Efficient Classification of Supercomputer Failures Using Neuromorphic Computing

Prasanna Date, Christopher D. Carothers, James A. Hendler, Malik Magdon-Ismail
Department of Computer Science Rensselaer Polytechnic Institute Troy, NY, USA

***Abstract*—Today's petascale supercomputers are comprised of ten's of thousands of compute nodes. Failures on these massive machines are a growing problem as the time for a single compute node to fail is shrinking. Ideally, the job scheduler would like the capability to predict node failures ahead of time in order to minimize the impact of node failures on overall job throughput. However, due to the tight power constraints of future systems, the online modeling of real-time error data must be accomplished using as little power as possible. To this end, the IBM TrueNorth Neurosynaptic System is used to create a Spiking Neural Network (SNN) model of supercomputer failure data and the classification accuracy of this model is compared to other Machine Learning (ML) and Deep Learning (DL) techniques. It is observed that the TrueNorth failure classification model yields a training accuracy of 99.41%, validation accuracy of 98.12% and testing accuracy of 99.80% and outperforms other machine learning and deep learning approaches. Moreover, the TrueNorth SNN consumes five orders of magnitude less power than the other ML/DL approaches during the testing phase. Additionally, it is observed that all ML/DL approaches investigated as part of this study are able to produce accurate models of the supercomputer system failure data.**

***Index Terms*—Neuromorphic Computing, Deep Learning, Machine Learning, Supercomputer Failures**

I. Introduction

In today's world, Machine Learning (ML) and Deep Learning (DL) tasks consume a considerable chunk of compute resources on most of our devices – all the way from cellular phones to workstations [1]. If this trend continues, ML and DL tasks will dominate the consumption of compute resources on supercomputers in the future as well [2]. In such an environment, it seems reasonable to have a dedicated piece of hardware that is responsible for running ML and DL tasks on every node of the supercomputer in a fast and energy efficient way. Neuromorphic processors satisfy this need by introducing a computation paradigm that emulates the human brain [3].

The IBM TrueNorth Neurosynaptic System [4] is an example of one such neuromorphic device. It is a flexible, programmable substrate for neural algorithms, suited to sensory processing and spatio-temporal pattern recognition. It features a many-core processor network on a chip design. Each TrueNorth chip contains 4096 neurosynaptic cores, each composed of 256 programmable neurons, which is over a million neurons per chip. It typically uses about 65 mW of power [5]. Research is currently being pursued to explore the possibility of having neuromorphic chips on the next generation supercomputing systems [6]. So, the setting that this work lies is one where the next generation supercomputers have a Neuromorphic Processing Unit (NPU) on each node and these NPUs are able to perform ML/DL tasks extremely fast and in an energy efficient way. In such a setting, we look at a particular application that would potentially be running on the NPU. The application is an operating system tool that can predict when a node on the supercomputer is about to fail – a node failure predictor.

This work was funded by Air Force Research Lab (AFRL), USA.

Failures in supercomputers, especially node failures, are an unfortunate and unpleasant reality that the modern day supercomputing systems have due to their size and overall system complexity [7]. They not only result in extensive downtimes, lower reliability and component loss, but also consume significant amount of human resources. Therefore it would be ideal to have the ability to predict such failures ahead of time so that the jobs running on a potentially failing node can be rerouted to other nodes, new jobs can be deferred to healthier nodes and the node itself can be repaired during a scheduled maintenance period.

The main contributions of this work are as follows:

- We demonstrate that node failures can be modeled and classified using a neuromorphic computing approach by leveraging the IBM TrueNorth chip.
- We show that the Spiking Neural Network (SNN) of TrueNorth outperforms other Machine Learning (ML) and Deep Learning (DL) approaches for our application.
- We show that all ML, DL and neuromorphic computing approaches yield accurate results for our application.

II. Related Work

A. Node Failures on Supercomputers

The majority of research in modeling and predicting node failures has been domain-specific. Gainaru and Cappello provide an overview of failures observed in large-scale High Performance Computing (HPC) systems and also point out their characteristics with emphasis on modeling, detection and prediction [8]. Schroeder and Gibson review the sources of failures, corresponding decrease in application effectiveness, and also the coping strategies like application-level checkpoint compression and system level process-pairs fault-tolerance for supercomputing [9]. Fiala et al. study the use of redundancy to detect and correct soft errors in MPI applications that lead to silent data corruption and eventually node failures [10]. Abawajy proposes a fault-tolerant scheduling policy which couples job scheduling with job replication so that jobs are

978-1-5386-9276-9/18/$31.00

efficiently and reliably executed for grid computing systems [11]. Chen et al. explore the use of a floating-point arithmetic coding approach to build survivable HPC applications that can adapt to node failures without being aborted [12]. Fagg et al. emphasize the importance of application-level fault-tolerant systems and present a fault-tolerant version of the Message Passing Interface (MPI), which lets applications recover from a node or link error and as a result continue execution normally [13]. These papers focus on fault-tolerance from the point of view of applications being run.

