Proactive Workload Management in Hybrid Cloud Computing

Hui Zhang, Guofei Jiang, Kenji Yoshihira, and Haifeng Chen

***Abstract*—The hindrances to the adoption of public cloud computing services include service reliability, data security and privacy, regulation compliant requirements, and so on. To address those concerns, we propose a hybrid cloud computing model which users may adopt as a viable and cost-saving methodology to make the best use of public cloud services along with their privately-owned (legacy) data centers.**

As the core of this hybrid cloud computing model, an intelligent workload factoring service is designed for proactive workload management. It enables federation between on- and off-premise infrastructures for hosting Internet-based applications, and the intelligence lies in the explicit segregation of *base* workload and *flash crowd* workload, the two naturally different components composing the application workload. The core technology of the intelligent workload factoring service is a fast frequent data item detection algorithm, which enables factoring incoming requests not only on volume but also on data content, upon a changing application data popularity.

Through analysis and extensive evaluation with real-trace driven simulations and experiments on a hybrid testbed consisting of local computing platform and Amazon Cloud service platform, we showed that the proactive workload management technology can enable reliable workload prediction in the base workload zone (with simple statistical methods), achieve resource efficiency (e.g., 78% higher server capacity than that in base workload zone) and reduce data cache/replication overhead (up to two orders of magnitude) in the flash crowd workload zone, and react fast (with an X^2 speed-up factor) to the changing application data popularity upon the arrival of load spikes.

***Index Terms*—Cloud computing, hybrid cloud, workload management, algorithms, load balancing.**

I. Introduction

CLOUD Computing, known either as online services such as Amazon AWS [1] and Google App Engine [2], or a technology portfolio behind such services, features a shared elastic computing infrastructure hosting multiple applications where IT management complexity is hidden and resource multiplexing leads to efficiency; more computing resources can be allocated on demand to an application when its current workload incurs more resource demand than it was allocated.

Despite the advantages at management simplification and pay-per-use utility model, Cloud Computing remains in doubt regarding enterprise IT adoption. The concerns on the current cloud computing services include service availability & reliability, lack of Service Level Agreements, customer data security & privacy, government compliance regulation requirements, and more [1]. For example, Payment Card Industry Data Security Standard (PCI-DSS) audit is required for e-commerce systems with payment card involved, and the auditors need a clear physical demonstration on server infrastructure, software configuration and network deployment; the outages of the state-of-the-art Amazon cloud services (e.g., the S3 service outage on February 5, 2008 [4]) refresh concerns with the ballyhooed approach of any fully cloud-based computing solution.

This paper proposes a hybrid cloud computing model which enterprise IT customers can base to design and plan their computing platform for hosting Internet-based applications with *highly dynamic* workload. The hybrid cloud computing model features a two-zone system architecture where the two naturally different components in the aggregated workload of an Internet-based application, *base* load and *flash crowd* load, are explicitly separated for individual management. *Base* load refers to the smaller and smoother workload experienced by the application all the time, while *flash crowd* load refers to the much larger but transient load spikes experienced at rare time (e.g., the 5%-percentile heavy load time). The base load platform can be setup and managed in the (small) enterprise data center with the expectation of effective planning and high utilization, while the flash crowd load platform can be provisioned on demand through a cloud service by taking advantage of the elastic nature of the cloud infrastructure.

An intelligent workload factoring service is designed as an enabling technology of the hybrid cloud computing model. Its basic function is to split the workload into two parts upon (unpredictable) load spikes, and assures that the base load part remains within plan in volume, and the flash crowd load part incurs minimal cache/replication demand on the application data required by it. This simplifies the system architecture for the flash crowd load zone and significantly increases the server performance within it. As for the base load zone, workload dynamics are reduced significantly; this makes possible capacity planning with low over-provisioning factor and/or efficient dynamic provisioning with reliable workload prediction.

We built a video streaming service testbed as a concept system of the hybrid cloud computing model. It has a local cluster serving as the base load zone and the Amazon EC2 infrastructure [1] as the flash crowd zone; the workload factoring service was implemented as a load controller to arbitrate the stream load distribution between the two zones. With analysis, trace-driven simulations, and testbed experiments,

Manuscript received January 31, 2013; revised October 3 and November 26, 2013. The special issue guest editors coordinating the review of this paper and approving it for publication were G. Martinez, R. Campbell, and J. Alcaraz Calero.

The authors are with NEC Laboratories America, Princeton, NJ 08540 (e-mail: {huizhang, gfj, kenji, haifeng}@nec-labs.com).

Digital Object Identifier 10.1109/TNSM.2013.122313.130448

[1] Please refer to [3] for an interesting discussion

1932-4537/14/$31.00 © 2014 IEEE

we showed the workload factoring technology can enable reliable workload prediction in the base load zone (with simple statistical method), achieve resource efficiency (e.g., 78% higher server capacity than that in base load zone), reduce data cache/replication overhead (up to two orders of magnitude) in the flash crowd load zone, and react fast (with an X^2 speed-up factor) to the changing application data popularity upon the arrival of load spikes.

Note that the technologies presented in this paper are by no means a complete solution for the hybrid cloud computing model. There are many technical components skipped for discussion such as load balancing schemes in the two zones, data replication & consistency management in the flash crowd zone, security management for a hybrid platform, and more. We focus on the workload factoring component in this paper as it is a unique functionality requirement in the hybrid cloud computing architecture. In addition, for the presentation concreteness, we describe and evaluate our technologies in the context of video streaming applications throughout the paper.

The rest of the paper is organized as follows. Section II describes the design rationale and architecture of the hybrid cloud computing model. We present the problem model and technical details of the workload factoring mechanism in Section III, and analytic results of the fast frequent data item detection algorithm used by the workload factoring mechanism at Section IV. Section V shows the evaluation results. We present the related work in Section VI, and conclude this paper with Section VII.

II. Hybrid Cloud Computing Model

Our target applications are Internet-based applications with a scaling-out architecture; they can duplicate service instances on demand to distribute and process increasing workload. Examples include stateless applications such as YouTube video streaming service [5] and stateful applications such as GigaSpaces XAP web applications [6]. The design goal of the hybrid cloud computing model is to achieve both resource efficiency and QoS guarantee upon *highly* dynamic workload when hosting those scaling applications.

A. Design Rationale

For the presentation concreteness, we discuss the design rationale through our observations on the measured workload of a real Internet web service.

Figure 1 shows the dynamics of the hourly workload measured [2] during a 46-day time period on Yahoo! Video [8], the 2nd largest U.S. online video sharing website [9]. Applying statistical analysis techniques including auto-regressive integrated moving average (ARIMA) and Fourier Transfer analysis, we observed that there were clear periodic components (exampled by the workload between July 23 and July 27 in Figure 1) in most of the time; however, the big spikes shown in Figure 1 were not predictable. Therefore, resource planning could be effective most of the time, but not always working. We also 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 overprovisioning over peak load highly inefficient. Based on the above observations, we believe an integration of proactive and reactive resource management schemes should be applied on different components of the aggregated application workload along the time: proactive management opportunistically achieves resource efficiency through reliable prediction on the base workload seen most of the time, while reactive management quickly responds to sudden workload surges in rare time with the requirement of agile and responsive performance.

