FailureSim: A System for Predicting Hardware Failures in Cloud Data Centers Using Neural Networks

Nickolas Allen Davis
Dept. of Computer Science and Engineering
New Mexico Tech
Socorro, New Mexico, USA
ndav01@nmt.edu

Abdelmounaam Rezgui
Dept. of Computer Science and Engineering
New Mexico Tech
Socorro, New Mexico, USA
rezgui@cs.nmt.edu

Hamdy Soliman
Dept. of Computer Science and Engineering
New Mexico Tech
Socorro, New Mexico, USA
hss@cs.nmt.edu

Skyler Manzanares
Dept. of Computer Science and Engineering
New Mexico Tech
Socorro, New Mexico, USA
smanzanares@nmt.edu

Milagre Coates
Dept. of Computer Science and Engineering
New Mexico Tech
Socorro, New Mexico, USA
mcoates@nmt.edu

Abstract— **Hardware failures in cloud data centers may cause substantial losses to cloud providers and cloud users. Therefore, the ability to accurately predict when failures occur is of paramount importance. In this paper, we present FailureSim, a simulator based on CloudSim that supports failure prediction. FailureSim obtains performance related information from the cloud and classifies the status of the hardware using a neural network. Performance information is read from hosting hardware and stored in a variable length windowing vector. At specified stages, various aggregation methods are applied to the windowing vector to obtain a single vector that is fed as input to the trained classification algorithm. Using conservative host failure behavior models, FailureSim was able to successfully predict host failure in the cloud with roughly 89% accuracy.**

Keywords-Failure prediction; Cloud; Simulation; Neural networks; Machine learning; FailureSim; CloudSim; Dynamic CloudSim.

I. INTRODUCTION

With increased access to the Internet and big data analytics technologies, users are looking for more accessible solutions to their computing problems. Recent advances in cloud software have made it easier for the average user to gain access to powerful computational resources. Companies like Google, Amazon, and Microsoft are leasing their massive cloud engines to users that require relatively hassle free access to computational resources. Using these publicly available clouds, it is possible for users to access and utilize resources without having to worry about the maintenance of the hardware itself. Instead, the upkeep of these massive data centers is placed on the service providers themselves. Unfortunately, these systems rarely work perfectly as intended. It is estimated that in a cluster's first year of usage, roughly 1000 individual machine failures will occur [1]. Each minute of downtime can cost roughly $7,900 on average [2].

Due to the high cost of downtime for the average data center, it should be an operator's goal to minimize downtime as much as possible. A crucial requirement to reduce the downtime is the ability to *predict* hardware failures. In this paper, we present FailureSim, a simulator based on CloudSim [7,9] that supports failure prediction. FailureSim uses a neural networks based approach to predict hardware failures in cloud data centers. In this approach, various performance data are gathered from the cloud data center and fed into a neural network which then makes predictions as to future potential hardware failures in the data center.

This paper is organized as follows. Section 2 discusses work related to the prediction of failures in cloud systems. As FailureSim is based on CloudSim and DynamicCloudSim, we first give an overview of both simulations frameworks in Section 3. We describe our FailureSim framework in Section 4. In Section 5, we present an overview of FailureSim's implementation. In Section 6, we describe the experimental environment that we used to evaluate FailureSim. In Section 7, we give the details of our experimental evaluation of FailureSim. Sections 8 concludes the paper.

II. RELATED WORK

Improving cloud functionality by applying neural networks is a well-researched topic. Chen, Lu, and Pattabiraman applied recurrent neural networks to resource usage patterns [3]. This allowed them to categorize the resource utilization time series data into different classes and predict when batch applications would fail. This job prediction method improved resource savings cluster-wide between 6% and 10% [3]. Duggan et al. introduce a more portable learning-based workload prediction tool specifically tailored to analytical database applications deployed in the cloud [4]. They showed that their models were able to accurately predict the workload throughput values of heterogeneous systems within a 30% margin of error [4]. These types of predictive models focus on the application more than the physical hardware itself.

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.75

Specifically, in [3], task predictions are made based on the performance of the job itself and is less reliant on the hardware.

There are also other works that are more closely related to predicting hardware failures in a cloud-like environment. Ganguly et al. presented a hard disk failure prediction model for usage in cloud environments [5]. Using a two-stage ensemble model and various data sources, they describe the techniques and challenges associated with deploying an operational predictive model on cloud-like environments. They also list the benefits associated with a predictive hardware model [5]. Our research directly correlates to the work done in this paper. In addition to predicting hard drive failures, however, processor and RAM failures are also predicted from simulated data. Bahga and Madisetti detailed a framework that processes and analyzes data collected from multiple sensors embedded in a cloud computing environment [6]. Using a large set of information obtained from previous failure cases, they can predict failure cases in real time before the failures occur [6]. The purpose of our work is to provide a simulation framework that would enable more research on failure prediction similar to the research mentioned in this section.

