# Using Residual Resource Consumption to Resample Top-$k$ Monitoring Reports

Thomas Gschwind, Metin Feridun
IBM Research
CH-8803 Rüschlikon, Switzerland

***Abstract*—Top-$k$ reports are compound metrics that provide useful information when diagnosing problems in a system, e.g., to identify persistent CPU usage by a process. In large systems, these reports are collected at regular intervals and need to be resampled to a coarser granularity to answer user queries for different sampling periods, or to save space and make it possible to keep historical data for long term performance analysis. However, resampling top-$k$ reports, i.e., aggregating several reports collected for small time intervals into a single top-$k$ report can introduce inaccuracies. For example, a process that consistently uses CPU over the aggregation interval but did not make it to the short term top-$k$ reports will be missing from the aggregated report. In this paper, we present an algorithm that collects top-$k$ reports at regular intervals and can aggregate them with little or no error. This is done by including residual resource consumption of unreported, but potentially significant entities in the top-$k$ reports, and using these residual values during aggregation. We show different approaches to including residual resource consumption in individual top-$k$ reports, analyze the error introduced, and demonstrate the effectiveness of the algorithm in real-world scenarios.**

## I. Introduction

The importance of monitoring tools for large distributed systems is well known and many tools such as Ganglia [1], Scuba [2], or our own performance monitoring system [3] have been developed. Many of these systems gather the performance monitoring data in a so called time-series database that stores performance values of a system at a given sampling rate. The collected data may be resampled for two reasons. One, the sampling rate at which data is collected may not always match the sampling rate requested by the user, hence, requiring to resample the data. Two, in a large clustered system several gigabytes of data will be collected quickly and therefore, monitoring systems need to discard or resample older performance data into a coarser granularity [3] in order to cope with the large volume of data. For simple metrics such as counters, or absolute values, resampling such data can be achieved simply by retaining the highest value or the average of the values over the resampling period.

In our performance monitoring system, the top-$k$ reports are metrics that capture the top-$k$ processes consuming CPU time. It lets us identify the processes responsible for CPU time consumption spikes which is important for detecting system faults. Top-$k$ reports, in itself are nothing new and have existed for a long time [4], [5] and used, for instance, to identify the largest customers of a company by examining its sales database. More recently, top-$k$ metrics have been applied to streaming data to show the most often visited web pages in the last week [6].

Several algorithms have been designed to efficiently collect such data. The original work on this topic is for top-$k$ queries in conventional databases [4], [5]. Mouratidis et al. [7] expanded it to cover top-$k$ queries of data streams whose algorithm was subsequently optimized by Yang et al. [6].

Algorithms to resample top-$k$ reports, however, have not been reported in the literature. However, resampling top-$k$ reports, i.e., aggregating several reports collected for small time intervals into a single top-$k$ report can introduce inaccuracies. For example, a process that consistently uses CPU over the aggregation interval but did not make it to the short term top-$k$ reports will be missing from the aggregated report.

Our contribution is to address this problem both from a research and an engineering point of view. We investigate the different strategies to resample top-$k$ metrics and analyze the situations under which an unreported entity may become eligible for becoming a top-$k$ entity. Based on this analysis, we develop a heuristic for resampling top-$k$ reports, and compute the margin of error that this and other resampling approaches may exhibit. We consider only resampling shorter intervals to longer intervals as the reverse would require recreating lost information.

As we will show, being able to accurately resample top-$k$ reports under any condition would require us to report all entities at every point in time. It is obvious that this approach is hardly scalable in a typical deep data analysis cluster.

The paper is structured as follows. Section II gives an overview of the background of this problem, and Section III of the terminology used throughout the paper. Section IV discusses a simple top-$k$ implementation and the margin of error when resampling the top-$k$ data into larger time windows. Section V presents our extension to the top-$k$ data reporting and how it improves the margin of error encountered. Section VI evaluates the margin of error of both approaches in different real-world scenarios and compares them with each other. Related work is considered in Section VIII and potential future work in VII. Section IX presents our conclusions.

## II. Background

Our performance monitoring system which is shipped as part of IBM Spectrum Scale [8] is a distributed performance monitoring environment consisting of a set of sensor nodes that send resource consumption data to a smaller set of collector nodes. Clients interested in the performance figures connect to one of the collectors which then collaborates with each other to fulfill the client's request.

Clients, depending on the purpose, may request performance metrics with any sampling rate which our performance monitoring system resamples as desired. Also storing performance monitoring data from a large set of nodes over an extended period of time requires lots of memory. To deal with this amount of data, our performance monitoring system keeps more recent performance data available at a finer granularity whereas older performance data is resampled to a coarser granularity to conserve memory [3]. In this paper we used this performance monitoring system to also show the top-$k$ processes for a node's CPU load.

For typical performance measurements, such as CPU load, standard aggregation operations such as the average or the sum may be used [3]. Keeping historic top-$k$ data in itself is straight forward but resampling this type of data is more challenging.

