eDoS Mitigation for Autonomic Management on Multi-Tier IoT

Rajsimman Ravichandiran, Hadi Bannazadeh, Alberto Leon-Garcia
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, Canada
Email: {rajsimman.ravichandiran, hadi.bannazadeh, alberto.leongarcia}@utoronto.ca

***Abstract*—In this age of the Internet of Things and ubiquitous computing, autonomic management has become a critical component in cloud platforms. Autonomic management helps systems adapt seamlessly and efficiently to rapidly fluctuating workloads. However, economic Denial of Sustainability (eDoS) attacks can directly target the autonomic management to waste resources. In this paper, we propose an eDoS mitigation framework that incorporates online anomaly detection with our Elascale autonomic management system to thwart eDoS attacks in real-time. This allows the detection system to be application-agnostic as this framework utilizes only resource statistics of the monitoring applications. We present the design and implementation of our anomaly detection framework with Elascale. We evaluate Hierarchical Temporal Memory (HTM) and Tukey with Relative Entropy against spatial and temporal anomalies. Our results prove that the HTM-based anomaly detection method outperforms with significant accuracy.**

I. Introduction

The Internet of Things (IoT) is a catalyst in connecting our everyday lives with ubiquitous devices around us. Numerous sensors constantly monitor or provide information to help make our lives comfortable and safe. With the collection and streaming of data comes a challenge for cloud infrastructures to handle the volume, velocity, and variety of the incoming data. Furthermore, the volatile behaviour of IoT applications forces cloud platforms to adapt to the workload efficiently. This problem motivates the use of Autonomic Management System (AMS) to monitor the usage of cloud resources and help the system adapt to increased (or decreased) workload.

However, incautiously increasing resources may lead to wastage in cloud infrastructures. There can be direct security attacks (economic Denial of Sustainability (eDoS) [1]) on the autonomic management to make it waste resources and deny proper distribution to other applications or services. These attacks exhibit abnormal behaviour patterns in the system [2] and this guides us to incorporate anomaly detection mechanisms in our system to mitigate eDoS attacks [3].

We envision future large-scale IoT deployments will be integrated with cloud resources in a multi-tier fashion [4]. The sensors in the physical environment stream data to an Aggregator (or IoT gateway) which can be located inside or outside the cloud. The Aggregator can gather the data, perform some low computing operations and send it to an Edge node located close to the IoT environment. The Edge node can perform heavier real-time processing (compared to the Aggregator), analytics, and send the data to the Core node (located in the data center) for storage or further processing. We implemented Elascale AMS to support this multi-tier IoT platform. Elascale can monitor both microservices and macroservices in cloud infrastructures [5]. A microservice is a component of an application that is designed to do a specific task. We define a macroservice as a resource that hosts multiple microservices of an application. If more microservices are spawned to handle a heavy workload, it may be necessary to scale macroservices as well to provide sufficient resources for those components.

978-3-903176-14-0 ©2018 IFIP

Our contribution in this paper utilizes known online anomaly detection methods to thwart eDoS attacks for autonomic management system on multi-tier IoT platform. We describe our implementation of the proposed framework and present experimental results by evaluating these methods on an IoT application against various anomalous scenarios.

The organization of this paper is as follows. Our proposed approach and anomaly detection implementation are described in sections 2 and 3, respectively. In section 4, we describe the experimental setup and in section 5, we present our evaluation results. The final section describes our conclusion and future work.

II. Proposed approach

The eDoS mitigation framework utilizes online anomaly detection to minimize eDoS attacks. The detection is performed before applying adaptive algorithms to scale resources. In order to choose the most accurate anomaly detection model, we evaluated Hierarchical Temporal Memory with anomaly likelihood (HTM+AL) and Tukey with Relative Entropy (Tukey+RE) for different attack scenarios (described in evaluation section). HTM is a machine intelligence framework based on interactions of pyramidal neurons in the neocortex of the brain and can be utilized to perform anomaly detection using Anomaly Likelihood mechanisms (HTM+AL) [6]. We chose HTM+AL as it has been evaluated against various techniques and proved to be the best algorithm according to the Numenta Anomaly Benchmark dataset [7]. We chose Tukey+RE [8] as it is more lightweight than HTM+AL [6] and can be deployed on devices with less computing power which can be useful for IoT environments.