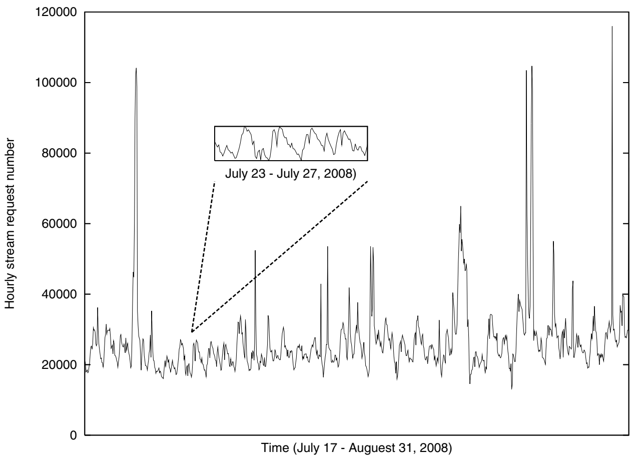

Fig. 1. Video stream workload evolution on Yahoo! Video Site.

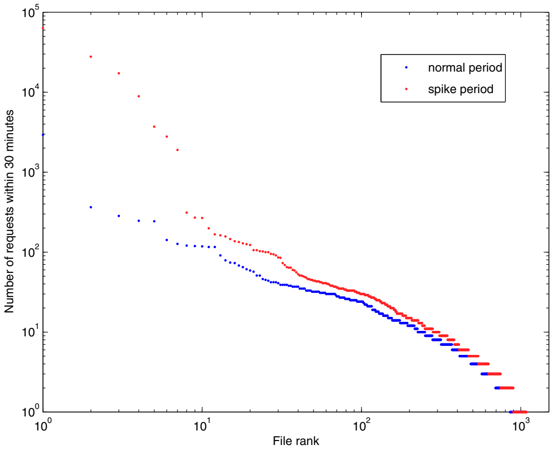

Fig. 2. Popularity comparison of the stable interval and bursty interval on the load in Figure 1.

While we can not predict these unexpected spikes in the workload, it is necessary to learn the nature of the burstiness and find out an efficient way to handle it once such events happen. The comparison of the data popularity distribution during one spike interval in Figure 1 and that in the normal interval right before the spike is shown in Figure 2. Clearly, the bursty workload can be seen as two parts: a base workload similar to the workload in the previous normal period, and a flash crowd load that is caused by a few very popular data items (video clips). Actually this phenomenon is not

[2] The measured workload was the number of video streams served per hour on Yahoo! Video site. For further details, please refer to [7].

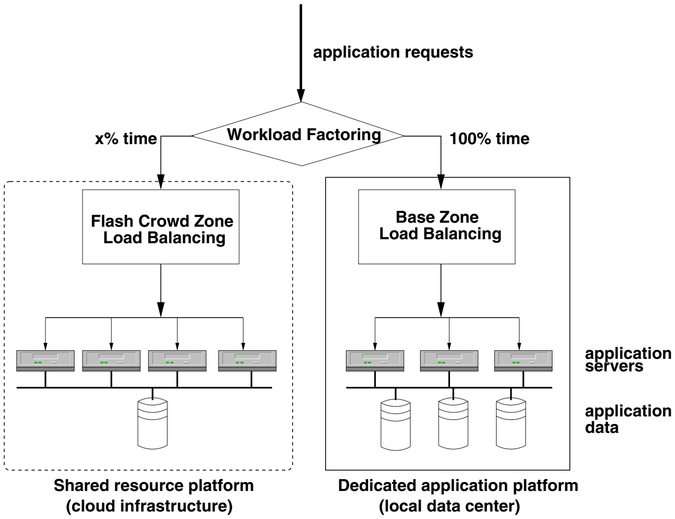
application requests
Workload Factoring
x% time
100% time
Flash Crowd Zone Load Balancing
Base Zone Load Balancing
application servers
application data
Shared resource platform (cloud infrastructure)
Dedicated application platform (local data center)

Fig. 3. Application hosting platform architecture in the hybrid Cloud Computing Model.

limited to our measurement; it is a typical pattern in flash crowd traffic and explained through *slash-dot effect* (or *Digg effect* et al). This pattern indicates that we may not need complicated workload dispatching scheme for the flash crowd load part because most requests will be for a small number of unique data items. As the flash crowd load has extremely high content locality, the operator can make the best of data caching and simply provision extra servers with the best-scenario capacity estimation based on maximal cache hit ratio (much higher server capacity than that with low cache hit ratio, as we will show in Section V-B2). The challenge of workload management here lies in the responsiveness of workload decomposition upon changing data popularity distribution.

B. Architecture

Figure 3 shows the application hosting platform architecture in the proposed hybrid Cloud Computing Model. It includes two resource zones: a *base zone* which is a dedicated application platform in a local data center, and a *flash crowd zone* which is an application platform hosted on a cloud infrastructure. The base zone runs all the time and processes the base load of the application. As the base load volume does not vary dramatically after removing the sporadic spikes, the local data center is expected to run in a compact and highly utilized mode even though small-margin resource overprovisioning is necessary for application QoS guarantee (Section V-A3 gives evaluation results on this argument). The flash crowd zone is provisioned on demand and expected to be on for transient periods. While the resource in the flash crowd zone might have to be overprovisioned at a large factor (e.g., several times larger than the compute resources provisioned in the base zone), it is expected to be utilized only in rare time ($X\%$ of the time, e.g., 5%). Each resource zone has its own load balancing scheme in managing the separated workload, and we do not discuss it further in this paper.

At the entry point lies the workload factoring component. The design goals of the workload factoring component include two: 1) smoothing the workload dynamics in the base zone application platform and avoiding overloading scenarios through load redirection; 2) making flash crowd zone application platform agile through load decomposition not only on the volume but also on the requested application data. By selectively dispatching requests for similar (hot) data objects into the flash crowd zone, the workload factoring scheme aims at minimizing the resulting application data cache/replication overhead. This will bring multiple benefits on the architecture and performance of the flash crowd zone platform:

- with only a small set of active application data accessed, the data storage component at the flash crowd zone can be designed as a data cache and decoupled from that at the base zone; therefore, the former can be a floating platform and does not have to be tied to the latter through shared physical resources (e.g., shared SAN, which otherwise has to be provisioned for the peak load).
- with only a small set of active application data served, application servers can reduce their warm-up time significantly with a cold cache, and therefore speedup the overall dynamic provisioning process.
- with only a small set of active application data cached, application servers can maximize their capacity with high cache hit ratio. Section V-B3 gives some evaluation results on this argument.
- with only a small set of active application data requested, simple load balancing schemes like random or round-robin can perform as well as more complicated schemes exploiting content locality such as job-size-based dispatching [10].

