SRATE: A Fast and Memory Efficient Request Rate Estimation by State-Rank based Scheme

Umega Kaul
LNCT ,Bhopal (M.P),India

Suneel Phulere
LNCT, Bhopal(M.P),India

Vineet Richariya
LNCT,Bhopal (M.P),India

ABSTRACT

In Cloud Computing or Content Distributed Cloud (CDN) workload happens between a local web server and proxy servers. The typical method is DNS redirecting and the workload factoring decision is predefined manually over a set of most popular objects. For such popular object detection in requests tails, many schemes have proposed for fast request rate estimation in traffic monitoring and fast counting for most popular data item for which request arrived and above mentioned jobs done. Accurate request rate estimation is necessary for resource planning & management, measuring compliance to SLAs and cloud security especially in Cloud Computing where we are talking about Integrated, Heterogeneous or Hybrid Cloud deployment model. With following ideals, we propose SRATE (State-Rank bAsed Traffic Estimation) has sufficient short estimation times with provable bounds on estimation error, low memory usage and also easily implementable in hardware for operation at high speeds. With developing such scheme, we achieve up to three orders of magnitude speedup in estimation time. The speedups are achieved with low memory usage by using "*State-Rank (0,1,ϕ)*" instead of runs or coincidences. This new scheme has many benefits at high oppressive workload including quicker detection of unhandled oppressive data items or spikes with incipient denial of service attacks. As result, we show that the proposed scheme is faster and more accurate than other schemes. We also prove bounds on the scheme's accuracy, request size or estimation time, average request rate estimation, memory needs, average request sending rate and also show that it performs well by efficient simulation techniques.

1. INTRODUCTION

In this paper, we address the problem of accurate measurement of traffic in a Cloud Computing Cloud. Measurement of traffic is an important component for resource planning & management, workload factoring, detecting DoS attacks, and in traffic engineering [2], [3], [6]. The traffic in the cloud can typically be classified into request and measurements are required on a per-request basis for particular service requirement. The definition of request can be very flexible. Examples are specific application to application traffic characterized by 'State-Rank', all traffic destined toward destination cloud. The standard approach used for measuring traffic is to sample the traffic arriving at the server, keep counts of the traffic arrivals on a per-request basis and then use this Counter & State-Rank to estimate the traffic. The main problem with this approach is scalability. If the number of requests is large, then keeping per-request counts consumes considerable memory as well as processing power. Now for active tracery, we used the Yahoo! Video. Yahoo! Video (India) [1] is 3rd ranked online video website just after YouTube and MetaCafe in terms of total number of video views, uploading & downloading during January 2011, it delivered totally 30.2 millions of video streams to 17.56 million of unique Indian viewers [18]. It is 72% of the online population of India who take services of video streaming.

In fact, measurements have shown that there are a large number of requests in the cloud, a significant fraction of the traffic are carried by only a small number of requests. These oppressive objects, which may only be a few hundreds of request, can constitute as much as 85-90 % of the traffic at a server. Therefore, detecting and measuring these oppressive objects are an important aspect of traffic measurement. Further, measuring sudden increase in activity towards a given destination can be a sign of a denial of service attack. Therefore, monitoring traffic is a very important component in cloud server. Another application area where per-request measurements are needed is for active queue management [8] for providing fairness in clouds. The main idea is to isolate large request to reduce their impact on the rest of the request in the cloud. This is especially important if the large requests are misbehaving short round-trip delays. However, to track these small numbers of misbehaving sources, we have to wade through tens of thousands (if not hundreds of thousands) of small sources likely we should not have to track millions of ants to track a few elephants.

So we exploit the fact that there are a few oppressive objects to reduce the amount of memory required to measure heavy sources. The basic idea is to sample data items with some probability and if the request to which the data item belongs is not already in memory, and then the request is added to the memory, and from that point on, all data items arriving to this request are counted and attested as State-Rank. Since every data item is counted, the sampled request are kept in a CSRT (hash table) and at every data item arrival, its Data-ID has to be attested into this CSRT in order to increment and attests the appropriate counter and State-Rank respectively if it is already in the table. However, since the size of the memory is reduced, they show that this scheme is easier to implement. They also give a more processing intensive multistage filter scheme to track large request. As we will show in this paper, this is a powerful technique of elastic nature for estimating traffic rates especially when the number of request is large. We can extend our approach according to required goodness of Request Estimation Scheme

- *"It should extremely estimate request rate to specified accuracy as well as proved their correctness above theoretical prediction"*. This implies that for a specified accuracy. The shorter sampling time is better for every scheme. Extreme estimation is critical for fast detection of anomalous events.
- *"The scheme should be suitable for run time processing of traffic streams of requested data items"*. This implies that the scheme performed should be simple or preferably amenable to cover each step of implementations.
- *"The scheme should be memory efficient practically"*. System may have millions of data items and it should not be necessary to maintain system according to state for large fraction of the data item for a few estimation results.

The rest of the paper is organized as follows: A related prior work defined is in Section-II. Section-III outlines SRATE scheme and describes the implementation Architecture. Section-IV presents the analysis of SRATE. The experimental results of SRATE along with extensive simulation are described in Section-V. Lastly, Section-VI concludes the paper.

2. RELATED WORKS

There has been much recent research on request measurement. The work of [10] presented a sampling method that first selects the oppressive objects, i.e., requests with rate above a certain threshold (say, 1% of the entire traffic), and then counts all data items belonging to these oppressive objects. For this scheme, deriving the number of samples needed to achieve specified estimation accuracy does not appear to be easy.