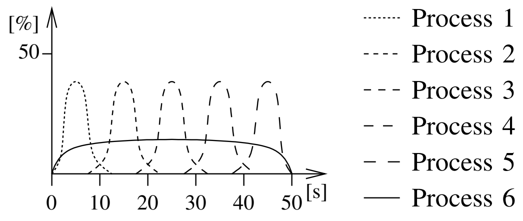


Fig. 1. Sample CPU Usage over Timer

Let us look at the sample CPU usage chart in Figure 1. It shows 6 processes, 5 running for ten seconds each and one running for 50 seconds. While the ten-second processes are the top process at each ten second interval, the overall top process for the entire period of 50 seconds is process 6.

Top-$k$ data may be of bounded or unbounded data. Bounded data describes finite resources that can be expressed as a fraction, such as CPU time utilization. In contrast, an unbounded top-$k$ report may indicate the top-$k$ accessed web pages of a web server. We focus on bounded data because most metrics we collect are bounded by some number such as the disk size, the network bandwidth, etc.

## III. Terminology

Figure 2 shows 5 top-$k$ CPU consumption reports with $k = 4$. The $x$ axis indicates the different reports, whereas the height of each box indicates the resources consumed by an entity in the top-$k$ report. The figure also shows unreported entities whose borders are drawn with dashed lines.

We represent the $i$-th report in a series of reports with $V_i$ and the $j$-th entry of a given report $V_i$ as $v_{ij}$. In formulas, we use $v_{ij}$ as a shorthand for its resource consumption $\text{cons}(v_{ij})$. As we mentioned previously, the values indicate the fraction of the consumption of a finite resource: $0 \leq v_{ij} \leq 1$ and $\sum_j v_{ij} \leq 1$.

The entries within a given report $V_i$ are sorted using their resource consumption ($v_{ij} \leq v_{ij'}$ iff $j \geq j'$). Hence, the largest value reported in a report $V_i$ is $v_{i1}$ and the lowest

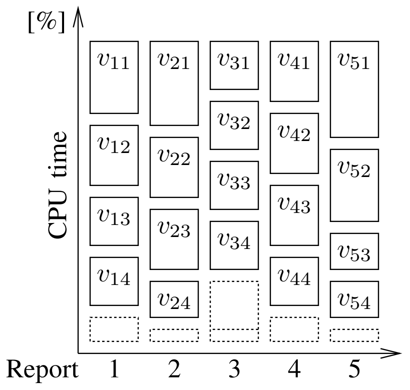


Fig. 2. Sample Set of Top-$k$ Reports

$v_{ik}$. By definition, no unreported entity may have a resource consumption higher than $v_{ik}$.

In order to correctly resample a series of $n$ top-$k$ reports, we would have to store the resource consumption of all processes, simply because we cannot possibly know the future resource consumption of an entity. For instance, an unreported entity in Figure 2 might not be included in any top-$k$ report but may consume enough resources for it to be included in a resampled report. This is the same problem as in the Heavy Hitter problem explained in [9].

To better illustrate our examples and without loss of generality, in the following we view top-$k$ entities as processes consuming CPU resources.

## IV. Simple Top-$k$ Reporting and Resampling

In this section, we use a simple top-$k$ resampling approach that will only consider the data reported in the top-$k$ reports. If a set, let's say of 5, such reports are to be resampled later on, all data items that had been reported previously will be considered and again the top-$k$ processes among them will be chosen.

Looking at our example in Figure 2, based on the height of the boxes, the resampled top-$k$ report would be $v_{51}$, $v_{21}$, $v_{11}$, and $v_{52}$ assuming all $v_{ij}$ are consumed by different processes. As indicated previously, this report may not be perfectly accurate as another process that may not have been relevant within each single report window, may very well have been significant when viewed over a longer time window.

To assess this approach, we are interested in answering two questions. First, given a set of top-$k$ reports to be resampled, what is the largest error that can be encountered for that given report. Second, what is the largest error that can be encountered on any set of reports to be resampled..

Let us answer the first question. The unreported value of a process cannot be larger than $v_{ik}$ and cannot be more than the remaining resource. Mathematically, the largest possible value of an unreported process can be expressed as:

$$\min(v_{ik}, 1 - \sum_j v_{ij}) \tag{1}$$

*Proof.* We prove by contradiction. We assume there is an unreported value that is larger then the result of (1). However, that means that this value is either greater than $v_{ik}$ or $\sum_j v_{ij}$ must be greater then one. Both being a contradiction of our initial definitions from Section III. $\square$

To compute the maximum amount of a resource consumed by an unreported process in a given set of top-$k$ reports $\mathcal{V}$ and to answer our first question, we sum up the largest possible unreported values of a given process for all the reports in the set of reports ($V_i \in \mathcal{V}$):

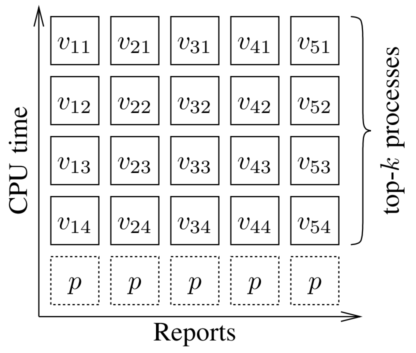
$$\sum_i \min(v_{ik}, 1 - \sum_j v_{ij}) \tag{2}$$

