Distributed Sketch Based Anomaly Detection in Edge Streams in GPU environment with Collision Aware Optimization

October 27, 2021

Abstract

Detecting anomalous edges in a dynamic network in an online manner is an important requirement in industrial settings for intrusion detection or denial of service attacks. While detecting anomalous edges in real-time is crucial, often in an offline setting, where data may be distributed across different clusters/nodes, it is imperative to solve the same problem within optimized time bounds. We thus propose MDistrib and its variants which provides (a) faster execution in a distributed fashion with GPU support for both online/offline setting than other approaches, (b) better false positive guarantees and (c) with collision aware based anomaly scoring for better accuracy results than state-of-the-art approaches. We describe experiments confirming that DistribMIDAS is more efficient than prior work.

1 Introduction

Studying the behavior of agents and detecting anomalies in streaming graphs have gained much importance in recent years due to rising denial of distributed attacks, financial frauds and intrusion in network services.

As graph structures have become more dynamic in nature, particularly in streaming environments where new edges arrive at each time stamp in an online setting, it is crucial to have a constant memory solution for quick detection of intrusion and faster recovery. At the same time, in various applications particularly in network domains, logs of incoming requests are usually maintained in different distributed nodes/clusters. For such situations a fast offline algorithm needs to be proposed that can process large data streams in a few micro-seconds and in constant memory, thus triaging/detecting anomalous network requests, so that recovery options can be deployed immediately.

In this paper, we propose a distributed version of anomalous edge detection for large streaming graphs using sketch data structures called MDISTRIB for both offline/online setting, as an extension to previous work on MIDAS [2], thereby reducing overall edge processing time and faster execution. In addition, we extend our method for any generalized frequency based sketch data structure with better theoretical guarantees for frequency estimates and better theoretical bounds on false positive probability as compared to MIDAS.

Our main contributions in the paper are as follows:

1. **Distributed Framework:** We propose a distributed framework for MIDAS with equivalent accuracy and better time complexity.
2. **Constant Memory Space:** The corresponding framework provides a constant space solution to anomaly detection.
3. **Theoretical Guarantee:** We propose better frequency estimation bound using MDISTRIB-FIS and false positive probability in subsequent theorems.
4. **Collision Aware in CMS storage** : We propose a weighted learning optimization for hashing to reduce the number of collisions as well as propose a novel method of storing collisions to get better accuracy.

Table 1: Comparison of relevant methods

	SedanSpot '18	ranshous2016scalable	Spotlight	PENminer	F-Fade	MIDAS-R	**MDISTRIB**
Microcluster Detection	-	-	-	-	-	✓	✔
Constant Memory	✓	✓	✓	-	✓	✓	✔
Constant Update Time	✓	✓	✓	✓	✓	✓	✔
Distributed Processing	-	-	-	-	-	-	✔
Collision Aware Optimization	-	-	-	-	-	-	✔

2 Related Work

The recent interest in anomaly detection from data streams has contributed quite a few studies in the literature [8].

However, the studies on anomaly detection from streaming graphs is relatively newer. A recent study by Bhatia et al. has presented a microcluster-based anomaly detection from edge streams [2]. However, this study presents only an online real-time streaming scenario but does not consider the scenario of network logs analysis in an offline setting.

Another study by Liu et al. on streaming Anomaly Detection with Frequency and Patterns (termed as Isconna) ... [7]

Very recently, Chang et al. have proposed a frequency factorization approach (termed as F-fade) for anomaly detection in edge streams [3].

The novelty that we introduce in the current paper is providing a fast and scalable parallel framework for distributed sketch based anomaly detection in edge streams in GPU environment with collision aware optimization.

3 Problem Setting

Given a dynamic time-evolving graph G where $\varepsilon = (e_1, e_2, \dots)$, e_i denotes (u_i, v_i, t_i), how can we detect anomalous edges and micro-cluster activities in a distributed fashion. Here, u_i and v_i denotes source and destination node respectively and e_i denotes the incoming edge between them in timestamp t_i. In a network setting, u_i and v_i may indicate source and destination IP addresses respectively and edge denoting a connection request occurring at timestamp t_i. Also the graph G can be modelled as a directed multi-graph indicating that $(u_i, v_i) \neq (v_i, u_i)$ and edges might appear more than once between any two specific nodes.

4 Motivation

Motivation is supposed to be part of the introduction. You don't need a separate motivation section.

In an offline setting, where there exists log history of network communication information among routers in a system, can we detect efficiently which among the network connections are currently anomalous or previously unseen ?

Most often these logs are spread across various timestamps and sometimes to check whether the incoming network connections in the current timestamp is anomalous or not, it is required to compare the same with the previous history of incoming network connection requests.

Hence its challenging to distribute these logs across timestamps and process them in parallel due to their interdependence. At the same time, most offline log processing nowadays dont consider constant memory

5 Proposed Algorithm

We propose here a distributed version of micro-cluster detection in edge streams with any general frequency based mergeable sketch datastructure. In subsequent sections, we propose two models MDISTRIB-CMS and MDISTRIB-FIS which uses separate sketch datastructures.