III. CloudSim and DynamicCloudSim

CloudSim [7] is an open source cloud simulator that provides support for the modeling and simulation of virtualized cloud environments including dedicated management interfaces for VMs, memory, storage, and bandwidth. In CloudSim, users define data centers that contain multiple types of hosts. A host is the simulated equivalent of a server and each host defines its own processor type, bandwidth, I/O, RAM, and storage amounts. Once a data center and its machines are specified, users can then create various types of VMs and cloudlets, or tasks, which can be submitted to the cloud data center. These VMs and cloudlets can then be added to a broker that manages the submission of these tasks to the data centers and the simulation can be initialized.

CloudSim allows users to simulate various cloud environments and stress test scheduling algorithms. The initial CloudSim framework did not take into account the variability of processing performance associated with clouds. In an effort to create a simulation that more closely mimicked the performance of physical clouds, the authors of [8] developed DynamicCloudSim, an extension to CloudSim that introduces resource variability and task failure. DynamicCloudSim simulations were shown to mimic the execution patterns of tasks run on physical clouds more accurately than standard CloudSim simulations.

CloudSim and DynamicCloudSim can simulate cloud computing environments and allow users to test various types of scheduling algorithms for deployment in actual clouds. There are also many other types of tools developed for CloudSim that aim to make the development and testing of algorithms easier for researchers. Many of these tools can be found at [9]. These tools attempt to improve the simulation environment to more closely mimic the execution environments available in clouds but none of these implementations simulate the causes or variability of hardware failures. Most implementations create statistical models using real-world data and then force failure of random nodes corresponding to these models [10]. These types of simulations, however, do not change the performance of the hardware with respect to these failures. A sustainable failure model needs to be created before a failure detection algorithm can be designed. This failure model must update hardware performance with respect to simulated environment changes and task execution behavior.

IV. FailureSim

In order to predict when hardware failures occur, a simulation system must allow varying host behaviors during simulation. While CloudSim is a powerful simulation tool, there is no extension that allows for the simulation of host performance that changes over time. To address this limitation, FailureSim extends CloudSim and DynamicCloudSim to support the simulation of dynamic host behaviors. It supports the simulation-time modification of hardware performance based on a configurable behavior. FailureSim also allows users to define various types of hardware behavior. This facilitates the implementation of various types of failure behaviors and allows users to modify the performance characteristics of failing hardware.

FailureSim continuously collects performance data from the given data center. These data are stored in a variable length windowing vector. This vector stores the most recent cloud data available and ejects older entries, adding a temporal relationship to the obtained data. When enough data are gathered, we apply multiple aggregation methods to obtain a single vector representing the status of the cloud hardware for a specific interval. These aggregation methods include integrals, slopes, weighted averages, and standard deviations, and assist in making the system more portable by focusing on failure trends as opposed to the direct values associated with failing hardware. This vector is then input to a trained classification algorithm that outputs the status of the hardware during that specific time interval.

In FailureSim, each host is assigned a behavioral execution pattern that defines how resources are modified at designated update intervals. For example, a host can be given the behavior of "sporadic processor performance," where at each update interval the processor's performance is varied by a random sample from a normal distribution with a predefined mean and variance. We chose to utilize a normal distribution based on the research performed in [8]. Using this new host functionality, it is possible to change behaviors mid-execution based on various updates to the cloud environment (such as overloading task executions) or randomly using samples from various distributions.

In addition to allowing users to easily create and define new types of host behaviors, FailureSim also provides failure based resource provisioners and failure aware data centers and scheduling strategies. The failure based resource provisioners extend DynamicCloudSim's resource provisioners while adding the ability to modify data center

resources mid execution, making it possible to add variability to the available resources while the simulation is running. The failure compliant data centers, hosts, and scheduling strategies take into account the behaviors of failing and failed hosts to correctly distribute incoming tasks to available VMs.

Standard CloudSim and DynamicCloudSim do not have a way to easily create tasks to submit to a cloud. Both systems rely on creating a large static workflow which is then run until completion. In order to improve upon this system, we designed an interface that spawns batches of tasks and submits these batches to the cloud at regulated intervals. This forces the cloud to execute either a specific number of tasks or to inject tasks at a constant rate until a specific simulated time.