When it comes to modeling failures, Zheng and Lan extend the Amdahl's Law and Gustafson's Law in order to consider the impact of failures and the effect of fault-tolerance techniques on applications [14]. Their reliability-aware models can predict application scalability in failure prone environments and also evaluate various fault-tolerant techniques. When it comes to tackling node failures, Bautista-Gomez et al. propose a low-overhead high-frequency multi-level checkpoint technique to tackle node failures by integrating a highly reliable topology-aware Reed-Solomon encoding in a three-level checkpointing scheme [15]. Egwutuoha et al. review the failure rates of HPC systems, survey fault-tolerance approaches, discuss the feature requirements of rollback-recovery techniques and develop a taxonomy for twenty checkpoint-restart solutions [16]. Brown and Patterson propose Recovery-Oriented Computing (ROC), which is an approach that recognizes the inevitability of unanticipated failure and focuses on recovery and repair as opposed to simple fault-tolerance [17].

Apart from the domain-specific approaches mentioned above, researchers have also used statistical methods as to analyze node failures. Schroeder and Gibson analyze failure data from two large HPC sites, and use a statistical approach to study the root cause of failures, the mean time between failures and mean time to repair [18]. Hacker et al. analyze the event logs of two IBM Blue Gene systems, characterize system failures statistically and propose a semi-Markov model for predicting the probability of a node failure [19]. This work found that components within the supercomputer often become "noisy" by issuing four or more warning events prior to complete failure and is a driver for our research here. Liang et al. have used RIPPER (a rule-based classifier), Support Vector Machines (SVM) and a Nearest Neighbor method to classify failures using error logs from the IBM Blue Gene/L machine [20]. SVM (recall: 80%, precision: 50%) and Bi-Modal Nearest Neighbor (BMNN) (recall: 85%, precision: 35%) were seen to perform best for 12-hour and 6-hour prediction windows respectively. Yu et al. have used event-based Bayesian model to classify failures using error logs from the IBM Blue Gene/P machine achieving 83.8% accuracy [21].

B. Neuromorphic Computing

Neuromorphic Computing is a computation paradigm that emulates biological neurons at the hardware level [22], [23]. The central idea is to deploy ML/DL models at the hardware level in order to run them fast and in an energy efficient manner. Given how widespread ML/DL models are in today's computation world, it would seem plausible to have a dedicated compute unit for them. The current analog neuromorphic devices are memristor based and boast of performing computations in a fast and energy efficient manner [24]. We find the earliest works on neuromorphic computing in the late 1980s [25]. In the late 2000s, Farquhar et al. proposed the field programmable neural array, which was an analog circuit that emulated biological neurons at the hardware level [26]. Pivotal work appeared in the field in 2012 when Pickett et al. proposed a scalable neuristor built with Mott memristors that displayed properties similar to the Hodgkin-Huxley axon [27].

Other examples of neuromorphic computing projects include Neurogrid at Stanford University [28], Human Brain Project in Europe that uses the SpiNNaker architecture [29], [30], the DARPA SyNAPSE program in the USA that has resulted in the IBM TrueNorth Neurosynaptic System and many others [31]. Our focus in this work will be on the TrueNorth chip. As of today, the research pertaining to neuromorphic computing is largely skewed towards the hardware side [32]. This leaves plenty of room for research on the software and theoretical side [33].

C. Machine Learning and Deep Learning

The ability of machines to *learn from data* forms the crux of the field of Machine Learning (ML) [34] Specifically, ML targets three problems: (i) Supervised Learning or Classification; (ii) Unsupervised Learning or Clustering; and (iii) Reinforcement Learning. Supervised Learning refers to learning from data when it is already available in a labeled format. Unsupervised Learning or Clustering refers to learning from data when it is not available in labeled format. Reinforcement Learning refers to learning by doing tasks, where a reward is awarded to the machine if it performs the task correctly and the aim of the machine is to maximize the reward.

In the field of machine learning, there is a plethora of learning algorithms and techniques to choose from: decision trees, rule-based learning, support vector machines, bayesian learning, evolutionary algorithms [35], [36] etc. Deep Learning (DL) also is one of the techniques that enable machine learning. While most of ML techniques target a specific problem (i.e. Classification, Clustering or Reinforcement Learning), DL can tackle all three problems. ML can be thought of as a class of problems in Artificial Intelligence (AI), and DL can be thought of as a technique that enables ML, and in turn, also enables the larger field of AI [37].

III. Data and Methodology

A. Data

In our study, we used the Reliability, Availability and Serviceability (RAS) logs from the IBM Blue Gene/L supercomputer that was stationed at the Center for Computational Innovation (CCI) at the Rensselaer Polytechnic Institute (RPI). RAS logs from Blue Gene systems have been used to model, classify and predict supercomputer failures in the literature [38]. Our Blue Gene/L system consisted of 16,384 compute nodes, each node having two IBM 440 PowerPC processors.