A white paper [11] proposed RATE, a flow estimation mechanism based on counting two-runs (flow id matches for two consecutive samples from the traffic stream). For a given accuracy level, RATE requires worst-case sampling time slightly longer than (1.38 times) the naive counting scheme. However, it uses significantly less memory (square root of number of samples). A drawback is that for the same sampling time, RATE has much worse accuracy than naive counting for flows with low rates. ACCEL-RATE [4] was proposed as an enhancement to the original RATE scheme. In ACCEL-RATE, each arrival packet is hashed into multiple buckets based on its flow id, so that packets of the same flow are more “concentrated" within each bucket. As a result, a larger number of two-run samples can be generated for each flow. It is shown that under uniform hashing the sampling time, in comparison to RATE, can be reduced to 7:3 k where k is the number of buckets. This is achieved with about 2/7 times more memory needs than RATE. Note that, however, such sampling time reduction only holds when hashing is uniform, which implies the rate of the largest request, should be significantly less than 1 k. Hence ACCEL-RATE is best suited for cases where the maximum flow rate is small, and a loose upper bound is known a priori. In general, all the proposed mechanisms tend to trade reduced estimation accuracy and/or increased sampling time for lower memory cost. For a given sampling time, naïve counting scheme can still produce the most accurate results regardless of flow rate but at the expense of much higher memory requirements. Other side, CATE [7] extends RATE with using the width of the coincidence interval of the naive counting scheme as the benchmark to determine the effectiveness of the sampling schemes developed. FastTop-K [15] is the first paper which contributes request patterns instead of flow mechanism with fully inspiration of CATE. Follow same ideals of FastTop-K, here we use ‘State-Rank’ terminology to access rate-estimation as well as efficient rate-concentration with memory usage also.

3. STATE-RANK BASED TRAFFIC ESTIMATION

In this paper, we propose a new traffic estimation mechanism called SRATE: State-Rank bAsed Traffic Estimation. The new scheme works by keeping registers for k previous arrivals and comparing the new arrival with each of them with attesting ‘State-Rank’ according to system state these are ‘Regular and Critical’. Matching between the new arrived request and one of the previous arrived requests in registers are updated their Count and State-Rank with follow system state. Now, we process algorithm, likely other estimation mechanism, SRATE maintains two tables-

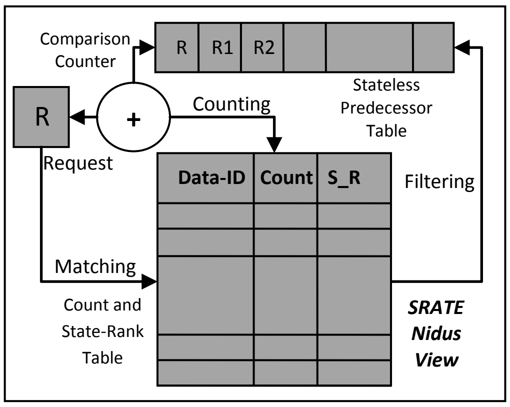

Fig.1. Descriptions of SRATE Nidus View approach

- *Stateless Predecessor Table (SPT)* maintains the first request which have no identical states since pick in CSRT to the system. It is FIFO based primary table & filled directly or by filtering as form of system buffer.
- *Count & State-Rank Table (CSRT)* where each arrived request into the system, through matching if the incoming request f of data item F, which already in CSRT then increase their count by 1 and update state-rank according to system state otherwise apply filtering approach. If match is done in FIFO then Nidus counts the request in CSRT according to algorithm.

As shown in fig.1, Given a request R, the algorithm outputs are “Regular State processing” if r will go to the Regular State with state-rank ‘0’ otherwise “Critical State Processing” if it has state-rank ‘1’ otherwise disposed from this approach. It works as following:

- If the system is in Regular State, the CSRT (Count & State-Rank Table) is always set as Empty when first request R is arrived.
- If R matches any data item of CSRT (for asking same data) increase the counter of data item by ‘1’ in counter column and update state-rank ‘0’.
- If system is setting as Regular State after going Critical State then on basis of MWL (Maximal Workload Left) [19], request is processed by Local Web Server.
- Otherwise, randomly draw M requests from FIFO queue named as SPT (Stateless Predecessor Table) & compare them with R, if R matches any of the M request (for asking same data), pick that data item & put in CSRT with initialization of counter as ‘1’ and state-rank ‘0’ & update CSRT.
- If any request doesn’t arrived again then system automatically dispose request from SPT, because there is no mean of that request to keep continue in SPT.
- In Critical State, reset all counters to ‘0’ and set state-rank ‘1’ and calculating the request rate of each data item participated in CSRT. For each data item in CSRT, the request rate is its counter value divided by total requests arrives since entering the system in Critical State. Also, calculate estimated request rate correspond of each data item according to Section-V.
- If requests of data item cross or overhead the “Empirical Threshold Indicator” as well as follows “CN-46 (Controversial NonEntityship-46)” means workload so high not managed by system then request will be signified with state rank ‘ϕ’ since disposing it automatically. Otherwise redirect to proxy server on basis of LWL (Least Workload Left) [19] as prerequisite

Workload Factoring scheme with increment count by ‘1’ and State-Rank ‘1’.