FailureSim provides an extensible way to create new types of behaviors that can mimic the variability of resources on a server at simulated time steps. These behaviors can be used to define multiple types of host performance types, including the behavior of a failing host. FailureSim also provides a task injection mechanism that allows users to easily simulate environments with variable workloads over long periods of time.

V. Implementation

There are four major steps in the implementation and training of the classification algorithm using FailureSim: (i) setting up the simulation, (ii) reading simulation data, (iii) training the classifier, and (iv) using the classifier.

A. Simulation Setup

Simulations are setup through a series of configurable experiment files. Configuration files are written according to the YAML machine parsable data format and contain information pertaining to the data center, VMs, and tasks. In addition to simulation specific information, there are also configurable experiment files that facilitate the modification of various parameters. Examples of these parameters include: execution length of the simulation, data sliding window size, and data to store for use in the classification algorithm.

B. Data Collection

In order to make the updating and polling of data easier, we designed a `Scheduler` class that facilitated asynchronous updates at designated intervals. Using the experiment configuration file, it is possible to specify when FailureSim should apply host behavior definitions and when to poll the hosts for data. At these specified intervals, data are obtained from the hosts. These data represent the current state of the system (how many Millions of Instructions Per Second (MIPS) are available for host x, the bandwidth currently utilized by host x, the number of tasks running on host x, etc.). These data are then recorded and placed into a sliding window queue. When the queue buffer is full, the system performs various types of analyses on these data (averages, std. dev., integral, etc.) and ejects this result vector out to a file that will be used to train the neural network. These analyses are repeated whenever enough data points have been gathered to fill the sliding window update buffer. The sliding window queue size and the sliding window update buffer size are both configurable and are based on the number of data points read by the asynchronous polling mechanism. This makes it possible to define the size of the window and also the rate at which calculations are performed.

C. Training the Classifier

We wrote the classification code using WEKA [11] as the base API. After obtaining enough training data, the training data file is input to a WEKA classification algorithm. On average, each simulation generated roughly 10 000 aggregated neural network input vectors (variation dependent on the number of hosts, queue size, and simulation time). With a total of 20 different training configurations and 3 simulations per configuration, that comes out to about 600,000 different training data-set points. Once the classification algorithm is trained, the algorithm is written out to a file and can then be used to predict future failures in the system.

D. Using the Classifier

It is possible to utilize the classifier both on a running cloud and on a batch set of testing data. Using the classifier on a running cloud requires pointing the cloud's host-data stream at the classifier input and then providing window sizes. The classifier then reads these data points and decides how to classify the current state of a host. If a test file has already been generated, it is possible to input that file to the classifier and have it output the accuracy of the classifications.

VI. Experiment Setup

There are four key factors involved in setting up the experiments: (i) determining simulation specific information, such as types of tasks, VMs, and hosts; (ii) defining the types of behaviors that various hosts can experience; (iii) selecting the types of data that will be used in the neural network training and testing phases; and (iv) selecting the neural networks and classification systems.

A. Simulation Specifics

There are two fundamental types of simulation set-ups: training simulations and testing simulations. Training simulations need to be numerous and diverse so that the classification algorithm can converge to an accurate representation. Testing simulations need to stress various break points in order to show that the classifier holds up well under multiple conditions.

1) Training

The simulations utilized for training purposes were randomized and designed in an attempt to capture the many possible types of host behaviors and system states. There were 20 different training configuration set-ups and each configuration file was executed once, resulting in a total of 20 simulations that generated training data. Each

configuration file had a different set of hosts (ranging from 20 to 400 machines) and host specifications (MIPS per core, number of processors, I/O rate, bandwidth, and memory).

Changing the types of hosts available in a data center can improve the accuracy of the classification algorithm by providing more robust and diverse data. Using multiple types of distinct hosts can make the algorithm train on the patterns seen in the data as opposed to training it to match a given specification for a set type of host. It should be possible to directly transfer the trained classification algorithm to different types of cloud systems while still maintaining a reasonable failure prediction accuracy. Since many cloud systems are of varying sizes and hardware specifications, the failure prediction model must be able to detect failures in different types of systems without needing to re-train for the specific hardware. This is accomplished by utilizing the patterns found on the different types of trained hosts and using data that does not rely on hard coded parameters such as size.

Between 5 and 80 VMs were randomly created per task submission interval depending on the number of hosts in the datacenter. Between 5 and a 200 tasks were executed on those VMs using a C20 task scheduling algorithm implemented in DynamicCloudSim. These values were randomized for each simulation and were regenerated at each task submission interval. This ensured that the datac enter was always busy and simulating a reasonable workload.

