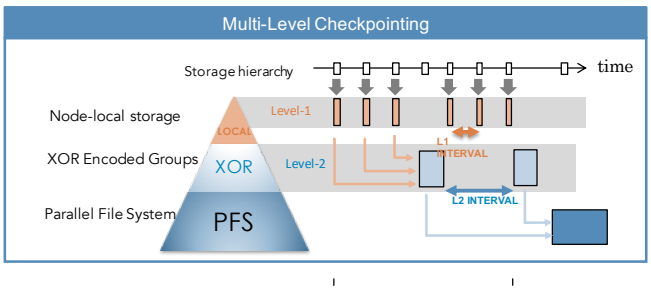

Fig. 1: Multi Level Checkpointing

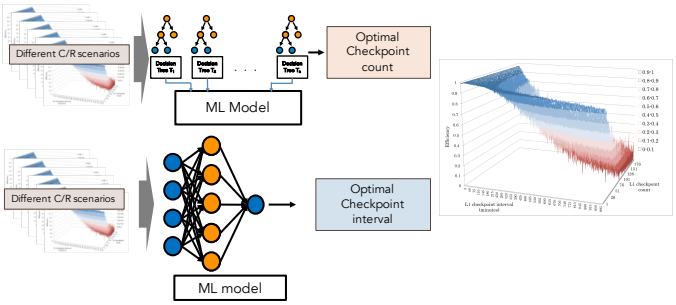

Fig. 2: Combining Simulation Approach with Machine Learning

simulation approach where we apply our machine learning models on existing simulated data to learn from the existing data and obtain the optimal checkpoint configuration with minimal error. The major contributions of this paper are:

- Development of multi-level checkpoint simulator to replicate the behavior of real world large scale systems.
- Novel machine learning (ML) approach that achieve convergence of traditional ML (Random forest) and state-of-the-art ML (NN), i.e. daisy chaining.
- Novel pre-processing for neural network (NN) to optimize CR i.e., parameter reduction. (Parameter correlation analysis is universal to other optimization area in general. Our work gives one instance in CR)
- Quantitative evaluation: Random Forest v.s. LightGBM v.s. Baseline NN v.s. NN after daisy chaining v.s. NN after parameter reduction with daisy chaining

With our approach and design optimizations, we show that our models can predict the optimized parameter values when trained with the simulation approach. We also show that using more advanced deep neural network techniques can improve the performance of neural network over the machine learning models by up to 50%. which can further be used in future research works to optimize the checkpoint restart configuration of large scale systems.

II. Background and Motivation

Checkpoint and restart is a commonly used technique for large scale systems running for a long time. Checkpoint is a snapshot of an application's state taken periodically and stored on to the available storage devices and it is used to restart the application when a failure occurs. Though CR is very important for large scale systems, however it is also one of the major contributors to the slowdown because of input output (I/O) workloads in HPC systems. To improve the efficiency of such systems, it is very crucial to use the optimal CR configuration[8]. Checkpoint interval is one of the most important checkpoint parameter that can configured to optimize the performance of large scale systems. Finding the optimal interval is very important as shorter interval will lead to frequent checkpoints where the system will spend more I/O time for saving checkpoints whereas longer interval can lead a less resilient system where a huge number of computation may be lost during a failure. Even with state of the art CR techniques, poorly determined checkpoint intervals can make the system less resilient.

For simple checkpoint models where execution states can be categorized into compute, checkpoint and recovery state without any overlapping, determining the optimal checkpoint interval can be relatively simple. However, this approach to checkpointing is inefficient due to its synchronous manner of execution and with the emergence of fast local storage such as non volatile memory (NVM), multi-level checkpointing (MLC) has become a standard approach for efficient checkpointing. In MLC, the system stores checkpoints on the local node storage in addition to the parity node storage and the PFS. Unlike the simple checkpoint model, the execution takes place in an asynchronous manner (Async MLC) such that all operations can continue while saving the checkpoint in the background. For Async MLC, in addition to the checkpoint interval it is also very important to determine the optimal checkpoint count for each checkpoint levels.

