AMAS: Adaptive Auto-Scaling for Edge Computing Applications

Abstract—Despite the emergence of edge computing as a key technology paradigm, there is a general lack of auto-scaling techniques specifically designed for edge computing applications. Further, existing auto-scaling solutions tailor-made for the cloud cannot be readily applied to an application running on an *edge cluster*, i.e., a cluster of edge devices. In this paper, we present AMAS - a novel auto-scaling algorithm, designed specifically for an edge cluster, which allows edge devices to be automatically and seamlessly added or deleted from an edge cluster according to dynamic variations in the workload. By design, AMAS can help users and enterprises minimize the cost of infrastructure while maintaining necessary SLO (i.e., Service Level Objective) deadlines for different edge applications. We demonstrate that AMAS outperforms the state-of-the-art auto-scalers in failure-prone conditions.

1 Introduction

We consider edge computing, such as smart health monitoring [1], agro-analytics [2], autonomous vehicle control [3], etc., which mostly deal with considerably large datasets and work under strict SLO deadlines and an infrastructure budget specified for acquiring and maintaining the devices, the cluster must be comprised of sufficient resources to be able to process such datasets, satisfying the SLO deadlines [4]. While edge computing allows us to spawn an ad hoc *edge cluster*, i.e., a cluster of cheap and easily available edge devices, the challenge is to enable the above cluster to be workload-aware for handling dynamic and intensive workload while restricted by an infrastructure budget for acquiring and maintaining such devices. This necessitates the incorporation of elastic scaling capabilities in the edge cluster. The performance of low-end computing devices comprised in an edge cluster is directly associated with the volume of workload being processed. For instance, if the image resolution of the camera feeds at traffic junctions is increased for analysing the images with a higher level of details and an improved accuracy, the computation latency could drastically rise with the existing cluster composition. In such a case, the system should be aware of the real-time workload volume and dynamically auto-scale the cluster to achieve a reduced processing latency without compromising the accuracy of the analytics task performed, and also keeping the operational cost under bounds.

Most IoT applications of today are characterized by time-varying workload patterns. Hence, despite the ad hoc nature of the edge cluster that hosts such applications, the cluster must be able to dynamically scale in or out in response to the variations in the workload, i.e., devices must be dynamically added/removed to/from a cluster depending on the current workload. With traditional cluster computing paradigms such as cloud computing, the servers comprising a cluster are housed in secure data centers with stable network connectivity, making elastic scaling a predictable and smooth process. However, in edge environment, elastic scaling has its own challenges. The principal aspects for an end-to-end elastic auto-scaler in edge computing involve continuous monitoring, real-time detection and proactive resolution of frequent joining and leaving of devices, network partitions, and sudden node failures which occur frequently during the processing of compute/data-intensive workload on resource-constrained edge devices. In cloud computing, additional physical servers are already available in secure data centers over a reliable network and can be smoothly added to the cluster in real-time. However, in edge applications, new devices to be added to the cluster need to be dynamically discovered through proactive probing of the network for "active" devices. Due to secure and

- *Saptarshi Mukherjee was with Indian Institute of Technology Bhilai during the duration of this work. E-mail: saptarshim@iitbhilai.ac.in*
- *S. Sidhanta is with Indian Institute of Technology Bhilai. Email: subhajit@iitbhilai.ac.in*

Manuscript received June 5, 2021; revised June 5, 2021.

well-equipped set up in the data centers, cloud environments can withstand fluctuations in the request rate quite conveniently. Frequent decisions of scale-in or scale-out won't be difficult to implement in those data centers. However, edge computing can't manage the fluctuations so comfortably, as the frequent process of powering ON and OFF shall affect the battery life of those not-so-robust devices, entail the extra cost of data/computation caching, etc. These factors are yet to be incorporated in autoscaler designs. Hence, though there is no dearth of auto-scaling tools in the space of cloud computing, there is an appetite for more research in the auto-scaling of containerized applications in the edge, which can address all the additional challenges faced on the edge.

To that end, this paper presents the design of Ant Man Auto-Scaler (AMAS) - an end-to-end framework that can address changes in time-varying workloads by applying automated elastic scaling on the edge cluster. AMAS enables the end-to-end automation of the dynamic discovery of devices and subsequent addition/removal of devices to an edge cluster to adapt to an increase/decrease in the application workload in the form of an augmented/reduced request rate, respectively. Consider real-world edge computing applications such as a traffic monitoring application which runs on edge devices such as android phones or embedded devices (i.e., embedded with traffic cameras or base stations), and processes video feeds received from traffic camera to detect congestion and traffic bottlenecks. During peak hours, the volume of the traffic feed from major traffic junctions increases at an exponential rate such that an edge cluster that was equipped to process the workload in an idle hour fails to retain the performance and SLO deadlines. This is where the autoscaler AMAS shall work towards reaching a favourable cluster composition. However, in the absence of availability of virtually unlimited resources as in the cloud, we have to assume that there is an upper bound to the number of available edge devices, and this bound changes dynamically and, often unpredictably, with time. Also, AMAS can only estimate the required scale-in/out factor for a given edge cluster, while it is unable to guarantee whether the cluster will have the requisite number of devices to perform the above scaling. While these remain out of scope of this work, this paper focuses on the major decision making aspects for addressing the aforementioned challenges of scaling an edge cluster characterized by resource-constrained devices under typically intermittent network connectivity.

The rest of the paper is organized as follows. We've first carried out a brief analysis of the performance of existing state-of-the-art scaling procedures, their shortcomings, and associated scopes of betterment. Following that, we discuss the heuristic steps involved in the making of AMAS, justifying how they're aligned in the direction of our larger objective. Finally, we've shown the results of simulation and benchmark experiments to justify the applicability of our model in Edge Computing environments.

2 Related Work

The papers [5] and [6] have enshrined the crucial VM-level metrics required for such scaling operations. Some other vital precursors include [7], [8], and [9] that describe the evolution of Edge Development practices, and [10], [4], and [11] that have illustrated the visions and challenges involved in the use of Edge devices. Finally, papers like [12] provide the baseline for outlining performance, resource and cost-based awareness as the necessary instruments for making scaling decisions. The KHPA [13] or KHPA-A algorithm [14] is unable to scale-out with dynamic workloads, primarily because the setting up of new containers in our edge cluster shall take some time and other reasons. [15] introduces a novel approach to multilayered auto-scalers performance measurement. Ren et al. [16] give an overview of dynamic auto-scaling algorithms taking into account the need for allocating redundant devices given a sudden surge in computational load and also to mitigate the expenses of powering on and off of devices at the hours of frequent fluctuations. Our paper draws this concept and walks a step further to address the scenario of edge computing by adapting the redundancy to the surroundings. AWS SS1 [17] suffers from under-utilization while AWS SS2, AWS TTS1, and AWS TTS2 [18] are unable to scale with dynamic changes in workload. THRES [19] have an overly large delay associated with them as they only change cluster composition by a single device at a time. Also, they have no adaptive inertial property, so a sudden decrease in the request load would undesirably scale them in. For the DM method [20], only one device is removed while performing scale-in. Also, the delay for switching on devices hasn't been considered here, so it does not scale well in dynamic workloads. Based on the description of these models and comparative analysis in [20], we may infer that DM Method is the strongest competitor for any new model. We demonstrate that in low failure rate situations, AMAS performs better or at least close to all the above models, and in failure-prone situations, outperforms the rest. Over the last few years, some autoscaling researches in edge computing are also worth mentioning. Xu et al proposed in [21] an autoscaling strategy for Energy Harvesting Mobile Edge Computing. Their online learning algorithm uses a decomposition of the (offline) value iteration and (online) reinforcement

learning. In our work, we've consciously avoided the incorporation of learning algorithms to ensure that the not-so-robust edge devices are not burdened with heavy Machine Learning tasks. In [21], the authors have also considered a mandatory presence of the cloud for offloading purposes. If the edge cluster is unable to process the workload, it can offload the same to the cloud and thus keep the system functioning. But in view of our time-critical application, where the reception of data and response to the stimulus is both at the edge, the avoidance of offloading to the cloud becomes absolutely necessary. And this is why we're keeping the autoscaler infrastructure free of heavy computing (involved in machine learning procedures). [21] also doesn't clearly depict in its simulation and experiment results what fraction had to be offloaded and what costs had to be paid for the same. Owing to the difference in outlooks, we're not going to directly compare AMAS with [21]. It has, however, shared a similar objective as ours, trying to incorporate the parameters of power consumption and battery level for the devices. In our application, the edge devices are likely to be mobile and hence we can't depend on constant power supply. So the parameter of power consumption becomes impertinent in our case. Regarding parameters like battery level, our work poses a noteworthy ideological difference with [21]. [21] has directly used parameters like wireless access and transmission delay cost, battery level, etc. We believe that this approach makes the algorithm distinctly dependent on these few parameters, and shall tend to become obsolete upon the arrival of newer metrics. Hence, we choose to measure not these parameters, rather the consolidated impact of these parameters on the average response time, battery life, etc. While it significantly differs from the formula based and state-based approaches in other researches, this paper attempts to establish that this paradigm effectively complements the mentioned counterparts, remains provably auto-adaptive in nature, and is more generic in the long run and across a wider variety of applications. Another autoscaler ENORM has been proposed in [22]. This also involves the compulsory provision of workload offloading to the cloud. The autoscaling approach adopted here is again static, with no step towards adapting to changing situations. Also, the experiments have tested the multi-tenancy, but didn't compare the autoscaler with others. While [22] introduces provisioning, handshaking and deployment mechanism focussed on fog-computing based gaming applications, the autoscaler as such possesses little novelty. Since ENORM is quite similar in design to the other auto-scalers like DM and KHPA that we are already comparing against, we feel that adding experiments to compare our algorithm with ENORM will add little value. Hence, we choose to restrict our experimental comparison to the models KHPA, DM, and THRES only.

