Resilient Workload Manager: Taming Bursty Workload of Scaling Internet Applications

Hui Zhang, Guofei Jiang, Kenji Yoshihira, Haifeng Chen, Akhilesh Saxena
NEC Labs America
Princeton, NJ, 08901
huizhang@nec-labs.com, gfj@nec-labs.com, kenji@nec-labs.com,
haifeng@nec-labs.com, saxena@nec-labs.com

ABSTRACT

In data centers hosting scaling Internet applications, operators face the tradeoff dilemma between resource efficiency and Quality of Service (QoS), and the root cause lies in workload dynamics. In this paper, we address the problem with the design of Resilient Workload Manager (ROM). ROM explicitly segregates *base* workload and *trespassing* workload, the two naturally different components in application workload, and manages them separately in two resource zones with specialized optimization techniques.

As a comprehensive workload management framework, ROM covers workload, data, resource, and Quality of Service of the target applications. It features a fast workload factoring algorithm for distributing incoming application requests, not only on volume but also on content, between the two resource zones; integrated two-dimensional workload shaping, resource planning, and request dispatching schemes for efficient utilization of base workload zone resource; and a simple and high-performance system architecture for dynamic provisioning in trespassing workload zone. Through extensive evaluation, we showed ROM can achieve resource efficiency (e.g., 54.9% server saving), guarantee QoS (based on client-side perceived service quality), reduce data access overhead in the trespassing workload zone during peak load (up to two orders of magnitude), and be adaptive at processing speed (running faster at peak load periods than at regular periods).

Categories and Subject Descriptors: H.m Information Systems:MISCELLANEOUS

General Terms: Design, Management, Performance

Keywords: algorithms, cloud computing, data center management, load balancing, workload management

1. INTRODUCTION

Internet websites can experience rapid growth on their service workload. For example, it took MySpace [7] only 20 months to reach 20 million users since its inauguration. To address the scalability issue, a scaling-out architecture is adopted in today's Internet applications; they can duplicate service instances on demand to distribute and process increasing workload. Examples include stateless applications such as YouTube video streaming service [13] and stateful applications such as GigaSpaces XAP web applications [5].

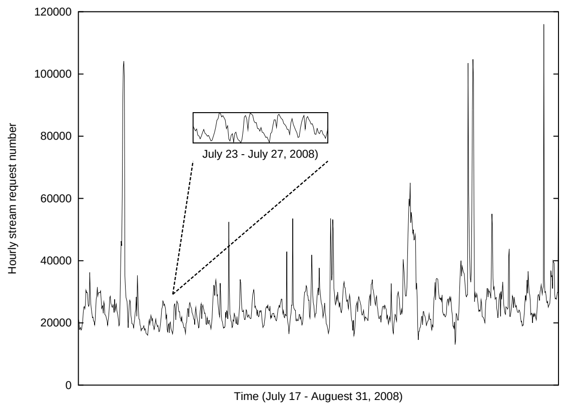

Figure 1: Video stream workload evolution on Yahoo! Video Site.

Resource management is an issue even with a scaling-out architecture. Figure 1 shows the dynamics of the hourly workload measured [1] during a 40-days period on Yahoo! Video [12], the 2nd largest U.S. online video sharing website [15]. We observed that the ratio of the maximum workload to the average load is as high as 5.3 (12.9 if workload was measured in half an hour interval), which makes over-provisioning over peak load unrealistic. Applying statistical analysis techniques including autoregressive integrated moving average (ARIMA), we observed that even though there were clear periodic components (exampled by the workload between July 23 and July 27) the big spikes shown in Figure 1 were not predictable; therefore resource assignment could not been done ahead of time for all the time.

Dynamic provisioning comes as a rescue for handling dynamic workload. With the mature of cloud computing technologies such as virtual machines (VM) [11, 9], and network and storage virtualization [4], it becomes a reality of real-

[1]The measured workload was the number of video streams served per hour on Yahoo! Video site. For further details, please refer to [31].

time dynamic provisioning on computing resources. Cloud computing, known either as online services such as Amazon AWS [2] and Google App Engine [6], or a technology portfolio behind such services, features a shared computing infrastructure hosting multiple applications where resource multiplexing leads to efficiency; more computing resources are allocated on demand to an application when its current workload incurs more resource demand than it is allocated. As reported in [19], VM based application server provisioning (reboot, resume VM, reconnect to load balancer) takes less than 2 minutes (only 45 seconds reported in [39]).

Simple reactive dynamic provisioning (such as threshold rules based) has well-known drawbacks. It is slow to workload changes and not effective at handling complex load patterns experienced in practice. Besides, over peak load reactive dynamic provisioning typically does no more than overprovisioning over one computing resource (aka, physical servers), while the provisioning over other resources including storage and networking is not explicitly addressed or optimized.

In this paper, we present Resilient Workload Manager (ROM), a proactive solution for dynamic workload management. Its core idea lies in the explicitly separate management of the two naturally different components in the aggregate workload of an Internet application: *base* load and *trespassing* load. With the workload in Figure 1 as an example, *base* load refers to the smaller and smoother workload experienced by the application platform most of the time (e.g, 95% of the time), while *trespassing* load refers to the short and transient load spikes experienced at rare time (e.g., the top 5%-percentile load).