Now, let us move on to find the worst case that can occur independent of a given report. Each report contains the resource consumption values of $k$ processes out of $k + m$ processes consuming the resource. In the worst case each reported process and $l$ unreported processes ($l < m$) consume the same amount of a given resource (unreported processes consuming less resources do not impact the analysis):

$$v_{ij} = 1/(k + l) \tag{3}$$

*Proof.* We prove by contradiction. We assume there must be an unreported process that can consume more of the resource then $1/(k+l)$. We can write this as $1/(k+l)+x, x > 0$. Clearly this process consumes more than any of the other processes, i.e., $1/(k + l) < 1/(k + l) + x$ considering $x > 0$. Hence, the process cannot be unreported which is in contradiction to our initial assumption. $\square$

Since all the processes consume the same amount of resources, the processes to be picked for the top-$k$ processes is random. To further construct the worst case, let us assume that those processes that are reported among the top-$k$ processes are short-lived and only exist for a single reporting period. In contrast, let us assume the non-reported processes are never reported as part of a top-$k$ report. Figure 3 shows such a scenario with $k = 4$, $l = 1$, and $n = 5$ reports to be resampled. As before, the processes with solid borders are reported among the top-$k$ processes whereas the long running process $p$ shown with a dashed border is never reported.



$v_{11}$ $v_{21}$ $v_{31}$ $v_{41}$ $v_{51}$
$v_{12}$ $v_{22}$ $v_{32}$ $v_{42}$ $v_{52}$
$v_{13}$ $v_{23}$ $v_{33}$ $v_{43}$ $v_{53}$
$v_{14}$ $v_{24}$ $v_{34}$ $v_{44}$ $v_{54}$
$p$ $p$ $p$ $p$ $p$
CPU time
Reports
top-$k$ processes

Fig. 3. Worst Case Scenario

Using equations 2 and 3, we can compute the resource consumption of the unreported process $p$ over the entire resampling period ($n = 5$) as $n \cdot 1/(k + l)$ and put this into relation with the other reported values:

$$\frac{n \cdot 1/(k + l)}{1/(k + l)}$$

The result nicely cancels down to $n$, which is also the intuitive answer when looking at Figure 3 and comparing the resource consumption of each reported top-$k$ process with that of $p$. In order to prove this being the worst case, we need to show that no such relation greater than $n$ exists.

Since we constructed the worst-case where $p$ was never reported among the top-$k$, we need to verify that no case where $p$ is reported once among the top-$k$ processes can yield a higher value. However, for that to be valid, this value must be removed when the top-$k$ reports are resampled, hence there must be at least $k$ other processes reported with the same or a higher value than the reported process $p$. Let us prove that no such possibility exists.

*Proof.* Again, we prove by contradiction. If a higher fraction exists, it must fulfill the following equation:

$$\frac{(n-1)/(k+l) + 1/(k+l) + x}{1/(k+l) + x} > n, x > 0$$

This equation describes $n-1$ reports constructed in the above way and one report that has a higher value ($+x$). That means that the top report of the resampled reports also must have a value of at least $1/(k + l) + x$ or otherwise the residual process $p$ would have been reported as part of the resampled top-$k$ reports. Maximizing the above equation gives:

$$\frac{(n-1)/(k+l)}{1/(k+l) + x} + \frac{1/(k+l) + x}{1/(k+l) + x} > n$$
$$\frac{(n-1)/(k+l)}{(1 + x(k+l))/(k+l)} + 1 > n$$
$$1 + \frac{n-1}{1 + x(k+l)} > n$$
$$1 > 1 + x(k+l)$$
$$x(k+l) < 0$$

Considering that $k, l > 0$, $x$ must be negative which violates our initial assumption that $x > 0$. $\square$

In the equations so far we have put the largest unreported process into relation with the largest reported process, hence computing a relative error. It is interesting to note that if we were interested only in absolute values, a higher maximum could be obtained for $k = n$. The following example with $n = 3, l = 1$ would give an unreported process with a consumption of $0.5 + 2 \cdot 0.1\dot{6} = 0.8\dot{3}$ which gives, however, gives only a fraction of $0.8\dot{3}/0.5$, less than that of our previous example which would be $0.75/0.25 = 3$.

$$V_1 : v_{11} = 0.5, v_{12} = v_{13} = 0.1\dot{6}, p = 0.1\dot{6}$$
$$V_2 : v_{21} = 0.5, v_{22} = v_{23} = 0.1\dot{6}, p = 0.1\dot{6}$$
$$V_3 : v_{31} = 0.5, p = 0.5, v_{33} = 0$$

## V. Residual Top-$k$ Reporting and Resampling

As we have shown, the problem with the simple top-$k$ reporting approach is that it may miss processes that consume resources at a low rate over an extended period of time. When resampling a period of $n$ such top-$k$ reports, we might be missing processes consuming $n$ times the resource consumption of the top process. Storing the resource consumption of all processes at any point in time would be a solution, albeit impractical.

To reduce the error of the simple top-$k$ reporting and resampling approach, we suggest to include residual resource consumption data (residual data, for brevity) in top-$k$ reports. In the following we discuss what we consider as residual resource consumption and when such residual data should be included in top-$k$ reports. We show how to resample such top-$k$ reports and the expected margin.

### A. Residual Resource Consumption

With residual resource consumption, we describe the part of a process's resource consumption that has not been reported in any top-$k$ report. For instance, in Figure 3, the residual resource consumption of process $p$ is $1/(k+1), 2/(k+1), \ldots$ when the first, second, ... report is generated. The resource consumption of a process reported as a top-$k$ entity we do not consider to be part of the residual resource consumption.