- At starting, if any data item requested at Critical State then it follow same procedure as when it is entered in Regular State.
- If data item of request R doesn’t belong to CSRT then add R into SPT FIFO queue for request logs & returns.

4. ANALYSIS

In this section, we present the performance analysis results of the SRATE for the workload factoring .

A. *Notation and Accuracy Requirement*

We assume that each request belongs one of F data item. The rate of request f denoted by r_f and let $\lambda = \sum_{f \in F} r_f$ denote total request rate to server. Let $P_f = \frac{r_f}{\lambda}$ denote the proportion of request rate or actual request rate to the server that belongs to request $f \in F$. We have to design an efficient scheme to estimate r_f for each $f \in F$. Since it is easy to measure λ, instead of directly estimating r_f, we solve the equivalent problem of getting an estimate $\widehat{P_f}$ of P_f for each $f \in F$. Then we use $\lambda\widehat{P_f}$ to estimate r_f. We can view P_f as the probability that an arriving requests belong to data item F. We assume that P_f is static or stationary over the time in which the estimation is done. We also assume that the probability that an arriving request belongs to a given data item is independent of all other requests. We can sample randomly in order to reduce this dependence. We now give the accuracy requirement for SRATE will determine an estimated $\widehat{P_f}$ such that-

$$\widehat{P_f} \in \left(P_f - \frac{\beta}{2}, P_f + \frac{\beta}{2}\right) \; if \; P_f \leq \Delta$$
$$\widehat{P_f} \in \left(P_f - \frac{\theta\beta}{2}, P_f + \frac{\theta\beta}{2}\right) \; if \; P_f > \Delta$$

With probability greater than α. In other words, we are willing to tolerate an error of β with probability less than α for all $P_f \leq \Delta$ and an error of $\theta\beta$ with probability less than α for all $P_f > \Delta$. We consider that the proportion for most data item lies below some threshold proportion Δ and we want the estimation to be accurate in the range $[0,\Delta]$. Formally, we are given threshold proportion $0 \leq \Delta \leq 1$ and parameter $\theta \geq 1$. A case, if there are data items with proportion greater than Δ, we still want the estimation to have a guaranteed performance but we are willing to sacrifice the quality of guarantee somewhat. The SRATE will estimate P_f with relative estimation error range of $\left[\frac{1}{2}, 1\right]$%. We use Z_α to denote the α percentile for unit normal distribution, such as if $\alpha = 99.9\%$ then $Z_\alpha = 4.00$.

B. *Main Result* As per follow processes of [7] with [9].

***Lemma*-1**: Given the modified accuracy requirement, let N be the number of request samples required by SRATE. If the accuracy requirement for large data item can be relaxed to-

$$\theta \geq \sqrt{\frac{\left(\frac{1}{\Delta}+1\right)^2}{3\left(\frac{2}{\Delta}+1\right)}} \qquad (1)$$

Then setting, $k = \frac{1}{2}\left(\frac{1}{\Delta} + 1\right)$

Proof: Follow Theorem-1 of [7].

C. *Correlations*

The analysis of SRATE is significantly more complicated than RATE, CATE and Fast Top-K. This is because that the different comparison in SRATE is not independent. Therefore we need to account for the covariance between different comparisons in order to accurately compute the variance of estimation of proportions. In this section we concentrate on data item and assume that the Stateless Predecessor Table (SPT) maintains all k-requests to the server. Therefore when there are total N requests to the server. We would have made Nk comparisons. We make less than k comparisons for the first k request arrival since k is small compared to N; we ignore this in the rest of the analysis. We assume that requests to server are approached as iid [16] where the probability that request belongs to data item f is given by P_f. We label the request arrival 1 to N based on request arrival’s sequence. Let-

$$C_{ijs}(f) = \begin{cases} 1 \; if \; i, j, s \in f \\ 0, otherwise \end{cases}$$

Let $M(N,f)$ denote the number of requests for data item f after N requests. Therefore-

$$M(N,f) = \sum_{i \leq N} \sum_{j=i-k}^{i-1} C_{ijs}(f) \qquad (2)$$

Before study the correlation structure of the comparisons, we state the following elementary result first-

***Lemma*-2**: Let $C_{ijs}(f)$ be defined above. Then-

$$E\left[C_{ijs}(f)\right] = P_f^3$$
$$VAR\left[C_{ijs}(f)\right] = P_f^3(1 - P_f^3)$$

Proof: this result follows directly from the assumption that request are independent and probability that an arrival belongs to data item f is P_f. But in SRATE, the comparisons are not always independent of each other. Let use the comparison $C_{ijs}(f)$ and $C_{imr}(f)$ where $(i \neq j \neq m) \& (r \neq s)$ as an example. Note that $P\left[C_{ijs}(f) = 1\right] = P_f^3$ due to independency of requests. But $P\left[\{C_{imr}(f) = 1 | C_{ijs}(f) = 1\}\right] = P_f^2, (r \neq s)$ because the condition is already implies that request $i \in f$. In general for any pair of comparison $C_{ijs}(f)$ and $C_{imr}(f)$ are independent if and only if all the indices are distinct. If any two of the indices are identical, then the comparisons are dependent. For example, $C_{ijs}(f)$ and $C_{imr}(f)$ are dependent. The next result gives the correlation between the random variables $C_{ijs}(f)$ and $C_{imr}(f)$. ■