Resource statistics from each microservice of an application are streamed into the detection model to identify any anomalies. If the application is new to the detection system, first-time anomalies will go undetected as the system is still learning the application's behaviour. Thus, if the application starts receiving a large workload, the autonomic management will provide more resources. Once the anomaly detection model learns the application's behaviour and recognizes anomalous behaviour regarding that large workload, the AMS will remove any extra resources provided to the application. Furthermore, the management will put the application in an *investigation phase*, where it 1) checks to see if other anomalies are detected within this phase; and 2) prevents the AMS from adding extra resources to minimize economic damage. In order to exit the *investigation phase*, no anomalies should be detected within the period. If any anomaly is found, the AMS resets the cycle

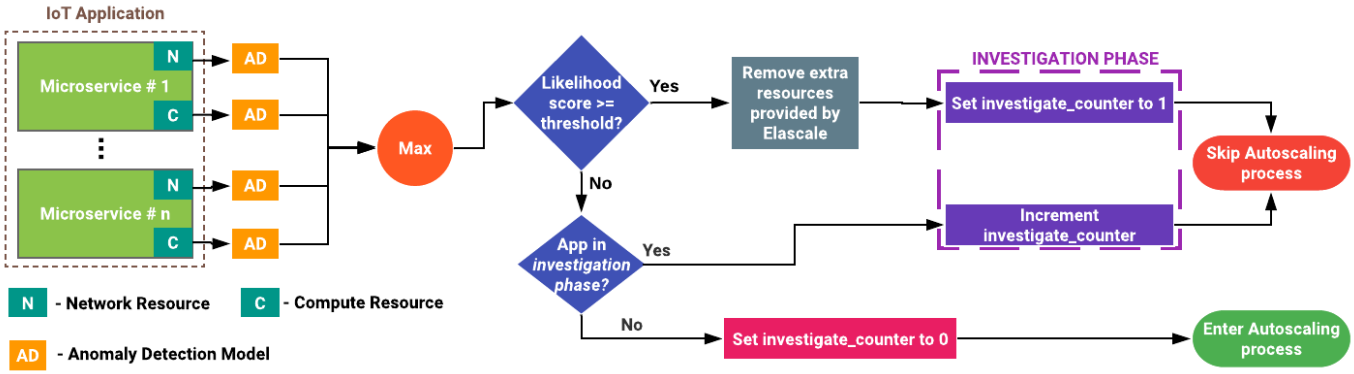


Fig. 1: eDoS mitigation framework flowchart within Elascale autonomic management. Once the application enters the *investigation phase*, we use the investigate_counter to represent the number of consecutive samples of the application that have been labelled as non-anomalous. If the investigate_counter equals the length of the *investigation phase*, we consider the application non-suspicious (set the counter to 0) and continue with autoscaling process.

of the *investigation phase* and continues observing. Figure 1 shows the flowchart for the eDoS mitigation framework for our AMS. The length of the *investigation phase* can be configured based on the application security requirements. Longer period allows the owner to monitor the abnormality longer, but risk denying better quality of service to legitimate users. This is because any resources provided to handle the large workload are removed during the period.

III. Anomaly Detection Implementation

Two separate detection models (for network and compute resources) are created for each microservice as they have distinct behaviours (based on the application's resources usage) and may not correlate with each other. After the learning phase (or probationary window), we collect the anomaly likelihood score predicted by both the models. The likelihood score defines how likely that the current data point is an anomaly. The range of the score is 0 to 1 and a larger value indicates a higher probability that the current value is an anomaly. Once we gather the likelihood scores from all the microservices (of an application), we find the sample maximum and if the value crosses some threshold, we consider the application anomalous. We chose the standard threshold 1 - ϵ where $\epsilon = 10^{-5}$ as suggested in [6] and it worked well during our evaluation as well. Since our sampling frequency is 6 samples/minute, we set the length of the *investigation phase* to be 6. For evaluation purposes, we set the probationary period to be 15% of the evaluation period as it is common practice using this methodology [9].

Furthermore, we utilize the scores from all the microservices because if one component of an application behaves abnormally, it is bound to affect other components as well. Hence, the decision takes into account of all microservices within the application and skip the autoscaling process if any component exhibits abnormality. We implemented the anomaly detection framework into the Elascale AMS and evaluated it on the SAVI Testbed [10]. The source code of the implementation is online and open-source [11].