There are mainly two approaches for optimizing the configuration of CR, namely modeling and simulation approach. However, with Async MLC, CR becomes more complicated due to the additional levels of checkpointing and the complicated scenarios that could arise from such design. It becomes very challenging for the existing approaches to determine the optimal configuration as the design of the system becomes more complex. The modeling approach is not accurate for such complicated CR scenarios and cannot provide accurate results, and though the simulation approach is accurate, it is very slow due to the large number of scenarios it has to simulate to obtain the optimal configuration. In this paper, we focus on maintaining the accuracy of the simulation without spending the performance of CR. We achieve this by predicting the optimized checkpoint count and interval for large scale systems using machine learning on the data generated by the simulator as shown in Figure 2.

III. Non-Blocking Checkpoint/Restart Simulator

Our multi-level checkpointing simulator is designed to replicate the behavior of real-world large scale systems for managing checkpoints across different levels of the storage hierarchy. The simulator supports three levels of checkpointing schemes, LOCAL, PARITY, and PFS. The first level

of checkpointing is LOCAL in which the system saves the checkpoints on the storage in the local node and no redundant data stored across other nodes. In the PARITY scheme, the systems write checkpoints to the local storage along with the storage of other parity nodes. The redundant data are stored on the parity nodes so that the system can recover a copy of the file, provided a node, and its parity nodes do not fail simultaneously. Reliability is further enhanced, some systems encode the checkpoint data across the parity nodes so that the resiliency of the system can be enhanced. One of the standard encoding schemes is XOR encoding, where the redundant data are collectively written to the parity nodes to withstand single node failures in a group. However, it can only restart successfully if two or more nodes from the same parity group do not fail at the same time. Another one is Reed-Solomon encoding, which allows the system to withstand failures as long as half the nodes in the parity group do not fail simultaneously. Unlike the other two schemes, in the PFS parity scheme the system saves and restores the checkpoints from the PFS. The PFS scheme is only required when there is a catastrophic failure where a large number of nodes fail simultaneously requiring the entire system to be restored from the PFS.

Our simulator stores the checkpoint data at different storage level, each of which may have a different cost and level of resilience. Each of the checkpointing storage is a level, for which lowest level or level 1 checkpoints are the least resilient with minimum overhead, while storing the checkpoints in PFS is the most resilient with maximum overhead. In our simulator, a level L failure is a failure that requires a checkpoint at level L or higher for restarting while a level L recovery refers to the process of restarting an application using a checkpoint saved at level L. Since lower level failures such as process failures occur more frequently as compared to the higher level failure such as PFS failure, the system records one or more lower level checkpoints for every higher level checkpoint.

The performance of the checkpointing system is dependent on several characteristics of the system such as the overhead of saving the checkpoint to different levels of storage and also the cost of recovering from these storage levels. The overhead of saving and recovering the checkpoint from different levels of storage are not the only factors to take into account, the performance is also dependent on how frequently does each of these storage levels fail as failure of a checkpoint level causes the system to lose compute time and delays the progress. Considering all these characteristics of a system, our simulator was developed. It simulates multiple failures using different configurations to determine the optimal interval and checkpoint count for the different levels. This simulator can provide an overview of multi-level checkpoint systems for current and future systems and motivate systems to optimize checkpoint configuration that provide adequate overall reliability and efficiency. The simulator can be used to optimize performance of a given multi-level checkpointing implementation on a specific HPC system. For a given configuration of a large scale system, we simulate the performance of the system across multiple failures. The simulator modulates the checkpoint count and interval and provides the optimized interval and count or minimizing the overhead of a particular configuration, accounting for delays due to checkpoints, failures, and restarts.