We will describe the workload factoring scheme in details in Section III.

C. Discussion

While the motivation of the hybrid cloud computing model originates from dynamic workload management, it addresses many concerns on the full Cloud Computing model where customers completely rely on public cloud services for application hosting. For example, enterprise IT legacy infrastructures do not need to be abolished and instead be powered with the capability to handle the long tail of their workload; public cloud service availability is no longer so critical with the sporadic utilization (a two-9s availability translates into a four-9s if load spikes are defined as the 1-percentile peak load); data security & privacy concerns will not be severe as application data are only cached temporarily on public clouds for a short time; data transfer bottlenecks can be largely alleviated as only a small portion of the application data is replicated on the public cloud.

III. Intelligent Workload Factoring

A. Problem Model

We model the general workload factoring process as a hypergraph partition problem [11]. Each object (e.g., a video clip or a DB table) in the application data [3] is modeled as

[3] The definition of data objects is specific to applications. For example, in video streaming the data items are video clips, in web services the data items can be URLs, HTML pages, database tables, or even database table entries.

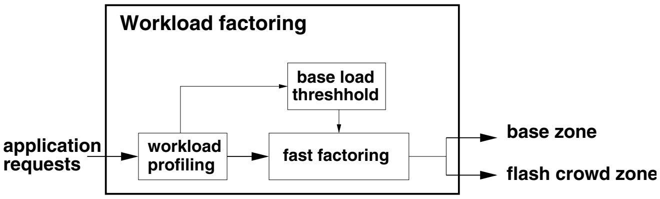

Fig. 4. Logic view of Workload Factoring Component.

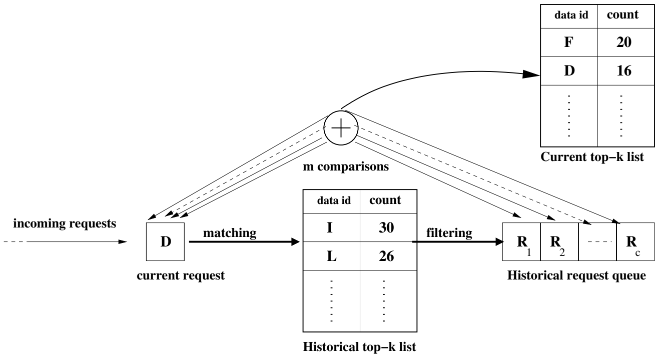

Fig. 5. Schematic description of fastTopK algorithm.

a vertex, each service request type (e.g., all stream requests for a video clip x or all HTTP requests of the "shopping cart"interaction category) is modeled as a net, and a link between a net and a vertex shows the access relationship between the request type and the application data object. This leads to the definition of a hypergraph $H = (V, N)$ where V is the vertex set and N is the net set; the weight w_i of a vertex $v_i \in V$ is the expected workload caused by this data object, which is calculated as its popularity multiplied by average workload per request to access/process this data; another weight s_i of a vertex $v_i \in V$ is the data size of this object; the cost c_j of a net $n_j \in N$ is the expected data access overhead caused by this request type, which is calculated as the expected request number of this type multiplied by the sum of its neighboring data objects' size.

The K-way hypergraph partition, an NP-hard problem [11], is to assign all vertexes (data objects) to K (K=2 in our case) disjoint nonempty locations without the expected workload beyond their capacities, and achieve minimal partition cost

$$Min(\sum_{n_j \in N_E} c_j + \gamma \sum_{v_i \in V_{flashcrowd}} s_i)$$

where $\sum_{n_j \in N_E} c_j$ is the net cut cost (the total weights of the nets that span more than one location, therefore bringing remote data access/consistency overhead); $\sum_{v_i \in V_{flashcrowd}} s_i$ is the total size of the data objects in the flash crowd zone and represents the data transfer/replication overhead; γ is a factor to assign different weights on the two overhead components.

There are fast partition solutions proposed like the bisection partition scheme [11]. For video streaming services where request-data relationship is simple and there is no net cut as one request accesses only one data item, the partition problem degenerates to the knapsack problem where our greedy scheme is moving vertexes from the base zone one by one ranked by their popularity until reaching the flash crowd zone's capacity. This is equal to redirect the requests for the most popular data items in a top-k list into the flash crowd zone, and the remaining question is on how to quickly generate the correct top-k list during a popularity transition time disturbed by the workload burst. Next we give the details of the workload factoring process.

B. Logic view

As shown in Figure 4, the intelligent workload factoring (IWF) scheme has three basic components: workload profiling, based load threshold, and fast factoring. The workload profiling component updates the current system load upon incoming requests, and compares it to the base load threshold to decide if the system is in a *normal* mode or a *factoring* mode. The base load threshold specifies the maximum load that the base zone can handle; it may be manually configured by operators according to the base zone capacity, or automatically set based on the load history information (e.g., the 95-percentile arrival rate) which then will also input into the base zone for resource provisioning decision. When the current system load is not higher than the base load threshold, the fast factoring process is in the "normal" mode, and it simply forwards incoming requests into the base zone. When the current system load is higher than the base load threshold, it is in the *factoring* mode and queries a fast frequent data item detection algorithm to check if an incoming request asks for data in a set of hot data objects; if yes, this request is forwarded to the flash crowd zone; otherwise, it is forwarded to the base zone.

C. Fast frequent data item detection algorithm

We call the fast frequent data item detection algorithm *FastTopK*. As shown in Figure 5, it has the following data structures: a FIFO queue to record the last c requests, a list to record the current top-k popular data items, a list to record the historical top-k popular data items, and a list of counters to record the data item access frequency. Given a request r, the algorithm outputs "base" if r will go to the base zone, and "flash crowd" otherwise. It works as following:

1) if the system is in the "normal" mode, the historical top-k list is always set as empty; go to step 4).
2) if the system is in the "factoring" mode and r is the first request since entering this mode, we 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), we increase 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, we 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), we increase 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, the algorithm always answers "base".
6) In the "factoring" mode, the algorithm combines the two top-k lists by calculating the estimated request rate

of each data item: for each item in the historical top-k list, the rate is its frequency counter value divided by the total requests arrived since entering the "factoring" mode; for each item in the current top-k list, the rate is given in Theorem 1.

7) if r's data item is in the top k of the $2k$ joint items, the algorithm answers "flash crowd", otherwise it answers "base".
8) if r's data item does not belong to the historical top-k list, the algorithm adds r into the FIFO queue for request history, and returns.