IV. Experiment Setup

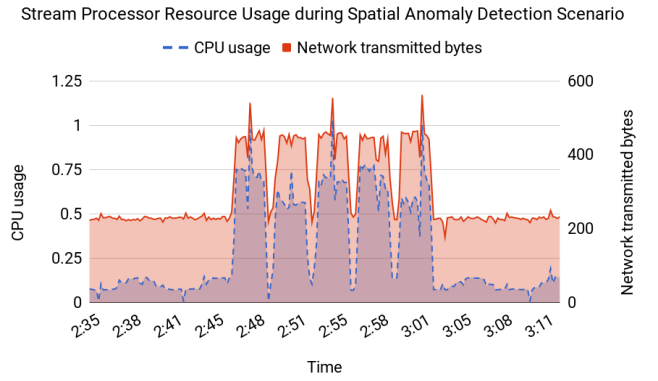
We developed a sample IoT application on the SAVI Testbed to test various scenarios. The application consists of four microservices: virtual sensors, aggregator, stream processor, and a database. The components are deployed in various macroservices in order to mimic various layers of a multi-tiered IoT application. The sensor and the aggregator are deployed on a macroservice that represents the sensor/aggregator layer, while the stream processor is on another macroservice that depicts the edge layer. Finally, the database is located at the core layer. All the macroservices are Ubuntu 16.04 virtual machines with 5 GB memory and 2 CPUs.

The virtual sensors are used to simulate added workload (e.g. DDoS attacks or legitimate workloads) in the form of streaming monitoring information (CPU and network usage, memory, processes, etc.). We can scale up/down the virtual sensors to increase/decrease the workload. The virtual sensors send the monitoring information to the aggregator, which uses Kafka to forward the sensor data to a stream processor. The stream processor analyzes the incoming sensor data to perform text classification and data filtering before feeding them to the database, which uses Cassandra to store historical data. Once the IoT application is deployed, we evaluate the detection mechanisms for different anomalous scenarios.

V. Evaluation

We evaluate our detection framework by comparing the accuracy between the anomaly detection models for spatial and temporal anomalous scenarios. For evaluation purposes, we need to assign the size of the anomaly and probationary windows. Anomaly windows are used since anomalous data do not occur at a single point, but rather occurs over time. Hence, having a window improves the effectiveness of the scoring policy. We chose the window size to be 10% of the evaluation period divided by the number of anomalous samples as this is common practice when evaluating anomaly detection mechanisms [9]. Our evaluation period for all the scenarios is 2 hours with the sampling frequency of 6 samples per minute. The period will contain $2 \times 60 \times 6 = 720$ samples, and our anomaly window will be $0.1 \times 720 = 72 \div$ number of anomalies during that period (depending on the scenario). One half of the window will be before the anomaly sample and the other half will be after the sample. If multiple anomaly windows overlap, we combine the windows as we assume they identify the same anomalous data.

Probationary windows are used by online detection mechanisms to model or learn the behaviour of the streaming data. It is usually the first x % of the evaluation period or dataset. As per [9], a common practice is to set the window as 15% of the period. Since our evaluation period is 2 hours, the probationary window will be 108 samples for all the scenarios. For each scenario, we collect the number of True Positives (TP), False Positives (FP) and False Negatives (FN) and compare the accuracy of the models based on the F1 score. We consider



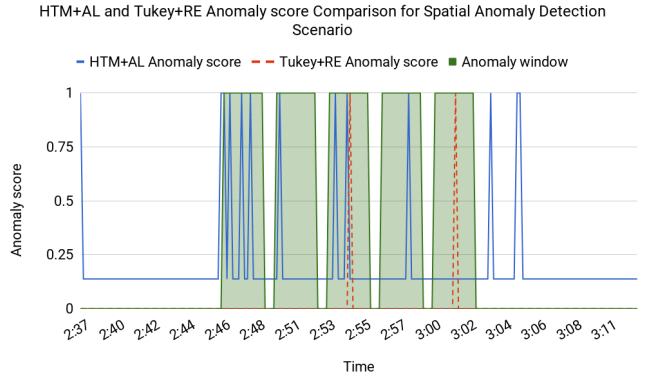
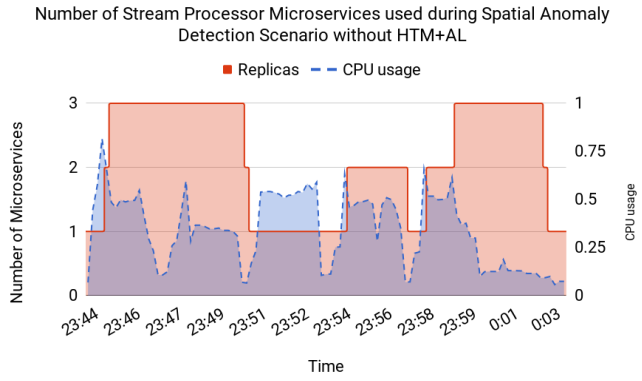