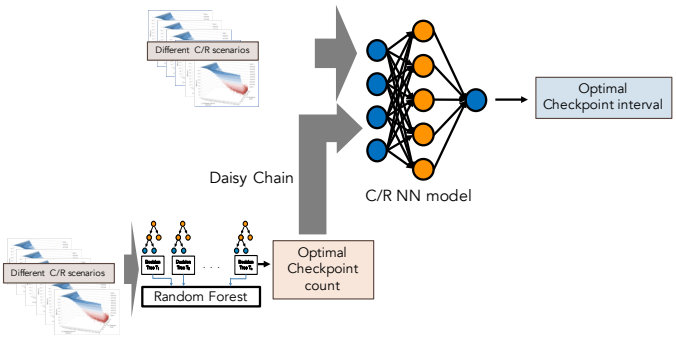

Fig. 3: Daisy Chaining output from checkpoint count prediction as an input to the neural network for checkpoint interval prediction

IV. Machine Learning for Checkpoint/Restart

The count and interval are two of the most important checkpoint parameters for optimizing the CR configuration. For determining the optimized checkpoint interval, the simulation approach is accurate but it can be slow as it explores different CR parameters before determining the optimized checkpoint interval and because of this, it is not a practical approach for real world scenarios such as while submitting a job. Our objective was to determine the optimized interval faster than the simulation approach without losing much of its accuracy. For our approach, we generated a limited amount of data using the simulator which provides us with optimized interval and count for the provided CR configurations and we use that data to determine the optimized count and interval for other CR configurations using the AI techniques described in the following sections.

A. Machine Learning

After running simple machine learning models such as linear and polynomial regression on the simulated data, we observed that these models were not able to learn from the data and provided inaccurate predictions for checkpoint count and interval. However, running more complex machine learning models for predicting checkpoint count and checkpoint interval showed that the optimized count can be predicted very accurately without much modification to the algorithms. Using different machine learning algorithms on the simulated data, we got the best results for predicting the checkpoint count using Random forest, Support Vector Clustering and Gaussian Naive Bayes. While random forest algorithm performed best among the different machine learning models for predicting the checkpoint interval.

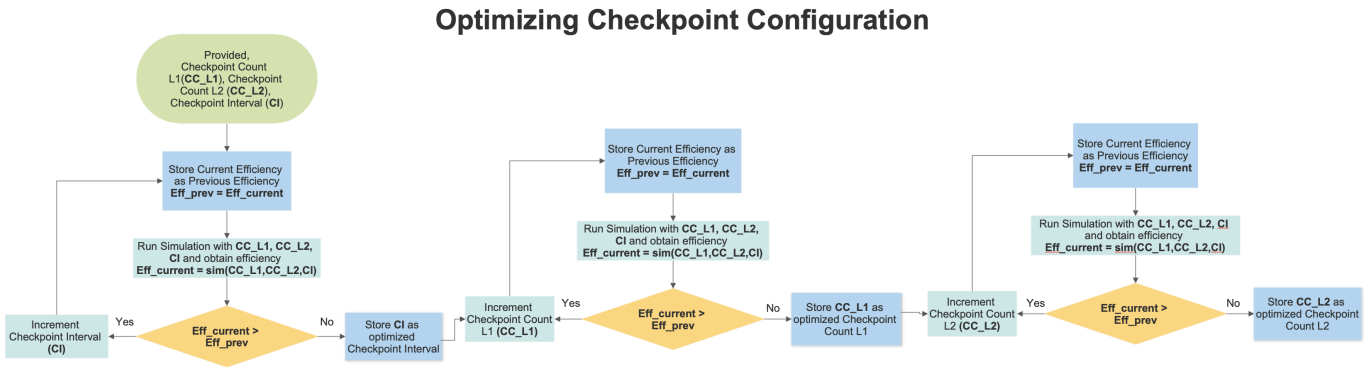

Fig. 4: Workflow for Optimizing Checkpoint Configuration

B. Neural Network

Since the random forest model could predict the optimized checkpoint count for the two level checkpoint model with an accuracy greater than 99%, we did not design a neural network for predicting checkpoint count. We designed a neural network only for predicting the optimized checkpoint interval, to obtain better performance than the machine learning models. For designing our neural network, we started with a simple neural network consisting of two hidden layers with 10 nodes on each hidden layer and ReLU as the activation function. The neural network showed an increase in error by a factor of 2.5 in comparison to the machine learning models for predicting the optimized interval. To improve the performance without significantly increasing training time, we added an extra hidden layer and increased the node on each hidden layer to 25. Though adding a layer and more nodes improved the performance, however, this baseline model still predicted the optimized checkpoint interval with an error that was greater than two times the error shown by the machine learning models. To improve the performance of this baseline model, we optimized its design using the techniques described in the following sections.