***Lemma*-3**: Consider $C_{ijs}(f)$ and $C_{imr}(f)$ for $i - k \leq j, m \leq i - 1$ and $r \neq s$. Then-

$$COV\left[C_{ijs}(f), C_{imr}(f)\right] = P_f^4(1 - P_f^2) \qquad (3)$$

Proof: Let $\tau = COV\left(C_{ijs}(f), C_{imr}(f)\right)$

$$\tau = E\left[\{C_{ijs}(f) - E[C_{imr}(f)]\}\{C_{imr}(f) - E[C_{imr}(f)]\}\right]$$
$$\tau = E\left[\{C_{ijs}(f) - P_f^3\}\{C_{imr}(f) - P_f^3\}\right]$$
$$\tau = E[C_{ijs}(f)C_{imr}(f) - P_f^3 C_{ijs}(f) - P_f^3 C_{imr}(f) + P_f^6]$$
$$\tau = E\left[C_{ijs}(f)C_{imr}(f)\right] - 2P_f^3 E\left[C_{ijs}(f)\right] + P_f^6$$
$$\tau = E\left[C_{ijs}(f)C_{imr}(f)\right] - 2P_f^6 + P_f^6$$
$$\tau = P_f^4 - P_f^6 =$$
$$P_f^4\left(1 - P_f^2\right)$$ ■

Where the third equality follow from the fact that $E[C_{ijs}(f)] = E[C_{imr}(f)]$. At last step follows from the fact that $C_{ijs}(f)$ and $C_{imr}(f)$ are both one if and only if request arrivals i, j, m all belongs to the data item f and $r = s$, which happen probability P_f^4. In fact it is easy to show that the covariance is $P_f^4(1 - P_f^2)$ for any two comparisons that are correlated. We can now derive the Mean and Variance for the request of data item f.

D. *Expectation and Variance of $M(N,f)$*

***Lemma*-4**: Let $M(N,f)$ denote the number of requests for data item f after N request arrivals to the server. Then-

$$E[M(N,f)] = NkP_f^3$$
$$VAR[M(N,f)] = NkP_f^4(1 - P_f^2)\left[1 + \frac{2(2k-1)P_f}{1 + P_f}\right]$$

Proof: Note that-

$$E[M(N,f)] = E\left[\sum_{i\le N}\sum_{j=i-k}^{i-1} C_{ijs}(f)\right]$$
$$E[M(N,f)] = NkP_f^3$$

(4) ■

To simplify the notation we assume that we index the comparisons using a single index m where $I_m(f)$ is set to one if comparison m result in a requests for data item f. The variance can be computed as follows-

$$VAR[M(N,f)] = E[M^2(N,f)] - \{E[M(N,f)]\}^2$$
$$\because E[M(N,f)] = NkP_f^3$$

Now,

$$E[M^2(N,f)] = E\left[\sum_{i=1}^{Nk} I_{is}^2(f) + \sum_{i=1}^{Nk}\sum_{i=1;1\le j\le Nk;j\ne i} I_{is}(f)I_{js}(f)\right]$$
$$\because E\left[\sum_{i=1}^{Nk}\{I_i^2(f)\}\right] = NkP_f^4$$

Now,

$$E\left[\sum_{i=1;1\le j\le Nk;j\ne i} I_{is}(f)I_{js}(f)\right] = E\left[\sum_{i=1}^{Nk}\{\sum_{j;1\le j\le Nk;j\ne i}^{COV[I_{is}(f)I_{js}(f)]} I_{is}(f)I_{js}(f) + \sum_{l;1\le l\le Nk;l\ne i}^{COV[I_{is}(f)I_{ls}(f)]} I_{is}(f)I_{ls}(f)\}\right]$$

Now, Variance becomes-

$$\therefore VAR[M(N,f)] = NkP_f^4 + \sum_{i=1}^{Nk}[2(2k-1)P_f^5 + \{Nk - 1 - 2(2k-1)\}P_f^6] - (NkP_f^3)^2$$
$$\therefore VAR[M(N,f)] = NkP_f^4(1 - P_f^2)[1 + \tfrac{2(2k-1)P_f}{1+P_f}]$$

(5) ■

Notice that $NkP_f^4(1 - P_f^2)$ is the variance of $M(N,f)$ when all request samples are independent from each other, therefore the correlations among request samples in SRATE increase the variance of $M(N,f)$ by factor of $\frac{2(2k-1)P_f}{1+P_f}$. Since we know the mean and variance of the number of requests are matched. We now use the "Central Limit Theorem" to obtain a normal approximation for the number of request match than use the result to estimate the proportion. The next theorem gives the expression for estimator of the proportion along with its variance.