Fig. 2: Spatial anomalies are injected by increasing the workload for the stream processors as represented by the five bursts in the first plot. The next plot shows the anomaly scores produced by both HTM+AL and Tukey+RE models during the anomalous periods.



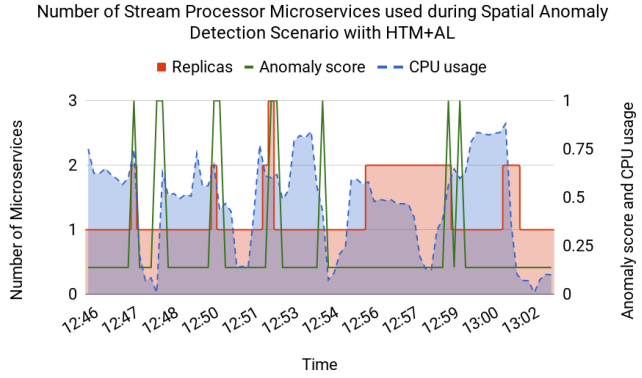


Fig. 3: Without HTM+AL, the AMS scaled microservices to handle the unnecessary workload produced by spatial anomalies, as shown in the first plot. In the next plot, the AMS was able to detect most of the spatial anomalies with HTM+AL and minimized the economic damage by 38.12%.

TP if the model can detect any point within the window and the score equals the number of samples within the correctly detected anomaly window. We consider FN if the detection method does not detect any sample within an anomaly window and the score equals the number of samples within the missed anomaly window. Finally, we consider FP if the detection method falsely detects the correct sample as anomalous and the score equals the number of erroneously detected samples. Once we correctly examined the TPs, FPs and FNs, we compute the Precision ($\frac{TP}{TP+FP}$), Recall ($\frac{TP}{TP+FN}$) and finally F1 score ($2 \times \frac{Precision \times Recall}{Precision + Recall}$).

A. Spatial Anomaly Detection

We emulate a scenario where multiple rogue sensors attack the stream processor by sending sensor data at a high frequency. This leads to a structural change in microservices' behaviours and implies a spatial anomaly in the series. We implement this scenario by increasing the number of sensor microservices. Without the anomaly detection mechanism, Elascale would simply increase the stream processors to handle the workload which results in wastage of resources. Hence, we evaluate whether our anomaly detection can help avoid the eDoS attack.

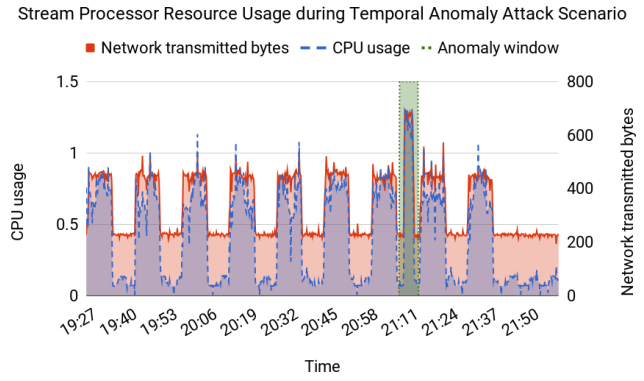
The first plot of Figure 2 shows the utilization of resources for stream processor microservices during the attack. As mentioned before, our evaluation period is 720 samples or 2 hours, and our probationary period is set 108 samples or 18 minutes. The attack can be observed by a period of spikes in the utilization metrics, where we increase the sensors for two minutes to create a heavy workload and decrease the sensors back for a minute. An anomaly represents one heavy workload period, and since there are 5 On attack periods, the number of anomalies for this evaluation period is 5. Since the anomalous window equals to $0.1 \times 720 \div 5 = 14.4$ samples, the window encapsulates an On attack period and we have total 5 anomaly windows for this dataset.

