Towards Implementing ML-Based Failure Detectors

Xiaonan Li and Olivier Marin
New York University Shanghai, Shanghai, China
emails: [xl2149, ogm2]@nyu.edu

***Abstract*—Most existing failure detection algorithms rely on statistical methods, and very few use machine learning (ML). This paper explores the viability of ML in the field of failure detection: is it possible to implement an ML-based detector that achieves a satisfactory quality of service? We implement a prototype that uses a basic long short-term memory neural network algorithm, and study its behavior with real traces. Although ML model has comparatively longer computing time, our prototype performs well in terms of accuracy and detection time.**

***Index Terms*—failure detection, machine learning.**

I. Introduction

Distributed systems should provide reliable and continuous service despite failures. In [1], Chandra and Toueg show that failure detection is the dominant factor in system unavailability, and introduce the notion of unreliable failure detector (FD) as a theoretical construct to extend the applicability of distributed algorithms such as consensus and atomic broadcast. An FD is an oracle which monitors a remote process P, and assesses in real time whether P is up or has crashed. The assessment is unreliable because an FD might provide incorrect information over limited periods of time.

Machine learning (ML) performs well upon analyzing extremely regular data, but network latency on a link can vary often and significantly over time, and failures or delays are transient events that occur irregularly. To achieve high accuracy rates consistently, an ML-based FD must constantly train over newly emerging data. Resource greediness can be prohibitive, since a crucial aspect of an FD is its output rate. Our main goal is to study the viability of FDs based on ML techniques; such FDs should require minimum resources and time during real-time training while maintaining high accuracy. In this paper, we present a preliminary approach that uses a basic long short-term memory neural network algorithm. After comparing its output with a baseline FD, we further optimize our model's performance by adjusting its parameters and structure. Our simulations based on real traces show that our model performs well in terms of accuracy and detection time, despite that the ML model incurs non-negligible computation time.

II. Related work

An FD is a process that monitors remote processes, and strives to estimate whether they're still up or whether they've crashed. The authors of [2] prove formally that there is no way to determine the failure of a node in a distributed environment with 100% certainty. To make up for this, distributed systems can establish a network of mutual observation via *unreliable* FDs [1]. Every node p sends heartbeats to its counterparts

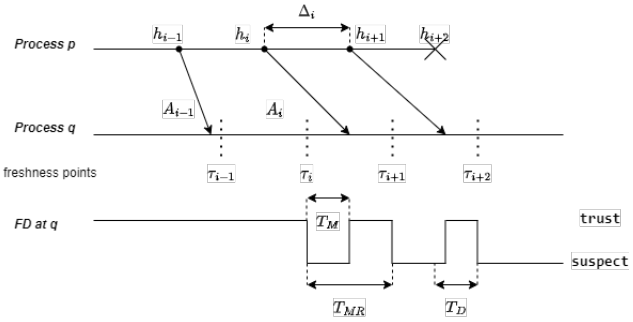

Fig. 1. Estimating the arrival time of the next heartbeat to suspect failures.

q periodically, every Δ_i (Figure 1). To assess the status of p ($trust$ or $suspect$), q computes an estimate of the arrival date τ_i of the next heartbeat h_i. If q receives h_i before τ_i, then it considers that p is up. Otherwise, q starts suspecting p of having crashed. The time that elapses between the emission of h_i and the estimated arrival date τ_i is the *detection time* T_D. If h_i was only delayed, and arrives on q after a *mistake duration time* T_M beyond τ_i, then q stops suspecting p. The *mistake recurrence time* T_{MR} is the time elapsed between two incorrect suspicions. These metrics are important because they can be used to measure the performance of an FD implementation. For now, let's just consider that a high performance FD implementation aims for the shortest possible average detection time and the largest possible average mistake recurrence time.

Implementing an FD is a challenge, because it must contend with the unpredictable and asynchronous nature of network links while preserving its set properties in terms of completeness and accuracy [3]. Early approaches [4], [5] propose adaptive FDs that adjust the timeout delay tolerance dynamically; but they assume an unrealistic timing model with no bound on the delays. Chen et al. [6] overcome this issue by introducing quality of service metrics: *detection time (T_D)* measures the collection speed of correct positives, while the *probability of availability (P_A)* measures the ratio of false positives.