ROM makes prediction on the base load, compensates inevitable prediction inaccuracy due to short-term dynamics with a new workload shaping scheme, and uses integrated offline planning and online dispatching schemes to deliver the guaranteed Quality of Service (QoS) with minimal resource in plan. On trespassing load, ROM does not attempt the impossible prediction; it relies on the overflow alarm from the workload shaping component to trigger the dynamic provisioning of a resource zone dedicated for trespassing load; ROM offers a fast workload factoring scheme to split the peak load into two parts and assures that the base load part remains within planned in amount, and the trespassing load part incurs minimal access demand on the application data associated with it. This simplifies the system architecture design for the trespassing load zone and significantly increases the server performance within it. As we will discuss in Section 2, it makes the two-zone resource management architecture an attractive option for evolving current enterprise IT infrastructures into the Cloud Computing era.

As a comprehensive solution for application workload management, ROM includes the necessary components to covers workload (online load balancing and dispatching), data (load factoring), resource (offline capacity planning), and QoS enforcement (workload shaping). While those components are functionally independent, many have dependency relationship for their performance maximization For example, the capacity planning component relies on the workload shaping component for a reliable load prediction; the load factoring component allows the dispatching scheme to be simple and reduce service warm-up time in the trespassing zone.

For the presentation concreteness, we describe and evaluate ROM in the context of the video streaming application

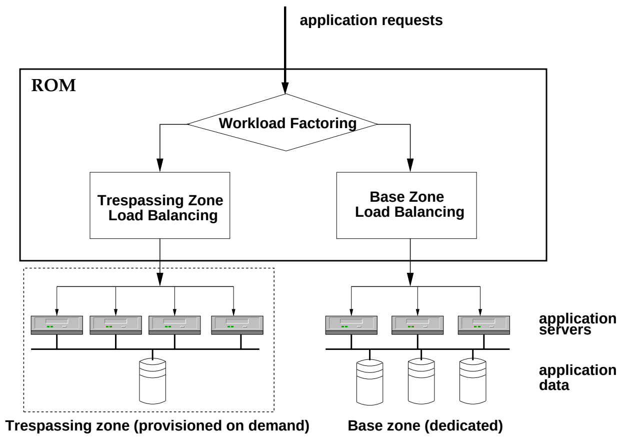

Figure 2: ROM architecture

in the rest of the paper. We have built a video streaming service testbed that has a local cluster as the base load zone and the Amazon EC2 infrastructure [2] as the trespassing zone, and implemented ROM as a load controller to arbitrate the stream load distribution between the two zones and the servers within each. With testbed experiments and real-trace driven simulations, we showed ROM can achieve resource efficiency (e.g., 54.9% server saving), guarantee QoS (based on client-side perceived service quality), reduce data access overhead in the trespassing workload zone during peak load (up to two orders of magnitude), and be adaptive at processing speed (running faster at peak load periods than at regular periods).

The rest of the paper is organized as follows. Section 2 describes the architecture and application scenarios of ROM. In Sections 3 4 and 5, we present the problem definitions and technical details that ROM offers in the workload factoring, workload shaping, and base zone load balancing components. Section 6 presents the evaluation results of the ROM framework. We present the related work in Section 7, and concludes this paper in Section 8.

2. ROM ARCHITECTURE

2.1 ROM Architecture

Figure 2 shows the ROM architecture. It has three basic components: workload factoring, base zone load balancing and trespassing zone load balancing. The functions of the workload factoring component include two: avoiding overloading of the base zone application platform through load redirection; making trespassing zone application platform simple through load decomposition not only on the volume but also on the requested content. We will describe the workload factoring in details at Section 3. Notice that in the ROM architecture, the data storage at the trespassing zone is decoupled from that at the base zone so that the former can be a floating platform and does not have to be tied to the latter through some shared resources (e.g., shared SAN, which otherwise has to be provisioned for the peak load).

Figure 3 shows the details on ROM base zone load balancing. Its basic function is to manage a dedicated platform for the hosted application, with minimal resource upon specified Service Level Agreements (SLAs) (e.g., request response time). Besides the common workload profiling and

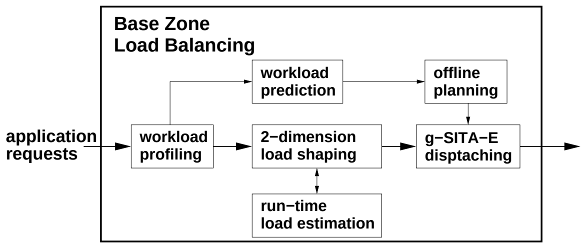

Figure 3: ROM base zone load balancer architecture

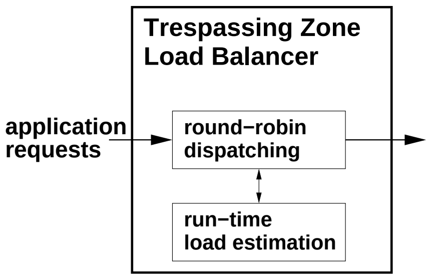

Figure 4: ROM trespassing zone load balancer architecture

prediction components, its new features include the two-dimensional workload shaping with a low-overhead run-time load estimation scheme, which will be described in Section 4, and the integrated offline planning and online dispatching schemes, which will be described in Section 5.

Figure 4 shows the details on ROM trespassing zone load balancing. It is designed to be simple and fast as it is activated only at peak load. The request dispatching is the well-known round-robin scheme [17]; it is supported with the same run-time load estimation scheme used in the base zone management for SLA enforcement (preventing server overloading). While simple, the trespassing zone load balancing keeps high performance with the help of the workload factoring component, which manages to make the requests flowing into this zone exhibit high data locality; therefore, no complex load balancing scheme is needed for performance optimization. The data storage can be a cache which will be populated in real time when the trespassing zone is provisioned on demand. This makes the trespassing zone being decoupled from the base zone so that it can float at a different physical infrastructure.

