Adaptive Tolerance Algorithm for Distributed Top-K Monitoring with Bandwidth Constraints

Michael Bauer, Srinivasan Ravichandran
University of Wisconsin-Madison
Department of Computer Sciences
{bauer, srini}@cs.wisc.edu

ABSTRACT

Distributed top-k monitoring is an important task that is being performed almost everywhere in the Internet today, and has applications in DDoS prevention, website access detection, and many other areas. In the past decade distributed top-k algorithms have emerged which limit bandwidth usage by applying constraints and only communicating when those constraints are violated. However, these algorithms don't perform efficiently when the data set is consistently oscillating, which can result in highly variable bandwidth usage. We propose an adaptive tolerance based system, which varies the amount of error to achieve a consistent bandwidth usage. This approach achieves the accuracy of the original top-k implementation when bandwidth is high and can achieve constant bandwidth even when the distribution of top-k objects is highly variable.

1. INTRODUCTION

We have entered into an age of big data. Large organizations that interact with end-users typically have data centers all over the globe for improved performance and availability. This leads to terabytes of monitoring information generated everyday. This monitoring information is then analyzed to decide where to add servers, replicate data, study usage patterns etc. However, these queries can be very expensive, both in computational complexity and network requirements. Many organizations make use of the naive approach of transferring all data to a centralized node, which performs all the processing. Limited availability of bandwidth and the geographical distribution of these nodes makes this an expensive task. The last decade has seen a large amount of research in reducing the network overhead imposed by such queries. In many applications, the specific behaviors of interest to monitor in detail are characterized by numeric values that are exceptionally large (or small) relative to the majority of the data.

One particular query which has been found to be particularly useful and frequent is finding the top k globally ranked objects among a large set of objects which are distributed across the world. In this model there is a central processing location, and a number of remote monitors. These monitors store a number of objects with associated values. A top-k query will supply the k objects with the largest values when summed over all monitor nodes. After calculating these values, the central processing location can issue more specific queries for these top-k objects only. This can reduce computational complexity, and minimize wasted network bandwidth. In this model, detailed statistics about the objects not in the top-k set will not be sent across the network.

A potential application of this kind of a query can be seen in detecting distributed Denial-of-Service (DDoS) attacks. These attacks are typically realized by a set of hosts issuing a large number of requests (such as DNS requests) to a server to overwhelm it. By monitoring the number of requests per unit time coming from hosts, one can stop a DDoS attack from becoming successful. However, DNS servers may normally receive large numbers of requests making it difficult to monitor these requests. Identifying these malicious requests can be implemented using a top-k monitoring scheme.

Another application that could be of interest is identifying the set of most frequently accessed web pages in a given website. Identifying these most frequently accessed web pages can help the website administrator to make these pages highly available by replicating these web pages over multiple servers or CDNs. This problem squarely fits into the distributed top-k framework that is mentioned above.

In this project, we started out by looking at the performance of the distributed top-k monitoring algorithm proposed by Babcock and Olsen in [1]. It is worth noting that the algorithm in [1] provides the top-k set within a certain tolerance ε. However, once set, the value of ε does not change. We observed that setting the tolerance as a constant irrespective of the distribution of the input values can result in highly variable bandwidth usage. In many cases, a user of a network can request a fixed portion of bandwidth across the internet, and send packets happily as long as they remain within this limit. With a fixed epsilon, bandwidth can potentially rise or fall significantly as the nature of the data oscillations change, causing unoptimal results. Instead, we propose an algorithm that takes as input a target bandwidth and adaptively changes its tolerance value based on the input distribution. This achieves low bandwidth usage and high tolerance with low data oscillations, and also does not

exceed the bandwidth limit in particularly variable environments.