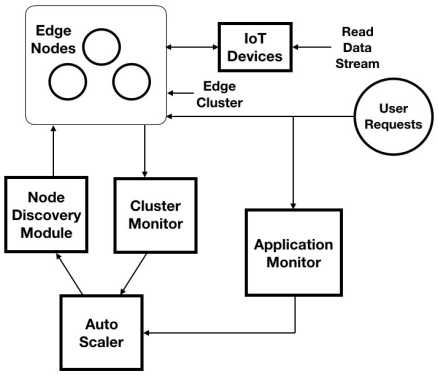

Fig. 1: AMAS System Model

3 Solution Design

3.1 Overall Architecture

Figure 1 illustrates the system model of AMAS where we have considered that an edge cluster processes data streams from IoT devices upon receiving user requests from client applications. AMAS comprises several modules that work together to provide end-to-end elastic auto-scaling of an edge cluster. The module Cluster Monitor keeps track of the availability of the edge devices, network status, utilization, etc., and keeps the module Auto-Scaler apprised of the current workload and latest statistics. Based on these statistics and the reports from the module Application Monitor, the Auto-Scaler suggests a change in cluster composition and requests the Node Discovery Module to find available devices for the said change. In response, the Node Discovery Module probes the network to dynamically discover active edge devices, communicates with them, and activates them.

3.2 Design Approach

One possible design approach for AMAS would be to follow KHPA [13] in constantly adapting the cluster to dynamic changes in workload by reactively probing for the optimal number of devices from the Node Discovery Module according to a fixed time interval. The above number of devices would be assigned based on the frequency of the requests observed and the optimal request load per device been decided. However, especially in situations characterized by a trend of drastic change in request rate, we postulate that it is important to consider the general trend of requests in addition to the request rate observed at that instant. This is because after a device is being added into the cluster, there is

always a delay associated with initializing the devices before they can start processing requests.

An example of a drastically changing workload pattern is the workload to be processed by an online news broadcasting channel in which a video or some news suddenly spreads in the social media world [20]. Such systems are generally identified by a characteristic short active period, after which the service can continue to be available with a relatively lower performance level. Applications such as batch processing systems are typically characterized by an on-off workload pattern in which requests cluster around regular batch execution over short periods of time. On the other hand, a gently varying workload pattern runs on typically predictable environments such as general household surroundings that allow application providers to specify their requirement in detail, and then accurately provision the correct amount of resources to process that workload on the provisioned system.

If the auto-scaler recommends scale-in of a cluster due to a sudden fall in the request rate, and the request rate surges in the next interval, the original size of cluster can not be immediately regained and this indicates a possibility of violation of SLO deadlines. Hence, though a scale-in may be recommended at a particular instant as an immediate response to the fall in request rate, it may not necessarily be the most effective solution in the longer run. The above principle has been effectively applied in the THRES auto-scaler where the algorithm has a distinct inertia against sudden fluctuations, i.e., the cluster may only scale-out (resp. scale-in) if there is a recommendation to increase (resp. decrease) the number of cluster-nodes over a longer duration such as at least two successive time intervals. This inertial property is a crucial aspect that can be incorporated in the design of AMAS by making the auto-scaler wait for a predefined period of time before performing a scale-in or scale-out by the recommended number.

However, static scaling parameters such as a fixed waiting period for scale-in or scale-out, would improve the performance only in specific scenarios and would be unsuitable for real-world workloads that have a time-varying nature. Let us consider that the auto-scaler experiences a workload where the request rate does not remain low throughout, but the occasional spikes can be effectively handled by a reduced cluster size. If the auto-scaler is bound by the static waiting period, it is unable to scale-in unless the change in request rate continues to persist beyond the assigned waiting period. Given the above considerations, we need to ensure that the auto-scaler can adapt to the changing environment and decide whether at any point of time it is prudent to bring a certain change in the cluster composition. Another aspect that we adopt from some of the prevalent auto-scalers, such as AWS SS1, AWS SS2, THRES, etc. is a notion of fractional alteration, based on the fact that the next time interval may show a contrasting trend in request rate that requires the nullification of the last decision that has been taken. This paradigm shall help in minimizing the fluctuation in cluster composition and frequency of reconfiguration of a cluster by avoiding scaling decisions that need to be reverted soon.

We consciously avoid the application of machine learning in favor of computationally lightweight solutions since learning a complex model, such as SVM (Support Vector Machine), CNN (convoluted neural network), and Random Forest, is usually computationally quite expensive as it involves an exhaustive training phase. Moreover, learning a model is feasible only when we have sufficient data which requires extensive experimentation on a real system which may not always be possible for an edge application.

3.3 Algorithm Parameters

We use the parameter W_{rise} which tells the system how long it should ideally wait before performing scale-out. Similarly W_{fall} guides the process of scale-in. The variable S_{out} represents the scaling-out factor that indicates that if the cluster gets the information that 'd' devices should be added to meet the current load with optimum utilization, then it chooses to scale-out by d*S_{out} devices for the time being. Similar goes for S_{in}, the scaling-in factor.

Now, at an instant, if we observe that the cluster is suffering a delay in requests or that all the devices are overwhelmed with a mean utilization above 95%, we infer that it (the cluster) should show greater readiness in performing a scale-out, and lesser readiness in scale-in. If the request rate drops for a moment, the auto-scaler needs to ascertain whether this is going to stay for sometime, else it shall not scale-in. Since this situation is concerning, we increase the W_{fall} and S_{out} by 50% of their corresponding current values, and reduce the W_{rise} and S_{in} by 50%. In contrast, if we observe that the cluster nodes are always running with utilization under 75%, we infer that the cluster should show greater readiness in performing scale-in, and lesser readiness in performing a scale-out; this shall help in improving the cost. If the request rate rises for a moment, the cluster needs to ascertain that this is going to stay for some time, else it shall not scale-out. Since this situation is less concerning, we increase the W_{rise} and S_{in} by 10% of their corresponding current values, and reduce the W_{fall} and S_{out} by 10%. In failure-prone situations, it is reasonable to scale-out the devices slightly in advance, because the delay involved after powering on a device coupled with the

failure rate indicates that the alarm should be triggered some time before the actual deficit is witnessed. This concept of alarm trigger is partially borrowed from [23]. We address this issue by adaptively maintaining the number of redundant devices that the cluster would fallback to in the case of failures. If failure rate would increase or a violation of SLO deadlines as a result of drastically changing trend is observed in the environment, the cluster would increase the number of redundant devices, and if under-utilization of devices is observed, the number of redundant devices shall be decreased.

AMAS introduces the following adaptive parameters. A) Waiting Period denotes the duration the algorithm should wait before scaling the cluster in/out. B) Scaling Factor denotes the fraction (or multiple) of the suggested change in cluster size that should be immediately carried out. C) Redundant devices denote a pool of additional devices maintained by the cluster as fallback options to deal with failures. This number shall indirectly influence the time when the alarm shall be triggered, recommending a scaling operation to be performed. The input and output parameters for our proposed auto-scaling algorithm (refer to Algorithm 1 and 2) are given in Table 1.

3.4 The AMAS Algorithm

Algorithm 1 depicts the inertial property which effectively cuts down the fluctuating behavior of frequent switching on and off of devices which would add up to adversely affecting the longevity of the devices as well as introduce further problems related to load balancing and data sharing among devices. in the case of network problems, it is likely that the request rate may drop significantly for a short duration, and other algorithms may suggest immediate shutting off of devices that can create a severe overhead once the normal flow of requests is restored. Thus, the inertial property enables the cluster to resist changes for some duration which enables it to handle real-world workload variations that may not always be smooth. In line 2 of Algorithm 1, AMAS obtains the number of failed devices from the Cluster Monitor and updates the count of active devices in line 3. In line 4, R_{extra} denotes the adaptive parameter which states the number of extra nodes that should be maintained to avoid the SLA violation due to network problems or node failures. Using R_{extra}, we compute R_{opt} - the optimum number of devices that should be maintained in the cluster. It uses L_{opt} to denote the optimum load that a single device can comfortably handle (as per experimental findings) with a pre-assigned 'sleep' duration in between batches of operations. In line 5, R_{min} denotes the minimum number of devices that shall be required to compute the current load if all the devices are operated at their maximum CPU utilization statistic. In line 6, AMAS fetches, from the Cluster Monitor, the set of devices that have been newly activated and are ready to process workloads. In line 7, we add them to the number of active devices. In line 8, the set of waiting devices, S_{wait} has been updated; these are the devices that have been identified recently by the Node Discovery Module and are being prepared to process workloads. In line 9, we compute N_{total} denoting the total number of devices that are currently present, this shall be used to keep the waiting devices in mind when a scaling-out decision is being taken. N_{add} denotes the number of devices to be added and N_{rem} denotes the number of devices to be removed. These values have been computed in the next lines 12-29. Line 12 is an indication of the scaling-out requirement. In line 13, we adopt the idea of THRES algorithm to record the duration T_{rise} for which the

Algorithm 1 Deciding the number of devices (Homogeneous)

1: **function** COMPUTE_REQUIREMENT(L_{curr})
2: $N_{failed} \leftarrow$ report_failed()
3: $N_{active} \leftarrow N_{active} - N_{failed}$
4: $R_{opt} \leftarrow \lceil \frac{L_{curr}}{L_{opt}} \rceil + R_{extra}$
5: $R_{min} \leftarrow \lceil \frac{L_{curr}}{L_{max}} \rceil + R_{extra}$
6: $S_{ready} \leftarrow$ find_new_active_devices()
7: $N_{active} \leftarrow N_{active}$ + len(S_{ready})
8: $S_{wait} \leftarrow S_{wait} - S_{ready}$
9: $N_{total} \leftarrow N_{active} +$ len(S_{wait})
10: $N_{add} \leftarrow 0$
11: $N_{rem} \leftarrow 0$
12: **if** $R_{opt} > N_{total}$ **then**
13: $T_{rise} \leftarrow T_{rise} + 1$
14: $T_{fall} \leftarrow$ max(T_{fall} - 1, 0)
15: **if** $T_{rise} > W_{rise}$ or $N_{total} < R_{min}$ **then**
16: $T_{rise} \leftarrow T_{rise}$ / 2
17: $N_{add} \leftarrow$ max($\lfloor (R_{opt} - N_{total}) * S_{out} \rfloor$, R_{min} - N_{total})
18: **end if**
19: **else if** $R_{opt} < N_{active}$ **then**:
20: $T_{fall} \leftarrow T_{fall} + 1$
21: $T_{rise} \leftarrow$ max(T_{rise} - 1, 0)
22: **if** $T_{fall} > W_{fall}$ **then**
23: $T_{fall} \leftarrow T_{fall}$ / 2
24: $N_{rem} \leftarrow \lfloor (N_{active} - R_{opt}) * S_{in} \rfloor$
25: **end if**
26: **end if**
27: **if** T_{steps} % 10 == 9 **then**
28: adjust_params()
29: **end if**
30: **return** N_{add}, N_{rem}
31: **end function**