2.2 ROM Application Scenarios

Scenario 1: private cloud within the data center. When the data center has ample resources (servers) and a private cloud will be built for resource multiplexing through server consolidation, ROM can be used to enable a hub-and-spoke infrastructure architecture. The base load zones of the applications are spokes, and a common shared platform acts as the hub and hosts (multiplexes) the trespassing zones from multiple applications. Through the hub-and-spoke architecture the multiplexing effect is maximized since the trespassing loads dispatched by ROM are random in nature. If unfortunately the applications reach their peak load at the same time, coordinated multi-service workload management schemes like in [40, 18, 26] can be applied on the trespassing zones.

Scenario 2: limited resources within the data center hosting the target applications, but on-demand resources available from outside (e.g., public cloud services). When the data center would like to take advantage of public cloud service to compensate for the resource limitation locally, ROM offers a cloud workload management service: the data center runs a small and fully utilized platform for base load zone hosting, while the remote cloud can be used for hosting the trespassing load zone.

3. WORKLOAD FACTORING

3.1 Problem Statement

We model workload factoring as a hypergraph partition problem [32]: each data item in the application is modeled as a node, each request is modeled as a net, and a link between a net and a node shows the access relationship between the request and the application data. The K-way hypergraph partition, an NP-hard problem, is to assign all vertices (data and then their requests) to K (K=2 in our case) disjoint nonempty locations without beyond their capacities, and with minimal net cut cost (the total weights of the nets that span more than one location, therefore bringing data transfer/consistency overhead).

The workload factoring solution contains two steps: the first step is on graph model generation addressed by a fast data popularity estimation scheme, and the second step in on hypergraph partition addressed by a greedy bi-section partition scheme. On video streaming where request-data relationship is simple (i.e., there is no hypergraph edge as one request accesses only one data), the partition problem degenerates to the knapsack problem where our greedy scheme is moving vertices one by one ranked by their popularity until reaching the other location's capacity. This is equal to redirecting the requests for the most popular data items in a top-k list into the trespassing zone. Next we present a factoring algorithm which quickly generates the correct top-k list during a popularity transition time disturbed by the workload burst.

3.2 Fast and memory-efficient Workload Factoring Algorithm

Shown in Figure 5, ROM workload factoring algorithm starts the factoring process when in the *panic* mode (which will be defined in ROM workload shaping algorithm in Figure 7). It features a fast data popularity estimation algorithm (called *fastTopK* in the rest of the paper) to detect if a request asks for one of the top-k most popular data items; if yes the request will go to the trespassing load zone, otherwise go to the base load zone. the key ideas in the fastTopK algorithm have two: speed up the top-k detection at changing data popularity distributions by pre-filtering old popular data items in a new distribution, and speed up the top-k detection at a data popularity distribution by pre-filtering unpopular data items in this distribution.

4. WORKLOAD SHAPING

4.1 Problem Statement

Traditionally Internet traffic has been modeled by Poisson arrivals because of the analytical simplicity and nice computational properties of this distributions. However, it has

Input:
request r

Output:
"base" if r will go to the base zone, "trespassing" otherwise.

Data Structures:

- a FIFO queue to record the last m requests.
- a list to record the current top-k popular data items.
- a list to record the historical top-k popular data items.
- a list to record the data items with its frequency counter not equal 0.

Algorithm:

1. if the system is in the "normal" mode, the historical top-k list is always set as empty.
2. if the system is in the "panic" mode and r is the first request since entering this mode, copy the current top-k list into the historical top-k list, reset all frequency counters to 0, and empty the current top-k list.
3. if r matches any of the historical top-k list (i.e., asking the same data item), increases the frequency counter of that data item by 1 in the counter list, and update the historical top-k list based on counter values.
4. otherwise, randomly draw m requests from the FIFO queue, and compare them with r; if r matches any of the m requests (i.e., asking the same data item), increases the frequency counter of that data item by 1 in the counter list, and update the current top-k list based on counter values.
5. In the "normal" mode, always answer "base".
6. In the "panic" mode, combining the two top-k lists by calculating the estimated request rate of each data item.
7. if r's data item is in the top k of the $2k$ joint items, answer "trespassing", otherwise answer "base".
8. if r's data item does not belong to the historical top-k list, add r into the FIFO queue for request history, and return.

Figure 5: ROM workload factoring algorithm

been discovered in [38] that such modeling of job arrivals may be inappropriate. Simply considering the first two moments in doing capacity planning when the traffic is highly variable may lead to insufficient capacity. Therefore in order to give accurate capacity planning, we apply workload shaping to the planned workload based on the technique introduced in [16].

4.2 "Better Than Reference Distribution" Shaping: a simple reference distribution

In Intelligent shaping[16], rather than simply comparing average values this shaper emits data streams with property "better than reference distribution" by employing a kind of nonparametric test. If all incoming data streams of the network considered are shaped according to "better than Poisson", network dimensioning is simplified in the sense that multiplexer performance can be predicted conservatively by assuming Poisson arrivals, hence, admission control and resource dimensioning can be based on a simple $M/D/1$ queueing model.

However, in our case the service distribution is clearly not D (deterministic service time). To shape the load, we have to shape along both volume and job size. Towards the goal, we use a simple reference distribution, the load distribution of the servers. A key challenge is how to obtain this load distribution at run time but with low overhead.

We design an open-loop run-time load estimation scheme, as shown in Figure 6. It does not need in-band monitoring which incurs heavy overhead and clearly does not scale to large server farms. Its performance will be negatively affected when the service time estimation is not accurate; at this case, it offers a conservative load estimation which will not cause critical results. For better and reliable load estimation, a server-side agent can run both the load monitoring and the same load estimation scheme, and only notify ROM to update its local data structure when the error between the estimation and the actual value is beyond some threshhold; this can give us both accuracy and efficiency.