2. RELATED WORK

A traditional approach to top-k monitoring queries is similar to a simple Map-Reduce style approach. Whenever the set of top k objects are queried for, a central processing node sends a request to every location to send the values for all objects. We call this central processing node the coordinator and the remote locations that collect and store data as monitor nodes. Once all data has been received, the coordinator collects the partial data from each node, uses rank aggregation techniques to obtain the total values of the objects and sorts these total values to achieve a global top-k list of objects. This approach is computationally intensive because of the rank aggregation step which may not scale well and is also bandwidth-intensive as it might require gigabytes of data transfer for every update. As a result, these queries are traditionally only performed a maximum of a few times a day.

Because of the aforementioned drawbacks, other approaches have been developed to compute the top-k values while minimizing network requirements and computational requirements. For the purposes of this course, we are considering only on the bandwidth-usage minimization part. The seminal work in this space is by Babcock and Olston in [1], which performs distributed top-k monitoring using a clever technique. They make use of the fact that only the changes in the partial data values at the data collection points are necessary to compute the global top-k ranked objects.

They show that this approach greatly reduces network bandwidth usage, because in most cases no communication is necessary. This work has had tremendous impact, with over 1200 citations in the past 12 years. The novelty in this algorithm lies in the fact that they make use of what they call adjustment factors for each object at each monitor node. They also come up with a set of constraints for every object at every node that need to be satisfied in order for the top-k set of objects to remain valid. Any violations of these constraints would lead to a process called resolution wherein the coordinator node will then recompute the top-k set based on these adjustment factors (and not with the actual data values) and installs a new set of constraints that should hold good for this new top-k set to remain valid. An important aspect of their algorithm is the tolerance factor that the user can set when initiating the algorithm. The tolerance factor is the extent to which the top-k set computed by the algorithm can be away from the actual top-k set. Varying this tolerance factor ε will have an effect on the amount of data transferred between the monitor nodes and the coordinator node.

3. BACKGROUND

In this section we will define the problem formally and also will explain the two approaches mentioned above in detail. In the next section, we will talk about the implementation of our comparison and present our results.

3.1 Problem Definition

We consider a distributed monitoring environment with $m+1$ nodes. There is a single coordinator node N_0 and m monitor nodes $N_1, N_2, N_3, ...$ There are a set of n data objects $\mathcal{U} = \{O_1, O_2, O_3, ...\}$ that are being monitored by these nodes. Each of these objects has a numeric value $V_i, i = 1, 2, ...m$ associated with it. The values of the data objects are not seen by any one individual node. But updates to these values occur over time as a sequence of $\langle O_i, N_j, \Delta \rangle$. When a tuple $\langle O_i, N_j, \Delta$ arrives, it represents that the object O_i's value has changed by Δ at node N_j. Note that Δ may be positive, negative or zero. Also note that the tuple $\langle O_i, N_j, \Delta \rangle$ can be seen only by node N_j.

3.2 Naive Centralized Aggregation

As mentioned earlier, the traditional approach to obtain the top-k objects has been to collect all the partial values from every node at the coordinator node and to use rank aggregation techniques to get the total values of the objects. These total objects can then be sorted to get the top-k objects. Although this approach provides accurate answers, it is not advisable to use this for scenarios where the data is being streamed as is the case in most applications. It is both computationally and bandwidth-wise inefficient to use this approach whenever the partial values at the nodes changes. In the worst case, even if one object's partial value changes in just one node, it could potentially change the top-k ranked objects, leading to a subsequent computation.

3.3 Distributed Top-K Monitoring

In order to address the issue of high network usage, the networking research community has tried to come up with a wide variety of techniques. One of the most influential of these works was done by Babcock and Olston in [1]. They came up with a clever technique to maintain a global top-k set of objects when each node received a stream of updates to the values. They make use of *adjustment factors* at each node for each object and define arithmetic constraints for each object at each node. The constraints are set up in such a way that whenever even one of them fails, there is a violation (or a potential violation) of the validity of the current top-k set.