TABLE 1: The Table of Parameters

Output variables		**Additional Variables**	
Variable	**Symbol**	**Variable**	**Symbol**
Total No. of devices	N_{total}	No. of requests delayed in last 10 sec	$REQ_{delayed}$
No. of active devices	N_{active}	Avg utilization of active nodes in last 10 sec	U_{mean}
Scaling-in factor	S_{in}	Total time for which current no. of devices is less than required no	T_{rise}
Scaling-out factor	S_{out}	Total time for which the current no. of devices is more than required no	T_{fall}
Set of devices ready to be added	S_{ready}	Max value of node utilization reached in last 10 sec	U_{max}
Set of waiting devices	S_{wait}	Current step number of the auto-scaling algorithm	T_{steps}
Optimum target no. of devices	R_{opt}	**Constants**	
Min required no. of devices	R_{min}	**Constant**	**Symbol**
Total time the algorithm waits before performing a scale-out	W_{rise}	Lower Limit for Scale-out Factor	j_1
Total time the algorithm waits before performing scale-in	W_{fall}	Upper Limit for Scale-out Factor	j_2
No. of redundant (extra) devices being kept	R_{extra}	Lower Limit for Scale-in Factor	k_1
Input Variables		Upper Limit for Scale-in Factor	k_2
Variable	**Symbol**	Lower Limit for Waiting Limit for performing Scale-out	l_1
No. of requests issued within next time interval	L_{curr}	Upper Limit for Waiting Limit for Scale-out	l_2
No. of node failures reported in last time interval	N_{failed}	Lower Limit for Waiting Limit for Scale-n	m_1
Max device load	L_{max}	Upper Limit for Waiting Limit for Scale-in	m_2
Optimum device load	L_{opt}		

Algorithm 2 Adjusting the parameters as per the changing environment

1: **function** ADJUST_PARAMS
2: **if** $REQ_{delayed} > 0$ or $U_{mean} > 0.95$ **then**
3: $S_{out} \leftarrow \min(S_{out} * 1.5, j_2)$
4: $S_{in} \leftarrow \max(S_{in} * 0.5, k_1)$
5: $W_{rise} \leftarrow \max(W_{rise} * 0.5, l_1)$
6: $W_{fall} \leftarrow \min(W_{fall} * 1.5, m_2)$
7: $R_{extra} \leftarrow R_{extra} + 1$
8: **else if** $U_{max} < 0.75$ **then**
9: $S_{out} \leftarrow \max(S_{out} * 0.9, j_1)$
10: $S_{in} \leftarrow \min(S_{in} * 1.1, k_2)$
11: $W_{rise} \leftarrow \min(W_{rise} * 1.1, l_2)$
12: $W_{fall} \leftarrow \max(W_{fall} * 0.9, m_1)$
13: $R_{extra} \leftarrow \max(R_{extra} - 1, 0)$
14: **end if**
15: **end function**

scaling-out procedure is being requested. Similarly, we update T_{fall} in line 14.

In line 15, we compare T_{rise} with an adaptive parameter W_{rise} to determine if scale-out should be performed immediately. This depicts the inertial property of AMAS: "scale-out only when necessary" and thereby avoid fluctuations. An additional condition in line 15 is that if the total number of devices falls below the minimum number of required devices, it should immediately scale-out. Hence line 15 depicts how urgency is being duly considered in addition to an attempt to maintain inertia. In line 16, we reduce T_{rise}. In line 17, we use an adaptive parameter S_{out} to indicate what fraction or multiple of the suggested increase of cluster size should be immediately carried out. in the case of a gently changing workload (with minor fluctuations), the value of S_{out} shall gradually decrease up to some threshold (less than 1), so the increase of cluster shall be restricted, as the workload statistics do not call for an immediate scaling of the cluster to handle the change in workload. If in due course of time, the workload pattern adopts a drastically changing trend, the values of S_{out} and R_{extra} shall rise and T_{rise} shall decrease as per Algorithm 2, and AMAS shall then show a greater propensity towards scaling-out. As a safety measure, line 17 depicts that (R_{min}-N_{total}) devices shall always be added to the cluster, by which the urgency is taken care of.

In line 19, we perform the initial check to determine a need for performing scale-in. Similar to lines 13 and 14, we update T_{fall} and T_{rise} in lines 20 and 21. In line 22, W_{fall} is the adaptive parameter that determines when the scaling-in operation should be performed. This prevents a wrong scaling-in decision from causing possibilities of SLA violations in the near future. In line 23, T_{fall} is reduced. In line 24, we use S_{in}, our final adaptive parameter, to decide the fraction of the suggested modification to be carried out. Considering the case that S_{out} and S_{in} may cause fractional modification, we don't reinitialize T_{rise} or T_{fall} to 0 in lines 16 and 23, rather reduce to half so that the remaining fraction can be addressed in the next iterations if a similar trend of workload prevails. In line 27, we check if it is time to verify the adaptive parameters and update them. After every 10 time intervals, the condition is satisfied and line 28 is executed. This leads to the execution of Algorithm 2 where the conditions are checked and parameters are updated accordingly. R_{min} indicates the bare minimum number of devices that seem necessary for the next interval, and R_{opt} indicates the total number of devices that are expected to be there. So if the current number is less than the R_{min}, it demands an immediate increase.
Function $ADJUST\_PARAMS$ adjusts the auto-adaptive parameters. If some requests have been delayed or the average CPU utilization turns out to be

above 95%, S_{out}, W_{fall}, and R_{extra} are increased, while S_{in} and W_{rise} are decreased. This indicates that the model shall show a greater tendency to scale-out, and a lesser tendency to scale-in. On the other hand, if maximum CPU utilization over the last duration is below 75%, the reverse operations are carried out, as depicted in the lines 8-14 of function $ADJUST\_PARAMS$. The CPU utilization threshold values of 95% and 75% have been reached through simulations.

At first, we've performed simulations by tuning our hyperparameters, and decided on an optimal alignment for them to be used in the final experiments. The values adopted for our other hyperparameters, which we then used in our main experiments, are as follows: $j_1 = 0.05$, $j_2 = 4$, $k_1 = 0.05$, $k_2 = 4$, $l_1 = 1$, $l_2 = 10$, $m_1 = 1$, $m_2 = 10$.

In all cases of minor rise or fall, AMAS provisions a slow change in the cluster and thereby, with a small overhead in the form of slightly increased operational cost, gives a significant guarantee with respect to the sudden instances of node failure or fluctuation in request rate.

4 Evaluation

In this section, we demonstrate the performance of AMAS by running different kinds of simulation workloads in two different scenarios with a low and a high-failure rate, followed by experiments performed with benchmark workloads. To render the simulation as close to real world scenarios, we use some standard baseline values for different characteristic workload parameters identifying an incoming request in a given interval for a real-world edge application with time-varying workload presented in the following subsection.

4.1 Experimental Setup

We conduct experiments on the GreenGrass Core service provided by Amazon AWS. At each iteration of our experiment, we execute a sample analytics application on the containerized deployment at the edge and record the number of requests processed in every 5 seconds. During the above experiments, the containers deployed on the GreenGrass Core are configured to run at a threshold CPU utilization level, which ranges around 90-95% average CPU Usage as per absolute metrics obtained from */proc*). We observe a resultant CPU load (i.e, throughput) in the range 961-1150 operations within each 5 second interval. We determine the 5th percentile value of the above observed values, which comes out to be $\frac{1020}{5}$=204 operations/second. The choice of 5th percentile value would ensure that in around 95% of situations, the device shall be capable of handling at least 204 requests in 1 second. Hence, the maximum device load (L_{max}) is fixed at 204 operations/second. We assign 80% of the max load as the optimal load, as this value has been widely chosen as a target utilization in the prior research literature [20], [24]. Thus the optimal request load has been fixed at 163.2 $\approx$ 163 operations/second.

4.2 Simulation Results

4.2.1 *Simulation with Real-World Workload*

Here, we evaluate the algorithms based on June 15 data collected from FIFA World Cup 1998 workload dataset [25] to test the performance on real-life workloads. This dataset has been used by many auto-scaling research works like [26] and [27]. In a low-failure rate scenario (probability of failure: 0.01%), AMAS uses

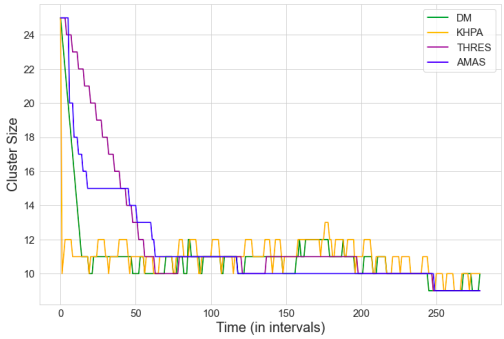

Fig. 2: Number of devices [AMAS vs others] in Real-World Workload with Low failure rate

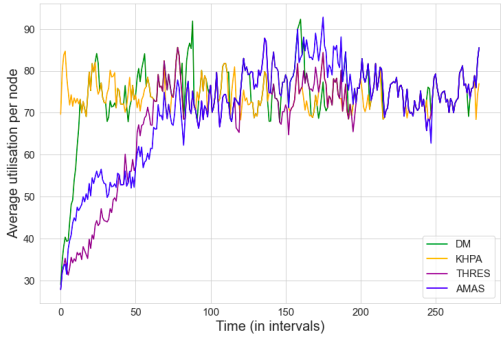

Fig. 3: CPU Utilization [AMAS vs others] in Real-World Workload with Low failure rate

11.42 devices at an average, which is comparable to the other algorithms. Figure 3 shows how AMAS has maintained the optimum CPU Utilization value. Results show that no request delays are caused by either algorithm, indicating that the workload didn't have much fluctuation.