Figure 7 describes the ROM workload shaping algorithm. It is the same as the shaping scheme in [16] except the runtime load estimation scheme and the buffer server group concept. The buffer server group is used to absorb the transient dynamics in the base load, in addition to the queue in ROM to hold the delayed requests. The size of the buffer server group is configured by operators and can also be calculated based on the SLA requirements and the expected workload variation. We skip the details on the buffer server group sizing problem in this paper.

5. BASE LOAD ZONE LOAD BALANCING

5.1 The Choice of Dispatching Schemes

First, we want to achieve both "workload-aware" and "locality-aware" in terms of the choice of the dispatching scheme. There are several policies studied extensively in recent liter-

Input:
request r, server i,

Output:
"Yes" if server i can serve request r, otherwise "No".

Data Structures:

- an integer variable to record the remaining capacity of server i,
- an m-length integer vector to record the current load on i,
- an index pointer to the latest access vector entry,
- an time variable to record the latest access time.

Algorithm:

```
1. Find the time difference between now and the latest access time, and update the time variable to now.
2. Move the index pointer to the new position according to the time difference; during the move, count the number of requests that should have finished by now.
3. Update the remaining capacity variable by counting into the freed capacity learned in 2.
4. If the remaining capacity is 0, answer "no" to the request.
5. Otherwise, find the estimated service time $s_r$ of request $r$, update the load vector by adding one into the entry $s_r$ away from the index pointer, and answer "yes" to the request $r$.
```

Figure 6: ROM run-time load estimation algorithm: an open-loop method

atures: Round Robin, Random task assignment, JSQ (Join-the-Shortest-Queue), LWL (least-workload-left), SITA (size-interval-task-assignment), other SITA based disciplines such as SITA-E (Size Interval Task Assignment with Equal Load) in [29] which choose the size cutoff point to achieve load balancing among servers so that mean sojourn time can be minimized). Basically, the optimality of the dispatching policy in distributed system depends on the request service time distribution and the service discipline on each server. When the ages of requests are known, then LWL is optimal when service time distribution has increasing failure rate. It is not clear what is the optimal dispatching scheme when each individual request's service time is known. In [29], extensive simulation results are shown to compare the performance of different dispatching schemes for requests with high variability, and discover that: at light traffic case, LWL works better than SITA-E because LWL can exploit idle servers, which are more likely in light traffic scenario; SITA-E works better in high load scenario because idle server are less likely to exist. A side result about LWL is that actually LWL works the same as if a FIFO queue is maintained at the dispatcher, i.e. a distributed system with s servers with LWL dispatching scheme is identical to an $M/G/s$ queue.

We use the following formulas for $M/G/s$ queue of traffic intensity ρ and service time distribution with kth moments m_k [44]:

1.
$$\Pr[W(M/G/s] > 0] \approx \Pr[W(M/M/s) > 0] = \frac{(s\rho)^s}{s(1-\rho)\left[\frac{(s\rho)^s}{s!(1-\rho)} + \sum_{k=0}^{s-1}\frac{(s\rho)^k}{k!}\right]}$$

2. $\mathbf{E}[W(M/G/s)] \approx \frac{m_2 \Pr[W(M/G/s)>0]}{2m_1 s(1-\rho)}$

3. $\Pr[W(M/G/s) > t] \approx \Pr[W(M/G/s) > 0]e^{-2sm_1(1-\rho)t/m_2}$

Equation 2) will be used to calculate the server size of an $M/G/s$ system given the workload and the SLA requirement on expected request waiting time, and Equation 3) will be used to calculate the server size of an $M/G/s$ system given the workload and the SLA requirement on waiting time tail distribution.

5.2 Integrated offline planning and online dispatching

Simulation results in [28] show that LWL and SITA-E can perform 100 times better than the naive task assignment policies such as random and round-robin. Between LWL and SITA-E, when task size is less variable (α is large), LWL exhibits better performance; when it is very variable (α approaches 1), SITA-E can perform 100 times better than LWL.

One Observation is that the essential reason for SITA-E dispatching lies in reducing load variance on individual servers, while there is no need to reduce load variance if it does not affect the server performance any more. We design a generalized SITA-E dispatching algorithm (g-SITA-E) that replaces the size range partition unit from "server" to "group"; a group can be a single server or a set of servers, and it depends on the actual load distribution; within a group the dispatching is LWL scheme. g-SITA-E is designed to achieve the specified quality of service (in terms of maximum mean sojourn time) with the minimum number of servers (efficiency) *and* and the minimum number of groups (scalability).

Figure 8 describes the g-SITA-E dispatching scheme, and it takes a subroutine *MGS_planning* , the offline planning on $M/G/s$ systems with LWL dispatching as shown in Equa-

```
Input:
    request $r$, SLA parameters on response time $t$ and maximum request drops allowed $d$.
Output:
    "Yes" if request $r$ is admitted for service, otherwise "No".
Algorithm:
    1. find the server group $G$ that $r$ belongs to based on its estimated service time and the grouping information from offline
       planning described in Figure 8.
    2. query the run-time load estimation scheme for server availability in $G$; if yes, answer "Yes" and returns.
    3. otherwise, check if it has been delayed before.
    4. if yes, send it to the buffer server group
       (a) if available capacity, answer "Yes" and return.
       (b) otherwise, dropped it, and base on the SLA parameter $d$, decide if trigger the "panic" model (by passing an alarm
           message to the workload factoring component), return.
    5. otherwise, generate a random delay based on $t$, go back to 1.
```