In addition to adjusting the amount of time a simulation runs, it is possible to modify the size of the sliding window and the data polling rate. These values are varied during training and represent the rates at which updates are applied to the data and the amount of data required before statistical calculations are performed.

2) Testing

There are six types of test simulations designed to stress various sections of the classification algorithms. The first simulation is a baseline test. It uses the same configurations as the first training configuration. This test demonstrates that the classification algorithm performs well in a simple scenario. The second simulation uses a data center that is smaller than any of the training simulations. This simulation illustrates the classification algorithm's accuracy at predicting failures in a previously un-encountered type of data center. The third simulation uses a data center that is larger than any of the training simulations. Similar to the previous simulation, this test aims to show how accurate the classification algorithm is at predicting failures in a different type of data center. The fourth simulation tests how well the classification algorithm can predict failure when data is obtained at a slower rate. In this simulation, data is obtained more slowly than in any of the other training simulations. The fifth simulation modifies the size of the sliding window queue and the rate at which it generates data. A larger sliding window will make the data less prone to sporadic changes. This test is used to conclude that the classification algorithm can still retain reasonable accuracy when the window size and update rate are modified. The sixth and final testing simulation reduces the size of the sliding window and increases the rate at which data is sent from the simulation to the classifier. Decreasing the size of the window and increasing the data acquisition rate will make certain types of system changes more noticeable. Classification accuracy should be maintained even though update rate is increased and the window size is decreased. Each of these simulations is run three different times for each classification test in order to get multiple data samples. The simulation results were then averaged together to obtain a final classification accuracy.

B. Host Behaviors

When a simulation is started, each host is given a working behavior. This behavior dictates the randomness in hardware performance that a host can experience. At every performance update interval, a host's hardware specifications are modified to reflect the behavior associated with that host. For example, if a host is set to experience processor performance variation according to a randomly sampled value from a normal distribution, then its processor performance will be modified accordingly. There are three types of working behaviors. Each working host behavior fluctuates the processor performance according to a normal distribution with a mean of 0 and a coefficient of variation (CV) between 0.6 and 0.8. Each working host also experiences fluctuations in available RAM, I/O, and available bandwidth following a normal distribution with a mean of 0 and a CV of 0.5.

In addition to the working host behaviors, there are also failing host behaviors. On simulation creation, a host can be set to fail at some point in time according to a configurable failure probability. If a host is set to fail, a time is chosen to have that host experience failure symptoms. At that point in time, a host's behavior will randomly change to one of the failure behaviors. The host will continue executing but performance will fluctuate according to a new failure behavior. This behavior will continue until the host fails and becomes unusable in the simulation.

There are currently 13 different types of failure behaviors, each of which updates host information according to a normal distribution. Each behavior affects a different aspect of the host performance (processor, RAM, I/O, and bandwidth), and has different normal means and normal standard deviations. The average means and standard deviations of these host behaviors are given for the various classifications in Figure 1 and Figure 2 respectively.

Currently, there is little research or reported information on how the performance of failing hardware varies at certain stages in its lifecycle. For example, it is possible to determine when RAM begins to fail by seeing that certain memory operations become slower, but the exact rates associated with this performance are undocumented. The idea is that, over time, there should be some sort of noticeable change in the performance of failing hardware, even if it is very difficult to detect. For the purposes of our experiments, the failing behaviors were set to be fairly

similar to the working behavior's performance, which makes the failing behaviors themselves difficult to detect.