Now we observe the results in a failure-prone scenario wherein nodes can fail with a probability of 2.5%. Figure 4 shows that AMAS has adapted its parameters to maintain a cluster size above the others, given the requirement. Figure 5 shows that AMAS has kept the average CPU utilization within recommended limits for most of the duration. Average CPU utilization in the case of AMAS is 70.73%, while for each of the others, it's above 80%, with KHPA being the highest (84.94%, ranging to about 89% in some experiments). From empirical results in Table 4, we infer that AMAS has incurred much less delayed requests, compared to KHPA, THRES and DM. We have also performed simulation with other kinds of workloads, namely a gentle workload characterized by small changes in the request load, and a sinusoidal workload which depicts sinusoidal variations in the workload. For lack of space, we have included these results in the appendix of extended version of the paper available at [**?**].

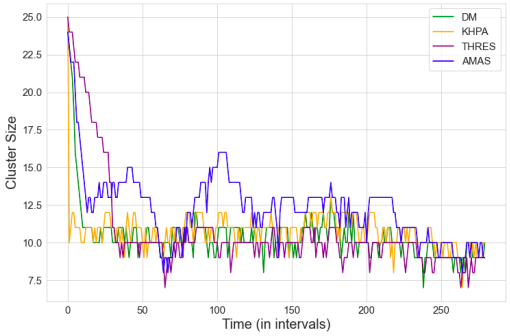

Fig. 4: Number of devices [AMAS vs others] in Real-World Workload with High failure rate

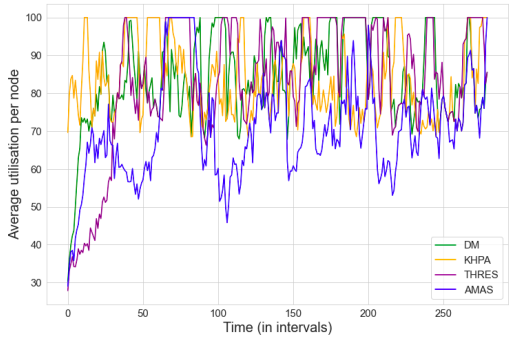

Fig. 5: CPU Utilization [AMAS vs others] in Real-World Workload with High failure rate

4.2.2 Analysis of Simulation Results

Table 4 illustrates that AMAS demonstrates a similar level of performance compared to other models in Low failure rate scenarios, whereas it performs significantly better in Failure-prone scenarios, especially with the sinusoidal workload and the real-world workload discussed above. THRES and DM reportedly use fractional alteration and some prediction-governed reasoning. While KHPA uses no such mechanism, it reactively orchestrates the cluster by deterministically predicting the required changes. Table 4 shows that this fractional alteration and prediction mechanism does not always yield favorable results. Fractional alterations induce considerably large delays for THRES, and approximation-based predictions adopted by DM result in huge delays in a few scenarios and cause over-provisioning of the cluster in other cases, such as with drastically changing workload as illustrated in Figure 9. On the other hand, we observe that AMAS maintains the same properties, namely fractional alteration, environment-aware predictions and reasoning-based heuristics, and yields favorable results with all workloads. This highlights the significance of dynamic parameters used in AMAS which prevents the algorithm from overfitting to a particular scenario. Hence, AMAS demonstrates improved performance in failure-prone scenarios, causes less delayed requests in low failure-rate scenario, and maintains optimal CPU utilization in all the discussed scenarios. This can be traced to the inertial mechanism adopted by AMAS to avoid fluctuations in cluster composition and the fact that it effectively implements fractional alteration and redundancy by dynamically adjusting the scaling parameters.

Table 3 shows that AMAS is the only algorithm that has mostly maintained the average CPU utilization in the optimal range 70-85%. The values signify the basic aspects of all the algorithms. in the case of the DM method, we find the average CPU utilization value at low failure situation, in the case of drastically changing workload, to be too low - below 50%. This statistic coincides with the paradigm adopted in DM. Whenever there is a sudden increase in workload, the algorithm predicts similar increase in the next interval and scales out the cluster by a significant amount. This is a step of voluntary overprovisioning, which results in unnecessary increase in cluster size and thereby very low CPU utilization in the devices. This fact gets highlighted from Table 2 where the number of devices is way higher in the case of DM compared to the other algorithms, in the Drastically Changing Workload in Low-Failure Rate. The algorithm works in a similar manner in High Failure Rate situation, but here the over-provisioning effect is not prominent, as the over-provisioning of devices has been counterbalanced by the high frequency of device failures.

Tables 2 and 3 highlight another interesting property of how the algorithms react to differences in the failure rate of devices. We observe that for all the

TABLE 2: Average cluster size - Number of devices in the cluster (Mean from 5 experiments)

	Low Failure Situation				failure-prone Situation			
	Gentle Workload	**Drastically Changing Workload**	**Sinusoidal workload**	**Real-World workload**	**Gentle Workload**	**Drastically Changing Workload**	**Sinusoidal workload**	**Real-World workload**
DM	47.85	143.22	52.74	10.93	46.69	94.80	50.76	10.49
KHPA	50.01	83.56	51.82	11.07	49.00	81.63	50.55	10.78
THRES	44.47	93.28	52.56	11.98	40.08	38.39	35.36	10.35
AMAS	49.82	97.08	51.50	11.42	52.04	95.34	54.81	12.29

TABLE 3: Average CPU utilization (Mean from 5 experiments)

	Low Failure Situation				failure-prone Situation			
	Gentle Workload	**Drastically Changing Workload**	**Sinusoidal workload**	**Real-World workload**	**Gentle Workload**	**Drastically Changing Workload**	**Sinusoidal workload**	**Real-World workload**
DM	79.53	48.58	76.28	74.21	89.49	75.48	86.20	84.65
KHPA	76.04	85.13	81.11	74.49	85.51	88.88	85.75	82.25
THRES	85.64	68.96	72.98	68.83	100	100	100	84.94
AMAS	76.53	64.47	75.66	70.61	80.81	71.58	76.42	70.73

TABLE 4: Total Number of Delayed Requests (Mean from 5 experiments)

	Low Failure Situation				failure-prone Situation			
	Gentle Workload	**Drastically Changing Workload**	**Sinusoidal workload**	**Real-World workload**	**Gentle Workload**	**Drastically Changing Workload**	**Sinusoidal workload**	**Real-World workload**
DM	0	874169.4	875.4	0	20473.4	1794907.4	176057.2	189611.8
KHPA	0	3535328.2	425.4	0	1174.0	5173051.0	123367.8	77064.2
THRES	0	60641833.4	10313.0	0	193638804.2	2202914507.2	584718161.8	433017.2
AMAS	0	3373387.4	684.0	0	2663.6	2243972.2	32004.6	37919.6

algorithms other than AMAS, the average cluster size is quite larger in the low failure rate situation compared to the failure-prone situation. However, AMAS ensures that a higher cluster size is maintained in the failure-prone situation to support the SLO deadlines in the adverse situation of device failures. As mentioned earlier, this is because AMAS has the property to sense the failure pattern and take appropriate steps by auto-adaptation of parameters. Despite the increase of cluster size, the average CPU utilization always stays in the optimal range, thereby justifying the parameters iteratively reached by AMAS. A similar observation is made from Table 3 where AMAS maintains the CPU utilization in the Failure-prone scenario to be close and comparable to the Low-Failure scenario, but the other algorithms depict that for all the workloads, the Failure-prone scenario witnesses a characteristically higher CPU utilization of available devices. This again supports that AMAS senses and handles the varied situations of failures and workloads better.

Table 4 depicts the average number of delayed responses for the various conditions. The last two columns show that AMAS attains the best statistics in the Failure-prone situation for Sinusoidal Workload and Real-World Workload. In the other cases, the number of delays is comparable to the other algorithms. Here it needs to be kept in mind that given the drastic changes in request rate, it is not always feasible to respond to all requests with no delays. However, the important takeaway at this point is that the restricted overprovisioning, the inertial mechanism of cluster modification, and the auto-adaptation of parameters always allow AMAS to perform as well as other algorithms with static parameters, and in some cases better than them.

Apart from these, a specialty of AMAS lies in its 'careful' scaling paradigm. To mathematically demonstrate this 'carefulness', we hereby employ a fluctuation scoring function that measures the occurrences of wrong scaling decisions. The following section highlights how AMAS avoids wrong scaling decisions and how it becomes useful for Edge scenarios.

4.2.3 Fluctuation Scoring Function

We hereby introduce a *Fluctuation Scoring Function* (FSF) whose value indicates the degree of fluctuation observed in the cluster size due to frequent scaling of an edge cluster. Due to the changing workloads processed by analytics applications run on an edge cluster, there is a possibility that immediately after the cluster is scaled in (resp. out) in response to a decrease (resp. increase) in request rate, the cluster needs to be subsequently scaled out (resp. in) to meet a sudden rise (resp. fall) in the request rate. This shall indicate that the scaling decision at such a time was inappropriate because a subsequent scale-out procedure (after a scale-in step) is always followed by a power-on delay which is likely to risk the SLO deadlines for that duration of switching

TABLE 5: Fluctuation Score (Mean from 5 experiments)

	Low Failure Situation				failure-prone Situation			
	Gentle Workload	**Drastically Changing Workload**	**Sinusoidal workload**	**Real-World workload**	**Gentle Workload**	**Drastically Changing Workload**	**Sinusoidal workload**	**Real-World workload**
DM	0.0	154884.8	5332.6	9.0	0.0	204363.6	11036.2	29.2
KHPA	0.0	178376.0	13991.6	369.0	0.0	98819.4	5996.0	452.4
THRES	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0
AMAS	0.0	173.6	0.0	0.0	0.0	29.8	2.0	0.0

on of the edge devices. Similarly, a scale-out request involves an extra cost of preparing the devices during switching on, which indicates that a power-off followed by a power-on is quite undesirable.