Figure 7: ROM workload shaping algorithm: two dimensions on volume and job size

tion 2) and 3). For Step 2) in the Planning phase, The cut-point decision is made as the following for workload with Bounded Pareto service time distribution and Poisson arrivals (enforced through workload shaping). The probability density function for a bounded Pareto distribution is

$$f(x) = \frac{\alpha k^{\alpha} x^{-\alpha-1}}{1-(k/p)^{\alpha}},\ k \le x \le p.$$

We denote it as $B(k,p,\alpha)$. The moments of X are given by

$$\mathbf{E}[X^j] = \frac{\alpha k^{\alpha}(k^{j-\alpha} - p^{j-\alpha})}{(\alpha - j)(1-(k/p)^{\alpha})}.$$

According to the results in [29], for $B(k,p,\alpha)$, if $\alpha > 1$, defining

$$x_i = \left(\frac{(h-i)k^{1-\alpha}}{h} + \frac{ip^{1-\alpha}}{h}\right)^{1/(1-\alpha)},\ \text{for } i = 0\cdots h,$$

then

$$\int_{x_0=k}^{x_1} xf(x) = \cdots = \int_{x_{h-1}}^{x_h=p} xf(x) = \frac{\mathbf{E}[x]}{h}.$$

And $h = 2$ in the each round of the planning procedure.

6. EVALUATIONS

6.1 Trace-driven Simulations

6.1.1 Yahoo! Video workload traces

We evaluate the two-zone resource management scheme at a large-scale system scenario through trace-based simulations. We crawled all 16 categories on the Yahoo! Video site for 46 days (from July 17 to August 31 2007), and the data was collected every 30 minutes. This measurement rate was chosen as a tradeoff between analysis requirement and resource constraint. Throughout the whole collection period, we recorded 9,986 unique videos and a total of 32,064,496 video views. This can be translated into a daily video request rate of 697064.

We model a single video server as a group of virtual servers with First Come First Served (FCFS) queues. The virtual server number corresponds to the physical server's capacity, which is defined as the maximum number of concurrent streams delivered by the server without loosing a quality of stream. In the analysis, the number 300 is chosen for the capacity of a video server based on the empirical results in [22]. In this way, we model the video service center as a queueing system with multiple FCFS servers.

We consider two QoS metrics for SLAs: the stream quality for an accepted connection, and the waiting time of a video request in the queue before accepted for streaming. Assume enough network bandwidth, then QoS on stream quality within the data center side can be guaranteed through admission control based on server capacity. For the waiting time W, we consider two types: *maximal average waiting time* (W_{avg}), defined as $\mathbf{E}[W] \le x$, $x > 0$;*bound on the tail of the waiting time distribution* (W_{tail}), define as $P[W > x] < y$ with $x > 0$, $y < 1$. For example, SLA could be that 90% of the requests experience no more than 5 seconds delays, i.e., $x = 5$ and $y = 90\%$.

Taking the one-week measurement data from Aug 13th to Aug 20th, the simulation calculated the server demands of random and g-SITA-E dispatching schemes with Poisson arrivals (based on results in [44]) and set the SLA requirement as W_{avg}: $\mathbf{E}[W] \le 1.5m_1$. Figure 9 shows the results; on average 54.9% of servers could be saved with g-SITA-E scheme as compared to random dispatching scheme.

Using the same data and set the SLA requirement as W_{tail}: $\Pr[W > m_1] < 0.3$, we repeat the simulation;on average 69.9% of servers could be saved with g-SITA-E scheme as compared to random dispatching scheme.

6.2 Testbed Experiments

We set up an application testbed which hosts YouTube-like video streaming service. As shown in Figure 10, the testbed consists of two parts: a local data center and a plat-

Input:

request r , service time histogram, bounded pareto distribution parameter k, p, a , SLA parameter qos (expected waiting time).

Output:

the server serving r; the number of servers needed for the current planned period and the grouping information.

Planning (revoked periodically, e.g., 1 hour):

1. Initialize $n = p + sqrt(p)$, where p is the load intensity.
2. Divide the total workload into two equal-weight pieces by calculating cutoff point x, Bernoulli splitting probability p1, p2
3. For the left-side load piece, the required server cluster size $snum_l = MGS\_planning(p1, qos, k, x, a, floor(n/2))$
4. calculate the QoS in the left cluster $E[W_l]$ with Equation 2).
5. $E[W_r] = (qos - E[W_l]p1)/p2$.
6. For the right-side load piece, the required server cluster size $snum_r = MGS\_planning(p2, E[W_r], x, p, a, ceil(n/2))$
7. the total servers needed $snum = min(snum_l + snum_r, n)$
8. If $(snum == n)$, go to next; otherwise, set $n = snum$, go back to 2 for each of the two workloads defined in the above and divide them recursively into smaller pieces (clusters).
9. assign physical servers into the smallest clusters (groups) based on the calculated cluster size, and notify the dispatching and workload shaping components about the grouping information in the form of (($group_1$, server list (s^1_{g1}, s^2_{g1},...), service time range (t_1,t_2),), ($group_2$, ...),...)

Dispatching:

1. find the group g based on the estimated service time of request r and the grouping information.
2. in the group g, find the server i through LWL dispatching and run-time load estimation scheme.
3. dispatch r to i.