The residual resource consumption from a given time $t$ of a process can be computed by storing the process's resource consumption $c_t$ at $t$. Any future residual resource consumption at time $t'$ ($t' > t$) would be $r = c_{t'} - c_t$. If at some point the process is part of a top-$k$ report the reported resource consumption $v_{ij}$ must be added to $c_t$ since we do not consider it to be part of the process's residual resource consumption $r$.

The next question is under what circumstances should the residual resource consumption data be reported. We can distinguish two scenarios:

1) The residual resource consumption data pertains to a process that is *not* part of a top-$k$ report.
2) The residual resource consumption data pertains to a process that *is* part of a top-$k$ report.

In the first scenario (Figure 3), we report the residual resource consumption whenever it is greater than the current resource consumption of any of the reported processes in the current top-$k$ report. In our process example, this implies that the accumulated resource consumption of process $p$ would be reported in every other report as additional item.

In the second situation (Figure 4), if we produced a top-2 report for every 10 seconds, we would get the following top-2 reports:

| | |
|---|---|
| 00-10s: P3 37%, P1 20% | 30-40s: P1 20%, P2 19% |
| 10-20s: P1 20%, P2 19% | 40-50s: P1 20%, P2 19% |
| 20-30s: P1 20%, P2 19% | 50-60s: P3 37%, P1 20% |

If we aggregate every 3 reports, we would get the following two reports:

00-30s: P1 20%, P2 $12.\dot{6}\%$
30-60s: P1 20%, P2 $12.\dot{6}\%$

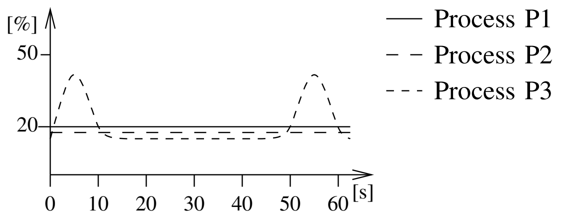


Fig. 4. Periodic Process

P3 would drop out because it has not been reported in between 10 and 50 seconds where its residual resource consumption is constantly below that of P2. Considering that in that time, it consumed approximately 18%, it should have been reported with an average consumption of $24.\dot{3}\%$.

If residual data is only reported when it exceeds the resource consumption of another entity, P3's residual data would not be reported because it never exceeds the resource consumption of P1 or P2. For instance, from 10 to 20 seconds, P1 consumes 2s whereas P3 only consumes 1.8s.[1] Similarly, from 10 to 30 seconds, P1 consumes 4s whereas P3 only consumes 3.6s and hence, would not be reported.

To solve this issue, we took the engineering decision to simply report the residual resource consumption if it is higher then the *currently* reported resource consumption for any of the top-$k$ processes. Looking at Figure 4, this implies that the residual resource consumption of P3 only has to exceed the current resource consumption of P2 which is 1.8s in the top-$k$ reports from 10 to 50 seconds.

This approach has the advantage that we do not have to relate historic resource consumption, thus simplifying residual data which simplifies the implementation of sensors. On the other hand, this approach creates more redundant residual resource consumption reports. For instance, were we interested only in the top-1 process, we would report the residual resource consumption of P2 although it is clear that it can only overtake P1 in the future and hence does not yet have to be reported.

There are different ways to reduce the size of residual resource consumption data. One is to report residual resource consumption data only when it is bigger then a given threshold. Another would be to report residual resource consumption data only if it covers a given duration. The affect on the theoretical error as well as the practical implications we discuss later.

### B. Resampling with Residual Resource Consumption

In this section, we discuss how a resampling algorithm shall take the residual data into account. If $n$ reports are to be resampled, there will be $kn$ top-$k$ items as well as residual data to be resampled into $k$ new items and new residual data. Since residual data may be reported some time after the resources were consumed, we increase accuracy of the algorithm by scanning later reports as well. The number of reports to look ahead is given by the parameter $o$.

[1]When aggregating data, we convert percentages to absolute time because it is easier to aggregate data when facing different time windows.

```
procedure ResidualResample($k$, $n$, $o$)
    $d_n \leftarrow 0$
    for $i \leftarrow 0, (n-1)$ do
        $d_n \leftarrow d_n + \mathrm{t}(V_i)$
        for $v \in V_i$ do
            $a(\mathrm{id}(v)) \leftarrow a(\mathrm{id}(v)) + \mathrm{cons}(v)$
        end for
        for $r \in R_i$ do
            if $\mathrm{t}(r) > d_n$ then
                $x \leftarrow \mathrm{cons}(r) * (\mathrm{t}(r) - d_n)/\mathrm{t}(r)$
                $u(\mathrm{id}(r)) \leftarrow u(\mathrm{id}(r)) + x$
                $\mathrm{cons}(r) \leftarrow \mathrm{cons}(r) - x$
            end if
            $a(\mathrm{id}(r)) \leftarrow a(\mathrm{id}(r)) + \mathrm{cons}(r)$
        end for
    end for
    $d_o \leftarrow 0$
    for $i \leftarrow n, (n+o-1)$ do
        $d_o \leftarrow d_o + \mathrm{t}(V_i)$
        for $r \in R_i$ do
            if $\mathrm{t}(r) > d_n + d_o$ then
                $x \leftarrow \mathrm{cons}(r) * (\mathrm{t}(r) - (d_n + d_o))/\mathrm{t}(r)$
                $u(\mathrm{id}(r)) \leftarrow u(\mathrm{id}(r)) + x$
                $\mathrm{cons}(r) \leftarrow \mathrm{cons}(r) - x$
                $\mathrm{t}(r) \leftarrow d_n + d_o$
            end if
            if $\mathrm{t}(r) > d_o$ then
                $x \leftarrow \mathrm{cons}(r) * (\mathrm{t}(r) - d_o)/\mathrm{t}(r)$
                $a(\mathrm{id}(r)) \leftarrow a(\mathrm{id}(r)) + x$
                $\mathrm{cons}(r) \leftarrow \mathrm{cons}(r) - x$
                $\mathrm{t}(r) \leftarrow d_o$
            end if
        end for
    end for
    ▷ sort $a$ by consumption, extract top-$k$
    ▷ aggregate remaining items with $u$
    ▷ compute residuals based on $u$
end procedure
```