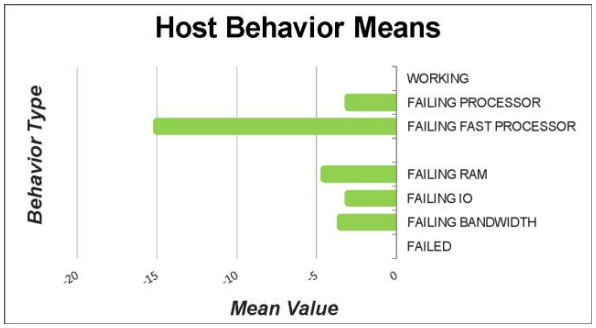

Figure 1. Host Behavior Means

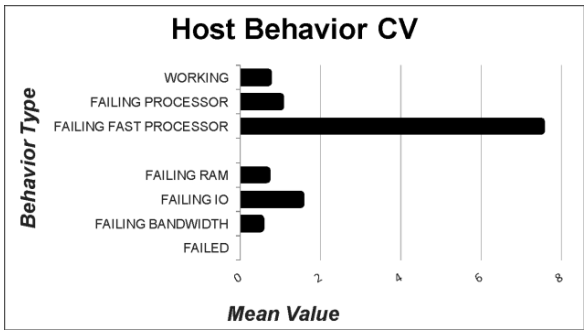

Figure 2. Host Behavior CV

C. Data Formatting

The classification and FailureSim systems deal with two primary types of data. The first type of data is host related data. CloudSim provides host descriptions of TotalMips and AvailableMips. for a specific host. TotalMips describes the total amount of processing capabilities a host possesses, and AvailableMips describes the processing power that can be assigned to new VMs. The resource utilization ratio is derived as: 1 – AvailableMips/TotalMips. In addition to processing usage information, CloudSim also provides this information for RAM and bandwidth and has a static I/O value. The current time of the simulation can also be obtained using CloudSim method calls. At each data gather interval, all of these values are read and stored in a sliding window vector.

The second type of data is the input vector used in the classification algorithm. After a sliding window has been filled, calculations are performed on the provided data and output to a single classification vector. There are currently 5 types of aggregation methods used for each type of host behavior. The first is an average of the utilization ratios at specific points in time. Taking the average of these values gives an average utilization ratio for a given period of time. Ideally, systems that are experiencing failure will have smaller utilization ratios since the amount of resources utilized will be less than the specified total. Using the average of utilization ratios for a specific type of host hardware should provide some insight into whether or not the system is experiencing certain failure symptoms.

The second aggregation method is the standard deviation of the utilization ratios. When performance starts to drop, it might become difficult to tell whether or not a specific host is either working harder or actually experiencing failure. Calculating the standard deviation in the values for a specific window should help distinguish between decreased available resources due to an increase in task executions or a decrease in available resources due to the host experiencing failure symptoms.

The third aggregation method is a slope calculation. If a host is experiencing failure symptoms, the slope of the utilization ratios should experience some noticeable fluctuations. The fourth aggregation method is an approximation of the integral using the trapezoidal method. Using simple available values creates the possibility that the integral would be biased towards specific host hardware types. To circumvent this issue, the area between the axis and the minimal available value is subtracted from the total area. This ensures that there is no bias related to the specific type of host hardware that the data is being gathered from. The final aggregation value outputs the most recent set of values in the sliding window.

D. Classification Information

As stated previously, the WEKA library is used as the classification engine. From WEKA, two different neural networking classification algorithms were selected: multilayered perceptron (MLP) and an Elman recurrent neural network. MLPs are one of the more classic neural networking algorithms. This algorithm uses a training data set to arrange weight vectors between different layers in a neural network. The goal is to create the weights in such a way that when a specific input vector is provided, the multilayered perceptron can correctly identify how these weights should be classified.

Elman recurrent neural networks are similar to the multilayered perceptron, except they also add the concept of memory. By creating an extra hidden layer with inputs and outputs to an internal hidden layer, it is possible to maintain a type of state. This state makes the Elman recurrent network better at predicting sequences, and can potentially improve classification accuracy. Since behavior changes over time, the recurrent network should be able to maintain behavioral state, which could potentially improve the accuracy of predictions.

Elman RNNs work well in cases where state information is required by the classification system; in this case we have a "memory" requirement related to the previous iterations of our data aggregation.

There are a total of 8 different classification states specified in the configuration of the system. The first state is *Working*. This state represents a host that is currently not experiencing failure symptoms. The second state is *Failing Processor*. This state represents a host that is currently experiencing failure symptoms in its processor. The third state is *Failing Fast Processor*. This state represents a host whose processor is failing at a significantly faster rate than normal. There are also states for *Failing RAM*, *Failing I/O*,

and *Failing Bandwidth*. The last failing state is *Dynamic Failing Processor*. In this state, the processor is failing variably according to its performance. For instance, a faster processor will experience greater variability in its failure than a slower processor. The final state is the *Failed* state, indicating that the host cannot perform further operations.

VII. Simulation Results and Analysis

Figure 3 shows a sample of the different host-behavior execution patterns for a specific simulation. The behavior exhibited by the failing hosts shows that there is a lot of noise in the availability of resources due to the assignment of tasks and VMs. In addition, we can see that there is no obvious failure trend, even though these machines are failing. However, by zooming in, one can see that there are some micro trends in the execution patterns of these systems. This is due, in part, to how the failure behavior was defined. While the means were set to be negative, the CVs were set to be fairly high, forcing the host to experience performance variations more frequently.