1) Daisy Chaining: Our initial design, consist of two independent models, the random forest model for predicting the checkpoint count and the neural network for predicting checkpoint interval. However, our analysis shows a strong correlation between the checkpoint count and interval with interval being inversely proportional to the checkpoint count. Since our random forest model was able to predict the checkpoint count very accurately, in order to improve the performance we fed the output from checkpoint count prediction as an input to the neural network as shown in Figure 3. The optimization reduced the error by 40% over the baseline model.

2) Parameter Reduction: After further analysis we observed some of the input parameters were inter dependent on each other and were impacting the performance of the neural network. On removing these input parameters, we observed a further reduction in error by 40% over the daisy chained model.

V. Implementation

The simulator has been developed to replicate the behavior of real-world scenarios when using a multi-level checkpoint for large scale systems. The simulator is provided with three critical parameters for each level, checkpoint overhead, checkpoint restart time, and failure rates. The checkpoint overhead of each level corresponds to the delay incurred when a checkpoint is stored at the desired level. Though the checkpoint storing mechanism is asynchronous and the majority of it is performed in the background, there still remains a fragment of time during which other useful operations need to be halted to perform the checkpointing. The checkpoint restart time refers to the time taken to retrieve the checkpoint data from the last stored checkpoint at the desired level, and failure rates provide how frequently the checkpoint levels fail.

The checkpoint overhead, restart time, and failure rates are used by the simulator as shown in Figure 4 to provide the user with elapsed time and the efficiency of the system. The efficiency corresponds to the amount of time that it would for the system to complete the execution without checkpoint to the increase in execution time due to checkpoint [17]. The elapsed time and efficiency are obtained by running the simulator for 3000 different failure scenarios. These failure scenarios are created based on the failure rate [12] provided by the user and generates a time interval until the next failure of one of the checkpoint levels. These time intervals are not fixed and vary across different simulations. Once the time interval to the subsequent failure is complete, the simulator uses the checkpoint restart time of the failed checkpoint level to determine the delay that will be incurred and add the delay to the total elapsed time. However, when no such failure occurs as there is still time until the next crash, the checkpoint uses the overhead parameter value of the checkpoint level where the current application status will be stored to determine the delay for storing the data at that desired level and add the delay to the elapsed time. At the end of all the failure scenarios, the simulator determines the total elapsed time and the efficiency of the system for that particular configuration, which is determined by comparing the elapsed time of the checkpoint system with delays to the elapsed time without

any checkpoint delays.

The above simulation run provides the elapsed time and efficiency of a system with a specific configuration. The simulator we have developed not only provides the elapsed time and efficiency for a system but also performs simulation across different settings of the checkpoint system to determine the optimal checkpoint count for each level of the multi-level checkpoint system. The optimized configurations are obtained by modulating the checkpoint configuration values starting with checkpoint interval. As shown in Figure 4, the simulator initially performs optimization of the checkpoint interval by comparing the current efficiency with the efficiency of the previous configuration until the maxima are obtained. The same procedure is also followed for both checkpoint count of level 1 and level 2. After the peaks for all the configuration parameters are captured by the simulator, the user is provided with the optimized configuration for the given system based on the overhead, restart time, and failure rates.

Once a significant amount of data is collected from the simulator, the information is passed on to different machine learning and neural network models to predict the optimized checkpoint configuration for other systems with different overhead, restart time, and failure rates. The models that were used from the scikit-learn [13] module, a python module integrating a wide range of state-of-the-art machine learning algorithms for training and predicting the optimized checkpoint configuration will be explained in the following section.