To measure this, we propose the following function as our FSF.

$$FL\_Score = \sum_{i} \sum_{j \in [i-\Delta t, i-1]} FL(i,j) \qquad (1)$$

where Δt indicates the size of the time interval that we wish to consider for our measurement of fluctuations. $FL(i,j)$ is given as:

$$FL(i,j) = \begin{cases} \dfrac{abs(v[i] * (v[j])^2)}{i-j} & if\ v[i] * v[j] < 0 \\ 0 & otherwise \end{cases}$$

$$\forall j \in [i - \Delta t, i-1].$$

Here, the FSF $FL(i,j)$ denotes the measure of fluctuation due to the varying scaling actions performed in the i^{th} and j^{th} time instant. The variable $v[i]$ denotes the scaling action performed in the i^{th} time instant and $v[j]$ denotes the scaling action performed in the j^{th} time instant, where $j \in [i - \Delta t, i-1]$.

In the first place, the above formula depicts that the cluster contributes to a positive increase of the score during a given pair of time intervals (i,j) if and only if $v[i] * v[j] < 0$. This relation holds in the following two cases.

- $v[i] > 0$ and $v[j] < 0$: This condition holds when a scale-out is observed in the cluster during the i^{th} time interval, while a scale-in is observed in the cluster during the j^{th} time interval.
- $v[i] < 0$ and $v[j] > 0$: This condition holds when a scale-in is observed in the cluster during the i^{th} time interval, while a scale-out is observed in the cluster during the j^{th} time interval.

The denominator in Equation 1 quantifies the time difference between successive scale-in and scale-out operations. If the time difference is smaller, it indicates that the urge of taking the opposite scaling decision arose rather prematurely, thereby highlighting that the previous scaling decision taken at the j^{th} interval was detrimental to the stability of the cluster. We refer to a scaling decision that does not result in any of the above cases, i.e., which results in lower values of FSF as *stability preserving*. Thus, we divide by the time difference to ensure that more frequently taken opposite scaling decisions are assigned a greater penalty.

The term $v[j]$ has been squared in the numerator of the function to add a greater penalty to the occurrences of scale-out where multiple instances have been added or removed together in a single time instant and the opposite decision being taken in the next time instants. For instance, let's consider the following two cases.

- $v[i-2]$ = -1, $v[i-1]$ = -1, $v[i]$ = +1
- $v[i-2]$ = -2, $v[i-1]$ = 0, $v[i]$ = +1

If the formula $\dfrac{abs(v[i] * (v[j]))}{i-j}$ would have been followed instead, it would yield a higher penalty to the first case. However, we need to appreciate the occurrence of gradual scaling activities, which, in turn, leads to reduced fluctuation levels in the cluster composition. This is implemented by squaring the term $v[j]$ in Equation 1, which ensures that stability preserving scaling decisions are less penalized in drastic conditions.

Hence, the final relation $\dfrac{abs(v[i] * (v[j])^2)}{i-j}$ (if $v[i] * v[j] < 0$) in Equation 1 provides a justified mathematical formulation to measure the stability preserving the ability of the scaling algorithm; the higher this value, the higher is the effect of fluctuations, and, in turn, less stability preserving is the algorithm.

Hence, keeping in mind the limitations of the edge computing environment discussed in Section 1, we prefer an algorithm that results in lower values of the fluctuation score. Table 5 shows that THRES and AMAS have much lower fluctuation scores compared to the other algorithms, namely DM and KHPA. (Here, the value of Δt has been considered to be 6.) We know that THRES follows a strong inertial policy, that dictates it to wait before every decision of scale-in and only allows addition or deletion of just one instance at a time. This is reflected in the FSF values in the table. However, as a result of this overly strict stability preserving and restricted scaling policy, THRES shows significantly higher potential for causing a violation of SLO deadlines, evident from the high values of the CPU utilization and the Delayed Responses given in Tables 3

and 4. On the other hand, AMAS shows good statistics in all these different yardsticks and thereby proves to be the scaling paradigm that allows for scaling the cluster while simultaneously satisfying the SLO deadlines and preserving the stability of the cluster.

In general, the fluctuation scores are slightly lesser in failure-prone situations than the corresponding low failure situations for many of the cases. This interesting observation attributes to the fact that in failure-prone situations, scale-out is mainly required, which leads to fewer fluctuations, since the failed devices would rarely recover, which makes scale-in hardly a necessity. Despite this, Table 5 signifies that the properties of AMAS and THRES influenced the fluctuations significantly.

4.2.4 Analysis by Overall Scoring

As depicted in Tables 2, 3, 4, and 5, there is a trade-off between preventing fluctuations by gradual scaling activities and reducing delayed requests along with optimizing cluster composition by adopting fast scaling activities. We intend to have a scaling policy that performs well on all these aspects simultaneously. Owing to this, we present below an overall scoring function to rank the algorithms, followed by justification for the choice of the coefficients.

$$
\begin{aligned}
Score &= \frac{350}{1 + f(|C|) + g(D) + h(FS)} \\
f(|C|) &= \log(1 + |C|) \\
g(D) &= 6 * \log(1 + D) \\
h(FS) &= 4 * \log(1 + FS)
\end{aligned}
$$

where $|C|$ = Cluster Size, D = Number of Delayed Requests, FS = Fluctuation Score.

The above weighted score provides a basis of comparing the algorithms based on the collective analysis of all the metrics. As apparent from the equation, the score increases with decrease in the values of cluster size, the number of delayed responses, and the fluctuation score. Higher the overall score, the better is the algorithm. Following are the basic points on which the formula is based.

- Lower cluster size implies lower running and maintenance costs incurred in maintaining less number of devices, which in turn, implies a higher score.
- Less number of delayed requests implies higher chances of SLA fulfillment due to lesser chances of deadline violations, which in turn, explains the higher scores.
- A lower fluctuation score implies better applicability to the resource-constrained edge environment due to greater stability of the cluster, which in turn, justifies a higher score.

Logarithms have been used to ensure that fractional changes do not affect the score significantly. Considering that the parameters have different ranges, logarithmic computation effectively results in normalization of the parameter values, which in turn, results in all parameters having a comparable impact on the formula. The weights for the corresponding parameters in the above weighted sum are assigned as per the notions of priority of the respective parameters. For edge applications, ensuring real-time response (by minimizing delayed requests) and preventing SLO violations due to arbitrary fluctuations in the cluster are more significant factors than the cluster size with respect to the weighted score. Hence we assign relatively higher weights to the factors of delayed requests and fluctuation scores while unit weight is assigned to the factor of cluster size. A weight of 4 for fluctuation scores guaranteed that it could override the influence of the cluster size, which has been assigned a unit weight, in our calculations. The delay of requests is always an important factor to be considered, and viewing the fact that algorithms with slow and safe scaling policy shall have very low fluctuation scores, their delay in requests need to be substantially accounted for in their scores, and through iterations, we thereby assign 6 as the coefficient of the term of delayed requests to make this term hold considerably greater significance in the score than fluctuation score. The unit terms have been added as logarithms in the denominator to restrict any problems due to accidentally applying logarithm over 0 or having 0 in the denominator, such as cluster size is added as $\log(1 + |C|)$ in the weighted score. The numerator has been decided in such a way that the resulting score lies in the range [0,100] for the average cluster size considered in our experiments and given the fact that the other terms shall contribute some number greater than or equal to 0.

Table 6 illustrates the scores obtained with different algorithms including AMAS under different workloads and environmental situations. In the case of Gentle Workload, AMAS performs comparably well to the best score which is recorded by THRES. In the case of low failure, DM and THRES record larger values of scores, due to the fractionally lower cluster sizes provisioned with those algorithms. In the absence of SLA violations for either of the baseline algorithms in our experiments with a gentle workload, the scoring is somewhat entirely dependent on the cluster size. In the case of high failure rates, KHPA attains a better score than AMAS due to fewer failures. Since with gentle workload, the variations in request rate with time are practically linear, hence the ad hoc paradigm followed by KHPA fares better than the inertial method in AMAS. However, in terms of the total score, AMAS follows

TABLE 6: Total Score (Mean from 5 experiments)

	Low Failure Situation				failure-prone Situation			
	Gentle Workload	Drastically Changing Workload	Sinusoidal workload	Real-World workload	Gentle Workload	Drastically Changing Workload	Sinusoidal workload	Real-World workload
DM	71.59	2.58	4.38	27.58	5.43	2.48	3.05	3.89
KHPA	70.94	2.43	4.40	12.89	7.40	2.43	3.18	3.67
THRES	72.66	2.95	5.79	98.23	2.94	2.62	2.78	4.31
AMAS	71.01	3.01	7.93	99.45	6.69	3.27	4.88	5.24

KHPA closely, while DM and THRES suffer huge volumes of delayed requests which bring their scores down radically. In the case of Drastically Changing Workload, AMAS produces the highest scores with both low and high failure rate of devices. All the above-mentioned algorithms suffered a comparable number of delayed requests, but DM and KHPA recorded large fluctuation scores as well, and DM additionally reported cluster over-provisioning in a low failure rate situations, which in turn, brought down the overall scores of both these methods. THRES suffered considerably higher delays than AMAS, and consequently, it reported an overall score that is fractionally less than AMAS.