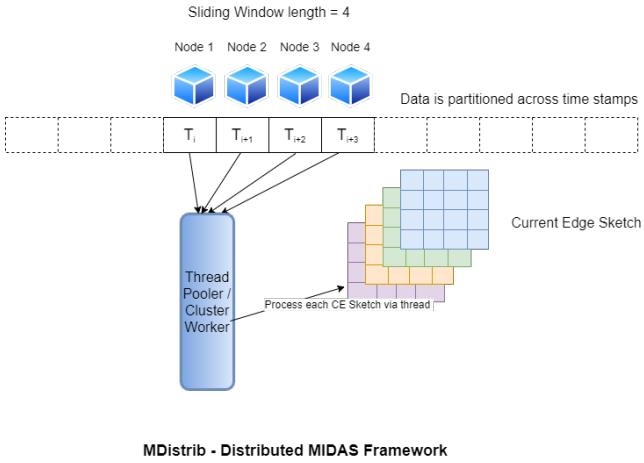


Figure 1: MDISTRIBArchitecture

5.1 Temporal Scoring Recent method [2] proposes a sequential online streaming based anomaly detection framework. It combines a chi-squared goodness-of-fit test with the Count-Min-Sketch (CMS) streaming data structures to get an anomaly score for each edge. In MIDAS, s_{uv} is defined to be the total number of edges from node u and v up to current time tick t, while a_{uv} is the number of edges between node u and v for the current time tick t (but not including the past time ticks). MDISTRIBthen divides the edges into two classes : edges at the current time tick ($= a_{uv}$), and edges in past time tick ($= s_{uv} - a_{uv}$), and computes the chi-squared statistic, i.e. the sum over categories of $\frac{(observed-expected)^2}{expected}$.

$$
\begin{aligned}
X^2 &= \frac{(observed_{curr_t} - expected_{curr_t})^2}{expected_{curr_t}} \\
&\quad + \frac{(observed_{past_t} - expected_{past_t})^2}{expected_{past_t}} \\
&= (\, a_{uv} - \frac{s_{uv}}{t})^2 \frac{t^2}{s_{uv}(t-1)}
\end{aligned} \tag{5.1}
$$

MIDAS uses two types of CMS data structures to maintain approximate counts $\hat{s_{uv}}$ and $\hat{a_{uv}}$ as an estimate for s_{uv} and a_{uv} sepectively.

Given an arriving edge indicated by tuple (u, v, t), our anomaly score is computed as:

$$score((u, v, t)) = (\hat{a_{uv}} - \frac{\hat{s_{uv}}}{t})^2 \frac{t^2}{\hat{s_{uv}}(t-1)}$$

5.2 Proposed Distributed Framework Architecture

5.2.1 Distributed MIDAS - MDISTRIB While most online streaming algorithms propose a real-time linear scan for computing anomalous edges as they arrive, a more optimized

approach is required for offline batch processing for historical log data even if the whole data exists in memory or is distributed across various clusters/ nodes. For simplicity, in offline phase we may consider a scenario where there exists network logs consisting of incoming requests during the past month and the data is partitioned across nodes/clusters by its corresponding timestamp (daywise).

Instead of a linear scan, our proposed framework models the data stream $S = (d_1, d_2, d_3, \dots d_t, \dots)$ s.t. $d_k = (i_k, t_k)$, where i_k indicates the corresponding item and t_k denotes the K^{th} timestamp. Considering there are n timestamps, for each single timestamps there might be multiple items which occur. Each of the timestamps can be considered for parallel processing rather than serially processing. Considering there are $m + 1$ hardware threads in a multiprocessor, the stream can be partitioned into m sub-streams $S_1, S_2, S_3, \dots S_m$.

For each substream, we consider only one frequency based sketch data structure denoted as CE, for maintaining the current frequency per item. And at the end of m sub-streams we maintain another sketch denoted as CS_t which stores the sum total frequency corresponding to each edge uptil all such previous substreams. Each such CE sketch data structure indexed by timestamp is stored under a particular thread Id.

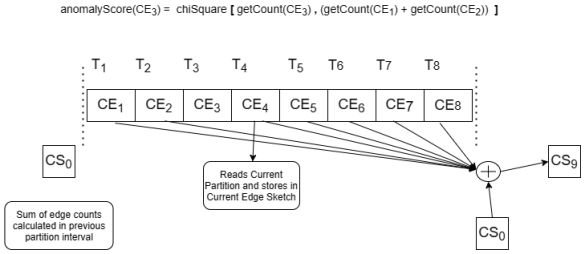


Figure 2: Calculation of CE and CS sketches

Since CS stores the total sum frequency for a particular source destination pair (u_i, v_i), for a particular time instant T_i, it can be written as follows $CS(T_i) = \sum_{j=0}^{j=i-1} CE(T_j)$ i.e summation over all previous *current Edge* frequency sketches.

5.2.2 Algorithm Here we propose an algorithm walkthrough for offline phase for our proposed MDISTRIB. Note, sketch datastructures mentioned here are general frequency based sketch datastructures that share mergeability properties.