TABLE I: Description of variables in a typical RAS log entry

Variable Name	Variable Description	Categorical?	Sample Value
RECID	Record identification number of log entry	No	1
FACILITY	Facility in the machine that registered the log entry	Yes	APP
NODE	Virtual location of the node	Yes	236
SERIALNUMBER	Serial number of the component that initiated the log entry	No	7.8E+56
YEAR	Time variable: year	No	2006
MONTH	Time variable: month	No	12
DAY	Time variable: day	No	6
HOUR	Time variable: hour	No	16
MINUTE	Time variable: minute	No	48
SECOND	Time variable: second	No	55
MICROSECOND	Time variable: microsecond	No	7849
LOC_RACK	Rack location from where log entry was initiated	Yes	0
LOC_MIDPLANE	Midplane location from where log entry was initiated	Yes	1
LOC_NODE	Node location from where log entry was initiated	Yes	8
LOC_LINE	Line location from where log entry was initiated	Yes	0
SEVERITY	Severity of the log entry	Yes	1

Each rack is organized into two midplanes each with 512 compute nodes or 1024 compute nodes total per rack. The network was a 3-D torus topology. The RAS log files span over one year worth of the machine's operational life. After cleaning and preprocessing the data, a typical log entry looks like the one shown in Table I. It documents data corresponding to sixteen variables. Table I also describes what each variable means, whether it is categorical or not and what a sample value for the variable looks like. The Severity feature in Table I denotes the severity of the log messages. It can contain one of five values: `INFORMATION`, `WARNING`, `ERROR`, `SEVERE` or `FATAL`. A `FATAL` log entry corresponds to occurrence of a node failure. We want to classify the RAS logs into these five severity levels using the remaining fifteen features.

Problem Description: Use neuromorphic computing, machine learning and deep learning techniques to classify RAS logs into five Severity levels using fifteen features, which contain information pertaining to origin of the log entry (Record ID, Facility, Virtual Node Address), serial number of the component that initiated the log entry, temporal information (Year, Month, Day, Hour, Minute, Second and Microsecond) and spatial information (Rack, Midplane, Node and Line).

Note that this is a time series dataset in the sense that the decision to label a node as failing or not at a certain point of time may depend on the log entries that the node has registered till that time. If it has registered a large number of `ERROR` or `SEVERE` messages, it is likely to fail soon and it would be advisable to raise a red flag saying the node is not healthy for computation. This finding was determined in a prior study by Hacker et al [19] using the same RAS log data. At no point do we want the node to register a `FATAL` log entry because that would mean that a node failure has occurred.

The raw data that we used consisted of $151,766$ log entries. The first task after obtaining the data was to clean it. This comprised of removing data entries that contained values that were not relevant to the study. For instance, some of the data entries contained 'NaN' entries and irrelevant strings like "????????" which were removed. We ended up discarding only a small portion (1939 out of $151,766$ data entries or 1.28%) of the data during this process. After cleaning the data, we were left with 149,827 log entries. We converted the data into a numeric format by assigning numeric values to categorical variables like Facility, Rack Location, Midplane Location etc. Next, we only kept the variables that were relevant to the task of node failure classification and discarded the rest. Finally, we were left with fifteen features and 149,827 data points.

The next task was to split the dataset into Exploratory Data Analysis (EDA) set, training set and test set. For this, we kept 10% of the data in the EDA set, 70% of the data in the training set and 20% of the data in the test set. Although our dataset is a time-series dataset, we did not want to be restricted to sequential classification techniques, which are generally used to analyze such data (for e.g. Recurrent Neural Networks), but also wanted to leverage non-sequential techniques (for e.g. Deep Neural Networks, K-Nearest Neighbors, Support Vector Machines). Since the non-sequential classification techniques are chronologically independent we picked the data points in each set uniformly at random. After this, each dataset was chronologically sorted in order to preserve time-dependence for sequential classification techniques.

B. Methodology

The IBM TrueNorth Neurosynaptic System is used to execute a Spiking Neural Network (SNN) model which classifies the previously described failure data. A Spiking Neural Network (SNN) is a type of neural network in the discrete domain, where each neuron in a layer may send discrete signal spikes (for e.g. voltage in analog SNN, or a digital packet in digital SNN like TrueNorth) to neurons in the next layer. This is unlike traditional Deep Neural Networks (DNNs) where all neurons send all data at all times to the next layer, regardless of their input, or stimuli.

The software environment used for SNNs was IBM's Energy-Efficient Deep Neuromorphic networks (EEDN), which is a framework built in conjunction with MatConvNet (MATLAB) [39]. To compare the performance of SNN with other techniques, we chose three machine learning (ML) techniques – Logistic Regression, K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) – and two deep learning (DL) techniques – Deep Neural Networks (DNN) and Recurrent Neural Networks (RNN). We used the Scikit-Learn [40] and TensorFlow [41] libraries in Python for ML and DL techniques respectively. All the above models were run on a machine that had 32 cores of two-way multi-threaded Intel Xeon CPUs running at 2.60 GHz, three NVIDIA GPUs

(GeForce GTX 1080 Titan, GeForce GTX 950 and GeForce GTX 670), 112 GB DIMM Synchronous RAM, 32 KB L1 cache, 256 KB L2 cache and 20 MB L3 cache.

The IBM TrueNorth Neurosynaptic System can cater to any application having streaming unstructured data that needs to be processed. It accepts input values that are 8-bit integers ($[0, \ldots, 255]$). For a computer vision application, these correspond to the red-green-blue (RGB) values that would be seen in an image. Furthermore, it supports Convolutional Neural Networks (CNN), which are the *de facto* deep learning models used in computer vision tasks. Each input data point must be reshaped and presented to the TrueNorth SNN as a 3-D array. This is because, the TrueNorth system lends itself very naturally to image data, as compared to any other data. Continuing the parallel to computer vision tasks, this corresponds to an image being read as a 3-D array (Image Width $\times$ Image Height $\times$ Number of Channels) in a CNN. Since images are 3-D arrays usually, TrueNorth expects its input data to be a 3-D array as well.