To give an intuition, let us look at a *sample run* of their algorithm. Let the number of nodes be 2 and let the number of objects also be 2. Suppose the current values of the objects O_1 and O_2 at node N_1 are $V_{1,1} = 9$ and $V_{2,1} = 1$ and at node N_2, they are $V_{1,2} = 1$ and $V_{2,2} = 3$ and if $k = 1$, then clearly the top-k set $\mathcal{T} = \{O_1\}$. To bring the local top-k set at each node into alignment with the overall top-k set, parameters called *adjustment factors* $\delta_{i,j}$ are defined for each partial value $V_{i,j}$. The purpose of the adjustment factors is to redistribute the data values more evenly across the monitor nodes so the k largest adjusted partial data values at each monitor node correspond to the current top-k set T maintained by the coordinator. However, in order to ensure correctness, they impose the additional constraint that the adjustment factors for each object across all nodes sums up to 0. They make use of these adjustment factors to define constraints that need to be satisfied at each node as the values at that node are being updated. If a constraint is violated, then a process called resolution takes place.

The resolution process involves 3 phases. In the first phase, the node at which one or more constraints have failed, N_f , sends a message to the coordinator N_0 containing a list of

failed constraints, a subset of its current partial data values, and a special border value it computes from its partial data values. Following this, the coordinator determines whether all invalidations can be ruled out based on information from nodes N_f and N_0 alone. If so, the coordinator performs a reallocation procedure to update the adjustment factors pertaining to those two nodes and sends to N_f its new adjustment factors. In this case, the top-k set remains unchanged and resolution terminates. If, on the other hand, the coordinator is unable to rule out all invalidations during this phase, a third phase is required in which *all the nodes* are asked to send their values and after a series of communications between the central coordinator and the monitor nodes, the new set of top-k objects is identified and the corresponding adjustment factors and constraints are installed at each node. The authors note in their paper that the above resolution process does not take a long time to complete. Specifically, they argue that the above process usually terminates in step 2 most of the time.

To further limit network usage, a global ε can be defined. Once defined, the adjustment factors at each node will be scaled slightly such that the global top-K can tolerate an error of at most ε. This means that any global oscillations between the top-k and non top-k objects will likely not produce any costly resolutions. However, we argue that fixing epsilon to a specific value will produce variable bandwidth usage as the distributions change. For example, consider two values which have aggregate values ranked k and $k+1$ respectively. If this values are almost identical, and oscillate back and forth so that the top-k changes, we will continue to see resolutions. However, a small epsilon will get rid of these oscillations, and simply pick one of the two objects to remain in the top-k set. The epsilon will adjust this value so that it no longer oscillates with the other. However, if the magnitude of the oscillations increase, this epsilon will no longer be suitable, and resolutions will constantly occur. This will lead to a significantly higher bandwidth.

4. ADAPTIVE EPSILON

As mentioned earlier, fixing the tolerance factor to a constant value leads to very high or very low bandwidth usage. It would be ideal if the algorithm would adjust the tolerance factor based on the available bandwidth and the input distribution. For this work, we consider that the available bandwidth does not change over time. It is only the rate at which the values grow at each node that changes over time. However, our model can be extended to the case where the available bandwidth is known as a function of time.

We approach this problem of using adaptive tolerance factors as follows. We first start with an arbitrary tolerance factor. Let the target bandwidth that is available be B. Now, if the traffic over the past T seconds averaged at more than a fraction F of B, we consider it as an implicit signal saying that the bandwidth is about to be over-used and so we increase the value of ε. This would result in a top-k set that is less accurate.

Similarly if the bandwidth goes below the target bandwidth B by more than a fraction $1/F$ of B, we use it as an implicit signal saying that the bandwidth is under-utilized and so we could decrease the value of ε to get even more accurate results.