In the case of Sinusoidal Workload, AMAS produces the largest value of the score, because it records considerably less delayed requests and fluctuation score. Low scores for the rest of the algorithms can be traced to the following root causes: the large number of delayed requests for THRES (which becomes more prominent in failure-prone situations), and the high fluctuation scores and delayed requests in comparison to AMAS. In Real World Workload, AMAS recorded the highest scores in both the low failure and failure-prone situations. From Table 2 we see that AMAS underwent a calculated over-provisioning in failure-prone situations, which helped in obtaining the highest score despite failure of devices. THRES obtains a score close to AMAS, owing to less number of delayed requests with this workload, while constantly maintaining an optimal fluctuation score. Hence, the difference in fluctuation scores becomes quite prominent with this workload, which in turn, results in low overall scores with DM and KHPA highlighting the fact that they produce relatively higher fluctuation scores in comparison to AMAS and THRES.

Based on the above analysis, AMAS proves to be the best choice as an auto-scaling model for edge applications, as it addresses the varied aspects related to cost, SLA, and resource constraints encountered in the edge while reporting a considerably high overall score in all the scenarios discussed and demonstrated above. As a notable observation in Table 6, AMAS proves to be more effective in adverse situations that include unpredictably varying request rates, arbitrary device failures, network instability, etc.

TABLE 7: Experimental Results

	KHPA	DM	AMAS
Average Cluster Size	16.68	18.12	18.40
Average proportion of active devices in cluster	89.23	85.10	92.20
Average CPU Utilization	82.40	80.47	72.96
Delayed Requests	137888500	254334900	50721400
Percentage of Delayed Requests	5.035	9.29	1.85
Fluctuation Score	288	348	3
Total Score	2.518	2.439	3.018

4.3 Benchmarking Experiments

To validate the results obtained from simulations presented in Section 4.2 with benchmarking experiments, we execute YCSB - *Yahoo! Cloud Serving Benchmark* benchmark suite on a cluster of edge devices emulated by virtual machine (VM) instances spawned on the public cloud platform Amazon AWS (Amazon Web Services). We used homogeneous AWS EC2 (Elastic Compute Cloud) VM instances as cluster devices and used AMIs (Amazon Machine Images) configured with the necessary database (MongoDB) and other prerequisites. Through empirical results from pilot experiments, we derived the length of time required to execute a request load comprising around 900000 operations, which is a sufficiently large enough load for our purpose, on a single AWS VM instance. We execute the FIFA World Cup 1998 Workload for 250 successive time intervals, each roughly 15 seconds long, such that the required cluster size ranges between 10 and 20. In our benchmarking experiments, we've compared AMAS against the state-of-the-art algorithms DM and KHPA. We have not compared AMAS with THRES as our simulation results in Section 4.2 had clearly indicated the incompatibility of THRES for the Edge environment, given the possibility of intermittent node failures. For AMAS, DM, and KHPA, we have performed the experiments with a cluster size reaching up to 35, such that the cluster could be comfortably spawned in AWS for bug-free experimentation without the risk of large-scale occurrence of arbitrary failures.

We model the lifetime of edge devices, the nature of their failure, and the subsequent need for recharging or repair, as a Poisson Distribution, as applied in earlier research works like [28]. As a result, every device, independent of other devices, is pre-assigned with its own probability of failing at an instant, that marks a charac-

teristic adherence to real-world scenarios of edge computing. As a disadvantage of using the cloud to emulate the edge computing, this approach involves additional communication overhead due to ssh-based authentication, parsing of commands, etc., while executing AWS-CLI commands needed to spawn new EC2 instances (or removing running instances) on AWS while scaling the cluster out (or in). We have introduced parallelization by leveraging multithreading while implementing the scripts to execute our experiments. This ensures that the tasks of starting and stopping of instances, scaling of a cluster, and running of workload is carried out by parallel threads, which in turn, ensures that the above overhead remains within a threshold. Since none of the metadata associated with the above tasks is concurrently updated by multiple threads, the tasks are safely executed by multiple concurrent threads. As an example, every queue-like structure in our experiments calls for push operation at the rear end and pop/read operation at the front. Ensuring that the push and pop operations are always performed by distinct threads at distinct intervals, the metadata of individual threads are protected. We emulate the power-on delay by using a dedicated thread to keep track of whether the instances are ready to be added, such that an instance is added only when it becomes active and usable.

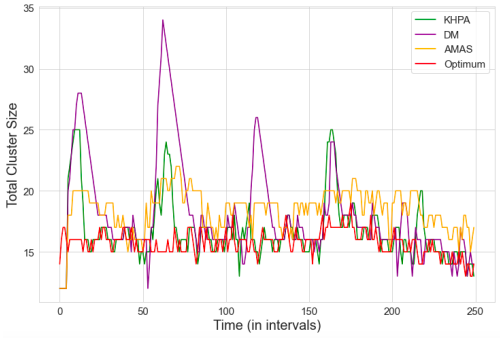

Fig. 6: Total Cluster Size observed in experiments

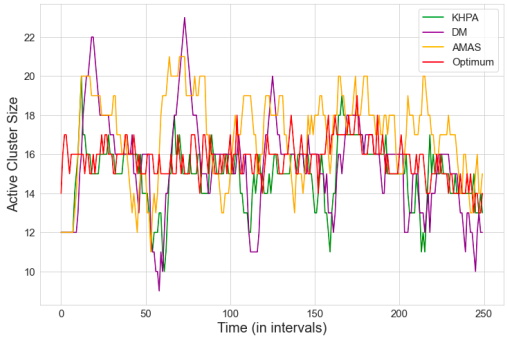

Fig. 7: Active Cluster Size observed in experiments

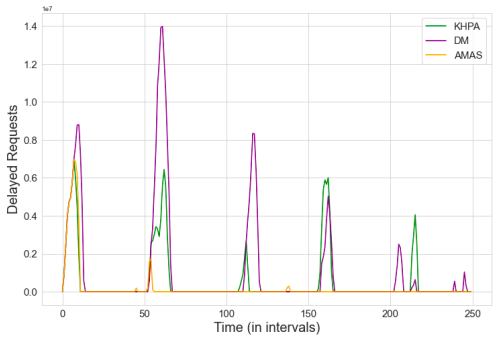

Fig. 8: Delayed Requests observed in experiments

Table 7 illustrates the results of the benchmarking experiments carried out individually for implementation of AMAS and the state-of-the-art algorithms with identical request rate. Figures 6, 7, and 8 depict the behavior of the algorithm in terms of the variations in the values of total cluster size, active cluster size, and the number of delayed requests with time. Here, the active cluster size refers to the instantaneous number of devices in the cluster that are processing workloads. The total cluster size includes both the active devices and the devices waiting to be ready for execution. While the total cluster size bears testimony to the cost of processing involved, the active cluster size defines the actual processing capability of the cluster. The red line in Figures 6 and 7 depict the optimal number of devices that should have been present at all the time instants, given the corresponding request rates and no failures of nodes.

The nature of the benchmarking results bears coherence with the observation made in the simulations, described in subsection 4.2. Figure 8 illustrates that, with a cluster size comparable to that in DM and KHPA, AMAS causes the least number of delayed requests. While DM and KHPA observe 9.2% and 5.03% delays, AMAS suffers only 1.85%. As per Table 7, AMAS reports the least Fluctuation Score by virtue of its strong inertial policy. Also, Figure 6 depicts that at certain moments, DM adopts over-provisioning beyond the requirement, which calls for a scale-in procedure very shortly, thereby increasing the Fluctuation Score. AMAS shows the best overall score (3.018), which is quite higher than that of KHPA (2.518) and DM (2.439). In Figure 7, we notice that after the 60^{th} time interval, the active cluster size for AMAS consistently remains above 13, despite the unavoidable periodic failure of nodes. However, for all the other algorithms, the value of active cluster size has fallen below 13 through the entire duration. This shows that once AMAS adapts itself to the right parameters, it has a stronger tendency of adhering to the SLA due to its inherent ability to safeguard against

arbitrary fluctuations in the cluster size. In turn, this ability to maintain stable cluster size enables AMAS to avoid significant increases in delay in processing requests which may arise due to any decrease in cluster size. This is illustrated by Figure 8, where DM and KHPA showed considerable delays through the entire interval, while AMAS significantly checked the delays after the 60^{th} time interval. We also notice that the points of the surge in delayed requests are directly linked with the sudden fall in thes active cluster size. Figure 6 depicts that AMAS has always tried to maintain the total cluster size slightly above the optimum value. A comparison between the results in Figures 6 and 7 shows that this tendency has enabled AMAS to suffer the least number of failures. The average percentages of active devices in the cluster observed in Table 7 show that in the case of AMAS, the maximum proportion of clusters is observed to be active on average. This property is inversely related to the number of fluctuations in the cluster as newly switched on instances require time to become active. Given the resource constraint nature and limited availability of devices in an edge environment, it is necessary that a considerable proportion of clusters remains active, and therefore AMAS turns out to be best suited for the edge.

5 Conclusions and Future Work

AMAS enables an edge cluster to quickly adapt to changes in the environment at any instant of time and maintain the availability of services in spite of failures. The inertial property possessed by the algorithm allows it to avoid somewhat erratic fluctuations which may be, in fact, aberrations that may disrupt the service availability over the long term. With such an adaptive auto-scaling paradigm, the clients shall be assured of reliable performance of any real-world edge application on a given edge cluster. In the future, we plan to address variable SLO deadlines, handle heterogeneous as well as priority-based forms of scaling.

6 Acknowledgement

This research has received funding under the NetApp Faculty Fellowship scheme from NetApp Inc.

References

[1] S. Oueida, Y. Kotb, M. Aloqaily, Y. Jararweh, and T. Baker, "An edge computing based smart healthcare framework for resource management," *Sensors*, vol. 18, p. 4307, 12 2018.