5.3 MDISTRIB-CMS → Distributed MIDAS using Count Min Sketch Here, we follow the exact architecture of MDISTRIBas proposed above and use count min sketch for each of the m *Current Edge Sketch* and *Current Sum Sketch*, considering m threads or partitions, with each having w and b number of hash functions and number of buckets respectively.

Algorithm 1 Distributed Microcluster Detection - Offline

Input: Stream of graph edges distributed across K partitions
Output: Anomaly score of individual edges
Initialize m Current Edge(CE) Sketches;
Initialize m Current Sum(CS) Sketches
while *All K partitions are visited* **do**
  Assign worker per m partition
  for *m partitions in parallel* **do**
    Execute $CalculateCE(partition_i^m)$ in parallel
  for *m partitions in parallel* **do**
    Execute $CalculateScore(partition_i^m)$ in parallel
Output final anomaly scores

Algorithm 2 CalculateCE - Current Edge Calculation

Input: Stream of graph edges for Partition id θ
while *a new edge (u, v, t_θ) arrives* **do**
  Assign edge (u, v) to a thread
  Update CE_i^m datastructures for the new edge uv
  EndWhile

5.4 MDISTRIB-FIS - Distributed MIDAS using Apache Frequent Item sketch [1] introduces Frequent Item Sketch which belongs to a class of Apache Incubator Datasketches that deals with estimating frequency of items in a streaming environment and is mainly aimed at applications like Heavy-Hitter estimation or estimation of counts of objects in a streaming environment. The Apache Incubator Datasketches framework provide a wrapper api for Frequent Item sketch with the mergeability functionality where two Frequent Item Datasketches f_1 and f_2 can be merged into one frequent item datasketch f_r with all the frequencies remaining intact $f_r = f_1 \oplus f_2$.

5.5 MDISTRIBR : Distributed MIDAS-R Like similar to MDISTRIB, we take into account [2]'s idea of relation attribute, both temporal and spatial relations.

5.5.1 Temporal Relations With the idea of most recent edges being a better representative of the statistic score than previous early edges, for each timestamp id Tid within the partition interval we apply a decay factor $\alpha^{Tid} \in (0, 1)$ to the corresponding

Similar to MDISTRIB, we propose the same distributed

Algorithm 3 CalculateScore - Current Edge Score Calculation

Input: Stream of graph edges for Partition id θ
Output: Anomaly Scores per edge
while *a new edge* (u, v, t_θ) *arrives* **do**
  Assign edge (u, v) to a thread
  Perform merge operation for $CS_i^m = \sum CE_i^m$
  Retrieve updated counts $a_{uv}^{\hat{}}$ from CE_i^m
  Retrieve updated counts $s_{uv}^{\hat{}}$ from CS_i^M
  Compute Edge Scores
  $score((u, v, t)) = (a_{uv}^{\hat{}} - \frac{s_{uv}^{\hat{}}}{t})^2 \frac{t^2}{s_{uv}^{\hat{}}(t-1)}$
  Output Scores
EndWhile

architecture for storing node degree counts as well.

5.6 Time complexity Analysis We focus or time complexity analysis and comparison with respect to MIDAS since MIDAS has performed significantly well in comparison with SOTA models in terms of time complexity.

Considering we have K partitions, $P_{j+1}, P_{j+2}, \dots P_{j+k}$, where each partition P_{j+i}; $i \in [1, K]$ belonging to timestamp T_{j+i} includes a CE datasketch. We denote the CE sketch update time as T_{CE}. If the average number of items during T_{j+i} is μ, then $T_{CE} \approx \mu$ and the total update time for all K partitions under the MIDAS framework would be $\mu * T_{CE}$. Similarly for calculating the sum of the previous edges seen uptil time T_{j+i} as per **??** it can be denoted as $T_{CS} = \tau_c * (K - 1)$ where τ indicates the merge time for two datasketches, Since in both MDISTRIBand MIDASthere is merging of previous current edge sketches, hence T_{CS} remains the same for both.

Comparing the total computation for both the methods, yields

$$T_{\text{MIDAS}} = \phi[\mu * T_{CE} + \tau_c * (K - 1)]$$

$$T_{\text{MDISTRIB}} = \phi[\mu + \tau_c * (K - 1) + Th_{AllocationTime}],$$

s.t. $\phi * K \approx C$ where C denotes the number of total partitions/nodes.

Considering thread allocation time $Th_{AllocationTime}$ to be very negligible compared to other factors like large number of streaming data points per partition μ and large number of partitions K, it can be seen that T_{MIDAS} is much less than T_{MDISTRIB}.

Further we also propose reading items per partition to be distributed across various subthreads instead of sequential reading, since each edge under a particular partition/ timestamp can be processed independently of the other. As a result, update time for CE denoted by μ can be further reduced to $\hat{\mu}$ where $\hat{\mu} < \mu$ thus resulting in much faster execution time.