This bandwidth is simply recorded at the central coordinator by determining the message size of all incoming/outgoing messages. This bandwidth is referred to as I, and averaged over the last 10 seconds to obtain a reasonable result. Every ten seconds, ε is scaled by the difference between the target bandwidth and the sampled bandwidth. This means that if there is a large difference between the target bandwidth and sampled bandwidth, we will more aggressively change ε to quickly arrive at the correct estimate. If the distribution remains relatively constant, this approach will lead to a relatively constant epsilon.

A simple version of the scaling formula is seen below:

$$\varepsilon_{new} = \varepsilon_{old} * \frac{I}{B}$$

Note that this is a multiplicative increase - multiplicative decrease feedback control (MIMD). This new epsilon value is stored at the coordinator, and used in any future computations for adjustment factors. This requires no additional communication to the monitor nodes. However, if we decrease the value of epsilon, we have an extra opportunity for accuracy. With a large epsilon, some object could be classified as not in the top-k set, even though its global sum is larger than some sum of a different object in the top-k. This is due to the large epsilon which allows for such a tolerance. However, as epsilon decreases, if the values at each node remained constant there would be no communication between the nodes. Each monitor would keep its old adjustment factors, and the global top-k would be incorrect, even though epsilon has decreased. Instead, we introduce the following policy.

If there has been no communication in the last 10 seconds ($I = 0$) and epsilon has decreased, we recompute the adjustment factors for the global top-k objects. We then pass these adjustment factors to each node. This is tolerable because we are well below the current target bandwidth (we are consuming no data), so this increased communication will not exceed the target. As these new adjustment factors are propagated, any false top-k obejcts will be detected and removed properly.

5. IMPLEMENTATION

We created an implementation of the Distributed Top-K Algorithm from [1] in Python (about 2200 lines total). We modeled each node (coordinator and monitors) as separate processes. These nodes are connected in a virtual topology in Mininet. At each monitor node, we run a generator process as a separate thread that generates updates to the objects. We make use of this generator process to provide the input according to different probability distributions. This essentially provides a simulated environment that mimics a distributed set of nodes that collect data constantly. The operation of the different nodes are detailed below.

5.1 Monitor Nodes

The monitor process deals with incoming data from the generator process and maintains adjustment factors for each object. Whenever a constraint is violated, the monitor node communicates violation of constraints to the central coordinator.

There are four threads operating at each monitor node. The first one is the generator thread that reads data from a configuration file, and adds this data to the partial object count once every second (adjustable to arbitrary time period). These data additions are maintained in a dequeue. A second thread checks the tail of the dequeue every second and removes any objects older than 10 seconds to maintain a rolling window. A third thread checks for any violations between adjustment factors and object counts. A fourth thread handles any messages from the coordinator, and updates necessary adjustment factors. Locks are used to provide correctness.

5.2 Coordinator

The coordinator node handles TCP connections from the monitor nodes and sends commands to the monitor node to carry out specific functions. This gives us more flexibility for generating data that mimic different scenarios with a single implementation. Once started, the coordinator first signals the monitor nodes to start generating data based on their local configuration files. The coordinator then, as a first step collects the global set of objects and their values from each monitor node and then uses a traditional sorting algorithm to find the top k objects in the set. Once this initial top-k set of objects has been determined, the coordinator will not use this initial, naive sorting algorithm again.

We have faithfully implemented the algorithm described in [1] using our setup on Mininet. The coordinator is threaded to allow for simultaneous communication, but as in [1] only one simultaneous resolution is allowed.

Any messages sent to and from the coordinator are recorded in a queue. Every ten seconds, a thread wakes up and determines the total number of bytes sent in the last ten seconds. This is the value I as mentioned in Section 4. This I is used to scale ε properly. If no communication has occurred and ε has decreased, a resolution is performed and

Add info about another thread that keeps running bandwidth and coordinator sends epsilon if changes are made.