[2] X. Zhang, Z. Cao, and W. Dong, "Overview of edge computing in the agricultural internet of things: Key technologies, applications, challenges," *IEEE Access*, vol. 8, pp. 141 748–141 761, 2020.

[3] C.-Y. Cheng, H. Liu, L.-T. Hsieh, E. Colbert, and J.-H. Cho, "Attribute-based access control for vehicular edge cloud computing," in *2020 IEEE Cloud Summit*, 2020, pp. 18–24.

[4] S. Bagchi, M.-B. Siddiqui, P. Wood, and H. Zhang, "Dependability in edge computing," *Communications of the ACM*, vol. 63, no. 1, pp. 58–66, 2019.

[5] F. Caglar and A. Gokhale, "ioverbook: intelligent resource-overbooking to support soft real-time applications in the cloud," in *2014 IEEE 7th International Conference on Cloud Computing*. IEEE, 2014, pp. 538–545.

[6] A. Meera and S. Swamynathan, "Agent based resource monitoring system in iaas cloud environment," *Procedia Technology*, vol. 10, pp. 200–207, 2013.

[7] L. Carnevale, A. Celesti, A. Galletta, S. Dustdar, and M. Villari, "From the cloud to edge and iot: a smart orchestration architecture for enabling osmotic computing," in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE, 2018, pp. 419–424.

[8] D. De, A. Mukherjee, and D. G. Roy, "Power and delay efficient multilevel offloading strategies for mobile cloud computing," *Wireless Personal Communications*, pp. 1–28, 2020.

[9] L. Mendiboure, M.-A. Chalouf, and F. Krief, "Edge computing based applications in vehicular environments: Comparative study and main issues," *Journal of Computer Science and Technology*, vol. 34, no. 4, pp. 869–886, 2019.

[10] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[11] S. Taherizadeh and V. Stankovski, "Auto-scaling applications in edge computing: Taxonomy and challenges," in *Proceedings of the International Conference on Big Data and Internet of Thing*, 2017, pp. 158–163.

[12] L. Logeswaran, H. D. Bandara, and H. Bhathiya, "Performance, resource, and cost aware resource provisioning in the cloud," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 913–916.

[13] Kubernetes, "Horizontal Pod Auto-scaling," https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/.

[14] E. Casalicchio and V. Perciballi, "Auto-scaling of containers: The impact of relative and absolute metrics," in *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 2017, pp. 207–214.

[15] A. Jindal, V. Podolskiy, and M. Gerndt, "Multilayered cloud applications autoscaling performance estimation," in *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, 2017, pp. 24–31.

[16] Y. Ren, T. Phung-Duc, J. Chen, and Z. Yu, "Dynamic auto scaling algorithm (DASA) for 5g mobile networks," in *2016 IEEE Global Communications Conference, GLOBECOM 2016, Washington, DC, USA, December 4-8, 2016*, 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1109/GLOCOM.2016.7841759

[17] amazon blogs, "Amazon Step Scaling," https://aws.amazon.com/blogs/aws/auto-scaling-update-new-scaling-policies-for-more-responsive-scaling/.

[18] amazon userguide, "Amazon target tracking scaling," https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-target-tracking.html.

[19] P. Dube, A. Gandhi, A. Karve, A. Kochut, and L. Zhang, "Scaling a cloud infrastructure," Mar. 29 2016, uS Patent 9,300,552.

[20] S. Taherizadeh and V. Stankovski, "Dynamic multi-level auto-scaling rules for containerized applications," *The Computer Journal*, vol. 62, no. 2, pp. 174–197, 2019.

[21] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge comput-

ing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 3, pp. 361–373, 2017.
[22] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "Enorm: A framework for edge node resource management," *IEEE transactions on services computing*, vol. 13, no. 6, pp. 1086–1099, 2017.
[23] salmant, "Autonomous Self-Adaptation Platform," https://github.com/salmant/Autonomous-Self-Adaptation-Platform/tree/master/SWITCH-Alarm-Trigger.
[24] M. KUMAR, "Load balancing algorithm to minimize the makespan time in cloud environment," 06 2018.
[25] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE network*, vol. 14, no. 3, pp. 30–37, 2000.
[26] K. Rattanaopas and P. Tandayya, "Adaptive workload prediction for cloud-based server infrastructures," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 2-4, pp. 129–134, 2017.
[27] D. Tran, N. Tran, B. M. Nguyen, and H. Le, "Pd-gabp—a novel prediction model applying for elastic applications in distributed environment," in *2016 3rd National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS)*. IEEE, 2016, pp. 240–245.
[28] Z. Sun, L. Wei, C. Xu, and Z. Lv, "An event-driven mechanism coverage algorithm based on sensing-cloud-computing in sensor networks," *IEEE Access*, vol. 7, pp. 84 668–84 679, 2019.

Saptarshi Mukherjee pursued his BTech (Honours) in Computer Science and Engineering at IIT Bhilai. He spent two years as a junior researcher under Dr. Subhajit Sidhanta, Asst Professor, IIT Bhilai. He's been a part of the National Blockchain Project, IIT Kanpur, in summer, 2020, as a fellow researcher. His areas of interests include Cloud Development, Edge Computing, Blockchain Technologies, Cryptography, Machine Learning, and Artificial Intelligence. He's assuming the role of Software Engineer at Google India.

Subhajit Sidhanta received his PhD in Computer Science from Louisiana State University, USA, in 2016. He spent one and a half years as a Postdoctoral Researcher with the Distributed Systems Research Group at INESC-ID research lab, affiliated with Instituto Superior Tecnico at University of Lisbon, Portugal, and then joined the Indian Institute of Technology Jodhpur in 2018 as an Assistant Professor in Computer Science and Engineering. His major research interest encompasses the areas of consistency in distributed systems, performance modeling of distributed storage and parallel computing systems.

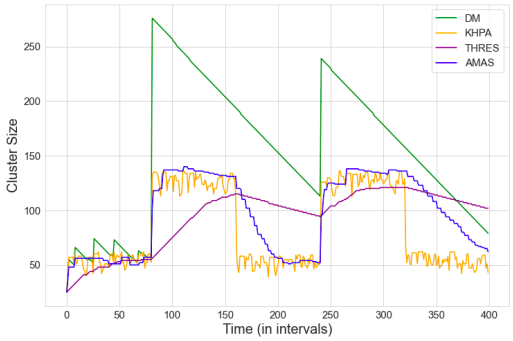

Fig. 9: Number of devices [AMAS vs others] in Drastically Changing Workload with Low failure rate

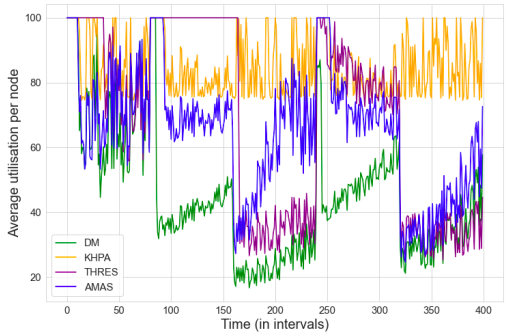

Fig. 10: CPU Utilization [AMAS vs others] in Drastically Changing Workload with Low failure rate

7 Appendix

7.1 More Simulation Results

7.1.1 Simulation With Drastically Changing Workload

Here we observe the results in a low-failure rate scenario wherein nodes can fail with a probability of only 0.01%. In Figure 9, DM's approach of estimating the next frequency of requests based on the past few intervals is observable, which suggests a huge rise beyond requirement. In Figure 9 we also observe that the fall in AMAS' curve is steeper in the period 150-200 intervals while it becomes gentle in 300-400 intervals. This shows how AMAS has adapted to the scenario; while the trend of DM appears to be largely the same in the entire duration. From Figure 10, we may infer that DM has used a much greater number of devices on average. The same has also been indicated in [20]. Finally, Table 4 shows that AMAS has performed similar to DM [20] and KHPA [13], while significantly better than THRES in protecting SLO. For AMAS, DM and KHPA, it is of the order of 10^6 in the duration, while ranging to 10^7 in the case of THRES. Overall, DM brings in much more additional cost of devices, THRES suffers too many request delays and KHPA experiences an average CPU utilization of 85.1% that is way higher than

the rest. These highlights that AMAS has done notably well, without posing any significant concern in either aspect. Now we observe the results in a failure-prone scenario wherein nodes can fail with a probability of 2.5%. Figure 11 shows that only DM [20] and AMAS

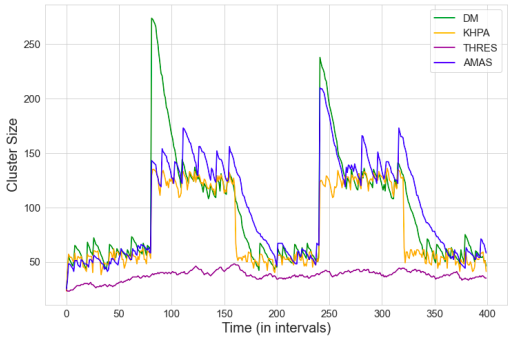

Fig. 11: Number of devices [AMAS vs others] in Drastically Changing Workload with High failure rate

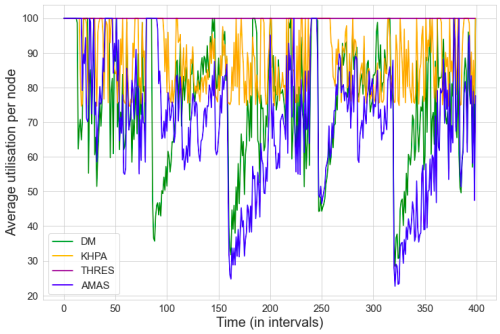

Fig. 12: CPU Utilization [AMAS vs others] in Drastically Changing Workload with High failure rate

scales well with a high rate of failures, but DM over-provisions the cluster at certain moments like the 80th interval whenever there is a sudden rise in the request load. From the summary of average cluster size and CPU utilization, we observe that while DM and AMAS maintain similar cluster size on average, AMAS experiences a much lesser CPU utilization, thereby depicting a far better distribution of computation. This happens because AMAS restricts the fluctuation in scaling which would have led to a greater percentage of inactive devices (due to power-on delay). Empirical results in Table 4 show that AMAS observes the least number of delayed requests in this scenario. Figure 12 further highlights that despite drastically changing workload, AMAS maintains the CPU utilization of the devices in the range of 0-80% for most of the duration. The observed intervals of over-utilization of CPU (above 90%) are spurious, and emulate surges caused due to resource constraints in edge devices.