***Lemma*-5**: Let $M(N,f)$ represents the number of requests for data item f after N request arrivals for SRATE with k comparisons for each arrival. Then-

$$\sqrt{Nk}\left[\sqrt{\frac{M(N,k)}{Nk}} - P_f\right] \sim N[0, \sigma_f^2]$$

Where, $\sigma_f^2 = \frac{(1-P_f^2)\left\{1+\frac{2(2k-1)P_f}{1+P_f}\right\}}{9}$

Proof: Though the comparisons are not independent, the comparisons are a stationary k^2 dependent sequence with finite exception and variance. The following the central limit theorem for dependent sequence. We can show that for large N-

$$\sqrt{Nk}\left[\sqrt{\frac{M(N,k)}{Nk}} - P_f^3\right] \sim N[0, \delta_f^2]$$

Where, $\delta_f^2 = P_f^4(1 - P_f^2)\left\{1 + \frac{2(2k-1)P_f}{1+P_f}\right\}$

Therefore the point estimates for $\hat{P}_f$ of P_f is-

$$\hat{P}_f = \sqrt{\frac{M(N,k)}{Nk}}$$

Then variance of estimation of P_f-

$$\hat{\sigma}_f^2 = \frac{(1-\hat{P}_f^2)\left\{1+\frac{2(2k-1)\hat{P}_f}{1+\hat{P}_f}\right\}}{9}$$

(6) ■

We know this expression for variance of the estimator and derive upper bounds on its value. This bound is derived in two regions. The first upper bound on the variance hold in the entire [0,1] range and is a function of k. The second bound on the variance is a constant independent of k and holds when the proportion is below the threshold.

***Lemma*-6**: Let $\sigma_f^2 = \frac{(1-\hat{P}_f^2)\left\{1+\frac{2(2k-1)\hat{P}_f}{1+\hat{P}_f}\right\}}{9}$ then-

$$\sigma_f^2 \le \frac{k^2}{4k-1}$$
$$\sigma_f^2 \le 0.33 \;\; if \; P_f < \frac{1}{2k-1}$$

Proof: Set the derivative of the variance with respect to P_f to zero gives us the first upper bound. When $P_f < \frac{1}{2k-1}$,

$$\sigma_f^2 \le \frac{3 - 2P_f - P_f^2}{9}$$

This variance takes on a maximum value of $\frac{1}{3}$ when $P_f = 0$. The above bounds on the variance can now be used to compute the sample size and estimation accuracy of SRATE. Note the setting $k = 1$ in variance of estimation, gives-

$$\because \hat{\sigma}_f^2 = \frac{\left(1 + 2P_f - 3P_f^2\right)}{9} \leq \frac{1}{3}$$

$$\therefore \hat{\sigma}_f^2 = \frac{\left(1 + 2P_f - 3P_f^2\right)}{3} \leq 1$$

E. *Minimum Sample Size*

$(2k-1)P_f \leq 1$, let β be the desired estimation accuracy and Z_α the desired α percentile. For any data item f with $(2k-1)P_f \leq 1$, since based on Lemma 6 its variance takes on a maximum value of $\frac{1}{3}$, the minimum request sample size N in order to satisfy the accuracy requirement is given by-

$$N = \frac{3Z_\alpha^2}{k\beta^2} \quad (7)$$

F. *Speedup*

***Lemma*-7**: Given the accuracy requirement described in section-IV and N be the number of request sample required for SRATE-

$$N_{CSRT,Matching} = \frac{3Z_\alpha^2}{k\beta^2}$$

Let us define the request rate amplification factor X for the rate change of data item before and after the Matching as-

$$\chi = \frac{R_{after}}{R_{before}} \quad (8)$$

For example if a data item takes 0.1% of the total requests and takes 0.2% of the total request matched with CSRT data items, the rate amplification factor for $\chi = 3$ for this data item. Then we can have the speedup results of the SRATE algorithm with rate amplification factor X.

***Lemma*-8**: Given the accuracy requirement described in Section-IV. $N_{CSRT,Matching}$ be the number of request samples required for SRATE-CSRT and $N_{SPT,Filtering}$ be the number of request samples required for SRATE-SPT(Filtering).

$$N_{SPT,Filtering} = \frac{N_{CSRT,Matching}}{\chi^2} \quad (9)$$

$$N_{SPT,Filtering} = \frac{N_{CSRT,Matching} - N_{To\ Dispose}}{\chi^3} \quad (10)$$

As well as $N_{To\ Disposing}$ be the number of request samples required for Disposing. Therefore we have X^3 speedup of the detection (Matching, Filtering, and Disposing) process even with "$X - factor$" on rate amplification factor due to historical information filtering.

G. *Request Concentration & cumulative distribution*

We define request concentration, denoted P, as the ratio of request belong to this data item to the total number of requests observed for a fixed time interval. If we consider the set of all data items observed within a time interval, then the distribution of request concentration can be characterized by either a frequency distribution function or a cumulative distribution function. For a given request concentration P, the value of $f(P)$ is defined as the ratio of the number of requests with request concentration equal to P, to the total number of requests observed. The request concentration cumulative distribution $f(P)$ is defined as the ratio of the number of requests that have request concentration less than or equal to P, to the total number of request. That is,

$$f(P) = \sum_{m \leq P} f(m) \quad (11)$$

It should be noted that the request concentration distribution are independent. Request concentration describes the nature of the distribution of data items in different requests observed at a particular observation point.