Our input data points were 15-dimensional vectors. So, in order to make them TrueNorth compatible, we first reshaped them into 'images' of shape $1 \times 1 \times 15$ and later on added two layers of zero padding, which were seen to work best for our data. Thus, the shape of each of our data points was $5 \times 5 \times 15$. This data reshaping, was the only way to run numeric datasets on the TrueNorth chip. It must be noted that this reshaping of data is simply a workaround, and to the best of our knowledge, does not have any learning advantages.

Developing neural network applications with TrueNorth follows a standardized workflow. Development is done over six phases: Dataset, Preprocess, Train, Build, Test and Application. In the Dataset phase, we converted the data from the raw Comma Separated Values (CSV) format into the Lightening Memory-Mapped Database (LMDB) format. In the Preprocess phase, we scaled the data down to 8-bit range $(0 - 255)$ by normalizing and then rounding to the nearest integer. In doing so, we did not incur a significant loss of information because almost all of our features had values in the 8-bit integer range to begin with. The features that did not have values in this range were scaled down. However, as can be seen from Section IV, any potential loss of information arising from this operation did not seem to compromise our results.

The Train phase comprised of designing a SNN in EEDN and training it on the data obtained from the Preprocess phase. Figure 1 shows the configuration of our SNN in a terminal window output format and Figure 2 presents it in a pictorial format. Our network comprised of five layers, of which the first was the input layer (`I`), followed by the transduction layer (`P1`) and subsequently three convolutional hidden layers (`C2`–`C4`). The $5 \times 5 \times 15$ input data was fed to the input layer, and was encoded into spikes in the transduction layer. All in all, our SNN used 45 TrueNorth cores. With 256 neurons per core, this gives an upper bound of $11,520$ TrueNorth neurons in total. Note that $11,520$ is just an *upper bound* – it does not mean that our model used those many neurons. It was not possible to get the exact neuron count using the TrueNorth

```
LAYER TYPES
I: Image
P: Preprocessing layer
C: Convolution layer
D: Dropout layer

      Layer Size                    Patch                  TN Cores   Patch-in-Image
Lyr   Row x Col x Ftr  (Grp)        Str  Row Col Ffr       Base+Split Str   Row x Col
I     1   x 1   x 15

P1    5   x 5   x 64   (1  )       1    [1 x 1 x 15 ]     0   +0     1     [1   x 1  ]
C2    5   x 5   x 32   (1  )       1    [1 x 1 x 64 ]     15  +12    1     [1   x 1  ]
C3    5   x 5   x 32   (1  )       1    [1 x 1 x 32 ]     9   +0     1     [1   x 1  ]
C4    5   x 5   x 16   (1  )       1    [1 x 1 x 32 ]     9   +0     1     [1   x 1  ]
```

Fig. 1: Network configuration of TrueNorth SNN

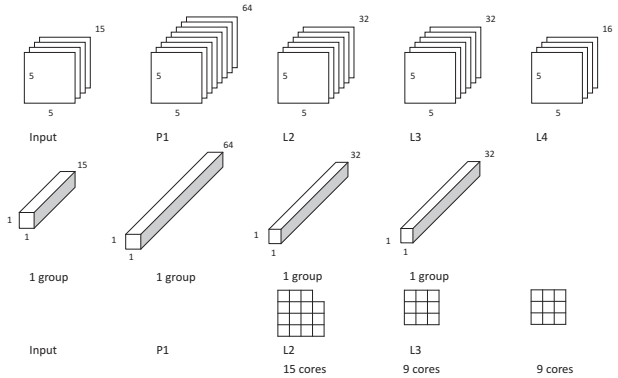


Fig. 2: Visualization of TrueNorth SNN

development kit at our disposal. Furthermore, note that spiking neurons in TrueNorth are hardware neurons and are different from the software neurons used in deep learning frameworks like TensorFlow. In general, a software neuron corresponds to multiple hardware neurons. The exact mapping is defined by the encoding algorithm, which in this case was not available in the TrueNorth development kit.

The Build phase comprised of generating the binaries that would be deployed on the TrueNorth chip. Finally, during the Test phase, we deployed our SNN model on the TrueNorth chip and obtained the test results. The Application phase runs an application on the TrueNorth chip in real time along with visualizations, analysis etc. In context of this work, it could potentially mean that the model running on the TrueNorth chip accepts a live stream of error data generated by the supercomputer and can predict failures ahead of time.

We now briefly describe all the machine learning and deep learning techniques that we have used. For any classification problem, Logistic Regression serves as the first step of analysis as it is a simple linear model and does not overfit the data easily unlike a complex model. We ran Logistic Regression in Python using the TensorFlow library. The second technique that we applied was K-Nearest Neighbors. We chose two values of K: $K = 3$, and $K = 289$. The former value has been known to perform well empirically and the latter value, which is the square root of number of data points in the training set (i.e. $\sqrt{83,903} \approx 289$), is known to perform well theoretically. The third technique that we used was Support Vector Machine (SVM) as it produces robust classifiers and can model nonlinear data as well using the kernel trick. To

run SVM and KNN, we used the Scikit-Learn library in Python. The two deep learning techniques that we used were Deep Neural Networks (DNN) and Recurrent Neural Networks (RNN). These were run using the TensorFlow library in Python. Deep Neural Networks (DNN) are the traditional deep learning model where each neuron is a perceptron. Recurrent Neural Networks (RNN) are known to perform well on sequential data. So, RNNs are the go to model for speech recognition and natural language processing. Since our data is a time series data, it is also sequential in nature and that was our motivation to pick RNN as one of our techniques.