Fig. 5. Residual Top-$k$ Resampling Algorithm

One version of such an algorithm is shown in Figure 5 and the symbols used are explained in Figure 6. As part of the first loop, the algorithm consolidates $n$ top-$k$ reports (lines 2–16). The loop starts by aggregating the existing top-$k$ reports into a map $a$ containing the aggregated resource consumption for the period to be resampled. Additionally, it adds the appropriate fraction of the residual data for the current period. This may extend past the current resampling period despite the fact that we look ahead for residual data. In this case it is added to the unhandled data map $u$. We will discuss the reasons for this later.

As part of the second loop, the algorithm looks ahead $o$ reports for residual data (lines 17–34). Again residual data may extend beyond the past in which case we add it to the unhandled data map $u$. If part of the residual data falls into the current resampling period, that part will be aggregated in $a$ as well.

| Symbol | Description |
|---|---|
| $a(id)$ | aggregated top-$k$ data |
| $\mathrm{cons}(e)$ | consumption of top-$k$ or residual item |
| $\mathrm{id}(e)$ | a unique identifier of top-$k$ or residual item |
| $\mathrm{t}(e)$ | the time over which the report or item collected |
| $u(id)$ | the currently unreported consumption of item id, this relationship is global and maintained across invocations |
| $V_i$ | the set of top-$k$ elements of report $i$ |
| $R_i$ | the set of residual data items of report $i$ |

Fig. 6. Symbols of the Residual Top-$k$ Resampling Algorithm

It may happen that residual data extends past the current reports to be resampled because residual data can reach back farther than the size of our look-ahead window $o$. In this case the data is aggregated as part of the unhandled data map $u$ which will be considered when the top-$k$ data is resampled again. It may be possible to patch this data into already resampled reports but we have not yet pursued this option.

Finally, we take the top-$k$ items from the map $a$ for the resampled top-$k$ report. The other items are added to the unhandled map $u$. The unhandled map contains all values that have not yet been reported as top-$k$ data and hence can be used to compute the new residual resource consumption data. Here the same rules apply as for initial reporting. If the data is bigger and older than a certain threshold, it will be added as residual resource consumption data. The map $u$ needs to be preserved for the resampling of the next window as it contains unaccounted data. Based on the algorithm we can derive the runtime complexity as $O(e{*}\log(e))$ where $e = (k+|R|)(n+o)$ is the number of entities inspected.

Two optimizations can be applied to this algorithm. The current algorithm assumes that each residual resource consumption item also reports resource consumption for the times that the corresponding entity was reported as top-$k$ item. These times can be filtered out by scanning when those items have been reported as top-$k$ data and removing them from the respective parts of the residual resource consumption data.

Second, the algorithm assumes that the residual resource consumption data is accrued at a constant rate. However, based on the top-$k$ data, we know that during a report $i$ (if the corresponding entity was not reported as top-$k$ data), the residual resource consumption data cannot have been higher than $v_{ik}$ or otherwise it would have been reported as a top-$k$ item in that report.

## C. Error Estimation

Let us estimate the maximum error of the algorithm presented in Figure 5. Residual resource consumption data is reported as soon as it exceeds a given threshold. The threshold can be relative such as $fv_{ik}$, an absolute threshold, or a combination thereof. Another parameter is the minimal duration $d$ that the residual data needs to cover before being reported.

Once a process consumes more than a given amount of a resource, its unreported resource consumption will be reported as residual data. This strategy implies that sooner or later every resource consumption is reported and no resources consumed by an entity are lost. However, since residual data is reported as a bulk value we do not know at which rate the entity consumed the corresponding resource. Our algorithm assumes that the resource is consumed at an average constant rate. This is captured by the following trivial formula where $r_i$ is the concrete resource consumption during a report $i$ of a residual data item $r$.