5.7 Better theoretical guarantees

5.7.1 Count Min Sketch Error Estimate The error estimate encountered via querying edge counts in MDISTRIB is less than that of MIDAS by a factor of $(\sum_x Count - \max(Count))$

Proof 1. MDISTRIB*CMS has the same sketch architecture as* MIDAS*, but however presents with a better theoretical guarantee as shown below:*

As per Thereom 1. [4], the estimate $\hat{a}$ *from a CMS has the following guarantees:* $a \leq \hat{a}$ *with some probability* $1 - \frac{\delta}{2}$,

$$\hat{a} \leq a + \epsilon * V \tag{5.2}$$

where the values of these error bounds can be chosen by $w = \lceil \frac{e}{\epsilon} \rceil$ *and* $d = \lceil \ln(\frac{1}{\delta}) \rceil$, w *and* d *being the dimensions of the CMS.* V *indicates the total sum of all the counts stored in the data structure. Again considering* K *partitions* $P_{j+1}, \dots P_{j+k}$ *, let* $P_{j+\theta}$ *where* $\theta \in [1, K]$ *denote the partition* θ*. As per* MIDAS *the sum sketch for time tick* t *is computed by summation of all previous current edge sketches* CE *uptil* $t - 1$*. Hence for partition* θ*,* V_θ^{MIDAS} *can be written as follows.*

$$V_\theta^{MIDAS} = \sum_{r=1}^{r=\theta-1} V_r^{MIDAS}$$

$$V_\theta^{MIDAS} = \sum_{r=1}^{r=\theta} \sum_{p=1,q=1}^{w,d} \textit{getCount}_r(p, q)$$

For MDISTRIB-CMS*, however since for the partition* θ *we have* $\theta - 1$ *CE sketches, instead of merging them for computing the same, we can individually query each of the* $\theta - 1$ *sketches instead of merging them. Hence in this case,*

$$V_\theta^{\text{MDISTRIB}} = max(V_r^{\text{MDISTRIB}}) \textit{ where } r \in [1, \theta - 1]$$

Hence, as per Eqn. (5.2) we compare the estimates of the count retrieved from the current Sum CS *CMS sketch of partition* θ *for* MDISTRIB *and* MIDAS.

$$\textit{MIDAS Estimate} : \hat{a_\theta} \leq a + \epsilon * (\sum_{r=1}^{r=\theta-1} \textit{count}_r) \tag{5.3}$$

(5.4)
$$\textit{MDistrib Estimate} : \hat{a_\theta} \leq a_\theta + \epsilon * \max(\textit{count}_1, \dots \textit{count}_{\theta-1})$$

Since such counts stored in the CMS being non-negative

$$\max(count_1, \dots count_{\theta-1}) <= \sum_{r=1}^{r=\theta-1} count_r \tag{5.5}$$

Hence, taking the inequality from 5.5 we can conclude that that the error bound in the estimate provided by MDISTRIB *is much less than that provided by* MIDAS.

$$\begin{aligned} \hat{a_\theta} &\leq a_\theta + \epsilon * \max(count_1, \dots count_{\theta-1}) \\ &\leq a_\theta + \epsilon * \sum_{r=1}^{r=\theta-1} count_r \end{aligned} \tag{5.6}$$

□

5.7.2 Apache Frequent Item Sketch Guarantees In case of a Frequent Item Sketch with configuration $< M, N >$, where M indicates the maximum map size in power of 2 and N being the total weights stored in the map, the error in the *estimate* of the frequency of a particular edge is always bounded within some T, $\hat{a} \leq a + T$, where T can be as follows, given α is the load factor :

$$T = \begin{cases} 0, & \text{if } N < \alpha * M \\ \epsilon * N, & \text{otherwise} \end{cases} \tag{5.7}$$

Since Frequent Item Sketch has a similar structure as a Count Min Sketch in terms of the error estimate, the proof for the false positive probability bound also holds here.

lemmahe error estimate encountered via merging two sketches A , B is the same as that obtained in an original sketch C which is contains frequencies all items that has been inserted to A and B as a whole.

5.7.3 Establishing for Count Min Sketch Considering count min sketches A and B having dimensions wxd each and inserting an item with actual frequency a in both the sketches, then the error estimate of a particular item can be represented as follows

$$\hat{a_A} \leq a + \epsilon * V_A \tag{5.8}$$

$$\hat{a_B} \leq a + \epsilon * V_B \tag{5.9}$$

where V_A and V_B indicate the total sum of all the counts stored in CMS A and B respectively.

Combining equations 5.8 and 5.9 and since all the elements involved are (≥ 0), we get

$$\hat{a_A} + \hat{a_B} \leq 2a + \epsilon * (V_A + V_B) \tag{5.10a}$$

$$a_{\hat{A+B}} \leq 2a + \epsilon * (V_A + V_B) \tag{5.10b}$$

$$\hat{a_C} \leq 2a + \epsilon * (V_C) \tag{5.10c}$$

where $\hat{a_A} + \hat{a_B} = a_{\hat{A+B}}$ indicates the total estimated frequency of that particular item, had that item be inserted into the sketch C having same dimension wxd which is a combination of A and B. Hence by this we showcase that the error estimate remains the same even after merging two Count Min Sketches compared to an original Count Min sketch which represents a summation of the two other sketches.