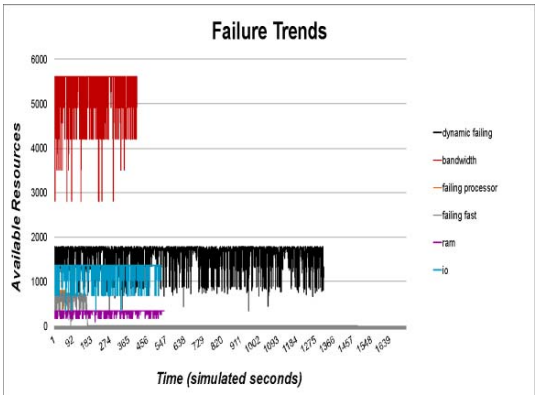

Figure 3. Failing Host Resource Availability

Figure 4 provides a more detailed look at the performance of the failing RAM host, making it easier to see the micro trends in failing performance characteristics. Various dips in available resources show that the host is experiencing fluctuations in the available RAM, even if those fluctuations themselves are very small. These isolated dips in available RAM lead to multiple micro trends of decreasing available resources. The goal of the classifier is to detect when these micro trends begin to occur in order to produce a correct behavioral prediction. Figure 5 shows the prediction accuracy of the MLP when using all the host specification information. The addition of simulation time adds bias to the classification algorithm. This bias forces an association between failure time and the behavioral classification, which leads to poor accuracy when testing the network on data where failures do not occur in similar time frames. This model can predict *Working* and *Failed* hosts and has a tendency to correctly select *Failing RAM* fairly accurately. *Failing RAM* host behavior has a high mean and smaller CV, which creates a more distinct utilization curve and improves prediction precision.

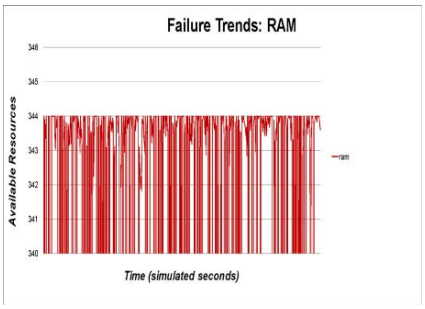

Figure 4. Detailed Failing Host RAM Availability

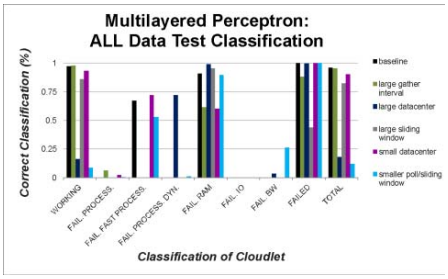

Figure 5. All Multilayered Perceptron Classification Accuracy

Figure 6 shows the prediction accuracy of the MLP when using just the averages of all the host specification information. Average is the most basic aggregation method and does not take into account trends over time. Consequently, the neural networks tend to directly associate certain average values with various classifications. However, multiple hosts experience variations in performance at multiple points in time, which makes the classification of a host's behavior difficult. As shown by the data, singling out averages resulted in poor prediction accuracy for every test case (excluding RAM). This discrepancy was partially due to the fact that failing RAM behavior was the most commonly predicted behavior.

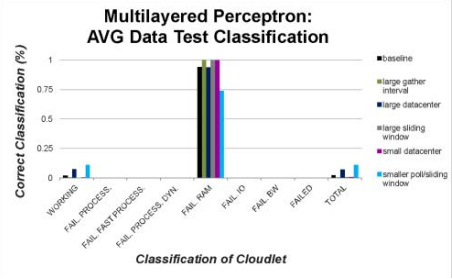

Figure 6. Average Only Multilayered Perceptron Classification Accuracy

Figure 7 shows the prediction accuracy of the MLP when not using time and the current aggregation method on all of the data. This network was the most accurate resultant network. Removing time from the equation made the classifier focus more on converging the weights to predict the trends in the data associated with the failure

behavior. This classifier was able to determine that a host was failing, and how it was failing, with roughly 50% accuracy.

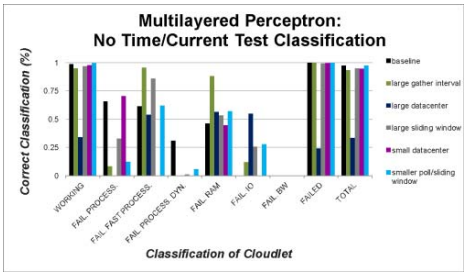

Figure 7. No Time and Current Multilayered Perceptron Classification Accuracy

There are a few outliers in this data, however. It can be seen that I/O failure and BW failure have very low prediction rates. This is due, in part, to how CloudSim handles BW and I/O. I/O doesn't have available or total values because it is a rate. Thus, it does not have a maximum performance value. Therefore, I/O can fluctuate drastically in the positive and negative directions, whereas other values, like RAM, have a maximum limit. This lack of a maximum limit ends up hurting the predictability of I/O performance: It is difficult to tell whether or not I/O is fluctuating due to failure or if it is normal behavior. If the CV was more limited for I/O, I/O failure could be easier to predict due to an obvious downward trend in the I/O rate. BW, on the other hand, has two extreme failure behaviors, one where the CV is 0.1 and the mean is -5, and another where the CV is 1 and the mean is -2. These differences in failure behavior create a split in the classification, which leads to poor accuracy. In addition to this split, BW is constantly drained by the hosts. While we currently are not able to attribute this to any specific cause, it leads the standard working behavior to showcase a negative trend. This conflicts with the prediction of failing behavior and leads to incorrect classifications a majority of the time.