H. *Average Request Sending Rate*

There are only two possible ways for a request to have a high request concentration: either it lasts a long time, or it sends at a fast rate. We suspect it is the combination of both factors that determines request concentration. We calculate the average request concentration and the average duration of requests for the top N percent of the requests (suppose N ranging from 0 to 100). We plot the distribution of average request concentration, in request trails, and average duration of requests against the cumulative distribution of requests, and show the graphs in Fig.6. We observe that requests with high concentration of request trails tend to last long as well. Suppose server wants to apply some QoS mechanisms to a limited set of requests, and the ideal choices of such requests are those that have the largest request concentration by the time the requests terminate. At any instant in time, however, server has no knowledge of how long an existing request is going to last. The only heuristic server can use is a request's past sending rate. If, for example, a high past sending rate is likely to predict that a request is going to be a highly concentrated request by the time the request terminates, then it is a safe bet for server to start specially treating those particular requests. Therefore, we study the correlation between the concentration of requests and their average sending rate. The average sending rate of any particular request is calculated using a low pass filter with a weight ω of 0.5, i.e.

$$AvgRate = \omega * CurrentRate + \{1 - \omega\}AvgRate$$

I. *Memory Requirement*

Now we focus on the memory requirement for SRATE that's nearly same as CATE. We calculate the worst case expected memory requirement due to dependencies of comparisons. Given the request sample size N, we now want to determine the expected amount of memory required in worst case. In other words, we want to find an allocation of P_f that maximizes the expected number of data item in CSRT-

***Lemma*-9**: Given the specified accuracy requirement described in Lemma 1, the maximum expected memory [11], [7] is-

$$at\ Primary\ web\ Server, E[L(N)] \leq 1.174\sqrt[3]{\frac{Z_\alpha^2}{\beta^2}}$$

$$at\ Proxy\ Server, E[L(N)] \leq 1.174\sqrt[3]{\frac{Z_\alpha^2}{\beta^2}} + \frac{1}{3}\left[Z_\alpha^2\sqrt{k}\right]$$

$$\approx 1.174\left[\frac{Z_\alpha^2}{\beta^2}\right]^{2/3} + \frac{1}{3}\left[Z_\alpha^2\sqrt{k}\right]$$

Proof: from theorem 8 and 9 of [11] the objective function is maximized when

$$P_i = \frac{1}{n}\forall i$$

Therefore a fixed value of n, the maximum expected list length through objective function is-

$$n\left(1 - exp\left(-\frac{Nk}{n^3}\right)\right)$$

We now want to determine the value of n for which this function is maximized. Assuming that n is continous, and differentiating this expression with respect to n. We get-

$$1 - exp\left(-\frac{Nk}{n^3}\right)\left(\frac{3Nk}{n^3}+1\right) = 0$$

If we set $\frac{Nk}{n^3} = \omega$ we can rewrite the expression with solving for ω we get $\omega = 1.903$. Therefore,

$$1 - e^{-\omega}(3\omega + 1) = 0$$

$$n = \sqrt[3]{\frac{Nk}{1.903}} = 0.81\sqrt[3]{Nk}$$

Setting this value of n in objective function, the maximum expected memory requirement is given by-

$$E[L(N)] \leq 0.81\sqrt[3]{Nk}\left[1 - e^{-\left(\frac{Nk}{(0.81\sqrt[3]{Nk})^3}\right)}\right]$$

$$E[L(N)] \leq 0.687\sqrt[3]{Nk}$$

Corollary-1 at Local Web Server: given the width of the confidence interval α and the error probability β, the expected memory-

$$E[L(N)] = 0.687\sqrt[3]{\frac{3Z_\alpha^2}{k\beta^2}k}$$

$$E[L(N)] = 1.174\left(\frac{Z_\alpha}{\beta}\right)^{2/3} = 1.174\sqrt[3]{\gamma} \quad (12)$$

This expression is particularly for local web server. But after performing analytical simulation & trace memory usage we found that the memory size at proxy server has been increased which at least equivalent to $\frac{1}{3}[Z_\alpha^2\sqrt{k}]$ named as Memory-Prediction-Error 'ε'.

Reason behind the naming error is that we can't predict about memory usage at proxy server as well as perform estimation for actual memory used accurately.

Corollary-II at Proxy Server: given the width of the confidence interval α and the error probability β, the expected memory-

$$E[L(N)] = 0.687\sqrt[3]{\frac{3Z_\alpha^2}{k\beta^2}k} + \frac{1}{3}[Z_\alpha^2\sqrt{k}]$$

If, $\gamma = \sqrt[3]{\frac{Z_\alpha^2}{\beta^2}}$ *then*

$$E[L(N)] = 1.174\sqrt[3]{\gamma} + \frac{1}{3}[Z_\alpha^2\sqrt{k}]$$

(13) ∎

5. PERFORMANCE CRITERIA

We conduct two sets of experiments. We first use observation tracery to estimation of all metrics, and then use 'Tri-Way Trajectory Simulation' traces counting of request arrived, sample size, memory usage, request concentration with average request sending rate with efficient accuracy orders and effective performance criteria than other schemes.