The key ideas in the fastTopK algorithm for speeding up frequent data item detection include two: speeding up the top-k detection at changing data popularity distributions by pre-filtering old popular data items in a new distribution, and speeding up the top-k detection at a data popularity distribution by pre-filtering unpopular data items in this new distribution.

IV. Analysis

In this section, we present the performance analysis results of the FastTopK algorithm.

A. Accuracy Requirement and Performance Metric

The correctness of the fastTopK algorithm relies on the accuracy of the frequency counter information, which is the estimation of the request rates on the corresponding data items. Formally, for a data item T, we define its actual request rate

$$p(T) = \frac{\text{total requests for T}}{\text{total requests}}.$$

FastTopK will determine an estimate $p(\hat{T})$ such that $p(\hat{T}) \in \Big(p(T)(1-\frac{\beta}{2}), p(T)(1+\frac{\beta}{2})\Big)$ with probability greater than α. For example, If we set $\beta = 0.01$, $\alpha = 99.9\%$, then the accuracy requirement states that with probability greater than 99.9%, FastTopK will estimate $p(T)$ with a relative error range of 1%. We use Z_α to denote the α percentile for the unit normal distribution. For example, if $\alpha = 99.75\%$, then $Z_\alpha = 3$.

Given the specified accuracy requirement, we measure the performance of the algorithm through

- *Sample Size:* Sample size is defined to be the number of request arrivals needed to perform the estimation. We use the term *estimation time* and sample size interchangeably.

B. Request Rate Estimation

We assume that request arrivals to the system follow an independent and identically distributed ($i.i.d.$) process. The following result is used to estimate the request rate p_f from the frequency counter value of a data item f.

Let $M_k(N,T)$ represent the frequency counter value for the target data item T after N arrivals for fastTopK with k comparisons. We label the requests 1 to N based on the arrival sequence. Let

$$C_{ij}(f) \quad = \quad \begin{cases} 1 & \text{both requests } i \text{ and } j \text{ ask for data item } f \\ 0 & \text{otherwise} \end{cases}$$

Clearly, the frequency counter of data item f will be increased by 1 when $C_{ij}(f) = 1$. We call it a *coincidence* in the rest of the paper. Therefore,

$$M_k(N,f) = \sum_{i \le N} \sum_{j=i-k}^{i-1} C_{ij}(f).$$

Before we study the correlation structure of the comparisons, we first state the following elementary results.

Let $C_{ij}(f)$ be as defined above. Then

$$E[C_{ij}(f)] = p_f^2$$

and

$$Var[C_{ij}(f)] = p_f^2(1-p_f^2).$$

The results follow directly from the assumption that arrivals are independent and that the probability that an arrival request asks for data f is p_f.

In FastTopK, the comparisons are not always independent of each other. To see this, let's use the comparisons $C_{ij}(f)$ and $C_{im}(f)$ $(i \neq j \neq m)$ as an example. Note that $P(C_{ij}(f) = 1) = p_f^2$ due to the independence of arrivals. But $P(C_{im}(f) = 1 | C_{ij}(f) = 1) = p_f$ because the conditioning already implies that request i asks for data f. In general, for any pair of comparisons $C_{ij}(f)$ and $C_{lm}(f)$, they are independent if and only if all the four indices are distinct. If any two of the indices are identical, then the comparisons are dependent. For example, $C_{ij}(f)$ and $C_{im}(f)$ are dependent. The next result gives the correlation between the random variables $C_{ij}(f)$ and $C_{im}(f)$.

Lemma 1 *Consider $C_{ij}(f)$ and $C_{im}(f)$ for $i-k \le j, m \le i-1$. Then*

$$Cov(C_{ij}(f), C_{xy}(f)) = p_f^3(1-p_f).$$

Proof: Let $\tau = Cov(C_{ij}(f), C_{im}(f))$. Then,

$$\begin{aligned} \tau &= \mathbf{E}[(C_{ij}(f) - \mathbf{E}[C_{ij}(f)])(C_{im}(f) - \mathbf{E}[C_{im}(f)])] \\ &= \mathbf{E}[(C_{ij}(f) - p_f^2])(C_{im}(f) - p_f^2)] \\ &= p_f^4 - 2p_f^2\mathbf{E}[(C_{ij}(f)] + \mathbf{E}[C_{ij}(f)C_{im}(f)] \\ &= \mathbf{E}[C_{ij}(f)C_{im}(f)] - p_f^4 \\ &= p_f^3(1-p_f) \end{aligned}$$

where the third equality follows from the fact that

$$\mathbf{E}[C_{ij}(f)] = \mathbf{E}[C_{im}(f)].$$

The last step follows from the fact that C_{ij} and C_{im} are both one if and only if requests i, j and m all ask for data item f, which happens with probability p_f^3. ■

Following the above lemma, the expectation and variance of $M_k(N,f)$ is:

Lemma 2 *Let $M_k(N,f)$ denote the the number of coincidences for data item f after N arrival requests to the system. Then*

$$\mathbf{E}[M_k(N,f)] = NkP_f^2$$

and

$$Var[M_k(N,f)] = Nkp_f^2(1-p_f^2)\left[1 + \frac{2(2k-1)p_f}{(1+p_f)}\right].$$

Proof: Note that

$$\begin{aligned}\mathbf{E}[M_k(N,f)] &= \mathbf{E}[\sum_{i\leq N}\sum_{j=i-k}^{i-1} C_{ij}(f)] \\ &= Nkp_f^2.\end{aligned}$$

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 results in a coincidence for data item f. The variance can be computed as follows:

$$\begin{aligned}
&Var[M_k(N,f)] = \mathbf{E}[M^2(N,f)] - (\mathbf{E}[M_k(N,f)])^2 \\
&= \mathbf{E}[M^2(N,f)] - (NkP_f^2)^2 \qquad (1)\\
&\mathbf{E}[M^2(N,f)] \\
&= \mathbf{E}[\sum_{i=1}^{Nk}(I_i^2(f)) + \sum_{i=1}^{Nk}\sum_{j:1\leq j\leq Nk, j\neq i} I_i(f)I_j(f)] \qquad (2)\\
&\mathbf{E}[\sum_{i=1}^{Nk}(I_i^2(f))] = Nkp_f^2 \\
&\mathbf{E}[\sum_{i=1}^{Nk}\sum_{j:1\leq j\leq Nk, j\neq i} I_i(f)I_j(f)] \\
&= \mathbf{E}[\sum_{i=1}^{Nk}(\sum_{j:1\leq j\leq Nk, j\neq i}^{cov(I_i(f),I_j(f))\neq 0} I_i(f)I_j(f) \\
&+ \sum_{l:1\leq l\leq Nk, l\neq i}^{cov(I_i(f),I_l(f))=0} I_i(f)I_l(f))] \qquad (3)\\
&= \sum_{i=1}^{Nk}(2(2k-1)p_f^3 + (Nk-1-2(2k-1))p_f^4) \\
&= Nk((Nk-1)p_f^4 + 2(2k-1)(p_f^3 - p_f^4)) \\
&Therefore, \\
&Var[M_k(N,f)] = Nk[p_f^2 - p_f^4 + 2(2k-1)(p_f^3 - p_f^4)] \\
&= Nk(p_f^2 - p_f^4)(1 + \frac{2(2k-1)p_f}{(1+p_f)}).
\end{aligned}$$

■

Now we know the mean and the variance for the number of coincidences, we use the central limit theorem to obtain a normal approximation for the number of coincidences and then use the result to estimate the request rates. The next theorem gives the expression for the estimator of the rate along with its variance.