Figure 8: ROM g-SITA-E dispatching algorithm: integration of SITA-E and LWL dispatching

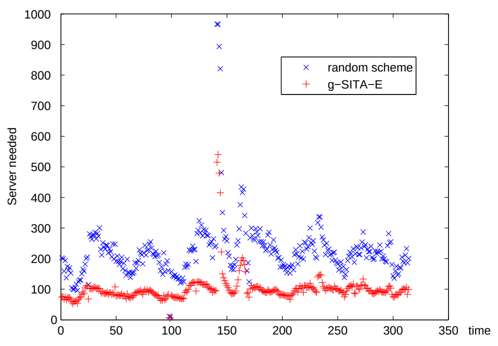

Figure 9: Resource Demand within one week: W_{avg} case

form set up at Amazon AWS infrastructure utilizing the EC2 and S3 services. In the local data center, 20 open source Darwin streaming servers [3] are provisioned all the time, while the streaming server instances at Amazon EC2 are activated on demand. A distributed workload generator based on openRTSP [8] was implemented to generate real video streaming load.

ROM was implemented as a load controller for the stream service. When a client request for a video clip comes into the testbed as a HTTP request, the Apache web server parses the request and asks ROM for which stream server (either a local server or a remote server at Amazon EC2.) the video clip will be served; it then returns to the client with a dynamic HTML page which automatically starts the media player for video streaming at the client machine.

6.2.1 Workload factoring - synthetic load traces

We evaluate the workload factoring algorithm by running experiments with synthetic load traces. In the traces, we generated workload with a stable data popularity distribution D_1 before time t, and then suddenly changed to another distribution D_2 after t where D_2 is the sum of D_1 and another distribution D_3. We generated D_1 and D_3 with uniform and *Zipf* distributions (different α values), and also changed the volume ratio of D_1 to D_3 with different numbers ($1 : k$ where $1 \leq k \leq 10$). For the ROM FastTopK algorithm, its goal is to decompose the aggregated workload D_2 into two parts so that their volume ratio is the same as that of D_1 to D_3 and minimize the unique data items contained in the load part with the volume $\frac{D_3}{D_1+D_3}$.

We compared fastTopK with 3 other workload factoring algorithms:

- *random*: the random factoring algorithm decides with the probability $\frac{D_3}{D_1+D_3}$ a request will go to the load group with the volume $\frac{D_3}{D_1+D_3}$.
- *Choke*: the Choke factoring algorithm is based on the

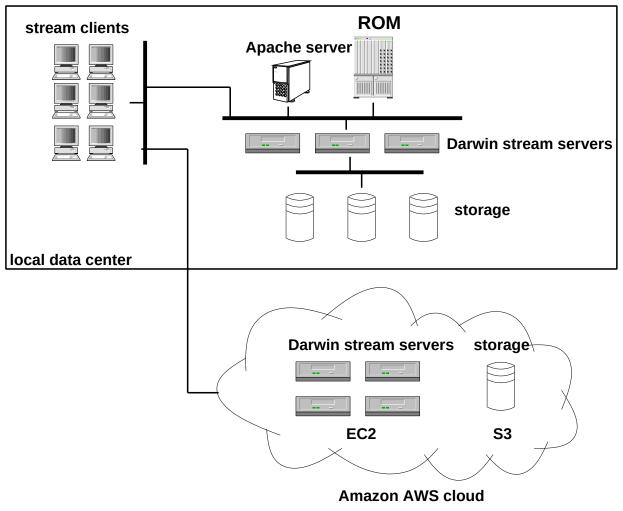

Figure 10: The video streaming service testbed with a hybrid platform

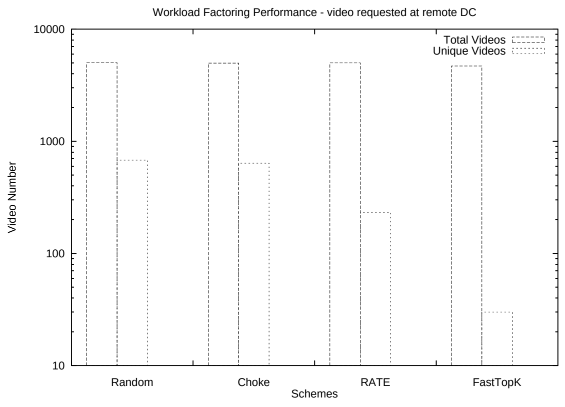

Figure 11: ROM workload factoring performance: comparison with other factoring algorithms

ChoKe active queue management scheme [37]. While ChoKe was originally proposed for approximating fair bandwidth allocation, it is a reasonable candidate for workload factoring when the optimization goal is minimizing the unique data items in one part (dropping the packets to the top popular IP destinations is similar to finding the top popular data items).

- *RATE*: the RATE factoring algorithm acts the same as fastTopK except that it uses the RATE scheme [27] to detect the top-K data items.

We ran one example trace where D_1 is Zipf and D_3 is uniform distributions, and the volume ratio is $1:1$. At the load changing point the trespassing load D_3 on a few hot data items jumped in. Figure 11 shows the factoring performance in terms of the number of unique data items contained in the load part with the volume $\frac{D_3}{D_1+D_3}$. When all algorithms dispatched the same amount of requests into the trespassing zone, fastTopK outperformed the other three significantly in terms of unique video files requested (two orders of magnitudes compared to random dispatching); actually it was quite close to the optimal performance (30 vs 10).