6. EXPERIMENTAL SETUP

We setup the following experiment to evaluate our algorithm against baselines for several different scenarios. The basic setup is that there are 10 nodes and at each node there are 25 key-value pairs (k, v) where $k \in \{a, b, c, ...y\}$ and $v \in N$, values are generated for keys according to distributions on values over time. For example, a plot of one of the distributions that we used is shown in Figure 1.

Once the experiment is started, a thread in the coordinator process stores all messages. For each message, a line is output containing the current timestamp, message size, message type, and the current value of epsilon.

We then use these values to make a number of plots. We

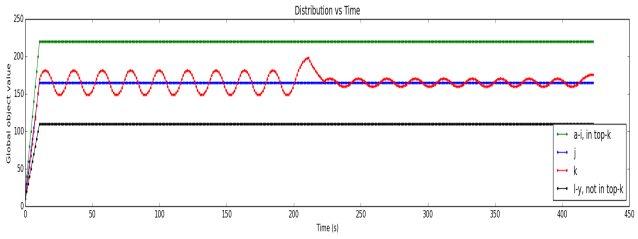

Figure 1: Distribution of values over time

calculate the global bandwidth by averaging the total bytes consumed over the last 20 seconds. This is done for ease in readability, but the graphing system can compute an average over any arbitrary amount of time. This bandwidth is plotted vs time for various settings.

Next, we use the recorded values of epsilon for the same experiments to plot this tolerance as a function of time. These plots are aligned with the bandwidth for the same experiment to see any correlations.

7. RESULTS

We compare our variable epsilon algorithm with the original Top-K implementation from [1] for two different distributions. The first distribution can be seen in 2. In this example, the values are constant until 200 seconds. We can see that a-i are always in the top-k set, and have a maximum global value. The key j is also in the top-k set, but has only a slightly bigger value than the $K + 1^{th}$ value k. However, at 200 seconds, the value of k begins to oscillate, such that the top-k set is also oscillating.

We examine four test cases that use this distribution. Two use a constant epsilon, while two use constant bandwidth. The results are shown in 2.

When the top-k set is static, we can see that all four tests use no bandwidth. The adaptive-epsilon examples successfully scale epsilon down to 0, as there is no communication. However, we can see that epsilon will begin to scale as the top-k set begins to oscillate. As expected, the adaptive example which targets less bandwidth (6000) scales epsilon to a larger value than the example that targets a higher bandwidth (8000). This increased epsilon leads to a higher tolerance, and thus lower communication cost.

It can be seen that our adaptive algorithm achieves more accurate results than the case where the tolerance is 10 but does not use as much bandwidth as the case where there is zero tolerance. Thus, our algorithm provides the user with the power to limit the bandwidth usage and still get the best possible result set given this bandwidth constraint. This could prove to be useful in wide area networks in which typically one of the transcontinental links turn out to be the bottleneck link. Also note that the algorithm does not incur any additional overhead other than just sending the updated tolerance factor from the coordinator to other nodes which amounts to sending an integer to every node from the coordinator node.