A. Random Forest

Random forest is a machine learning method used for classification and regression that operates by building multiple decision trees in randomly selected subspace of the feature space at training time and predicting the output class based on the combined decision of all the trees for classification or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set [11]

B. Gaussian Naive Bayes

The naive Baiyes classifier is a learning method that assumes that features are independent given class. It is a simple probabilistic classifiers that learns based on Bayes' theorem with an assumption of strong independence between the features. To allow a systematic study of classification accuracy for several classes of randomly generated problems it uses Monte Carlo simulations. It works well for completely independent features and functionally dependent features.[14]

C. Support Vector Clustering

It is a clustering method which uses the approach of support vector machines. It uses Gaussian kernel to map the data points in a high dimensional feature space, where we search for the minimal enclosing sphere. When mapped back from the enclosing sphere to data space, can separate into several components, each enclosing a separate cluster of points. The width of the Gaussian kernel controls the scale at which

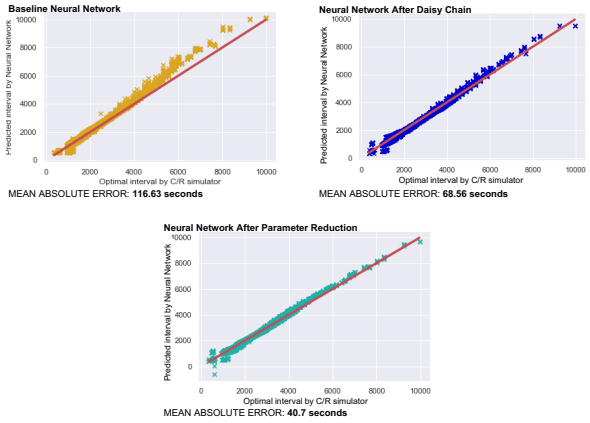

Fig. 5: Performance improvement with NN optimization

the data is probed while the soft margin constant helps to cope with outliers and overlapping clusters. The structure of a dataset is explored by varying the two parameters, maintaining a minimal number of support vectors to assure smooth cluster boundaries.[4]

D. Neural Network

Neural networks are a wide class of flexible nonlinear regression and discriminant models, data reduction models, and nonlinear dynamical systems. They consist of an often large number of **neurons**, i.e., simple linear or nonlinear computing elements, interconnected in often complex ways and often organized into layers [15]. In our implementation the neural network used ReLU [1] as the activation function, Adam optimizer as the solver with a batch size of 200 and learning rate of 0.001.

VI. Evaluation

For our evaluation, initially, we started with a two-level checkpoint model and then further evaluated on a three-level checkpoint model. The following sections describe the evaluation method and observations from both the multi-level checkpoint models.

A. Two-Level Checkpoint Model

Two independent data sets were generated for evaluation using the two-level checkpoint simulator. The training data was simulated by providing realistic overhead, latency, and restart time obtained using characteristics of some of the Top500 systems. We used the base failure rate of $2 \cdot 10^{-6}$ and $4 \cdot 10^{-7}$ failures/sec for Level-1 (L1) and Level-2 (L2) respectively [12] and explored the impact of these failure rates by increasing them up to 50X the base value. To validate our models for unique CR configurations, the test data set was simulated using data mutually exclusive to the training data set but within the parameter space of the training data set

Our random forest model could predict optimized checkpoint count with 99.365% accuracy when compared to the simulated result. For predicting the checkpoint interval with