More recent work introduces ML-driven methods for FD-related problems. In [7], the authors use a stacked-LSTM model to classify potential anomaly events in cloud services by analyzing labeled static sensor logs. They conclude that their S-LSTM approach has the ability to quickly learn from the historic patterns and adjust to unpredictable anomalous events. The authors of [8] propose three different ML methods for high-performance computing systems failure detection by analyzing informative hardware usage data. Their best solution, based on the Support Vector Machine method, achieves

a precision of 90%. [9] acknowledges the significance of timestamp data in system logs, and passes both time and token sequence data into the model under the form of interpolated vectors. By using a CNN model, this approach reaches a 99.5% F1 score.

In this paper, we focus on the crash-stop failure model, where nodes cannot recover from failures. Timestamps are the only retrievable information from the monitored sub-systems, yet this information is self-sufficient for detection training. The ML-driven methods mentioned above address a different, extended model where nodes may behave arbitrarily, recover from failures, and send erroneous data. Because of this, they don't apply well to the classic FD problem. For instance, they require pre-labeled data to tag anomalies in the original datasets. Neither [7] nor [8] take timestamps into account while training their models, because the systems they target produce nonuniform timestamps.

III. System Model

In our distributed system set up, every sub system (or node) sends periodical heartbeat messages to the FD, every t milliseconds. The FD generates an expected arrival time for the monitored node's next heartbeat based on its previous behaviors. The FD adds a safe margin to the estimated arrival timestamp to reduce the probability of a false positive detection. We assume a crash-stop model. However, it can easily be extended to a crash-recovery model: The FD will consider a node that recovers as a new node. Restricting the model to crash-stop ensures that all nodes behave normally in the time period where incoming data is recorded. Therefore, the main goal of our algorithm is not to detect behavior anomalies, unlike [7]–[9]. Instead, our FD aims to learn the pattern of heartbeat signals sent by each node in real-time, and to determine an expected arrival time with a high quality of detection: shortest possible detection time, highest possible accuracy.

We use three different metrics to assess FD performance. The first and the most significant one is the *Probability of Availability* (P_A), which is the ratio of the safe predictions over the total predictions. Making a safe prediction means the arrival time of the next heartbeat is earlier than the predicted time, and this is to avoid false-positive results where a node is wrongly suspected. Second, the *Detection Time* (T_D) is the difference between the forecast time stamp and the actual reception time stamp. This metric complements the P_A: it reduces the gap between the predicted time and the actual time, since a model can achieve a high P_A by always predicting absurdly long arrival times. Thus, both metrics balance each other: improving T_D aggressively increases the rate of false-positives, while doing so for P_A increases the risk of false-negatives. Our last metric is the Computation Time T_C. This is important because of the generally high computational cost of non-linear machine learning methods. An FD whose next prediction takes longer to compute than the next arrival date is pointless.

IV. Baseline

We use Chen's FD (CFD) [6] as our baseline for comparison. CFD predicts the next arrival time (Expected Arrival *EA*) upon receiving a heartbeat message. Equation 1 shows how CFD computes its prediction as a statistical analysis of the n latest reception times. To reduce the ratio of false positives, CFD adds a constant safety margin (α) to *EA*.

$$EA_{k+1} \approx \frac{1}{n}\left(\sum_{i=k-n}^{k} A_i - \Delta_i * i\right) + (k+1) * \Delta_i \qquad (1)$$

One of the weaknesses of Chen is the constant α. Setting α by default reduces the performance of CFD for highly unstable traces. Besides, selecting the optimal value for α is a delicate task. It depends heavily on the network environment, and requires thorough monitoring prior to the deployment of CFD.