7.1.2 Simulation With Gentle Workload

Here we observe the results in low failure rate scenario wherein nodes can fail with a probability of only 0.01%. From Figure 13, we observe that despite the small

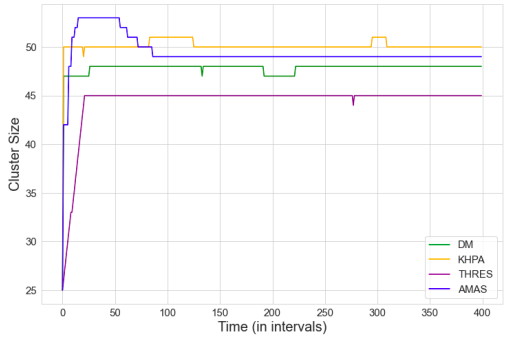

Fig. 13: Number of devices [AMAS vs others] in Gentle Workload with Low failure rate

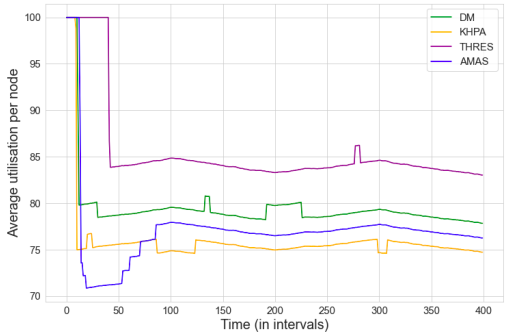

Fig. 14: CPU Utilization [AMAS vs others] in Gentle Workload with Low failure rate

changes in the requests' load, AMAS has kept the cluster size fixed at 49 devices, with the CPU utilization around 75-80% in the entire duration (as observed in Figure 14). This enshrines the targeted inertial property. AMAS allocates 49.82 devices at an average which is fractionally greater than THRES and DM and less than KHPA. Also, the slightly greater operational cost for AMAS is justified by the fact that no scaling cost needs to be incurred (unlike the other algorithms) as no changes in cluster size take place. Since it's a very gentle workload with no failures in the cluster, no SLO violations are observed in either algorithm - none has incurred any delayed requests in the entire duration. Now we observe the results in a failure-prone scenario wherein nodes can fail with a probability of 2.5%. In Figure 15, we observe that after the cluster size drops at around 100 intervals (due to node failures), AMAS adjusts its parameters to maintain a larger cluster size than required. In Figure 16, we observe that DM and THRES have led to CPU over-utilization throughout

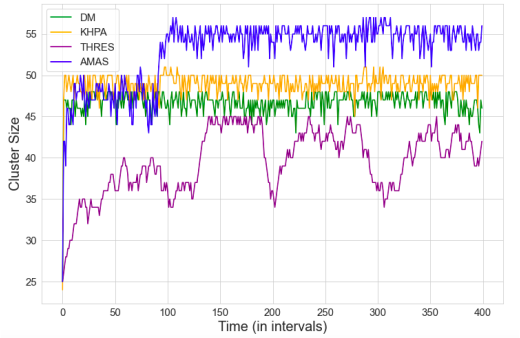

Fig. 15: Number of devices [AMAS vs others] in Gentle Workload with High failure rate

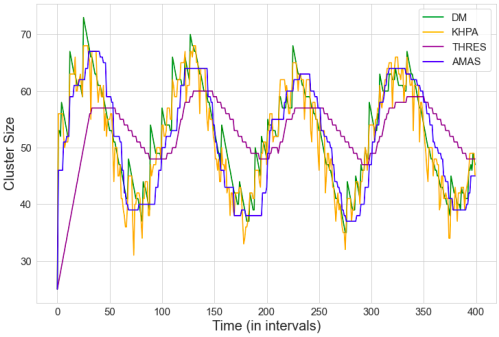

Fig. 17: Number of devices [AMAS vs others] in Sinusoidal Workload with Low failure rate

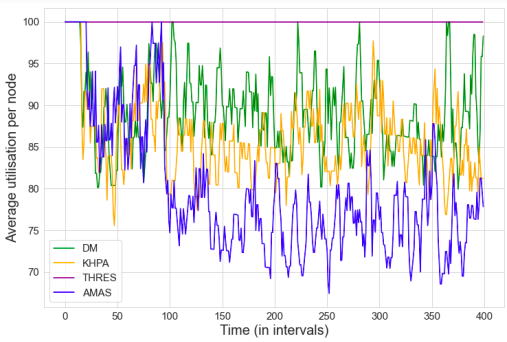

Fig. 16: CPU Utilization [AMAS vs others] in Gentle Workload with High failure rate

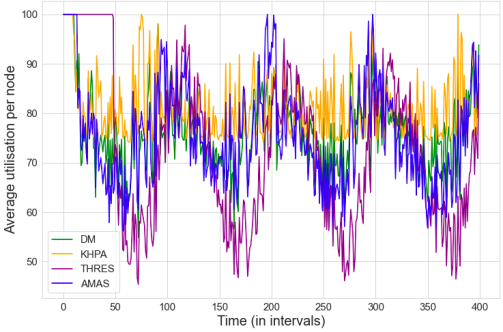

Fig. 18: CPU Utilization [AMAS vs others] in Sinusoidal Workload with Low failure rate

the duration, while AMAS has gradually reached the optimum range of 70-80% in the last 100 intervals, and even after node failures, the CPU utilization stayed below 85% for most of the time. The average CPU utilization is the least for AMAS - 79.84%, depicting that the working devices have not been overloaded, which is quite crucial in failure-prone situations. For all the others, the average CPU utilization value ranges from 80 to 95%. Table 4 confirms that AMAS is a better candidate compared to DM and THRES, as AMAS experiences much lesser delays in requests. The average number of delayed requests in AMAS (2663.6) is far less than in DM (20473.4) or THRES (193638804.2), while comparable to KHPA (1174.0). KHPA being the most static approach, we may infer that it coincidentally possesses the right constants for the scenario. On the other hand, AMAS has reached those values adaptively over the first 300 intervals, thereby promising a favorable outcome in other workloads as well.

7.1.3 Simulation With Sinusoidal Workload

Here we observe the results in low failure rate scenario wherein nodes can fail with a probability of only 0.01%. In Figure 17, we observe that AMAS and THRES differ from the other algorithms in the sense that these two algorithms follow a more gentle trend of scale-in or scale-out of devices. The other algorithms show frequent spikes that depict such moments where a scale-in decision is instantly followed by a scale-out decision, thereby indicating a wrong scale-in decision being taken, given the costs involved in scaling of devices and data sharing procedures. The inertial property in AMAS helps maintain a fairly uniform and gentle trend in scaling the edge cluster. Figure 18 shows that the CPU utilization stays in the desired range of 70-80% for most of the time in the case of AMAS, with over-utilization at very few time intervals. Results show that AMAS has used fractionally lesser devices (51.50) on an average compared to all the other algorithms. This enshrines that despite the inertial property delaying the scaling process, AMAS helps to reduce the overall cost incurred in the operation of the devices. Table 4 shows that AMAS, KHPA, and DM have shown fewer delayed requests. In AMAS, only 684.0 requests (on average) have got delayed in the duration. Now we observe the results in failure-prone scenarios wherein nodes can fail with a probability of 2.5%. From Figure 19, it is evident that AMAS maintains a higher cluster size in the last 200-300 intervals. This signifies that AMAS captures the failure rate in the surroundings effectively

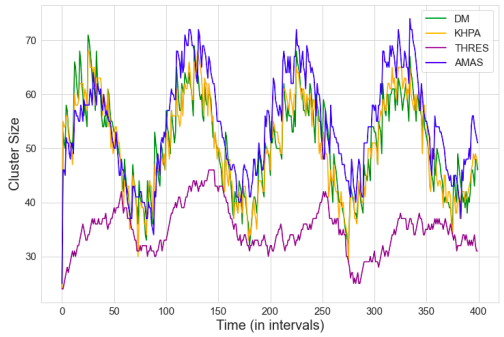

Fig. 19: Number of devices [AMAS vs others] in Sinusoidal Workload with High failure rate

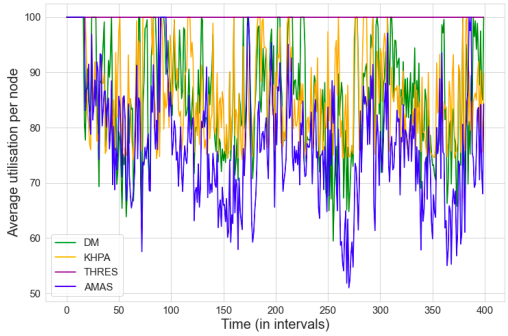

Fig. 20: CPU Utilization [AMAS vs others] in Sinusoidal Workload with High failure rate

and thereby maintains a greater cluster size than all the other algorithms. Figure 20 shows that as a result of the failure rate, all the other algorithms experience over-utilization (above 85%) for most of the time, while AMAS maintains the CPU utilization in the range of 70-80% for most of the time. Here, AMAS has maintained a greater cluster size than other algorithms, with the average CPU utilization being about 76.42%, which depicts that AMAS has adapted to the scenario well. Table 4 shows how the prime objective of protecting SLO in a failure-prone scenario has been met. AMAS leads to only 32005 delayed responses (fairly low considering a total of around 10^{10} requests), while KHPA, THRES, and DM methods have caused much more delayed requests, showing that AMAS is a better candidate at such difficult scenarios.