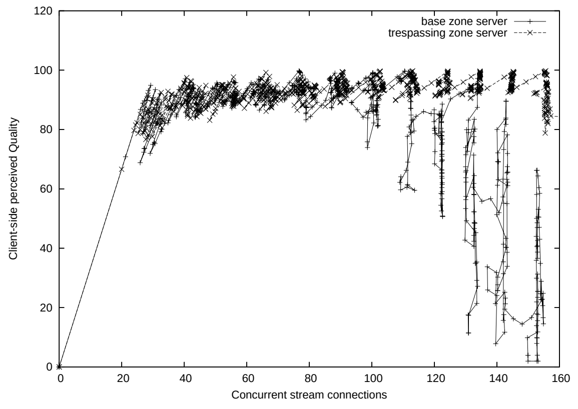

Figure 12: ROM performance: client-side perceived stream quality at two zones

Figure 12 shows the client-side perceived stream quality from a base load zone server and a trespassing zone server at different load with the above workload factoring experiment. In this case both servers have the same hardware/software configuration and are in the local testbed. The client-side perceived stream quality is a metric reported by the Darwin Stream server itself and has a score between 0 (worst) and 100 (best). We see when the concurrent connections went up, the trespassing load zone server delivered more reliable streaming quality than the base load zone server. It could support up to 160 concurrent stream connections while keeping the client-side quality at above 80, while the base load zone server could only support around 90 concurrent stream connections to keep the client-sided quality at above 80. In the testbed configuration, we set the trespassing load zone server capacity at 160 concurrent connections and that for base load zone servers at 90, and enforce it during dispatching.

6.2.2 End-to-end performance - stress testing

We used 20 servers to generate the streaming workload; starting from the beginning, each of them kept sending RTSP requests at a fixed rate of 40 request per second. The experiment stopped when ROM stayed at the panic mode for a short time. The goal of this stress testing is to observe the performance behavior of ROM in and not in the panic mode in terms of processing time.

Figure 13 shows ROM per-request processing time with 100 streaming servers (20 in base zone and 80 in trespassing zone). It kept increasing initially before entering the panic mode; the increase was due to the decrease of available servers in the base zone and the lookup time spent in run-time load estimation kept increasing. Then it suddenly dropped to to below 2 milliseconds as the base zone had run out of resource and the trespassing zone was activated; since the dispatching in the trespassing zone was fast (round-robin), ROM performed much faster when more load came in. This is one attractively adaptive property ROM has.

7. RELATED WORK

ROM borrowed, extended, and integrated many ideas from the research of service management, integrated resource man-

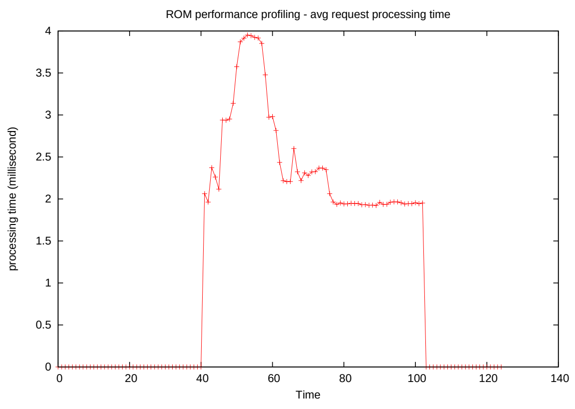

Figure 13: ROM performance: per-request processing time

agement, traffic shaping, traffic monitoring, and web caching. In the following we gave a brief description.

There have been works on mapping the SLA into the required system resource requirements and sizing the system [23, 21, 14]. Also [35] investigate the resource allocation problem with the objective of maximizing profits under SLA contracts using the methods from probability theory, queueing theory and combinatorial optimization. Our work on integrated offline planning and online dispatching contributes a new method in this domain.

Enhanced workload management solution like Cataclysm [42] integrates admission control and dynamic provisioning components in addition to the load balancing algorithm. It applies admission control to make sure flash crowd requests will not overload the provisioned server pool by dropping requests, and uses queueing-model based dynamic provisioning technology to decide the server pool size to be provisioned. Integrated resource management such as [43] had similar architecture on dynamic provisioning and addressed complex n-tier web services. Technologies enabled QoS-aware workload management for multiple applications include [40]Quorum [18] [26]. Compared to those solutions, ROM features the two zone load management mechanism on a hybrid (local and remote) infrastructure.

Web caching [20] [30] [45] addresses locality exploration for cache hit ratio improvement, and their focus is on compact data structure design to exchange data item information. For fast frequent data item detection in data streams, many schemes have proposed for fast rate estimation in traffic monitoring [27] [34] [24], and fast data item counting in CDN [36]; their focus is still on compact data structure design to memorize request historical information at a static distribution. Compared to those solutions, ROM features the real-time performance of the workload factoring scheme against the changing data popularity.

Network admission control (traffic shaping) for statistical QoS [41] [33] [16], is exampled by leaky bucket algorithm, and the focus was on one dimension of the workload - traffic rate since the network packet size does not vary widely. Compared to those solutions, ROM looks at the two dimensions - traffic rate and job size.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we present the design of ROM, a proactive application workload manager. With the two-zone resource management, ROM allows enterprise IT systems to adopt a hybrid cloud computing model where a dedicated resource platform runs for hosting application base loads, and a separate and shared resource platform serves trespassing peak load of multiple applications.

In November 2008, Amazon launches CloudFront [1] for its AWS customers who can now deliver part or all application load through Amazon's global network; around the same time period VMWare also proposed in its Virtual Data Center Operation System blueprint the vCloud service concept [10], which helps enterprise customers expand their internal IT infrastructure into an internal cloud model or leverage off-premise computing capacity. When current IT systems evolve from the dedicated platform model to the shared platform model along the cloud computing trend, we believe a core technology component in need is on flexible workload management working for both models, and ROM is proposed as one answer for it.

For the future work, extending ROM application to stateful applications such as web services is a natural and challenging step. Many new problems arise such as session maintenance, service time estimation, and data consistency. We are working on a fast data on demand service and integrating the dynamic web service scaling approach proposed in [25] with ROM.