V. Our ML-Based Failure Detector Design

Similarly to Chen's FD, our approach also bases its prediction on an analysis of the η latest receptions. We use the long short-term memory (LSTM) model: it can effectively retain important long-term information, and it can quickly fit the data compared to traditional time series forecasting models [10]. Moreover, its fitting method is non-linear, which makes its fitting potential stronger than ordinary linear models.

As our goal is to make real-time predictions, we train our LSTM model real-time and non-accumulatively: we only keep the η most recent data items as the training set, and always feed the model with the nearest fixed-size of data. Although we abandon data that falls out of the range, their impact are kept in the parameters. For each real-time training, the model retains the memory of the last training results, and will not restart from scratch. Since we don't store historical data for future training, their impact gradually decreases over time. This training method has two main advantages. First, it speeds up the training process and reduces its computational cost by retaining the memory of the last training results. Second, it gradually reduces the impact of the least recent historical data; this is essential to accommodate the high volatility of network link behaviours. Figure 2 illustrates the mechanism of our model.

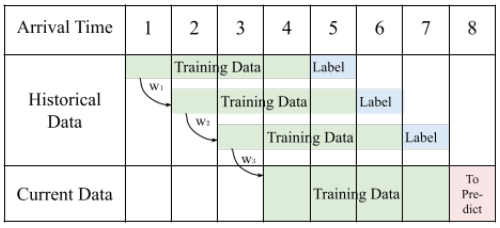

Fig. 2. Our real-time non-accumulative LSTM construction mechanism

We need to customize the adjustment of the prediction of our LSTM algorithm, because loss functions such as *mean squared error* treat negative and positive errors equally. This cannot do for an FD, whose objective is to minimize false

positives. Our solution is to write our own loss function: its goal is to tilt our FD's output towards overestimations of the next arrival date. For this purpose, we design a new loss function that adds a multiplier to the results whenever a negative prediction occurs. By using such a loss function, our model achieves a 95% accuracy rate on its predictions.

To improve the PA further, we add a dynamic safety margin based on the ϵ most recent errors, unlike our baseline model which uses a static one. A safety margin is used to help the model adapt to any unexpected and unstable delay when it occurs, and such a delay is unbounded and hard to detect. So the goal is to adapt the model as soon as such a series of delays occur, so having a dynamic array that keeps the ϵ most recent errors enables the model to adjust in time from the latest error. Altogether, our model computes its next prediction with Equation 2.

$$EA_{k+1} \approx LSTM.predict(k-\eta) + \frac{1}{\epsilon}\sum_{i=k-\epsilon}^{k} error_i \quad (2)$$

In the LSTM model, three parameters are decided by exhaustive search: η (training data set size), the batch size, and the epoch number. The training data set size η decides how much data to learn each time while real-time training; the batch sizes and epoch numbers influence the accuracy and timeliness of training from the structure and nature of training. Grid search is used to search for best combinations. Table I shows the top ten combinations of P_A among all since P_A is the most critical property for evaluation. The top three combinations share the same P_A and similar T_D, however, the second one has an outstanding T_C than the other two. As a result, we used a η of 500, a batch size of 64 and an epoch of 5.

η	*batch size*	*epoch*	P_A	T_D	T_C
500	32	5	0.9957	9.4416	72.9630
500	64	5	0.9957	9.7832	48.1018
500	64	10	0.9957	9.1369	74.2897
1000	32	10	0.9956	11.1584	150.0853
1000	64	10	0.9947	12.2609	89.4486
500	32	10	0.9945	9.2335	116.3436
100	32	5	0.9941	8.0131	45.8082
1000	32	5	0.9938	11.3694	83.6161
1000	64	5	0.9938	11.4039	53.5777
100	32	10	0.9931	7.5950	61.3970

TABLE I
PARAMETER COMBINATIONS THAT ACHIEVE TOP-10 P_A PERFORMANCE

VI. PERFORMANCE EVALUATION

We assess our approach on top of real traces: heartbeat transmission logs collected over a week from 9 nodes on the PlanetLab network (http://www.planet-lab.org/). Nodes 1 through 9 generate a heartbeat message containing the sender ID and a sequence number every 100 milliseconds, and send it to node 0, which we consider as the monitoring node. Upon reception of a heartbeat, node 0 records its arrival time in the log associated with the sender.