5.8 Why Distributed MIDAS using Apache Frequent Item sketch is better than MIDAS using CMS Freq. Item Sketch only stores those items at any point that's larger in freq than the global median frequency, whereas CMS keeps on storing. Estimation Errors. [5] introduces the concept of transient keys which are items that are not active and have low frequency relative to the group.

5.9 Space Complexity

5.9.1 Constant Space for MDISTRIB-CMS Considering K distributed partitions in our system, at any point we have only K current Edge CE CMS sketches. After the end of K partitions, again when we consider the next K partitions, in order to store the previous iteration's sum we maintain another CMS. Hence the total space complexity $(K+1) * (wd)$ where w and d indicates the number of hash functions and buckets in the CMS structure.

5.9.2 Constant Space for MDISTRIB-FIS Similarly for Frequent Item Sketch, the total space complexity can be denoted as $(K+1) * (M)$ where M is the maximum map size of the datasketch.

5.10 Proofs

5.10.1 Proposition 1 : False Positive Probability Error is more tightly bound in case of MDISTRIB *Count Min Sketch*

As we know for a particular partition θ our chi-square statistic can be written as

$$\hat{\chi^2_\theta} = (a^{\hat{\theta}}_{uv} - \frac{s^{\hat{\theta}}_{uv}}{t})^2 \frac{t^2}{s^{\hat{\theta}}_{uv}(t-1)}$$

Theorem 1 (False Positive Probability Bound)
Let $\chi^2_{1-\frac{\epsilon}{2}}(1)$ be the $1 - \frac{\epsilon}{2}$ quantile of a chi-squared

random variable with 1 degree of freedom. Then:

$$P(\hat{\chi^2} > \chi^2_{1-\frac{\delta}{2}}(1)) < \delta \tag{5.11}$$

[2] mentions that the motivation of the above theorem proposal is to show that with a threshold of $\chi^2_{1-\frac{\epsilon}{2}}(1)$, χ^2 as a test statistic results in a false positive probability of at most δ.

For CMS guarantees 5.2 we have the following

$$P(\hat{a} \leq a + \tau) \geq 1 - \frac{\delta}{2} \tag{5.12}$$

where $\epsilon * V$ is shortened as τ

We define an adjusted version of our earlier score for the total summation score :

$$s^{\tilde{\theta}}_{uv} = s^{\hat{\theta}}_{uv} - \tau \tag{5.13}$$

Combining 5.11 and 5.12 by union bound, with probability of atleast $1 - \delta$

$$\tilde{\chi^2_\theta} = (a^{\hat{\theta}}_{uv} - \frac{s^{\tilde{\theta}}_{uv}}{t})^2 \frac{t^2}{s^{\tilde{\theta}}_{uv}(t-1)}$$

We denote $\tilde{\chi^2_{\theta_{MIDAS}}}$ and $\tilde{\chi^2_{\theta_{MDISTRIB}}}$ as the chi-square test statistic for MIDAS and MDISTRIB methodologies respectively.

(5.14a)

$$\tilde{\chi^2_{\theta_{MIDAS}}} = (a^{\hat{\theta}}_{uv} - \frac{\tilde{s^{\theta}_{uv_{MIDAS}}}}{t})^2 \frac{t^2}{\tilde{s^{\theta}_{uv_{MIDAS}}}(t-1)}$$

(5.14b)

$$\tilde{\chi^2_{\theta_{MDISTRIB}}} = (a^{\hat{\theta}}_{uv} - \frac{\tilde{s^{\theta}_{uv_{MDISTRIB}}}}{t})^2 \frac{t^2}{\tilde{s^{\theta}_{uv_{MDISTRIB}}}(t-1)}$$

(5.14c)

We simplify the equation for MDISTRIB as it holds the same for MIDAS. From equation 5.13, equation 5.14b can be simplified as follows.

$$\tilde{\chi^2_{\theta_{MDISTRIB}}} = (a^{\hat{\theta}}_{uv} - \frac{\hat{s^{\theta}_{uv_{MDISTRIB}}} - \tau}{t})^2 \frac{t^2}{\tilde{s^{\theta}_{uv_{MDISTRIB}}}(t-1)} \tag{5.15}$$

Considering $a^{\theta}_{uv} \approx \gamma \frac{s^{\theta}_{uv}}{t}$ under the assumption that the current edge count is always an estimation of the previous mean plus some positive weight factor,

(5.16)

$$\tilde{\chi^2_{\theta_{MDISTRIB}}} = (\gamma \frac{\hat{s^{\theta}_{uv_{MDISTRIB}}}}{t} - \frac{\hat{s^{\theta}_{uv_{MDISTRIB}}} - \tau}{t})^2 \frac{t^2}{\tilde{s^{\theta}_{uv_{MDISTRIB}}}(t-1)}$$

which can be simplified as considering large value of t, that is for higher timestamps.