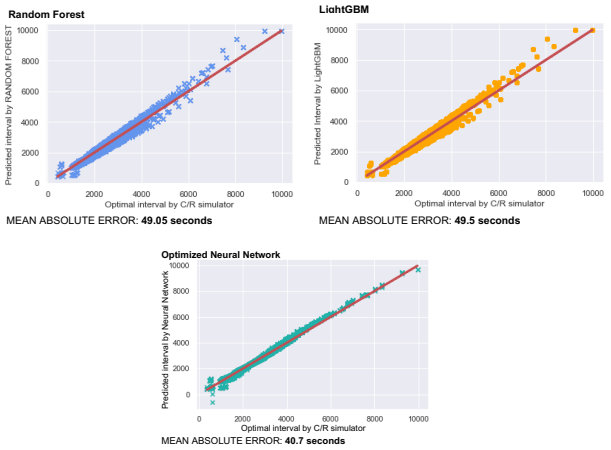

Fig. 6: Performance comparison between Machine Learning and Neural Network model for two-level checkpoint model

its values ranging up to 9960 seconds, both machine learning models could determine the optimized interval with a mean absolute error of 49.5 seconds. As shown in Figure 5, our baseline neural network predicted optimized interval with a mean absolute error of 116.63 seconds. However, with our optimizations described in Section 2.2, we see that daisy chaining shows a 40% performance improvement over the baseline design followed by a further 40% improvement with parameter reduction over the daisy chained model. The optimized neural network model performs 18% better than the machine learning models with a mean absolute error 40.7 seconds as shown in Figure 6. With majority of our dataset in the region 1000 - 8000 seconds, this converts to 0.5% - 4% mean error from the simulated values of the optimized interval for two-level checkpoint model.

B. Three-Level Checkpoint Model

For evaluating the three-level checkpoint model, we generated a dataset consisting of 112,000 different system configurations with varying checkpoint overhead, restart time, and failure rates for each level. We used the base checkpoint overhead and restart time of 0.5, 4.5, and 1052 seconds for Level-1 (L1), Level-2 (L2), and Level-3 (L3)checkpoints and a base failure rate of $2 \cdot 10^{-7}$, $1.8 \cdot 10^{-6}$ and $4 \cdot 10^{-7}$ failures/sec, respectively [12] and similar to the two-level checkpoint model, we evaluated the performance of different models by modulating these parameters between their base value and upper limit of 50X the base value. However, unlike the two-level checkpoint model, only one large dataset was generated using the parameter values ranging between their base value and upper limit. This dataset was further randomly broken into the training set and testing set for evaluation of the performance.

The training set was used for training the machine learning, and neural network model and validation for predicting the checkpoint count accurately for each level was performed on the test set. As shown in Figure 7 , the machine learning

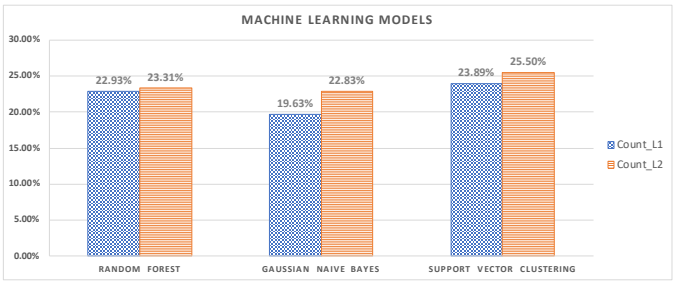

Fig. 7: Machine Learning Models Performance for three-level checkpoint model

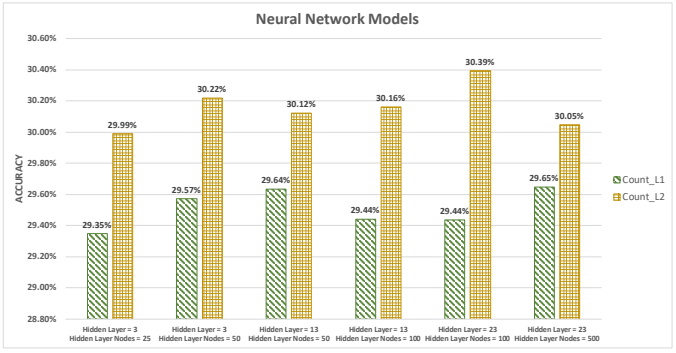

Fig. 8: Neural Network Models Performance for three-level checkpoint model

models show very similar accuracy for both level-1 and level-2 checkpoint count prediction with Support Vector Clustering performing the best with an accuracy of 23.89% and 25.50% for level-1 and level-2 count prediction. Random forest gives an accuracy of 22.93% and 23.31% respectively, and Gaussian Naive Bayes performs the worst among the tested machine learning models with an accuracy of 19.63% and 22.83% for level-1 and level-2 prediction.