IV. Results and Discussion

A. Exploratory Data Analysis (EDA)

The first step after cleaning the data was to perform Exploratory Data Analysis (EDA). EDA summarizes the main characteristics of the dataset so that we can form a solid understanding of the dataset and decide on further data analysis. Because we are using machine learning and deep learning in our analysis, we cannot perform exploratory data analysis on the whole dataset. This is in line with the machine learning principle that once any kind of inference is drawn from any part of the dataset, we should not use it to train the learning models. We ran a Singular Value Decomposition (SVD) on the dataset to check if dimensionality reduction was possible. This revealed that all the chosen features were significant and dimensionality reduction was not necessary.

The next step in EDA was to plot histograms of the categorical variables and the output variable, i.e. `SEVERITY` to get an idea of their distributions (see Figure 3). We used a value of -1.0 for all those data entries which did not have a value for the variable. The variable `FACILITY` is bimodally distributed with majority of log files initiated from `DISCOVERY` and `KERNEL`. The histogram for the Rack location is unimodal, meaning that in the 10% of the data that forms the EDA set, all the data points happened to have a Rack value of zero. It is, however, very unlikely that this would be true for the training and test datasets as well. Histogram for Midplane is uniformly distributed with some noise present in the data. The midplane can have two values, and in the EDA set, they are more or less equally distributed. When it comes to the `LINE` location, the EDA dataset was extremely noisy, as is evident from a large portion of -1.0 values. The histogram for the `NODE` location is more or less uniformly distributed around the frequency value of 1000 and contains very little noise.

B. Classification Results and Comparative Analysis

Our Spiking Neural Network (SNN) model consists of an input layer followed by a transduction layer, which is followed by three convolutional layers (see Fig. 1 and Fig. 2). The output layer is a softmax layer. This spiking convolutional neural network uses a total of 45 TrueNorth cores. Given that each TrueNorth core has 256 neurons, this gives us an upper bound of 11,520 TrueNorth neurons. We use the phrase 'upper bound' because it could be the case that not all 256 neurons in each of the 45 cores are being used. Nevertheless, we count it as part of the total system cost because most likely, these unused neurons cannot be used for other applications running on the TrueNorth chip. We used IBM's EEDN software framework, which is built in conjunction with MatConvNet (MATLAB) for SNN.

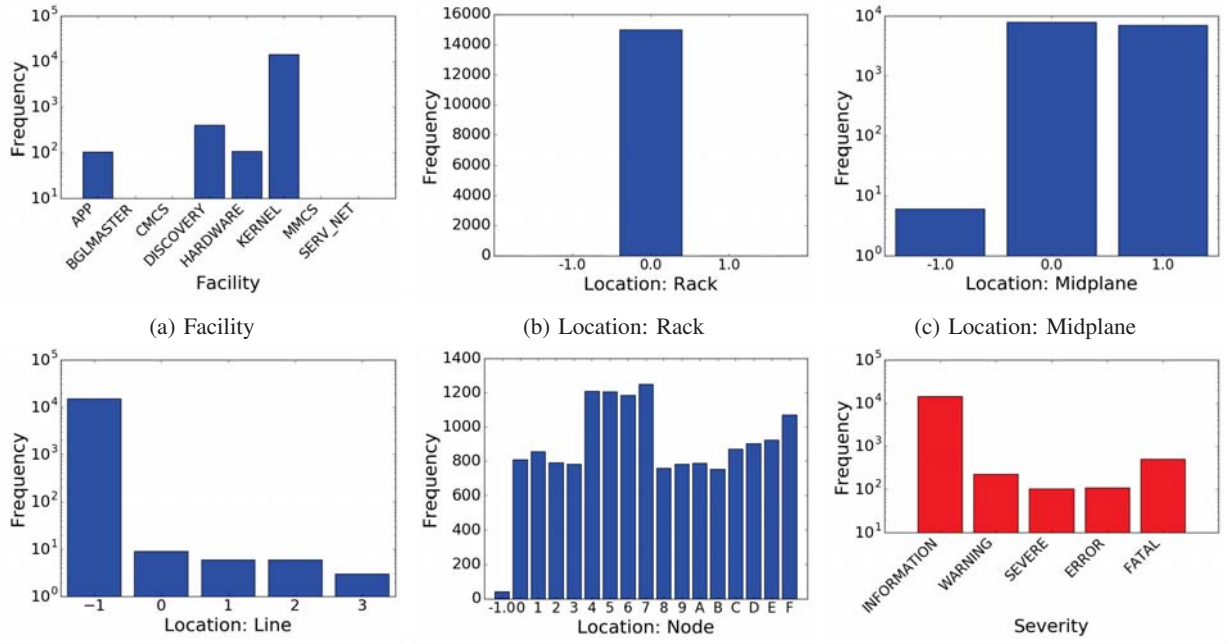
TABLE II: Performance comparison of all techniques