1) Comparison of Results

The last plot of Figure 2 shows the anomaly scores produced by both models during the attack period. The HTM+AL model was able to detect 4/5 windows, hence TP = $14.4 \times 4 = 57.6$ or 58, and FN = 72 - 58 = 14. The model also produced 3 FP during this period. The Tukey+RE model was able to detect only 2/5 windows, hence TP = 28.8 or 29, and FN = 43. However, the model did not produce any FP, hence Precision was 100%. Based on the F1 scores (as summarized in Table I), we see that HTM+AL outperformed Tukey+RE significantly for the spatial anomaly detection scenario.

2) Economic Impact

Since HTM+AL is the better detection method, we can measure the economic impact by the difference in average resource usage (in units of microservice-seconds) with and without the mitigation algorithm. The first plot of Figure 3 shows the number of stream processor replicas used during the attack when HTM+AL is not used. The average resource usage is 129,553 microservice-seconds without HTM+AL. When detection method was used, the average resource usage equals to 80,172 microservice-seconds. Thus, HTM+AL minimized **38.12%** of economic damage.



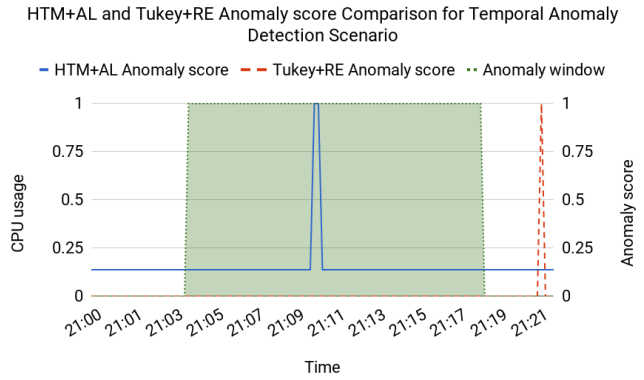
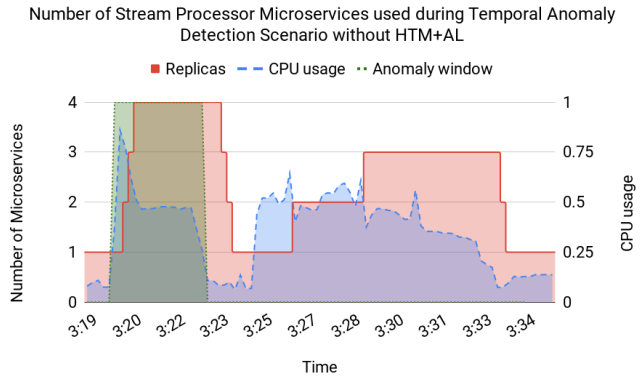


Fig. 4: Regular workload consists of high workload followed by low workload. Temporal anomalies were injected where high workload was observed but low workload was expected, as shown in the first plot. The next plot shows the anomaly scores produced by both HTM+AL and Tukey+RE models during the anomalous period.



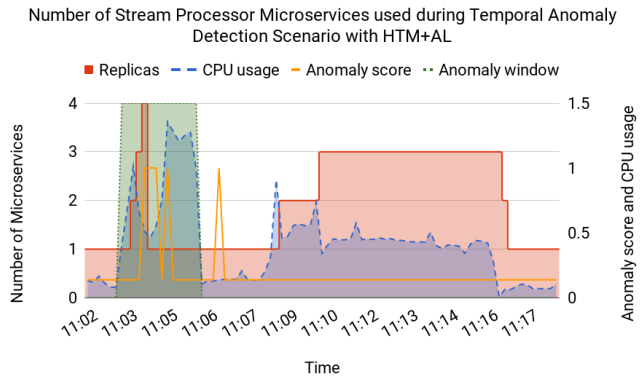


Fig. 5: Without HTM+AL, the AMS scaled microservices to handle the unnecessary workload produced by temporal anomalies, as shown in the first plot. In the next plot, the AMS was able to detect most of the temporal anomalies with HTM+AL and minimized the economic damage by 64.17%.

B. Temporal Anomaly Detection

In this scenario, we simulate an attack where the stream processors display a temporal abnormality. Before presenting the anomalies, we need to provide temporal context for the stream processor. We first create a cyclic behaviour for the stream processors and then insert abnormal behaviour in-between these periods. In our use case, we create a 15-minute periodic pattern where the processors perform a high workload for half the period (7.5 minutes) followed by a low workload for the other half. We simulate high workload by increasing the sensor microservices to increase the stream processor's CPU usage and network resources. Since we evaluate for 2 hours, we will have 8 cycles of this behaviour. We insert an anomaly by increasing the workload (for 2.5 minutes) during the expected low workload pattern, as shown in the first plot of Figure 4. Since we insert only one anomaly, our anomaly window is $0.1 \times 720 \div 1 = 72$ samples or 12 minutes.