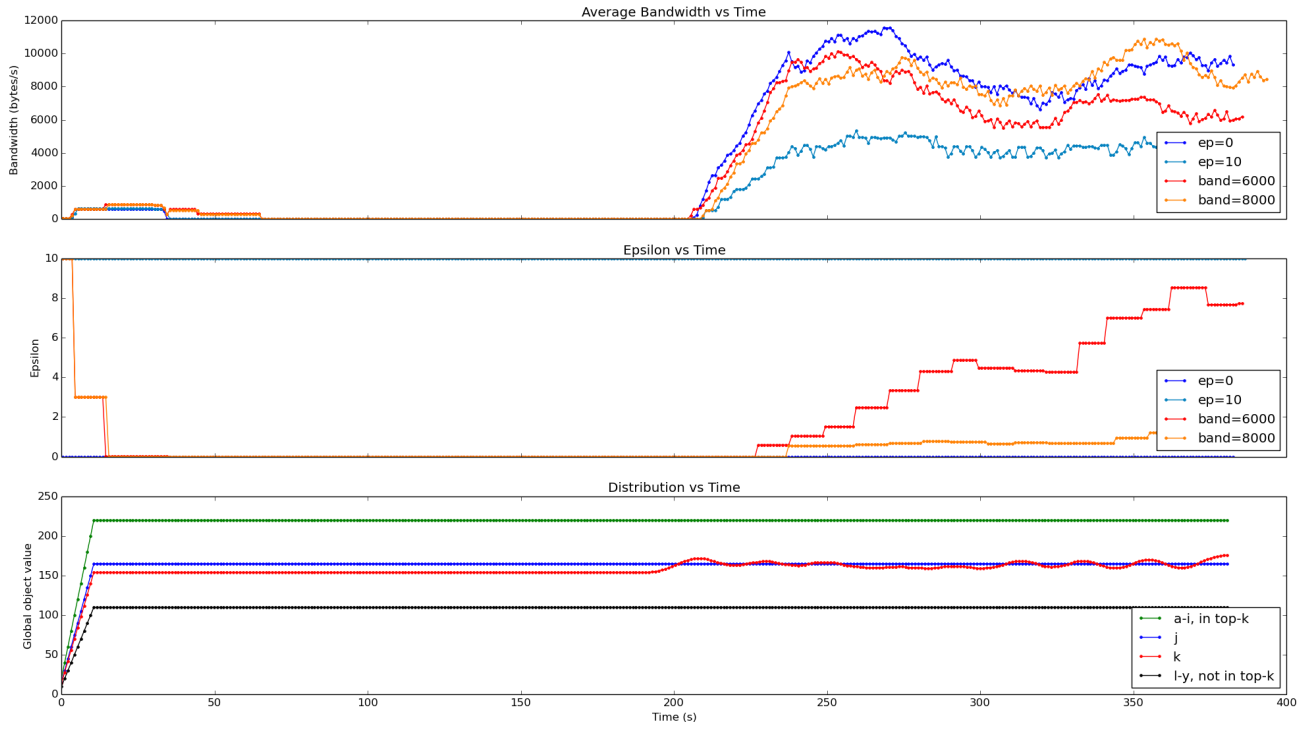

Figure 2: Statistics recorded for a distribution which is static, then begins oscillating. The variable epsilon algorithm scales epsilon to achieve the correct bandwidth.

In the next plot (Figure!3), we show the case where there is a difference in the magnitude of the oscillation in the top-k set. There are $K-1$ objects in the top-k set at all times but the K^{th} element keeps changing because of the sporadic generation of values for the keys j and k. This can be seen as a perturbation distribution and we observed that our algorithm does not aggressively respond to perturbations for high bandwidth bounds as can be seen in Figure 3. Thus when provided with sufficiently large bandwidth, our algorithm converges faster to a steady state (non-varying ϵ).

In this example, we see that for a fixed epsilon of 10, the communication is highly variable. When the distribution is oscillating at a high magnitude, the bandwidth is around 7000 bytes/s. However, when the magnitude of the oscillations decrease, the average bandwidth is around 4000 bytes/s. Our approach achieves more consistent results for bandwidth. The epsilon graph in Figure 3 shows how epsilon successfully scales to achieve consistent bandwidth. For both magnitudes of oscillation, allowing for a higher target bandwidth allows for a smaller epsilon, and thus more accurate results.

8. FUTURE WORK

One possible problem that needs to be addressed is to account for the variation in the available bandwidth in the network. This becomes particularly important in wide area networks where transcontinental links may be congested during some time of the day and free during other times. Another interesting problem would be to estimate the tolerance factor with a better update rule using other feedback control algorithms such as AIMD, AIAD and MIAD.

9. SOURCE

Github is available at `github.com/cs740wisc/simulator`.

Code tarball and report source are available at `pages.cs.wisc.edu/~bauer/topk/`.