ML/DL Technique	Training Accuracy (%)	Validation Accuracy (%)	Testing Accuracy (%)
Logistic Regression	93.59	98.63	94.52
K-NN (K=3)	97.84	98.67	96.75
K-NN (K=$\sqrt{83,903} = 289$)	94.14	98.48	94.97
SVM	92.85	97.65	93.75
DNN	96.57	98.54	96.23
RNN	96.27	98.36	95.98
SNN (TrueNorth)	**99.41**	**98.12**	**99.80**

Our TensorFlow DNN model consists of six fully connected layers containing 105 neurons and 1750 synapses. The layer-wise neuron configuration is as follows: 30-25-20-15-10-5. We use hyperbolic tangent (tanh) and rectified linear unit (relu) as our activation functions and we alternate them for each layer. The output layer is a softmax layer. Our TensorFlow RNN model consists of a Long Short Term Memory (LSTM) layer followed by a softmax layer. The maximum sequence length of this RNN was 100, meaning we would monitor information from the previous 100 log entries to make a prediction for the current log entry. To store relevant information from older as well as current sequence of log entries, we chose the RNN cell size as 50. We also observed that the DNN and RNN models with number of neurons in ballpark of the SNN overfitted the training data, and were thus discarded. The DNN and RNN configurations mentioned above gave the best performance metrics amongst all the configurations that we tested.

Table II shows the results of our ML/DL classifiers. Although we present the results for just one hyperparameter configuration of each of the ML/DL techniques described above, we fine tuned the hyperparameters for these techniques over multiple iterations to obtain the best possible configuration. So, for example, the DNN model presented in Table II was seen to be the best performing DNN model. We used three performance metrics – Training Accuracy, Validation Accuracy and Test Accuracy. All our ML/DL models perform better than 92% classification accuracy. The best model as per all the performance metrics was SNN with a test accuracy of 99.80%. KNN ($K = 3$) seems to be the second best model with test accuracy of 96.75%. The two deep learning models (DNN and RNN) show comparable results with respect to each other with test accuracy of 96.23% and 95.98% respectively. These are followed by KNN ($K = 289$), logistic regression and SVM with test accuracy of 94.97%, 94.52% and 93.75% respectively.

We also computed the design index [42] for all seven ML/DL and neuromorphic models used in this paper. The design index is a tool which functions as an aid during the designing phase of DNN. Although it was defined for DNNs,



(a) Facility (b) Location: Rack (c) Location: Midplane

(d) Location: Line (e) Location: Node (f) Severity

Fig. 3: Histograms of categorical variables

TABLE III: Design Index computation

ML/DL Technique	Training Error (%)	Validation Error (%)	Accuracy Index[1] (I_a)	Overfitting Index[2] (I_o)
Logistic Regression	6.41	1.37	1.1931	-0.6701
K-NN (K=3)	2.16	1.33	1.6655	-0.2106
K-NN (K=289)	5.86	1.52	1.2321	-0.5861
SVM	7.15	2.35	1.1457	-0.4832
DNN	3.43	1.46	1.4647	-0.3709
RNN	3.73	4.02	1.4283	-0.3569
SNN (TrueNorth)	**0.59**	**1.88**	**2.2291**	**0.5033**

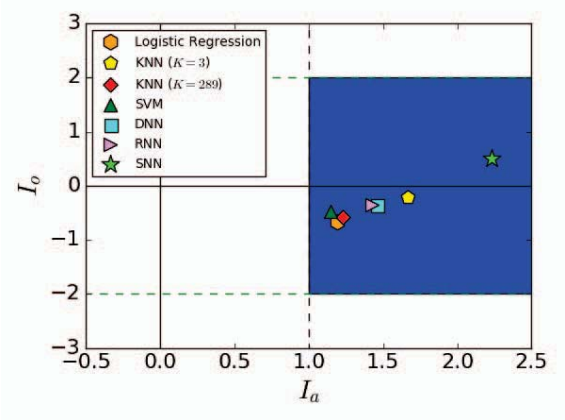


Fig. 4: Plot of design index for all techniques

it can easily be extended to other ML/DL and neuromorphic computing techniques. It consists of an index tuple containing Accuracy Index (I_a) and Overfitting Index (I_o), which gives a quantitative estimate of how accurate and how overfitted a trained DNN model is. Moreover, it is easy to represent visually. In this work, the threshold error was chosen as 1.0, the accuracy threshold was chosen as 1.0 and the overfitting threshold was chosen as 2.0. The threshold error is the order of magnitude of the smallest enumerated label. In our case, since the enumerated labels go from 1 to 5, the order of magnitude of smallest enumerated label is 0 and thus, the threshold error error was $10^0 = 1$. We chose an accuracy threshold of 1.0 because for this application, we would be happy to have a training error which is an order of magnitude less than the smallest enumerated label. We chose an overfitting threshold of 2.0 because for our application, we would render a trained model as overfitted if the training error is two orders of magnitude less than the validation error.