$$\bar{r} = \frac{1}{N} \sum_{i=1}^{N} r_i \tag{4}$$

This may lead to an underestimation of the resource on one side combined with an overestimation on another side. Let us construct the maximum underestimation that can occur for a given residual resource consumption $r$. Ignoring the minimal duration $d$ for the time being, the maximal residual resource consumption that can be reported is $fv_{ik}$, for instance, if $r_{i+1} = r_{i+2} = \ldots = r_{i+f}$. Based on this and equation 4 we can arrive at a value of $\bar{r} = 0$ if this sequence is preceded by a near infinite sequence of $r_1 = r_2 = \ldots = r_i = 0$ because

$$\lim_{N\to\infty} \frac{1}{N}((N-f)0 + fv_{ik}) = 0$$

Next, we need to ensure that all the zeros fall into a different resampling window than the other values which implies $r_{i+1}$ is the start of a new resampling window. Hence, we can construct a case where a value of $fv_{ik}$ is unreported. Given equation 3, $v_{ik}$ may equal $v_{1k}$, the largest unreported value may be $fv_{1k}$. Given the long setup time, the minimal duration $d$ plays no role in this case.

The construction for the maximum overestimation is analog to the above. The maximum unreported sequence of residual data is $r_1 = r_2 = \ldots = r_N, N = \max(d, f)$. Since we want to overestimate a value, we need to include at least one 0 which gives $r_1 = \ldots = r_{N-1}, r_N = 0$. Again, analog to the above, $r_N$ must be in a resampling window different from the other values. This gives us the following estimation for $r_1$:

$$r' = \frac{(N-1)v_{ik} + 0}{N} = \frac{N-1}{N} v_{ik}$$

Unlike for reports 1 to $N-1$, equation 3 does not need to hold for report $N$ because $r_N = 0$. Hence, $v_{N1}$ may theoretically be 0 giving a theoretical relative overestimation of

$$r' = \lim_{v_{N1}\to 0} \frac{\frac{(N-1)}{N} v_{ik}}{v_{N1}} = \infty$$

However, this problem can be fixed with our second optimization. Since we know that at any point in time $r_i < v_{ik}$, this must also hold for $r_N$ and hence the worst case relative overestimation is:

$$\frac{v_{Nk}}{v_{N1}} = 1, \quad \text{i.e.}, 100\%$$

Depending on $f$ and $d$, sequences with multiple zeros can be constructed that may lead to a slightly higher overestimation. Additionally, it is important to point out that each overestimation goes hand in hand with an underestimation and therefore an error that may arise from resampling the data will resolve itself at least partially when the data is again resampled.

The first optimization does not play a role because in the above construction because we assumed that a residual resource consumption entity is not reported among the top-$k$ items. The second optimization does not play a role in the construction of the underestimation because it does not provide a lower bound for values.

So far, we have constructed worst case examples to derive the worst case error. If the values from the worst case examples above are substituted with the concrete values of a given report, they give the worst case error for concrete data. In a user interface, this could be displayed alongside the data to give users an idea of the accuracy of the data collected. For this type of error estimation the above two optimizations, however, would indeed make a difference and would have to be factored into the equations.

### D. Maximum Residual Data Items

In our typical worst case scenario, where all $k+l$ processes consume an equal amount of resources, every $f$, respectively $d$ reports, whichever is greater, $l$ processes would have to be reported. If $l$ is big this might be a considerable overhead. Of course, $f$ and $d$ can be chosen to minimize this overhead. This, however, is a theoretical value and in the next section we will investigate the typical residual data that is generated on a production system.

As a separate consideration, one may also set a limit at which residual data is reported such as the maximum number of residual data items to report per top-$k$ report. However, we leave this discussion for future work.

## VI. Evaluation

Considering that our worst-case error computations require a highly artificial process distribution, we are interested in the error we can expect in a production system. Similarly, we are interested in the number of residual data items that are typically reported per top-$k$ report.

As test systems, we use one of the master nodes of our Hadoop cluster. It exhibits some long running processes consuming CPU time at a regular rate, some periodic processes, as well as some shorter-lived processes, providing an interesting mix for analysis. The trace spans over 35 hours at a 10 second base sampling rate. For the measurement of the CPU time consumed, we use the data from the proc filesystem [10]. Linux assumes that the maximum CPU utilization for an $n$ process system is $n \cdot 100\%$ (for all our test machines: $n = 4$). Since all our error estimations are relative, they apply without loss of generality.

Residual top-$k$ reports were generated with $k = 4$ and residual data was reported when it exceeded a resource usage of $2.5v_{ik}$ ($f = 2.5$) and if the data spanned at least the duration