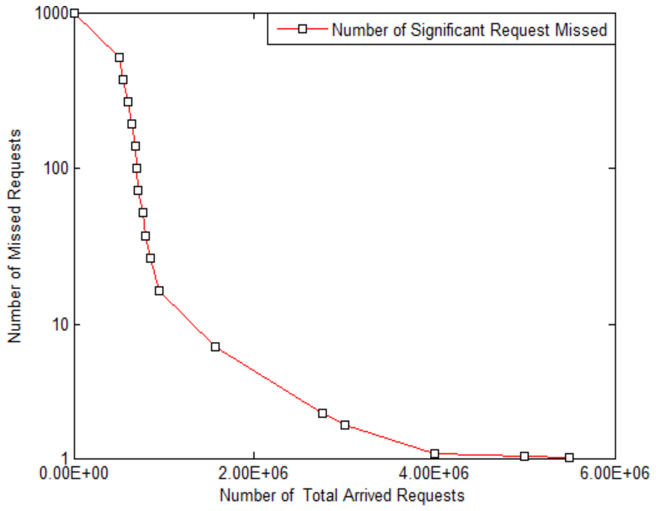

Fig.2. Desired Accuracy performed by SRATE

Dedicated runtime implementation of SRATE scheme is developed on the basis of fast & complicated comparisons between necessary information for updating its system state as well as desired changes in state-rank to follow system state of incoming requests, evaluated through "Tri-Way Trajectory Simulations" as implemented part of 'Dispatching Decision' in proposed Integrated Cloud Computing Model described in fig.1 of [17]. Following all of these, it gives state process prediction for particular request may be ready to process or dispose under limitation of threshold. On Yahoo! Video site, all the videos are classified into 16 categories. Each video is assigned a unique ID (integer number), and has the following information posted on its webpage: title, number of views (so far), video duration, average rating, number of ratings, added (uploaded) time, source (video producer), content description, tags (keywords), and comments. We trace the Yahoo! Video site for 2 months (from January 11 to March 12, 2012), and the data was collected in every 30 minutes to 1 hour. Due to large scale of Yahoo! Video site, we limited the data collection to the first 10 pages of each category. Since each page contains 10 video objects, each time the measurement collects dynamic workload information for 1230 video files in total. Throughout the whole collection period, we recorded 2,843 unique videos which durations range from 2 to 6350 seconds and a total of 1,755,186 video views. This can be translated into a daily video request rate of 29253.

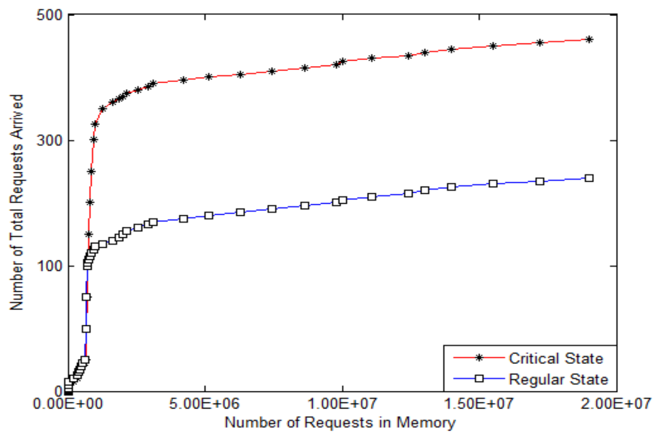

Fig.3. Memory cost of SRATE for both system state

So we decide the accuracy level $\beta = .0002$ with $\alpha = 99.99\%$. In other words, we want to estimate the proportion $\pm .0001$ with error probability less than 0.0001. This corresponds to a $Z_\alpha = 4.00$.

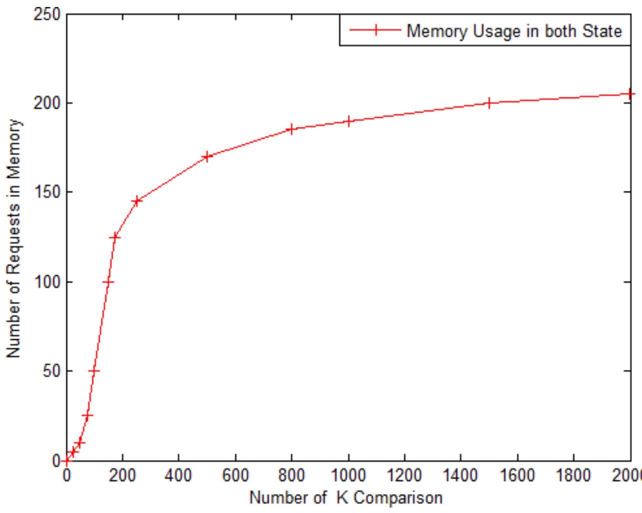

Fig.4. Memory Cost under different K values

SRATE is the fastest; it can capture all requests after about 6 million request arrivals. Fig.2 shows the number of significant requests that each approach fails to capture under various sampling time. Here K for SRATE is set to be 100. We find that all approaches can easily capture the have large requests with rate above 0:1. But their ability of capturing medium-sized requests varies significantly. The number of requests that SRATE captures remains almost constant, determined by buffer size. Since most buffer were occupied by data items from large requests, the scheme missed about 75% of all the medium-sized requests.

Next, fig.3 & fig.4 shows the change in memory size with increase in sampling time. From both Figures, we observe SRATE has lowest memory size but worst estimation accuracy. SRATE is the most accurate and fastest; it uses less memory than other schemes. We also note that both SRATE use about the same amount of memory by the time they capture all medium-sized requests (not shown in the figure), which indicates that it need similar memory size for the same estimation accuracy.