Table III and Figure 4 show the computation and scatter plot of of design indices. A peculiar feature in this figure is that the overfitting indices of all ML/DL techniques except SNN are negative, meaning that their validation errors were less than their training errors. We think this happened because in spite of picking the log entries identically and independently for training and validation datasets, their distributions were slightly different. However, as denoted by the green star in Figure 4, the SNN was able to correctly model the error data,

[1]Accuracy Index is computed as $\log_{10}\left(\frac{e_p}{e_t}\right)$, where e_p is the threshold error and e_t is the training error [42].

[2]Overfitting Index is computed as $\log_{10}\left(\frac{e_v}{e_t}\right)$, where e_v is the validation error and e_t is the training error [42].

showing the highest accuracy index (2.2291) and an acceptable overfitting index (0.5033). The blue region in the figure shows the acceptable region – we will accept a particular model if it lies in this region. All of our ML/DL and neuromorphic models lie in the acceptable range – i.e. not only do they demonstrate a certain acceptable level of accuracy, but also show that they do not overfit or underfit the training data. This confirms that ML/DL and neuromorphic techniques can be used to model and classify node failures on supercomputers as they display a respectable level of accuracy for this application.

C. Speed and Power Consumption

We now shed some light on the running times and power estimates of our models. During the test phase, all the techniques used in this study were able to classify $29,965$ data points in the test dataset fairly quickly. The fastest was DNN (9.6 milliseconds) and the slowest was RNN (2.2 seconds). The TrueNorth SNN took 23.5 milliseconds to classify the test dataset. When it comes to power consumption, DNN, RNN and Logistic Regression models were running on GPUs and consumed around 250 Watts of power. The models running on CPUs (i.e. K-NN and SVM) consumed around 60 Watts of power. The SNN ran on the TrueNorth chip and was estimated to consume less than 2 milliwatts of power – five orders of magnitude less than GPU and four orders of magnitude less than CPU. To the best of the authors' knowledge, a more detailed breakdown of power consumption is not possible with the TrueNorth development kit at our disposal.

D. Discussion

From the analysis so far, we understand that while all the ML/DL techniques fall within the acceptable range, the convolutional SNN seems to be performing the best for our dataset as per the performance metrics used in this work. This result is unexpected because CNNs are inherently not designed for sequential data such as the error data used in this paper. The model that is known to perform well on sequential datasets is the RNN owing to its inherent dynamical nature. Consequently, we expected RNN to outperform all other models, but that was not the case. Although our error log dataset is a sequential dataset, it also is a spatio-temporal dataset, because it contains information about the physical location and timestamp of the log entries. Spiking Neural Networks (SNN) have been shown to perform well on spatio-temporal datasets [43], [44]. While the exact reason for good performance of SNN on spatio-temporal datasets is unknown, a brief intuition is as follows. SNNs were modeled after the human brain and resemble it very closely. Since human brain is excellent at perceiving, interpreting and analyzing spatio-temporal data, one might expect SNNs to behave similarly – and they indeed perform well on spatio-temporal data, as demonstrated previously in the literature and by our results.

It must be pointed out that although SNN used 45 TrueNorth cores, which correspond to a maximum of 11,520 neurons, they cannot be compared directly to hundreds of neurons used in our DNN and RNN models. First of all, 11,520 is the number of *hardware* neurons used and is different from the *software* neurons used by DNN and RNN models. A single software neuron usually maps to multiple hardware neurons – the exact mapping depends on the encoding algorithm. Furthermore, we had to add two layers of padding to our dataset, which required 25 times more neurons to represent a single data point – this directly adds to the neuron cost (see Section III-B for more details on padding, and why we had to do it). Secondly, the 11,520 neurons is just an upper bound – it does not necessarily mean that our SNN used all 11,520 neurons. With the TrueNorth development kit that we had, it was not possible to know the exact number of neurons used.

V. Conclusion

We set out to explore the possibility of using a neuromorphic computing approach to classify node failures on supercomputers. Furthermore, we compared this approach to five other machine learning and deep learning approaches. We demonstrated that the neuromorphic computing approach outperforms all the other machine learning and deep learning approaches for our application. Moreover, we also showed that all the techniques used in this work do a good job of classifying node failures. We used the IBM Blue Gene/L (BG/L) dataset, which consisted of RAS logs spanning over a year of the machine's operational lifetime. We used Python (TensorFlow and Scikit-Learn) and MATLAB (IBM's EEDN framework) to run our ML and DL techniques.

The bigger picture that we are trying to paint is to design the next generation of supercomputers that will have a Neuromorphic Processing Unit (NPU) on every node of the supercomputer. In this setting, our paper sheds light on classifying node-level failures on such supercomputers. As part of our future work, we would like to put the neuromorphic computing approach in a live setting where failure data is read as a live stream of data. Another aspect that we would like to incorporate is to use more complex SNN architectures running on TrueNorth to see if they deliver better results. We would also like to get an estimate of how far into the future can we *predict* these failures.

VI. Acknowledgment

This work was funded by the Air Force Research Lab (AFRL), USA (Award Number: FA8750-15-2-0078). The authors are grateful to IBM Research for lending the TrueNorth chip, development kit and organizing the TrueNorth Bootcamp, and would specifically like to thank Dr. Ben Shaw for getting our paper reviewed by IBM. Dr. Catherine Schumann of the Oak Ridge National Laboratory was very generous to provide her feedback on the paper. The authors would further like to thank Rensselaer Polytechnic Institute (RPI) and the Center for Computational Innovation (CCI) at RPI.