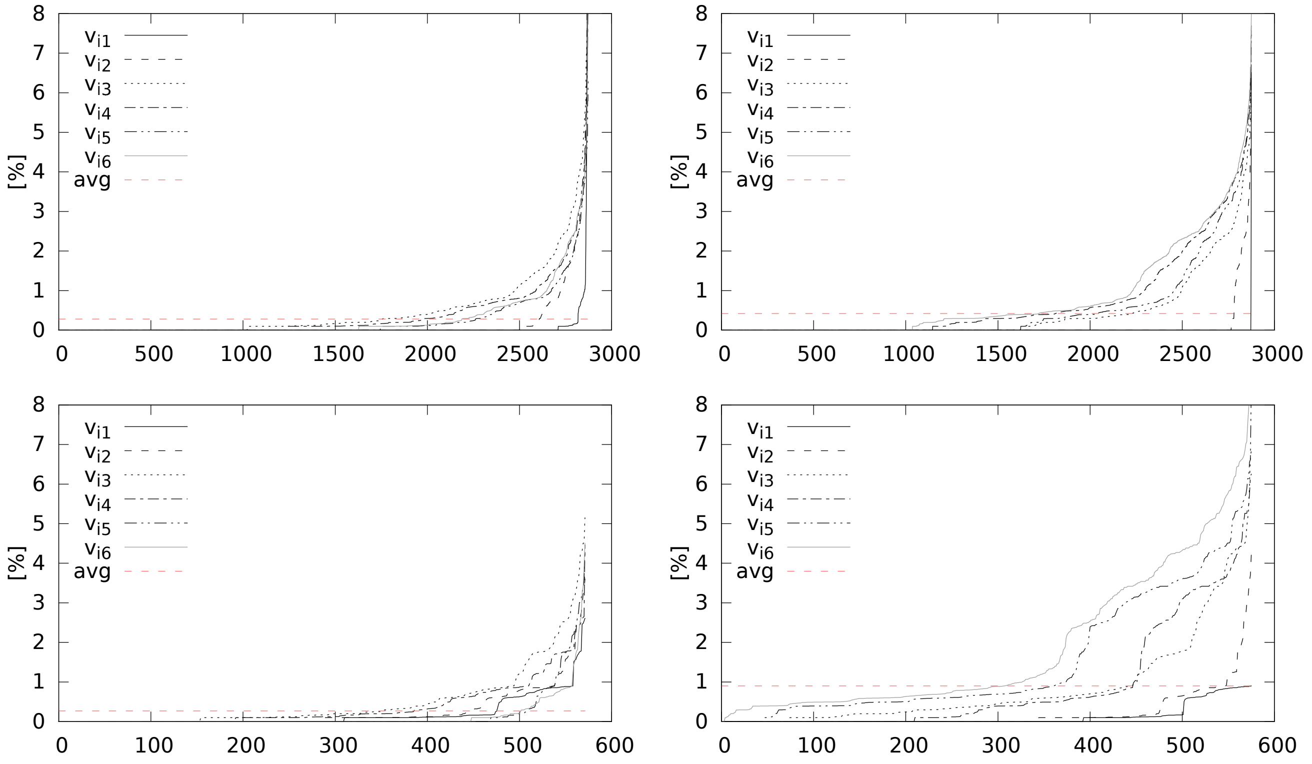


Fig. 7. Residual (left) and Simple (right) Top-$k$ Resampling (**x-axis: entities $v_{ij}$ sorted by ascending error rates**)

of 8 reports. In average this generated $1.72$ residual data items per report. We also decided to include an evaluation of the simple approach from Section IV for which we have chosen to report $k = 6$ data items per report to match the bitrate of the residual top-$k$ reports.

We use the residual resampling algorithm from Figure 5 with a look-ahead window $o$ of 10 reports and the first optimization applied (not appropriating residual data for reports where a corresponding top-$k$ item is present). The resampling rate for the simple and residual resampling algorithm is $n = 5$. The error rates we measured are shown in Figure 7 in the top row. While the residual reports only capture the top-4 items, our algorithm based on the residual data items is still able to compute the top-6 or more data items. To do so, one only has to collect the top-6 items from the $\mathrm{a}$-map in the algorithm. For subsequent resampling, however, only the top-4 items were considered.

The average error of the residual top-$k$ approach is $0.28\%$ and $20.3\%$ of the reports exhibit an element that has been reordered. For the simple approach, it is $0.42\%$ and $38.1\%$ respectively. The plots show the error rates for each of the top-6 items as a separate series. The errors are sorted and the error rate is shown in percent. Interestingly, the residual approach shows about 5 bigger errors, even bigger than the simple approach. These are mostly from smaller top-$k$ values due to the effect we have explained in Section V-C.

The strength of the residual approach becomes apparent when the resampled data needs to be resampled again (lower plots). This also shows in practice that due to the residual data no resource consumption actually gets lost. The average error rates are $0.27\%$ and $0.9\%$ respectively. The number of reports with an incorrectly sorted element are $12.6\%$ and $72.9\%$ respectively.

In conclusion, our evaluation shows that even our simple approach performs surprisingly well. It also confirms that the residual resource consumption data captures all resource consumption which is useful if a historically resampled report again needs to be resampled to match users' resampling needs as the error rate will not increase. On the other hand, we have seen that our concern that there may be spikes in the margin of error, albeit rarely, did manifest itself. Another, for us surprising result, is that our simple resampling approach does work considerably well, as long as it is not applied onto already resampled data.

## VII. Future Work

While our residual top-$k$ approach works well and especially so for repeatedly resampling data, it has a weakness when residual resource consumption of an entity spikes without being in the top-$k$ elements of the report followed by a time of low resource consumption. While this is rare, it did happen a handful of times within 35 hours (entities with the highest error rates in Figure 7). Residual resource consumption data is reported if it spans a minimal duration and its consumption is bigger than a threshold $fv_{ik}$. We believe the decision of when to report this data may be refined to solve this problem.

For a given top-$k$ report, we could compute a tighter error margin then the general analysis in the paper. This computation would give users the ability to estimate the accuracy of each individual top-$k$ report.

Also, sometimes, the top-$k$ items may consist of many similar processes hiding other processes. In this case it would be nice to allow users to group a set of entities into a single entity.