Theorem 1

$$\sqrt{Nk}\left[\sqrt{\frac{M_k(N,f)}{Nk}} - p_f\right] \sim \mathcal{N}\left[0, \sigma_f^2\right]$$

where

$$\sigma_f^2 = \frac{(1-p_f^2)(1+\frac{2(2k-1)p_f}{(1+p_f)})}{4} \qquad (4)$$

Proof: Though the comparisons are not independent, the comparisons are a stationary k^2-dependent sequence with finite expectation and variance. Following the central limit theorem for dependent sequences [12], we can show that for large N,

$$\sqrt{Nk}\left[\frac{M_k(N,f)}{Nk} - p_f^2\right] \sim \mathcal{N}\left[0, \delta_f^2\right]$$

where

$$\delta_f^2 = p_f^2(1-p_f^2)\left[1 + \frac{2(2k-1)p_f}{(1+p_f)}\right]. \qquad (5)$$

The above result can be shown as in Theorem 5 of [13]. ■

Theorem 1 says that in fastTopK, the estimated request rate is $\sqrt{\frac{M_k(N,f)}{Nk}}$ for the data item f.

C. Estimation Time

The historical top-k list serves as a filter on the requests entering the FIFO queue. We call fastTopK a basic version when without using the historical top-K list (such as in the normal mode), and that actively using the historical top-K list as a filtering version. For the estimation time of basic FastTopK,we have the following result:

Lemma 3 *Given the accuracy requirement (α, β) on p_f described in Section IV-A, and N_{basic}^C be the number of arrivals required for basic fastTopK,*

$$N_{basic}^C = \frac{(4k-1)Z_\alpha^2}{(kp_f\beta)^2}.$$

Proof: First, we consider the variance of the estimated request rate and derive the upper bound on its value. This upper bound on the variance holds in the entire $[0,1]$ range and is a function of k and p_f. As

$${\sigma_f}^2 = \frac{(1-p_f^2)(1+\frac{2(2k-1)p_f}{(1+p_f)})}{4},$$

setting the derivative of the variance with respect to p_f to zero gives us

$${\sigma_f}^2 \leq \frac{k^2}{4k-1}.$$

The above bound on the variance can now be used to compute the sample size given the accuracy requirement. Let $(p_f\beta)$ be the desired estimation accuracy and Z_α the desired α-percentile. To achieve the accuracy requirement,

$$\frac{Z_\alpha^2}{N(p_f\beta)^2} \leq \frac{k^2}{4k-1}.$$

Therefore, the minimum sample size N in order to satisfy the accuracy requirement is $\frac{(4k-1)Z_\alpha^2}{(kp_f\beta)^2}$.

■

Now, let us define an amplification factor X for the rate change of a data item f before and after the historical top-K filtering as

$$X = \frac{p_f^{after}}{p_f^{before}}$$

For example, if a data item takes 0.1% of the total requests, and takes 0.2% of the requests filtered with the historical top-K data items, the amplification factor $X = 2$ for this data item. We have the speedup result of the fastTopK algorithm given the rate amplification factor X.

TABLE I
ECONOMICAL COST COMPARISON OF THREE HOSTING SOLUTIONS

Hosting solution	Annual cost
local data center	running a 790-server DC
full cloud computing	$US\$1.384$ millions
hybrid Cloud Computing	$US\$58.96K$ + running a 99-server DC

Theorem 2 *Given the accuracy requirement* (α, β) *on* p_f *described in Section IV-A,* N^C_{basic} *be the number of arrivals required for basic fastTopK, and* $N^C_{filtering}$ *be the number of arrivals required for filtering fastTopK,*

$$N^C_{filtering} = \frac{N^C_{basic}}{X^2}.$$

Proof: Following Lemma 3,

$$N^C_{basic} = \frac{(4k-1)Z^2_\alpha}{(kp^{before}_f \beta)^2}$$

$$N^C_{filtering} = \frac{(4k-1)Z^2_\alpha}{(kp^{after}_f \beta)^2}$$

$$\frac{N^C_{basic}}{N^C_{filtering}} = \frac{(p^{after}_f)^2}{(p^{before}_f)^2} = X^2$$

■

Theorem 2 shows that we have a X^2 speedup of the detection process with a X-factor on rate amplification due to request filtering based on historical information. For example, if a data item takes 0.1% of the total requests, and takes 0.2% of the requests filtered with the historical top-K data items, the estimation time will be reduced by 4 times to accurately estimate its request rate with the frequency counter.

V. EVALUATION

Through the evaluation, we aim to answer the following questions:

1) What is the economical advantage of application hosting solutions based on the hybrid cloud computing model?
2) What is the benefit on the base zone workload management with the intelligent workload factoring (IWF) scheme?
3) What is the benefit on the flash crowd zone resource management with the IWF scheme?
4) What is the performance of the IWF scheme upon a changing data popularity distribution?

For the first two questions, we rely on trace-driven simulations to evaluate the hybrid cloud computing model in a large-scale system setting. For the rest two questions, we rely on testbed experiments to evaluate the IWF scheme in a dynamic workload setting.

A. Trace-driven simulations

1) Yahoo! Video workload traces: We use the Yahoo! Video workload traces presented in [7], and it contains a total of 32,064,496 video stream requests throughout the collection period of 46 days. The hourly request arrival rates are shown in Figure 1.

2) Economical cost comparison of three hosting solutions: We compare three application hosting solutions to host the measured Yahoo! Video stream load [4]:

- Local data center solution. In this solution, a local data center is overprovsioned over the peak load in the measurement.
- Full cloud computing solution. In this solution, a rented platform on Amazon EC2 infrastructure is provisioned over the peak load in the measurement. The rent price is $US\$0.10$ per machine hour based on the Amazon EC2 pricing policy [1] at the time when the paper was written.
- Our hybrid cloud computing model based solution. In this solution, a local data center is provisioned over the 95-percentile workload, and an Amazon EC2 based platform is rented on demand for the top 5-percentile workload.

Table I shows the annual economical cost of the three solutions when we repeated the 46-day workload through one year. For the presentation simplicity, we use a simple cost model which only includes the server cost even though there are many other cost factors such as bandwidth, storage, power, cooling, physical plant, and operation costs.