We set the parameter values of Equations 1 and 2 as follows: the heartbeat arrivals window size n for Chen's FD (CFD) is 1000; for our Machine Learning-based FD (MLFD), the heartbeat window size η is 500, and the error window size ϵ is 10. We aim to compare the performance of MLFD with that of CFD. However, CFD's performance depends highly on the value of its constant safety margin α. To allow for a fair comparison, we align the P_A values of both models before comparing their T_D; this leads to a value of 680ms for α.

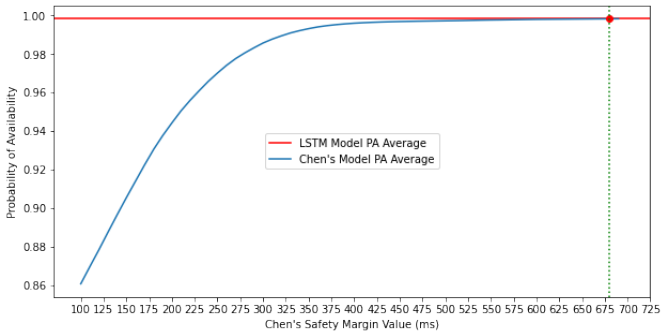

Fig. 3. Influence of CFD's safety margin on its accuracy

Figure 3 shows the alignment mechanism after running both FDs on all 9 links: the blue curve shows the correlation between P_A and α for CFD, and the red line is the average P_A for MLFD (approximately 99.84%). The plot shows that, for CFD's P_A to exceed 99.84%, its α must be above 680ms.

Table II gives the comparison of CFD and MLFD's P_A and Table III compares results in terms of detection time T_D. These results show how much the safety margin of CFD affects its detection time. For a similar P_A outcome, MLFD detects failures much faster than CFD on any given link. There is a caveat to these excellent results: MLFD consumes a lot of CPU to compute its predictions. In the third column, we add an extra metric: the average computation time T_C for MLFD's next prediction. We obtained these results on Linux CentOS v6.5 running on 2 cores of an Intel Xeon E5 2.2GHz with 128GB of memory.

We don't include the computation times for CFD because they remain way below the detection times, and are therefore insignificant. As the table shows, MLFD's computation time will delay the detection time in most cases. The non-linear characteristics of the calculation and the algorithm's complexity make the computation time longer than its calculated prediction interval. Nevertheless, even with such high computation times, MLFD still performs significantly better than CFD.

Link	Chen's P_A (680ms)	MLFD's P_A	Chen's P_A (690ms)
1	1	0.999127	1
2	1	0.997104	1
3	1	0.998439	1
4	1	0.996781	1
5	1	0.999165	1
6	1	0.998457	1
7	0.989449	0.998765	0.989559
8	0.995549	0.998366	0.995732
9	1	0.999055	1
Average	0.998333	0.998362	0.998365

TABLE II
P_A COMPARISON BETWEEN CFD AND MLFD

Link	CFD T_D (680ms)	MLFD T_D	MLFD T_C	MLFD $T_C + T_D$	CFD T_D (690ms)
1	679.093	12.926	123.847	136.773	689.093
2	510.860	96.087	124.529	220.616	520.860
3	676.351	21.827	124.682	146.509	686.351
4	584.397	148.897	124.836	273.733	594.397
5	680.019	12.410	125.254	137.664	690.019
6	678.190	20.400	125.199	145.600	688.190
7	672.419	26.823	125.754	152.578	682.343
8	667.905	32.072	126.229	158.302	677.781
9	675.992	16.160	126.670	142.830	685.992
Average	647.247	43.067	125.222	168.290	657.225

TABLE III
T_D COMPARISON BETWEEN CFD AND MLFD

VII. DISCUSSION

At the beginning of the experiment, we were concerned that the data fluctuations caused by the delays are arbitrary and non-regular, which might prevent the LSTM model from simulating the prediction curve well. But the excellent results show that LSTM can handle data sets that do not present strong regularity. We think the main reason is that, when a significant delay occurs, the following data delays will show a certain degree of consistency. Although LSTM cannot predict a sudden and major delay variation, it can sensitively capture the change and adjust the model parameters to adapt to the delays if they occur in bursts. And once the burst ends, the LSTM model will adapt back to the normal trend.