1) Comparison of Results

The last plot of Figure 4 shows the anomaly scores produced by both models during the attack period. The HTM+AL model was able to detect the anomaly within the window, hence TP = 72 and FN = 0. The model also produced 10 FP during this period. The Tukey+RE model was not able to detect any sample within the window, hence TP = 0, FN = 72 and the model produced 2 FP. Based on the F1 scores in Table I, we see that HTM+AL significantly outperformed Tukey+RE for the temporal anomaly detection scenario.

2) Economic Impact

Figure 5 shows plots of stream processor replicas used during the attack with and without HTM+AL, respectively. We calculated the resource usage to be 9,963 microservice-seconds without HTM+AL. When detection method was used, the resource usage is 3,570 microservice-seconds, thus minimizing **64.17%** of economic damage. Note that these quantities are substantially smaller than the values from spatial anomaly detection scenario, as the attack period for the temporal case is significantly shorter than the spatial case.

TABLE I: Evaluation results summary

Anomaly	Model	TP	FP	FN	Precision	Recall	F1
Spatial	HTM+AL	59	4	13	93.5%	81.9%	**87.4%**
	Tukey+RE	29	43	0	100%	40.4%	**57.5%**
Temporal	HTM+AL	72	10	0	87.8%	100%	**93.5%**
	Tukey+RE	0	2	72	0%	0%	**0%**

VI. Conclusion and Future Work

In this paper, we presented our approach to mitigating eDoS attacks using an HTM+AL mechanism for Elascale's autonomic management on a multi-tier IoT application. Our detection system is application-agnostic as it monitors only utilization of resources by applications. We developed and deployed the application on the multi-tier SAVI Testbed and evaluated it against other detection techniques for various scenarios. From the results, we show that HTM+AL is more accurate in detecting anomalies for the AMS than the Tukey+RE model. As future work, we will investigate techniques that can detect correlated spatial-temporal anomalies and incorporate intrusion detection systems to improve accuracy.

VII. Acknowledgement

This research was supported in part by NATO Grant Science for Peace and Security Programme.

REFERENCES

[1] C. Hoff. (2008, Nov.) Cloud Computing Security: From DDoS (Distributed Denial Of Service) to EDoS (Economic Denial of Sustainability. Accessed: Apr. 2018. [Online]. Available: http://rationalsecurity.typepad.com/blog/2008/11/cloud-computing-security-from-ddos-distributed-denial-of-service-to-edos-economic-denial-of-sustaina.html

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009.

[3] R. Ravichandiran, H. Bannazadeh, and A. Leon-Garcia, "Anomaly detection using resource behaviour analysis for autoscaling systems," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, June 2018, pp. 192–196.

[4] H. Khazaei, H. Bannazadeh, and A. Leon-Garcia, "SAVI-IoT: A Self-Managing Containerized IoT Platform," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, Aug 2017, pp. 227–234.

[5] H. Khazaei, R. Ravichandiran, B. Park, H. Bannazadeh, A. Tizghadam, and A. Leon-Garcia, "Elascale: autoscaling and monitoring as a service," in *CASCON*, 2017.

[6] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134 – 147, 2017, online Real-Time Learning Strategies for Data Streams.

[7] "The Numenta Anomaly Benchmark," GitHub, Online Code Repos, Accessed: Apr. 2018. [Online]. Available: https://github.com/numenta/NAB

[8] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical Techniques for Online Anomaly Detection in Data Centers," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, May 2011, pp. 385–392.

[9] "The Numenta Anomaly Benchmark," White Paper, Numenta, Inc., Redwood City, CA, 2015, Accessed: Apr. 2018. [Online]. Available: https://github.com/numenta/NAB/wiki

[10] T. Lin, B. Park, H. Bannazadeh, and A. Leon-Garcia, *SAVI Testbed Architecture and Federation*. Cham: Springer International Publishing, 2015, pp. 3–10.

[11] "Elascale_secure," GitHub, Online Code Repos, Accessed: Jun. 2018. [Online]. Available: https://github.com/RajsimmanRavi/Elascale_secure