As we can see, the local data center solution requires the running of a 790-server medium-sized infrastructure. The full cloud computing solution, seeming cheap on the 10-cent unit price, results in a bill of millions of dollars simply for computing cost. Lastly, our hybrid cloud computing model offers an economical solution with a 99-server small-sized local data center (affordable even for most SMB customers) and an annual bill of only $US\$58.96K$ for handling sporadic load spikes.

Nowadays content providers on the Internet rely on content distribution networks (CDNs) to leverage their presence across different geographical locations to serve video content, and lower the TCO (Total Cost of Ownership). While we do not include the CDN solution into the Cloud-based cost comparison, we note that some of these systems offer quite flexible rules to split CDN traffic among multiple CDNs; there are many CDN load balancers commercially available, including Level 3 intelligent traffic management [14], Akamai Cotendo CDN balancer [15], and LimeLight traffic load balancer [16]. However, a missing component of these existing systems is the algorithm to compute the allocation automatically (e.g., the percentages and/or the served content) [17]. Our IWF scheme can be applied to automatically configure these systems, where the CDNs serve the same purpose as the flash crowd zone defined in our hybrid Cloud Computing model does.

3) Workload smoothing effect in the base zone: We define the load spikes as the top 5-percentile data points in terms of request rates in Figure 1. By removing them with the workload factoring mechanism, we observed that the ratio of the maximum load to the average load was reduced to 1.84 in the base zone, where overprovisioning over peak load became a reasonable choice. We also applied statistical techniques on the short-term workload prediction in the base zone. Figure 6 shows the CDF of the prediction error using a simple heuristic: it uses the arrival rate from last interval as the

[4] In all hosting solutions, we assume the capacity of a video streaming server is 300 (i.e., supports 300 concurrent streaming threads).

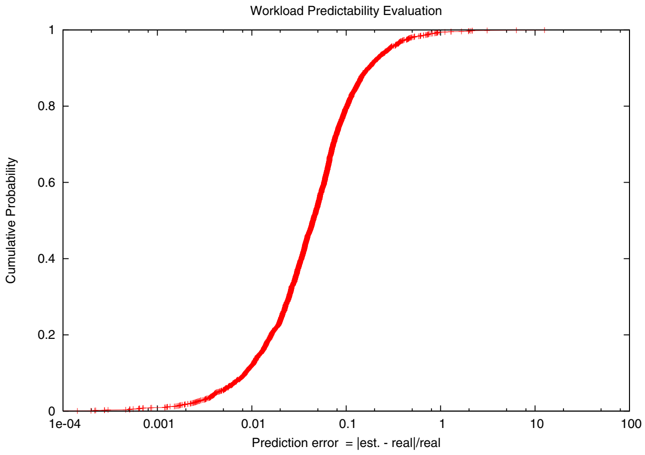

Fig. 6. Prediction error with a simple heuristic on the base load.

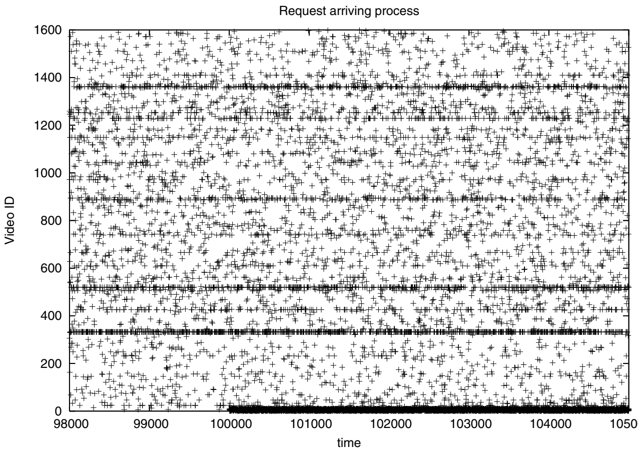

Fig. 8. Incoming streaming requests.

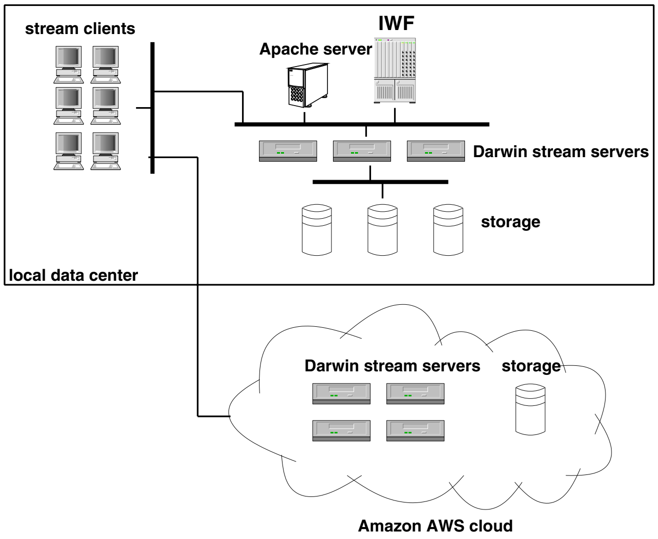

Fig. 7. The video streaming service testbed with a hybrid platform.

prediction of the following interval's arrival rate, with an 30-minute interval length. It turned out that 82% of the time the prediction had an error no more than 10%; 90% of the time the prediction had an error no more than 17%; Therefore, simple statistical prediction techniques with small margin factor on the predicted value could be reliable for dynamic provisioning in the base zone.

B. Testbed experiments

1) Testbed: We set up an application testbed which hosts YouTube-like video streaming service. The testbed consists of two parts: a local data center and a platform set up at Amazon AWS infrastructure utilizing the EC2 and S3 services. In the local data center, 20 open source Darwin streaming servers [18] are provisioned all the time, while the streaming server instances at Amazon EC2 are activated on demand.

The IWF component was implemented as a load controller for the streaming workload. 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 IWF for which streaming 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.

The load controller also contains the implementation of two well-known load balancing algorithms [19]: the Least Connections balancing algorithm for the local server farm and the Round-Robin load balancing algorithm for the server farm on Amazon Cloud.

We developed a distributed workload generator based on openRTSP [20] to generate real video streaming load. Depending on the test scenarios, up to 20 machines were provisioned for client-side workload generation.

2) Methodology: We evaluate the fast 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 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 $\frac{D_3}{D_1+D_3}$.

Figure 8 shows one example traces where D_1 is Zipf and D_3 is uniform distributions, and the volume ratio is $1 : 1$. One data point at the coordinate (x, y) in the graph represents one streaming request asking for the video file with ID y at time x. The changing time $t = 100000$ when a load spike on a few hot data items jumped in.