$$\tilde{\chi^2_{\theta_{MDISTRIB}}} \approx \hat{s^{\theta}_{uv_{MDISTRIB}}} (\frac{\gamma - 1}{t}) \frac{t^2}{(t-1)}$$

$$\tilde{\chi^2_{\theta_{MDISTRIB}}} \approx \hat{s^{\theta}_{uv_{MDISTRIB}}} (\gamma - 1) \tag{5.17}$$

$$\tilde{\chi^2_{\theta_{MIDAS}}} \approx \hat{s^{\theta}_{uv_{MIDAS}}} (\gamma - 1) \tag{5.18}$$

As per lemma 5.7.1 we know that $\hat{s^{\theta}_{uv_{MDISTRIB}}} \leq \hat{s^{\theta}_{uv_{MIDAS}}}$

Hence combining both the equations we get $\tilde{\chi^2_{\theta_{MDISTRIB}}} < \tilde{\chi^2_{\theta_{MIDAS}}}$

6 MDISTRIB-COA - Collision Aware Anomalous Edge Detection

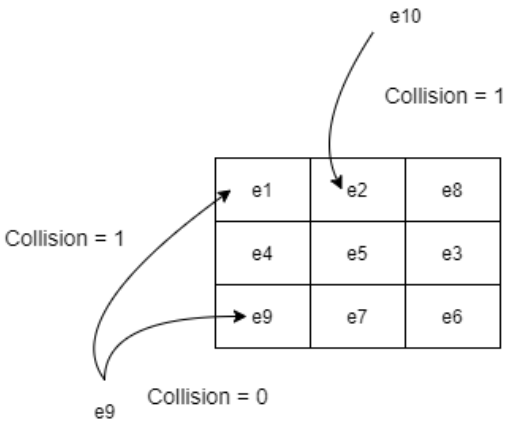
We extend our proposed algorithm on MDISTRIBCMS using only the Count Min Sketch for collision aware optimization. This can also be extended to Apache Frequent Item Sketch but for simplicity we keep it out of scope.

needs clarification - collision between edges or IP addresses?

6.1 Defining collisions First, here we define what exactly is meant by a collision. Collision is the phenomenon when a particular item gets assigned to a slot which was previously occupied by some other item.

Here we show an illustration of how collisions can occur in the CMS. Given a Count Min sketch with dimensions $w = 2$ and $d = 400$, the total slots in the collision array is $w * d$ = 800. Using the above definition of collision, we observe the following cases via experiments in one of our datasets (DARPA).

6.2 Observation 1 Figure 4 showcases the number of non-zero collisions for each unique timestamp. From the plot, we observe the following, that the highest collisions is almost correlated with high number of anomalous edges in that particular timestamp. For example in timestamps *389* and *698*, the number of non-zero collisions achieve a peak. At those timestamps also we observe a high $\frac{\text{anomalous edges}}{\text{anomalous edges+normal edges}}$ ratio. And the possibility arises only for one particular case.



(a) Collisions illustration in a CMS

1	1	0
0	0	0
0	0	0

(b) Collision Array

Figure 3: Number of collisions recorded when there is a new incoming edge

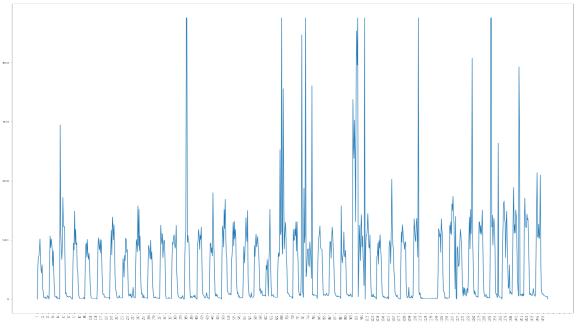


Figure 4: *Current Edge* collisions per timestamp in MDISTRIB-CMS (DARPA)

- If there are a lot of unique incoming edges for a particular timestamp that are not observed in previous timestamps i.e. more variance in the number of unique incoming edges, then we observe a high collision.

The correlation can indicate that due to high number of unique edges arriving during a particular timestamp, it might be an indication of anomaly in the system since such unique edges even if they have low frequency may actually try coordinating an attack simultaneously. Hence MDISTRIB may not be able to detect such low spikes in frequencies as anomaly but actually these unique edge occurence resulting in more collisions in the CMS resulting in more anomalousness in the system.

This doesn't seem convincing. Firstly, figure 4 is probably just correlated with the overall time series of the number of events at each time. Secondly, this claim needs more justification if it is true.