Following the machine learning model evaluation, the neural network was trained and tested using the same set of data. The baseline neural network with 3 hidden layers and 25 hidden layer nodes gave an accuracy of 29.35% for predicting level-1 checkpoint count, while level-2 checkpoint count also gave a similar result with an accuracy of 29.99%. To evaluate checkpoint count prediction with a much more complex neural network, the hidden layers were increased from 3 to 13 layers and then further increased to 23 layers, and this was accompanied by an increase in the number of nodes in the hidden layers. The increase in the number of hidden layers and the hidden layer nodes does not improve the accuracy. As shown in Figure 8, the accuracy remains within 1% of the baseline neural network accuracy.

On comparing the prediction accuracy of the neural network with the three machine learning models, we see a noticeable improvement in checkpoint count prediction accuracy in the neural network. As shown in Figure 9, the neural network outperforms the machine learning models by a margin of 24% - 51% for Level-1 count prediction. For Level-2 count

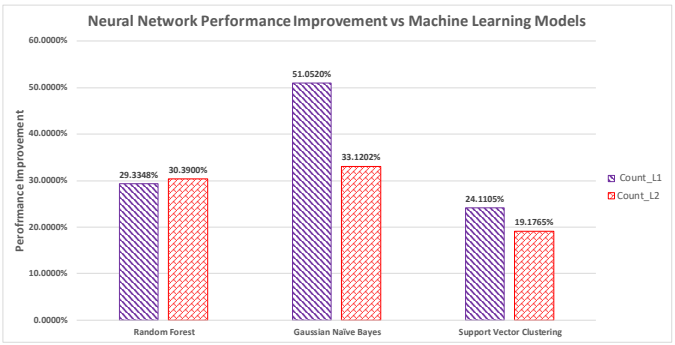

Fig. 9: Neural Network Models Performance Improvement vs Machine Learning Models for three-level checkpoint model

prediction, it shows a performance improvement ranging from from 19% - 33%.

VII. Related Work

Multi-level checkpointing [5] [6] [2] has become a promising technique for dealing with fault-tolerance in large scale or even exascale systems. With the increase in demand for such large scale systems [18] [10] [19], it has been studied thoroughly . The multi-level checkpoint-restart model providing an effective way to control the checkpoint overhead. In this paper, we focus on multi-level CR model and optimize the checkpoint count for multiple levels.

SCR[12] is a library which uses Markov model to fit the multi-level CR mechanism. There are some assumptions made in this paper such as the failure rates at different checkpoint levels are completely independent and also assumes that the failure rates at different levels are known beforehand. The other assumptions include that the checkpoint overhead remains constant throughout the job as well as infinite pool of spare nodes available at all time. Using the Markov model, Sato Kento et al. [16] combines the benefits of non-blocking and multi-level checkpointing. The work presents the design of the system and models its performance to show that the system can improve efficiency by 1.1 to 2.0 on future machines. Additionally, applications using that checkpointing system can achieve high efficiency when using a PFS with lower bandwidth.

FTI [3] is another multi-level checkpointing proposed by Bautista-Gomez et al. which uses local SSDs and a PFS. Since PFS usage is costly when compared to local storage, the model reduces PFS usage by encoding the data using Reed-Solomon (RS) encoding [7] for highly resilient cached checkpoints. However, increasing failure rates require checkpoints to a PFS more frequently. However, increasing failure rates require checkpoints to a PFS more frequently. Thus, checkpointing to a PFS is crucial for future multi-checkpointing systems. Sheng et al. [9] builds a mathematical model to fit the multi-level CR mechanism with large scale applications for various types of failures and theoretically optimize the entire execution performance for each parallel application by selecting the best checkpoint level combination and corresponding checkpoint intervals. They evaluated their optimal solutions using both simulation with millions of cores and real environment with real-world MPI programs running on hundreds of cores and showed that with optimal checkpoint intervals at each level, the system outperforms other state-of-the-art solutions by up to 50 percent.