References

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1223–1231, Curran Associates, Inc., 2012.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] D. Monroe, "Neuromorphic computing gets ready for the (really) big time," *Communications of the ACM*, vol. 57, no. 6, pp. 13–15, 2014.

[4] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pp. 1–10, IEEE, 2013.

[5] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[6] J. Hasler and B. Marr, "Finding a roadmap to achieve large neuromorphic hardware systems," *Frontiers in neuroscience*, vol. 7, 2013.

[7] M. J. Zaki, C. D. Carothers, and B. K. Szymanski, "Vogue: A variable order hidden markov model with duration based on frequent sequence mining," *ACM Trans. Knowl. Discov. Data*, vol. 4, pp. 5:1–5:31, Jan. 2010.

[8] A. Gainaru and F. Cappello, "Errors and faults," in *Fault-Tolerance Techniques for High-Performance Computing*, pp. 89–144, Springer, 2015.

[9] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," in *Journal of Physics: Conference Series*, vol. 78, p. 012022, IOP Publishing, 2007.

[10] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, p. 78, IEEE Computer Society Press, 2012.

[11] J. H. Abawajy, "Fault-tolerant scheduling policy for grid computing systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 238, IEEE, 2004.

[12] Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra, "Fault tolerant high performance computing by a coding approach," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 213–223, ACM, 2005.

[13] G. E. Fagg, E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, J. Pjesivac-Grbovic, and J. J. Dongarra, "Process fault tolerance: Semantics, design and applications for high performance computing," *The International Journal of High Performance Computing Applications*, vol. 19, no. 4, pp. 465–477, 2005.

[14] Z. Zheng and Z. Lan, "Reliability-aware scalability models for high performance computing," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pp. 1–9, IEEE, 2009.

[15] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka, "Fti: high performance fault tolerance interface for hybrid systems," in *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*, p. 32, ACM, 2011.

[16] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013.

[17] A. Brown and D. A. Patterson, "Embracing failure: A case for recovery-oriented computing (roc)," in *High Performance Transaction Processing Symposium*, vol. 10, pp. 3–8, 2001.

[18] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, 2010.

[19] T. J. Hacker, F. Romero, and C. D. Carothers, "An analysis of clustered failures on large supercomputing systems," *Journal of Parallel and Distributed Computing*, vol. 69, no. 7, pp. 652–665, 2009.

[20] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pp. 583–588, IEEE, 2007.

[21] L. Yu, Z. Zheng, Z. Lan, and S. Coghlan, "Practical online failure prediction for blue gene/p: Period-based vs event-driven," in *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pp. 259–264, IEEE, 2011.

[22] R. E. Pino, "Neuromorphic computer," Sept. 25 2012. US Patent 8,275,728.

[23] A. Calimera, E. Macii, and M. Poncino, "The human brain project and neuromorphic computing," *Functional neurology*, vol. 28, no. 3, p. 191, 2013.

[24] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis, "Integration of nanoscale memristor synapses in neuromorphic computing architectures," *Nanotechnology*, vol. 24, no. 38, p. 384010, 2013.

[25] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[26] E. Farquhar, C. Gordon, and P. Hasler, "A field programmable neural array," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pp. 4–pp, IEEE, 2006.

[27] M. D. Pickett, G. Medeiros-Ribeiro, and R. S. Williams, "A scalable neuristor built with mott memristors," *Nature materials*, vol. 12, no. 2, p. 114, 2013.

[28] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[29] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[30] H. Markram, "The human brain project," *Scientific American*, vol. 306, no. EPFL-ARTICLE-183377, pp. 50–5, 2012.

[31] T. Hylton, "Darpa synapse project," *Arlington, VA*, 2009.

[32] M. Chu, B. Kim, S. Park, H. Hwang, M. Jeon, B. H. Lee, and B.-G. Lee, "Neuromorphic hardware system for visual pattern recognition with memristor array and cmos neuron," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 4, pp. 2410–2419, 2015.

[33] A. Disney, J. Reynolds, C. D. Schuman, A. Klibisz, A. Young, and J. S. Plank, "Danna: A neuromorphic software ecosystem," *Biologically Inspired Cognitive Architectures*, vol. 17, pp. 49–56, 2016.

[34] S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas, "Machine learning: a review of classification and combining techniques," *Artificial Intelligence Review*, vol. 26, no. 3, pp. 159–190, 2006.

[35] B. Lantz, *Machine learning with R*. Packt Publishing Ltd, 2013.

[36] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[37] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[38] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner, "Co-analysis of ras log and job log on blue gene/p," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp. 840–851, IEEE, 2011.

[39] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for matlab," in *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[42] P. Date, J. A. Hendler, and C. D. Carothers, "Design index for deep neural networks," *Procedia Computer Science*, vol. 88, pp. 131–138, 2016.

[43] N. K. Kasabov, "Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data," *Neural Networks*, vol. 52, pp. 62–76, 2014.

[44] B. Ermentrout, "Neural networks as spatio-temporal pattern-forming systems," *Reports on progress in physics*, vol. 61, no. 4, p. 353, 1998.