In an online streaming phase for a new timestamp, when there is a simultaneous arrival of edges for different src-dest pairs, its important to notice how we update them in the CMS. The goal is to have as less number of collisions possible in CMS, thus resulting in better theoretical guarantees and thus less false positives. Due to simultaneous edge arrival, the CMS can be updated in any specific order. Hence taking into account the information about the previous collisions that has already happened can be useful in reducing the error in the count estimate. We thus propose the following solution. In addition to CMS structure for s_{uv} and a_{uv} we also keep an additional CMS streaming datastructure $CollisionCMS$ for storing the number of collisions encountered upto the current time tick t for each src, $dest$ pair. For the current time tick t, $\hat{a_{uv}^{CN}}$ indicates the estimated number of collisions that was encountered when edge uv was inserted during the current time in the corresponding Current Edge CMS Sketch. While $\hat{s_{uv}^{CN}}$ indicates the estimated number of collisions that was encountered when edge uv has been inserted into the Sum sketch uptil time tick t

At each time tick t we check within the CMS for which of the slots the number of collisions fall below some threshold. For this we consider a quantile threshold of 25%, i.e for considering hashing to slots that fall below 25% in the collision number distribution for the CMS. For this we use a separate datastructure called Apache Quantile Sketch for calculating the quantile threshold and determining which slots fall below that threshold.

Algorithm 4 Distributed Microcluster Detection with Collision Awareness

Input: Stream of graph edges for Partition id θ
Output: Anomaly Scores per edge
while *a new edge* (u, v, t_θ) *arrives* **do**
  Assign edge (u, v) to a thread
  Perform merge operation for $CS_i^m = \sum CE_i^m$
  Retrieve updated counts $\hat{a_{uv}}$ from CE_i^m
  Retrieve updated counts $\hat{s_{uv}}$ from CS_i^M
  Compute Edge Scores
  $score((u, v, t)) = (\hat{a_{uv}} - \frac{\hat{s_{uv}}}{t})^2 \frac{t^2}{\hat{s_{uv}}(t-1)}$
  Output Scores
  EndWhile

7 Experiments

In this section we try to evaluate our proposed anomaly scoring algorithm MDISTRIB and try to answer the following questions:

Q1. Accuracy: How accurately does MDISTRIB-COA detect real-world anomalies compared to baselines, as evaluated using the ground truth labels?

Q2. Scalability: How does it scale with input stream length? How does the time needed to process each input compare to baseline approaches?

Q3. Execution Time: How fast does the proposed method perform with respect to baseline methods while generating anomaly scores in the offline phase?

Baselines: MIDAS-R, SedanSpot, PENnimer, F-Fade

Datasets In this section, we evaluate performance of MDISTRIB-COA as compared to other baselines in Table **??** based on 5 real world datasets : *DARPA* [6] and *ISCX-IDS2012* are popular datasets for graph anomaly detection. According to survey [52], there has been proposal of more than 30 datasets and it has been recommended to use *CIC-DDoS2019* and *CIC-IDS2018* datasets. We also showcase the same for *CTU-13* dataset. Dataset details are discussed in appendix B. Hyperparameters for baselines are also provided in Appendix H. Appendix F describes the experimental setup and results with some additional parameters are shown in Appendix G.

All methods output an anomaly score corresponding to the incoming edge, a high anomaly score implying more anomalousness. As part of the experiments, we follow the same suite as the baseline papers in reporting the Area under the ROC Curve (using the True Positive Rate TPR and the False Positive Rate FPR) and the corresponding running time. All experiments for calculating the AUC are averaged for 8 runs and the mean is reported along with its spread.

7.1 Experimental Setup All experiments are carried out on a $1.00GHz$ Intel Core $i5$ processor, $8GB$ RAM, running OS $Win10$ 10.0.19042. We used an open-sourced implementation of MIDAS, provided by the authors, following parameter settings as suggested in the original paper (2 hash functions, 719 buckets). We implement MDISTRIB in python using dask framework. We follow the same parameter settings as (2 hash functions, 719 buckets).

7.2 Q1. Accuracy Table 2 includes the AUC of edge anomaly detection baselines along with our two proposed methods MDISTRIB and MDISTRIB-COA. PENimer is unable to finish in large datasets like CIC-DDos and CIC-IDS within 10 hours range, thus we dont report the result. Among the baseline results MDistrib-COA provides better accuracy results than the baselines on datasets like DARPA, CIC-DDoS and ISCX-IDS.

7.2.1 AUC comparison of DistributedMIDAS with respect to baselines for different datasets How does frequent item sketch data structure size changes How AUC changes based upon DS Size changes

7.2.2 Observation 1: How Distributed MIDAS scales with different node sizes

7.2.3 Online Phase Plot Plot showing the Anomaly score variations as new incoming edge comes in.

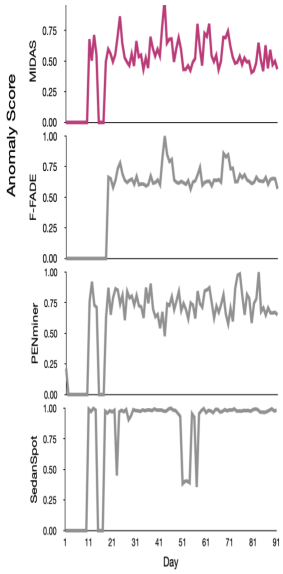
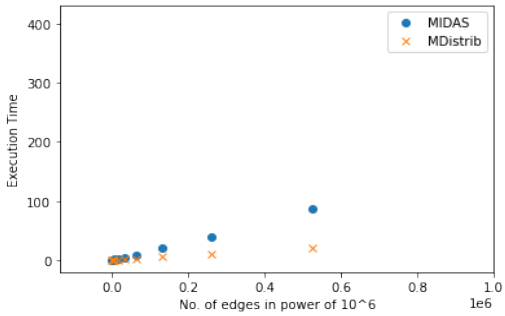


Figure 5: Online Phase- Show variations of the anomaly score when a new edge arrives in the stream

7.3 Q2. Scalability



number of workers used?