Similarly, the dynamic processor failure behavior is difficult to classify for the majority of classifiers. Dynamic processor behavior takes the performance of the machine and divides it by a constant in order to obtain the mean for the normal distribution from which the behavior is derived. The CV is then calculated by multiplying the mean by a constant value. In this way, individual performance over time is directly related to the processing power of the host itself. In general, this type of behavior would be difficult to characterize while simultaneously analyzing static behavior. Additionally, if the static behavior matches the dynamic behavior, as it did in most cases, it becomes very difficult to distinguish the difference. The only time the dynamic behavior can be characterized is when the mean and CV are different from all of the static behaviors.

Figure 8 shows the prediction accuracy of the MLP when using only the ratios for each of the measurable host specifications. Utilizing only the ratios, it was possible to attain high accuracy for the I/O failure behavior. Due to the lack of an actual available/total ratio for the I/O value, a ratio wasn't used and the available I/O was used instead. This biased the classification algorithm to focus more on the I/O value because it was experiencing larger fluctuations in comparison to the ratio values, regardless of the actual host behavior. If this test were to be repeated, then I/O would need to either be removed or represented differently.

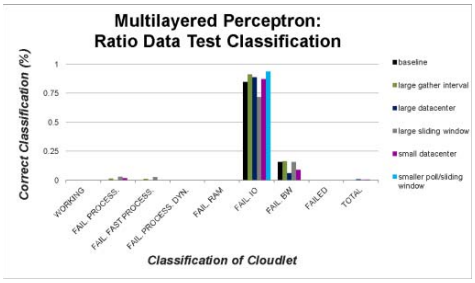

Figure 8. Ratio Only Multilayered Perceptron Classification Accuracy

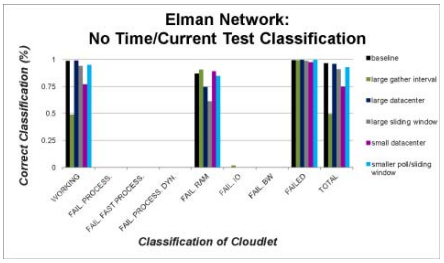

Figure 9. No Time and Current Elman Recurrent Network Classification Accuracy

Figure 9 shows the prediction accuracy of the Elman neural network when not using time or the current aggregation method on all of the data. The Elman network had poor accuracy in predicting the majority of cases due in part to the nature of the algorithm. Ideally, the Elman network would create a classification based on the state of the machine from a previous entry. However, because each host is input to a single network, it is unable to form a state analysis because the hosts themselves are unrelated. In order to take advantage of this algorithm, each host would need its own Elman network. Unfortunately, the WEKA Elman network implementation takes up too much space in memory to be utilized in a simulation on a single machine. In an actual scenario, the network would be running on each host, not a centralized server. Future work will involve testing the Elman network on a per-host basis.

Figure 10 shows a comparison between all of the MLP classifiers for the baseline test case. Figure 11 shows a comparison for the large data center test case. Figure 12 shows a comparison for the smaller sliding window/smaller data gather rate test case. It can be seen that the network that did not train on time and current information had the highest overall accuracy. In addition, we can see that the large data center test was the most difficult to classify correctly. This is due, in part, to the lack of training data for

a large data center. While the slope and integral values should be useful in helping predict host behavior for virtually any size data center, there is still a lot of average information associated with the smaller data centers. Future tests would involve removing the average value and seeing if that affects prediction accuracy when the data centers themselves change sizes.