9. REFERENCES

[1] Amazon. http://aws.amazon.com/cloudfront/.
[2] Amazon web services. http://aws.amazon.com/.
[3] Darwin streaming server. http://developer.apple.com/darwin/projects/streaming/.
[4] Emc invista: Network-based storage virtualization. http://www.emc.com/products/detail/software/invista.htm.
[5] Gigaspaces. http://www.gigaspaces.com.
[6] Google app engine. http://code.google.com/appengine/.
[7] Myspace. http://www.myspace.com.
[8] openrtsp. http://www.live555.com/openRTSP/.
[9] Vmware. http://www.vmware.com.
[10] Vmware cloud vservices. http://www.vmware.com/technology/virtual-datacenter-os/cloud-vservices/.
[11] Xen. http://www.xen.org.
[12] Yahoo! video. http://video.yahoo.com.
[13] Youtube. http://www.youtube.com.
[14] A capacity planning framework for multi-tier enterprise services with real workloads. In *Proceedings of 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 781–784, 2007.
[15] ComScore Video Metrix report: U.S. Viewers Watched an Average of 3 Hours of Online Video in July. http://www.comscore.com/press/release.asp?press=1678, July 2007.
[16] D. Abendroth and U. Killat. Intelligent shaping: Well shaped throughout the entire network? In *Proceedings of INFOCOM 2002*, 2002.
[17] M. Arregoces and M. Portolani. *Data Center Fundamentals*. Cisco Press, 2003.

[18] J. M. Blanquer, A. Batchelli, K. Schauser, and R. Wolski. Quorum: flexible quality of service for internet services. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 159–174, Berkeley, CA, USA, 2005. USENIX Association.

[19] P. Bodik, M. P. Armbrust, K. Canini, A. Fox, M. Jordan, and D. A. Patterson. A case for adaptive datacenters to conserve energy and improve reliability. Technical Report UCB/EECS-2008-127, EECS Department, University of California, Berkeley, Sep 2008.

[20] E. Casalicchio, V. Cardellini, and M. Colajanni. Content-aware dispatching algorithms for cluster-based web servers. *Cluster Computing*, 5(1):65–74, 2002.

[21] Y. Chen, S. Iyer, X. Liu, and D. M. abd Akhil Sahai. Sla decomposition: Translating service level objectives to system level thresholds. In *ICAC '07. Fourth International Conference on Autonomic Computing*, June 2007.

[22] L. Cherkasova and L. Staley. Measuring the capacity of a streaming media server in a utility data center environment. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 299–302, New York, NY, USA, 2002. ACM.

[23] L. Cherkasova, W. Tang, and S. Singhal. An sla-oriented capacity planning tool for streaming media services, July 2004.

[24] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *SIGCOMM Comput. Commun. Rev.*, 30(4):271–282, 2000.

[25] L. Gao, M. Dahlin, A. Nayate, J. Zheng, and A. Iyengar. Application specific data replication for edge services. In *Proceedings of the 2003 International World Wide Web*, May 2003.

[26] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *DSN*, pages 326–335, 2008.

[27] F. Hao, M. Kodialam, T. V. Lakshman, and H. Zhang. Fast payload-based flow estimation for traffic monitoring and network security. In *ANCS '05: Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems*, pages 211–220, New York, NY, USA, 2005. ACM.

[28] M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2):204–228, 1999.

[29] M. Harchol-Balter, M. E.Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system, 1998.

[30] S. Jin and A. Bestavros. Greedydual* web caching algorithm: Exploiting the two sources of temporal locality in web request streams. In *In Proceedings of the 5th International Web Caching and Content Delivery Workshop*, pages 174–183, 2000.

[31] X. Kang, H. Zhang, G. Jiang, H. Chen, X. Meng, and K. Yoshihira. Understanding internet video sharing site workload: a view from data center design. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 1097–1098, New York, NY, USA, 2008. ACM.

[32] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 343–348, New York, NY, USA, 1999. ACM.

[33] E. W. Knightly and N. B. Shroff. Admission control for statistical qos: theory and practice. *Network, IEEE*, 13(2):20–29, 1999.

[34] 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.

[35] Z. Liu, M. S. Squillante, and J. L. Wolf. On maximizing service-level-agreement profits. *ACM SIGMETRICS Performance Evaluation Review Special section on the MAMA 2001 workshop*, 29(3):43 – 44, 2001.

[36] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 767–778, Washington, DC, USA, 2005. IEEE Computer Society.

[37] R. Pan, B. Prabhakar, and K. Psounis. Choke - a stateless active queue management scheme for approximating fair bandwidth allocation. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 942–951 vol.2, 2000.

[38] V. Paxson and S. Floyd. Wide-area traffic: the failure of poisson modeling. *SIGCOMM Comput. Commun. Rev.*, 24(4):257–268, 1994.

[39] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. pages 75–93, 2003.

[40] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based internet services. *SIGOPS Oper. Syst. Rev.*, 36(SI):225–238, 2002.

[41] J. S. Turner. New directions in communications (or which way to the information age). *IEEE Communications*, 24(10):8–15, Oct. 1986.

[42] B. Urgaonkar and P. Shenoy. Cataclysm: Scalable overload policing for internet applications. Jul 2008.

[43] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In *ACM proceedings of the Second International Conference on Automonic Computing*, pages 217–228, June 2005.

[44] W. Whitt. Partitioning customers into service groups. *Management Science*, 45(11):1579–1592, Nov 1999.

[45] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. *SIGOPS Oper. Syst. Rev.*, 33(5):16–31, 1999.