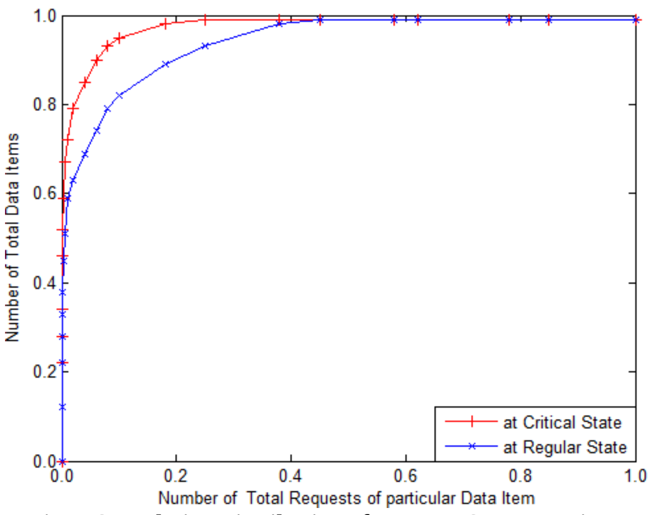

Fig.5. Cumulative Distribution of Request Concentration

From fig.5 for cumulative distribution of request concentration then we use a 60-second-timer mechanism to examine request states. Whenever a timer goes off, all non-expired requests are checked, and their average rates are updated. The requests are expired by a different mechanism: if no data items have arrived in the last 32 seconds on this request, the request is automatically timed out. We then keep track the top 10% of requests with the highest sending rates, and calculate number of data items & requests; they contributed as a percentage of entire traffic on a minute by minute basis.

So, compare with other fast estimation approaches that-

$$N^{ESRATE} \approx N^{CATE} = N^{Fast\ Top-k} = \frac{3Z_\alpha^2}{k\beta^2} = \frac{1}{0.46k} N^{RATE}$$

Now compare in case of processing speed & number of K comparisons-

$$SRATE > Fast\ Top - k > CATE > RATE$$

6. CONCLUSIONS & FUTURE WORKS

In this paper we have developed request rate estimation scheme called SRATE that uses counting, matching and state-ranking between different arriving data items to estimate the request rate and request concentration with memory usage. There is nearly dissimilarity in statistics to calculate mean duration due to the different video upload policies on the two sites: YouTube [11] imposes the limit of 10 minutes on regular state, while Yahoo! Video allows the video size up to 100MB (more than 40 minutes at 300Kb/s bit rate) in regular state. Even we have shown through both theoretical analysis and extensive simulations that SRATE is fast and memory efficient. It estimates small request quickly and accurately, and still gives good estimation accuracy for large requests. Many proposed QoS mechanisms can take advantage of this phenomenon to make the implementations scalable. A discussion on the implication of this phenomenon on different QoS mechanisms is offered. Dependence within data item arrivals may affect the estimation accuracy. Although the buffer-based approach can alleviate the problem, a better approach that can further minimize the impact of dependence without additional over-head may be more preferable. We attempt to address this issue as part of future work on SRATE based workload factoring to overcome the problem of workload in cloud computing with enhancing new trend as Integrated Cloud Computing.

7. REFERENCES

[1] "*Yahoo! Video*". http://video.yahoo.com/.

[2] Duffield, N, Lund, C., and Thorup, M., "*Charging from Sampled Network Usage*", SIGCOMM internet Workshop 2001.

[3] Duffield, N, and Grossglauser, M., "*Trajectory Sampling for Direct Traffic Observation*", Proceedings of ACM SIGCOMM 2000.

[4] Hao, F., Kodialam, M., and Lakshman, T. V., "*ACCEL-RATE: a faster mechanism for memory efficient per-flow traffic estimation*", proceedings of ACM SIGMETRICS 2004.

[5] Fang, W, and Peterson, L., "*Inter-as Traffic Patterns and their Implications*", Proceedings of IEEE GLOBECOM 1999.

[6] Feldmann, A. et al., “*Deriving Traffic Demands for Operational IP Networks: Methodology and Experience*”,Proceedings of ACM SIGCOMM’ 2000.

[7] F. Hao, M. S. Kodialam, T. V. Lakshman, and H. Zhang. “*Fast, memory-efficient traffic estimation by coincidence counting*”. in INFOCOM, 2005, pp. 2080-2090.

[8] Feng, W. et al., “*The Blue Queue Management Algorithms*”, IEEE/ACM Transactions on Networking, Vol.10, Number 4, 2002.

[9] Aldous, D., “*Probability Approximations via the Poisson Clumping Heuristic*”, Springer-Verlag, 1987.

[10] Estan, C. , and Varghese, G., "*New Directions in Traffic Measurement and Accounting*", Proceedings of ACM SIGCOMM 2002.

[11] Kodialam, M., Lakshman, T. V., and Mohanty, S., "*Runs bAsed Traffic Estimator (RATE): A simple, Memory Efficient Scheme for Per-Flow Rate Estimation*", Proceedings of INFOCOM'2004.

[12] Ferguson, T. S., "*A Course in Large Sample Theory*",Chapman and Hall, 1996.

[13] Estan, C., Savage, S., and Varghese, G., “*Automatically Inferring Patterns of Resource Consumption in Network Traffic*", Proceedings of ACM SIGCOMM 2003.

[14] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, and A. Saxena. “*Intelligent workload factoring for a hybrid Cloud Computing model*”. NEC Labs America Technical Report 2009-L036, Feb 2009.

[15] “*Wikipedia- IID*”. http://www.wikipedia.com/.

[16] “*ComScore Video Metrix report: U.S. Viewers Watched an Average of 3Hours of Online Video in July*”. Available: http://www.comscore.com/press/release.asp?press=1678.

[17] Du_eld, N, Lund, C., and Thorup, M., “*Charging from Sampled Network Usage*", SIGCOMM internet Work-shop 2001.

[18] A. Kumar, M. Sung, J. xu, and J. Wang, *“Data streaming algorithms for efficient and accurate estimation of flow distribution”*, in *Proc. Of ACM SIGMETRICS*, June 2004, to appear.