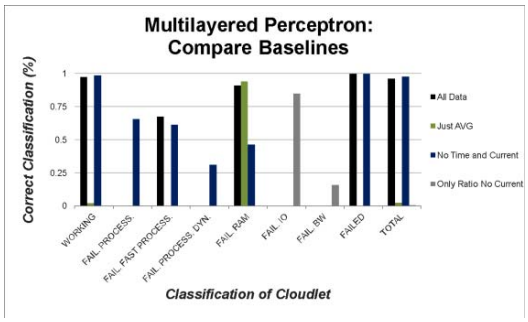

Figure 10. Comparison between Baseline Testing Configurations

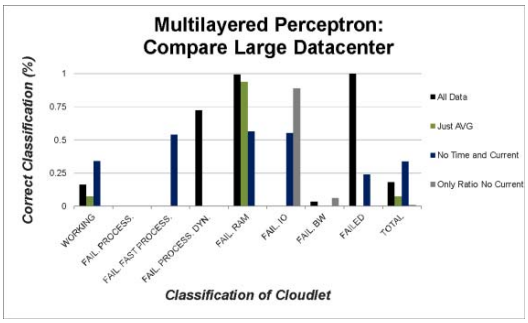

Figure 11. Comparison between Large Data Center Testing Configuration

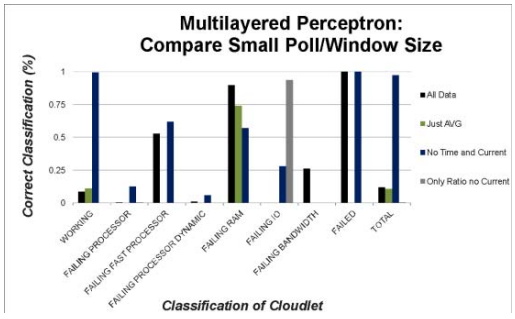

Figure 12. Comparison between Small Window and Poll Size Testing Configurations

An additional point to note is that a host destined to fail is always classified, in order, as *Working*, *Failing*, *Failed*. There are multiple points in time that a host is considered *Failing* before being designated as *Failed.*

The classification data and algorithm can classify failure correctly roughly 50% of the time, and when BW, I/O, and dynamic cases are removed, this rises to roughly 58% accuracy. Additionally, this classification data and algorithm can determine a host will fail before it has failed with an 89% accuracy, which rises to 98% with the removal of outliers. With improved failure modeling, this classification system has the potential to be incredibly useful for cloud management systems as it can help determine when a host is experiencing failure symptoms before actual failure. With this programmatic prediction, it becomes possible to improve the performance of the cloud by assigning tasks on working hardware.

VIII. Conclusions

In this paper, we presented FailureSim, a new simulator of cloud data centers that supports failure prediction. The system allows users to assign behaviors to hosts in a cloud data center. FailureSim defines a set of working and failing host behaviors. The simulator includes a classification system that allows users to predict when hosts begin to exhibit symptoms of failure. Using the most accurate model, our classification system was able to correctly predict 50% of failing hosts correctly, while successfully managing to predict 89% of all failing cases before they occurred.

IX. Acknowledgment

This work is supported by a grant from NASA through the NM Space Grant Consortium.

X. References

[1] R. Miller, "Failure rates in Google data centers," in Data Center Knowledge, 2008.

[2] Y. Sverdlik, "One minute of data center downtime costs US $7,900 on average," DatacenterDynamics, 2016.

[3] X. Chen, C. D. Lu and K. Pattabiraman, "Failure Prediction of Jobs in Compute Clouds: A Google Cluster Case Study," *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),* Naples, 2014, pp. 341-346.

[4] J. Duggan, Y. Chi, H. Hacigümüş, S. Zhu and U. Çetintemel, "Packing light: Portable workload performance prediction for the cloud," *IEEE 29th International Conference on Data Engineering Workshops (ICDEW),* Brisbane, QLD, 2013, pp. 258-265.

[5] S. Ganguly, A. Consul, A. Khan, B. Bussone, J. Richards and A. Miguel, "A Practical Approach to Hard Disk Failure Prediction in Cloud Platforms: Big Data Model for Failure Management in Datacenters," *IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, Oxford, 2016, pp. 105-116.

[6] A. Bahga and V. K. Madisetti, "Analyzing Massive Machine Maintenance Data in a Computing Cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 10, pp. 1831-1843, Oct. 2012.

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Aug. 2010.

[8] M. Bux and U. Leser, "DynamicCloudSim: simulating heterogeneity in computational clouds," 2nd ACM SIGMOD Work. Scalable Work. Exec. Engines Technol. (SWEET), 2013.

[9] http://www.cloudbus.org/cloudsim

[10] A. Alwabel, R. Walters, and G. Wills, "DesktopCloudSim: Simulation of Node Failures in the Cloud," *Sixth International Conference on Cloud Computing, GRIDs, and Virtualization*, 2015, pp. 14–19.

[11] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten, "The WEKA Data Mining Software: An Update," SIGKDD Explorations, Volume 11, Issue 1, 2009.