We compared IWF 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 ChoKe active queue management scheme [21]. While ChoKe was originally proposed for approximating fair bandwidth allocation, it is a reasonable candidate for

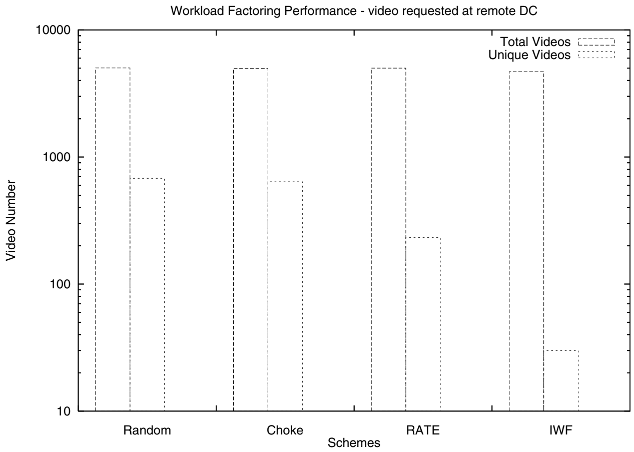

Fig. 9. IWF performance: D_1 - Zipf distribution, D_3 - uniform distribution.

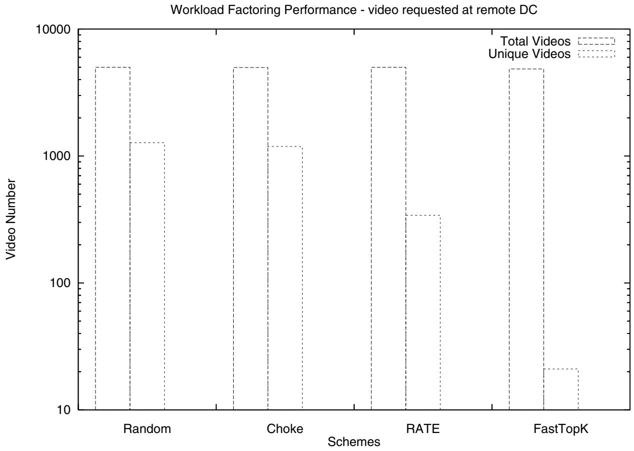

Fig. 10. IWF performance: D_1 - uniform distribution, D_3 - uniform distribution.

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 IWF except that it uses the RATE scheme [22] to detect the top-K data items.

3) Results: We generated one load trace over a video library of 1600 unique video clips, which all have video bit rate of $450Kb/s$. D_1 is Zipf distribution with $\alpha = 1$ and D_3 is uniform distribution, and the volume ratio is $1:1$. One dot at the coordinate (x, y) in Figure 8 represents that one streaming request asking for video file y arrives at time x. The changing time $t = 100000$ when a load on a few hot data items jumped in. Figure 9 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 factoring algorithms dispatched the same amount of requests into the flash crowd zone, IWF outperformed the other three significantly in terms of unique video files requested (two orders of magnitudes compared to random dispatching); in the flash crowd zone totally 5000 streams were served on only 30 unique video clips.

In Figure 10 we used another trace where both D_1 and D_3 are uniform distributions, and the volume ratio is $1:1$. In this case IWF's performance still outperformed the other three schemes; actually, it was quite close to the optimal performance (21 vs 10) in terms of unique video clips requested.

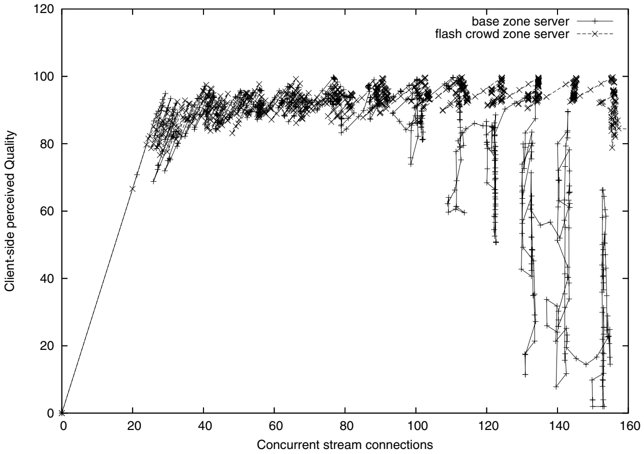

Fig. 11. IWF performance: client-side perceived streaming quality at two zones.

Figure 11 shows the client-side perceived streaming quality from a base zone server and a flash crowd zone server in the above workload factoring experiment. For fair comparison, in this case both servers have the same hardware/software configuration and reside in the local testbed. The client-side perceived streaming quality is a metric reported by the Darwin Stream server itself; it has a score between 0 and 100 to mainly reflect the packet loss ratio during the streaming process. We see when the concurrent connections went up, the flash crowd zone server delivered more reliable streaming quality than the base zone server. It could support up to 160 concurrent streaming connections while keeping the client-side quality at above 80, while the base load zone server could only support around 90 concurrent streaming connections to keep the client-side quality steadily at above 80. In the testbed configuration, we set the base zone server capacity at 90 concurrent connections and that for flash crowd zone servers at 160 (78$\%$ higher than that in the base zone), and enforce it during dispatching.

VI. Related Work

In November 2008, Amazon launched CloudFront [23] 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 [24], 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 a flexible workload management scheme working for both models, and our workload factoring technology is proposed as one answer for it.

Berkeley researchers [25] offer a ranked list of obstacles to the growth of Cloud Computing. Similar to the points we made in the introduction, the concerns on public cloud computing services include service availability, data confidentiality and auditability, performance unpredictability, and so on. While some concerns could be addressed technically, some are due to physical limitations naturally. On the service architecture level, we believe a hybrid cloud computing model makes sense to enterprise IT and can eliminate (or significantly alleviate) many issues raised from a full Cloud Computing model.

In Content Distributed Network (CDN) and web caching, workload factoring happens between a primary web server and proxy servers. The typical method is DNS redirecting and the workload factoring decision is predefined manually over a set of "hot" web objects. Automatic solutions [26] [27] [28] exist for cache hit ratio improvement through locality exploration, and their focus is on compact data structure design to exchange data item access information.

Many content publishers, such as Netflix, Hulu, Microsoft, Apple, Facebook, and MSNBC, use multiple CDNs to distribute and cache their digital content [17]. This allows them to aggregate the diversity of individual CDN providers on features, resources, and performance. Accordingly, two new strategies, telco-CDN federation and hybrid P2P-CDN, are emerging to augment existing CDN infrastructures [29]. Telco-CDN federation is based on the recent development among various CDNs operated by telecommunication companies to federate by interconnecting their networks, ensure better availability, and benefit the participating ISPs in terms of provisioning costs [30], [31]. A hybrid strategy of serving content from dedicated CDN servers using P2P technology provides the scalability advantage of P2P along with the reliability and manageability of CDNs [32]–[34]. Like the traditional CDN infrastructures, workload factoring happens between a primary server and proxy servers. Our IWF scheme can be applied to automatically configure these systems, and decide the load allocation (e.g., the percentages and/or the served content) for the primary server and proxy servers.