## VIII. Related Work

The top-$k$ problem has been addressed by many researchers. However, we were not able to find any reported work that is directly related to the topic of resampling top-$k$ data.

The work closest to our work is by Mouratidis et al. [7] which handles the top-$k$ problem in the context of data streaming. They study continuous monitoring of top-$k$ queries over a fixed window size, using entities that have multiple but constant attributes that are weighted differently for different top-$k$ queries. In order to handle this weighing they compute a $k$-skyband. Yang et al. [6] also look at the top-$k$ problem in the context of data streaming. They reduce the runtime complexity by using entities with a single constant attribute. Our approach does not use fixed time windows but, based on user queries, top-$k$ data is resampled to different time windows. Additionally, in [7] and [6], the values associated with an entity do not change. In our case, the resource consumption of a process changes over time and therefore is not constant.

Another related approach by Metwally et al. [11] discovers most popular $k$ elements, as well as frequent elements in a possibly infinite data stream, consisting of a large set of different data types. In our approach, the number of elements in a top-$k$ report are bounded by the resources reported by a machine, or in the worst case, a cluster. The challenge is not the discovery of the overall top-$k$ data, but being able to resample the collected top-$k$ data to be able to resample the collected top-$k$ data to different, potentially user-requested, sampling intervals.

Count-min sketch [12] can efficiently estimate frequencies, specially when new objects are being added. Removing elements is harder but is handled in [13] using time-decaying sketch. Neither approach deals with resampling of the collected data.

More distantly related work is presented by Yi et al. [4], where top-$k$ database queries are optimized in the presence of table updates. Hristidis at al. [14] optimize the top-$k$ problem for differing scoring functions $f$ by pre-computing a set of pre-materialized ranked views, and based on the function $f$, selecting a view that answer the query most efficiently. Ilyas et al. [15], [16] explore the top-$k$ problem in the context of databases to rank search results from multiple search engines and answering multi-feature queries. The challenge here is to return the results to the user in a combined relevance order.

## IX. Conclusions

We have presented a new approach that allows the aggregation of fixed time period top-$k$ reports into arbitrary time periods with minimal error. This enables performance analysis using historical data, such as determining trends in CPU usage, with greater accuracy. Our approach augments top-$k$ reports with residual resource consumption data, thereby capturing besides the top-$k$ entities, data on background, long running entities. We have shown that this data is instrumental if accurate top-$k$ reports need to be generated for time periods that are longer than the original collection period. Our computations of the algorithmic bounds show that this novel approach provides minimal error rate while allowing flexible resampling of top-$k$ reports. We also have both proved formally as as well as through an evaluation of the algorithm in an operational system to confirm our claim.

## References

[1] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(5–6):817–840, 2004.

[2] Lior Abraham, John Allen, Oleksandr Barykin, Vinayak Borkar, Bhuwan Chopra, Ciprian Gerea, Daniel Merl, Josh Metzler, David Reiss, Subbu Subramanian, Janet Wiener, and Okay Zed. Scuba: Diving into data at facebook. *Proceedings of the VLDB Endowment*, 6(11):1057–1067, 2013.

[3] Daniel Bauer and Metin Feridun. A scalable lightweight performance monitoring tool for storage clusters. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 1008–1013. IEEE, 2015.

[4] Ke Yi, Hai Yu, Jun Yang, Gangqiang Xia, and Yuguo Chen. Efficient maintenance of materialized top-k views. In *Proceedings of the 19th International Conference on Data Engineering*, pages 189–200. IEEE, 2003.

[5] Vagelis Hristidis and Yannis Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *The International Journal on Very Large Data Bases*, 13(1):49–70, January 2004.

[6] Di Yang, Avani Shastri, Elke A. Rundensteiner, and Matthew O. Ward. An optimal strategy for monitoring top-k queries in streaming windows. In *Proceedings of the 14th International Conference on Existsing Database Technology*, pages 57–68. ACM, 2011.

[7] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 635–646. ACM, 2006.

[8] IBM. An introduction to IBM Spectrum Scale. Technical report, IBM, February 2015.

[9] Tim Roughgarden and Gregory Valiant. Lecture #2: Approximate heavy hitters and the count-min sketch. In *CS168: The Modern Algorithmic Toolbox*. Stanford University, 2015.

[10] Terrehon Bowden, Bodo Bauer, Jorge Nerin, and Shen Feng. *The /proc Filesystem*, 2009.

[11] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent top-$k$ elements in data streams. In *Proceedings of the 10th International Conference on Data Theory (ICDT05)*, pages 398–412, 2005.

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

[13] Graham Cormode, Srikanta Tirthapura, and Bojian Xu. Time-decaying sketches for robust aggregation of sensor data. *SIAM Journal on Computing*, 39(4):1309–1339, October 2009.

[14] Vagelis Hristidis and Yannis Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *The VLDB Journal*, 13(1):49–70, January 2004.

[15] Ihab F. Ilyas, Walid G. Aref, and Ahmed K.Elmagarmid. Joining ranked inputs in practice. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 950–961, August 2002.

[16] Ihab F. Ilyas, Rahul Shah, Walid G. Aref, Jeffrey Scott Vitter, and Ahmed K. Elmargarmid. Rank aware query optimization. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 203–214. ACM, June 2004.