VIII. Conclusion

In this paper, we present an idea to combine the simulation approach with machine learning models to reduce the time taken to determine the optimized parameter values of checkpoint interval and checkpoint count for different configurations of CR. With our approach and design optimizations, we show that our models can predict the optimized parameter values when trained with the simulation approach. We have also demonstrated that using techniques such as neural networks can improve the performance over the machine learning models with neural network sometime exceeding the performance of a machine learning model by 50%. We plan to validate the results on real systems in the future. From our study and exploration, we can say that using much more advanced techniques can further improve this performance and predict optimized checkpoint configuration accurately.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This material was based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This work is also supported in part by the National Science Foundation awards 561041, 1564647, 1763547, and 1822737. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] Guillaume Aupy, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. Checkpointing algorithms and fault prediction. *Journal of Parallel and Distributed Computing*, 74(2):2048–2064, 2014.

[3] Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, and Satoshi Matsuoka. Fti: high performance fault tolerance interface for hybrid systems. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2011.

[4] Asa Ben-Hur, David Horn, Hava T Siegelmann, and Vladimir Vapnik. Support vector clustering. *Journal of machine learning research*, 2(Dec):125–137, 2001.

[5] Marin Bougeret, Henri Casanova, Mikael Rabie, Yves Robert, and Frédéric Vivien. Checkpointing strategies for parallel jobs. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2011.

[6] Mohamed Slim Bouguerra, Ana Gainaru, Leonardo Bautista Gomez, Franck Cappello, Satoshi Matsuoka, and Naoya Maruyam. Improving the computing efficiency of hpc systems using a combination of proactive and preventive checkpointing. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 501–512. IEEE, 2013.

[7] Zizhong Chen and Jack Dongarra. A scalable checkpoint encoding algorithm for diskless checkpointing. In *2008 11th IEEE High Assurance Systems Engineering Symposium*, pages 71–79. IEEE, 2008.

[8] S. Di, M. S. Bouguerra, L. Bautista-Gomez, and F. Cappello. Optimization of multi-level checkpoint model for large scale hpc applications. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 1181–1190, May 2014.

[9] Sheng Di, Mohamed Slim Bouguerra, Leonardo Bautista-Gomez, and Franck Cappello. Optimization of multi-level checkpoint model for large scale hpc applications. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 1181–1190. IEEE, 2014.

[10] Kurt Ferreira. Keeping checkpoint/restart viable for exascale systems. 2011.

[11] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[12] AT Moody, Greg Bronevetsky, KM Mohror, and Bronis R de Supinski. Detailed modeling, design, and evaluation of a scalable multi-level checkpointing system. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2010.

[13] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[14] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[15] Warren S Sarle. Neural networks and statistical models. 1994.

[16] Kento Sato, Naoya Maruyama, Kathryn Mohror, Adam Moody, Todd Gamblin, Bronis R de Supinski, and Satoshi Matsuoka. Design and modeling of a non-blocking checkpointing system. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 19. IEEE Computer Society Press, 2012.

[17] Nitin H Vaidya. *On checkpoint latency*. Citeseer, 1995.

[18] Long Wang, Karthik Pattabiraman, Zbigniew Kalbarczyk, Ravishankar K Iyer, Lawrence Votta, Christopher Vick, and Alan Wood. Modeling coordinated checkpointing for large-scale supercomputers. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 812–821. IEEE, 2005.

[19] Gengbin Zheng, Xiang Ni, and Laxmikant V Kalé. A scalable double in-memory checkpoint and restart scheme towards exascale. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, pages 1–6. IEEE, 2012.