Figure 6: MDISTRIBw.r.t MIDAS - Scalability with respect to the number of edges

7.3.1 Plot of MDISTRIB with respect to MIDAS for exponential no. of edges in powers of 2. In Figure 6 we showcase how via distributed processing of individual nodes and computing the sketch results in significant time improvement of execution. At the same time, we showcase that the

	DARPA	CIC-DDoS	CIC-IDS	CTU-13	ISCX-IDS
MDistrib-R	0.953 ±0.001	0.986 ±0.003	**0.979 ±0.01**	0.908 ±0.002	0.806 ±0.001
Spotlight	0.89 ±0.001	0.67 ±0.002	0.42 ±0.003	0.74 ±0.002	0.63 ±0.001
MIDAS-R	0.95 ±0.005	0.984 ±0.003	0.96 ±**0.01**	0.973 ±0.005	0.81
PENimer	0.87	-	-	0.72	0.51
F-Fade	0.91 ±0.005	0.72 ±0.02	0.61 ±0.03	0.58 ±0.001	0.62 ±0.004
MDistrib-COA	**0.98 ±0.0004**	**0.986 ±0.001**	0.93 ±0.0005	0.89 ±0.004	**0.93 ±0.0001**

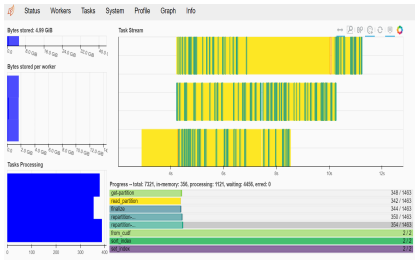
Table 2: AUC scores of the proposed methods in comparison with the edge anomaly detection baselines. The best values over a column (for a specific dataset) are shown in bold.

number of edges increase exponentially, MDISTRIB shows linearity of increase in execution time with a slope factor of $\approx K$ where K is the number of partitions considered in our setting.

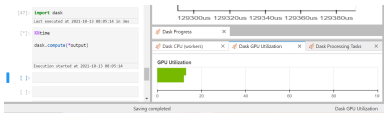
7.4 Q3. Execution Time

7.5 GPU based Execution In addition to our proposed solution, we also extend MDISTRIB to a GPU setting. For this we created a dask cluster hosted on Saturn cloud and ran our analysis on the same. We use Python's JupyterLab for running our experiments with server configuration of T4-XLarge, with 4 crores, 16GB of RAM and 1 GPU. In addition we use cuDF, python's GPU dataframe library to store our data.

We showcase here the experiments done for the DARPA dataset here. Using the above configurations, we achieved an execution time of **672ms** which is 1.37 times faster than MDISTRIB in non-GPU environment.



(a) DashBoard Status



(b) GPU Utilization

Figure 7: MDistrib - GPU

8 Conclusion

In this paper, we proposed MDISTRIB. Our contributions are as follows:

1.

2.

3.

References

[1] Daniel Anderson, Pryce Bevan, Kevin Lang, Edo Liberty, Lee Rhodes, and Justin Thaler. A high-performance algorithm for identifying frequent items in data streams. In *Proceedings of the 2017 Internet Measurement Conference*, pages 268–282, 2017.

[2] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. Midas: Microcluster-based detector of anomalies in edge streams. In *AAAI 2020 : The Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[3] Yen-Yu Chang, Pan Li, Rok Sosic, MH Afifi, Marco Schweighauser, and Jure Leskovec. F-fade: Frequency factorization for anomaly detection in edge streams. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 589–597, 2021.

[4] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[5] Xenofontas Dimitropoulos, Marc Stoecklin, Paul Hurley, and Andreas Kind. The eternal sunshine of the sketch data structure. *Computer Networks*, 52(17):3248–3257, 2008.

[6] Richard Lippmann, Robert K Cunningham, David J Fried, Isaac Graf, Kris R Kendall, Seth E Webster, and Marc A Zissman. Results of the darpa 1998 offline intrusion detection evaluation. In *Recent advances in intrusion detection*, volume 99, pages 829–835, 1999.

[7] Rui Liu, Siddharth Bhatia, and Bryan Hooi. Isconna: Streaming anomaly detection with frequency and patterns. *arXiv preprint arXiv:2104.01632*, 2021.

[8] Nguyen Thanh Tam, Matthias Weidlich, Bolong Zheng, Hongzhi Yin, Nguyen Quoc Viet Hung, and Bela Stantic. From anomaly detection to rumour detection using data streams of social platforms. *Proceedings of the VLDB Endowment*, 12(9):1016–1029, 2019.