However, the heavy calculation cost does indeed weigh on the performance of our LSTM model. On average, the T_C of the LSTM is three times as long as its T_D, which seriously affects the performance of the LSTM model on Detection Time. This is obviously a major challenge for the application of machine learning in the field of failure detection. Nevertheless, we remain optimistic about our approach. One reason is that, despite the computational cost issue, the accuracy and detection time of our LSTM based FD is still significantly better than that of CFD. The other reason is that the processing power of the server at our disposal to run the LSTM model is relatively low. Running our model on a more powerful server would shorten the calculation time significantly.

VIII. CONCLUSION

In this paper, we present a preliminary study about the feasibility of failure detector implementations based on machine learning algorithms. Our results suggest that ML may be a viable approach: our LSTM-based FD implements a unilateral penalty loss function, dynamic safety margin, and it trains on the 500 most recent message receptions in real-time. Upon comparison with Chen's, our FD achieves much shorter detection times for a similar probability of availability, but at a significant computation cost.

Our results illustrate the potential of machine learning models in this field, and we hope it will elicit further research. Our next step is to refine our prototype, and to test it against more aggressive FDs with dynamic safety margins [3]. We also intend to pursue our work with a focus on the trade-off between probability of availability and computation time. Evolved machine learning models, such as the transformer's attention model, seem like strong candidates for better performance. The advent of DPUs [11] also opens another promising avenue of research for our work: FDs are obvious candidates for network function virtualization, and SmartNICs have the power to support advanced implementations such as our MLFD.

REFERENCES

[1] T. D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *Journal of the ACM*, vol. 43(2), p. 225–267, 1996.

[2] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, p. 374–382, 04 1985.

[3] M. Bertier, O. Marin, and P. Sens, "Implementation and performance evaluation of an adaptable failure detector," in *Proc. of the Int. Conf. on Dependable Systems and Networks*, 2002, pp. 354–363.

[4] C. Fetzer, M. Raynal, and F. Tronel, "An adaptive failure detection protocol," in *Pacific Rim Int. Symposium on Dependable Computing*, 2001, pp. 146–153.

[5] I. Sotoma and E. R. M. Madeira, "DPCP (Discard Past Consider Present)-a novel approach to adaptive fault detection in distributed systems," in *Proc. 8th IEEE Workshop on Future Trends of Distributed Computing Systems*, 11 2001, pp. 76–82.

[6] W. Chen, S. Toueg, and M. K. Aguilera, "On the quality of service of failure detectors," in *Proc. of the Int. Conf. on Dependable Systems and Networks*, vol. 51, no. 1, 01 2002, p. 13–32.

[7] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Long short-term memory based operation log anomaly detection," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 236–242.

[8] K. Yamnual, P. Phunchongharn, and T. Achalakul, "Failure detection through monitoring of the scientific distributed system," in *2017 International Conference on Applied System Innovation (ICASI)*, 2017, pp. 568–571.

[9] Y. Huangfu, S. Habibi, and A. Wassyng, "System failure detection using deep learning models integrating timestamps with nonuniform intervals," *IEEE Access*, vol. 10, pp. 17 629–17 640, 2022.

[10] D. Gers, Felix A.and Eck and J. Schmidhuber, "Applying LSTM to time series predictable through time-window approaches," in *Proc. of the Int. Conf. on Artificial Neural Networks*, 2001, pp. 669–676.

[11] Y. Qiu, J. Xing, K.-F. Hsu, Q. Kang, M. Liu, S. Narayana, and A. Chen, "Automated smartnic offloading insights for network functions," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21, New York, NY, USA, 2021, p. 772–787.