10. CONCLUSION

We have implemented an algorithm that actively adapts its tolerance based on the distribution of values subject to bandwidth constraints. We have shown that it is stable and carried out experiments to analyze the performance of our algorithm against the one in [1]. The major contribution of our work is that the algorithm automatically chooses the tolerance factor against [1] where the tolerance is fixed and there is no direct way to set a tolerance value at the start without knowing what the distribution of values will be.

11. REFERENCES

[1] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 28–39, San Diego, California, June 2003.

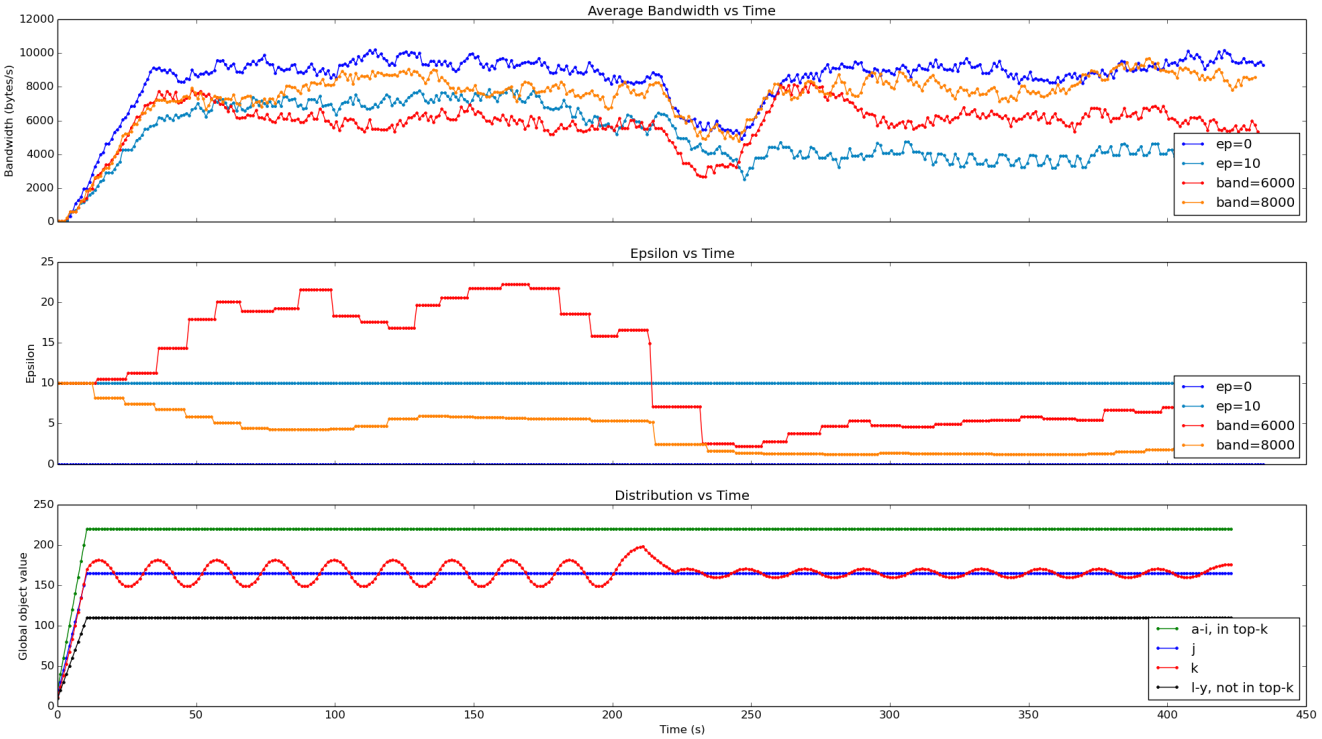

Figure 3: Statistics recorded for a distribution in which the top-k set is oscillating. The variable epsilon algorithm scales epsilon to achieve the correct bandwidth.