For fast frequent data item detection in data streams, many schemes have been proposed for fast rate estimation in traffic monitoring [22] [35] [36], and fast data item counting in CDN [37]; their focus is on compact data structure design to memorize request historical information at a static distribution.

VII. Conclusion

In this paper, we present the design of a hybrid cloud computing model. With the proposed proactive workload management technology, the hybrid cloud computing model allows users to develop a new architecture where a dedicated resource platform runs for hosting base service workload, and a separate and shared resource platform serves flash crowd peak load. Given the elastic nature of the cloud infrastructure, it creates a situation where cloud resources are used as an extension of existing infrastructures.

References

[1] "Amazon web services," http://aws.amazon.com/.

[2] "Google app engine," http://code.google.com/appengine/.

[3] C. Goolsbee, "Don't buy cloud computing hype: business model will evaporate," in www.searchdatacenter.com, 2008.

[4] "Massive (500) Internal Server Error.outage started 35 minutes ago," Feburary 2008. Available: http://developer.amazonwebservices.com/connect/message.jspa?messageID=79978#79978

[5] "Youtube," http://www.youtube.com.

[6] "Gigaspaces," http://www.gigaspaces.com.

[7] X. Kang, H. Zhang, G. Jiang, H. Chen, X. Meng, and K. Yoshihira, "Measurement, modeling, and analysis of Internet video sharing site workload: a case study," in *Proc. 2008 IEEE International Conference on Web Services*, pp. 278–285.

[8] "Yahoo! video," http://video.yahoo.com.

[9] "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. Available: http://www.comscore.com/press/release.asp?press=1678

[10] M. Harchol-Balter, M. E.Crovella, and C. D. Murta, "On choosing a task assignment policy for a distributed server system," pp. 231–242, 1998.

[11] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. 1999 ACM/IEEE Conference on Design Automation*, pp. 343–348.

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

[13] M. S. Kodialam, T. V. Lakshman, and S. Mohanty, "Runs based traffic estimator (rate): a simple, memory efficient scheme for per-flow rate estimation," in *2004 INFOCOM*.

[14] "Level 3 intelligent traffic management," http://www.level3.com/en/products-and-services/data-and-internet/cdn-content-delivery-network/.

[15] "Akamai cotendo cdn balancer," http://www.akamai.com/cotendo.

[16] "Limelight traffic load balancer," http://test.limelight.com/traffic-load-balancer/.

[17] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian, "Optimizing cost and performance for content multihoming," in *Proc. 2012 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 371–382.

[18] "Darwin streaming server," http://developer.apple.com/darwin/projects/streaming/.

[19] M. Arregoces and M. Portolani, *Data Center Fundamentals*. Cisco Press, 2003.

[20] "openrtsp," http://www.live555.com/openRTSP/.

[21] R. Pan, B. Prabhakar, and K. Psounis, "Choke—a stateless active queue management scheme for approximating fair bandwidth allocation," in *Proc. 2000 IEEE INFOCOM*, vol. 2, pp. 942–951. Available: http://dx.doi.org/10.1109/INFCOM.2000.832269

[22] F. Hao, M. Kodialam, T. V. Lakshman, and H. Zhang, "Fast payload-based flow estimation for traffic monitoring and network security," in *Proc. 2005 ACM Symposium on Architecture for Networking and Communications Systems*, pp. 211–220.

[23] "Amazon," http://aws.amazon.com/cloudfront/.

[24] "Vmware cloud vservices," http://www.vmware.com/technology/virtual-datacenter-os/cloud-vservices/.

[25] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: a Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[26] E. Casalicchio, V. Cardellini, and M. Colajanni, "Content-aware dispatching algorithms for cluster-based web servers," *Cluster Computing*, vol. 5, no. 1, pp. 65–74, 2002.

[27] S. Jin and A. Bestavros, "Greedydual* web caching algorithm: exploiting the two sources of temporal locality in web request streams," in *Proc. 2000 International Web Caching and Content Delivery Workshop*, pp. 174–183.

[28] 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.*, vol. 33, no. 5, pp. 16–31, 1999.

[29] A. Balachandran, V. Sekar, A. Akella, and S. Seshan, "Analyzing the potential benefits of CDN augmentation strategies for Internet video workloads," 2013.

[30] "Cisco report on CDN federation—solutions for SPS and content providers to scale a great customer experience."

[31] D. Rayburn, "Telcos and carriers forming new federated CDN group called OCX (operator carrier exchange)," http://goo.gl/wUhXr.

[32] "Akamai netsession," http://www.akamai.com/client/.

[33] C. Huang, A. Wang, J. Li, and K. W. Ross, "Understanding hybrid CDN-p2p: why limelight needs its own red swoosh," in *Proc. 2008 International Workshop on Network and Operating Systems Support for*

Digital Audio and Video, pp. 75–80. Available: http://doi.acm.org/10.1145/1496046.1496064

[34] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, “Design and deployment of a hybrid CDN-p2p system for live video streaming: experiences with livesky,” in *Proc. 2009 ACM International Conference on Multimedia*, pp. 25–34.

[35] A. Kumar, M. Sung, J. Xu, and J. Wang, “Data streaming algorithms for efficient and accurate estimation of flow distribution,” in *Proc. 2004 ACM SIGMETRICS*.

[36] N. G. Duffield and M. Grossglauser, “Trajectory sampling for direct traffic observation,” *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 271–282, 2000.

[37] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston, “Finding (recently) frequent items in distributed data streams,” in *Proc. 2005 International Conference on Data Engineering*, pp. 767–778.

Hui Zhang is a senior researcher at NEC Laboratories America at Princeton, New Jersey. He received the B.Eng. degree in Electrical Engineering from Hunan University, China in 1996, the M.Eng. degree in Electrical Engineering from the Institute of Automation, Chinese Academy of Sciences, China, in 1999, and the Ph.D. degree in Computer Science from the University of Southern California in 2005. His research interests include next-generation data centers, Peer-to-Peer and overlay networks, design and analysis of algorithms.

Guofei Jiang is the Vice President of NEC Laboratories America at Princeton, New Jersey. He leads a large research group consisted of members from the global network of NEC R&D units. His group conducts fundamental and applied research in the areas of Big Data Analytics, Distributed System and Cloud Platforms, Software-defined Networking, and Computer Security. He has published over 100 technical papers and also has over 40 patents granted or applied. His inventions have been successfully commercialized as Award Winning NEC products and solutions, and have significantly contributed to NEC’s business.

Kenji Yoshihira received the B.E. in EE at University of Tokyo in 1996 and designed processor chips for enterprise computer at Hitachi Ltd. for five years. He employed himself in CTO at Investoria Inc. in Japan to develop an Internet service system for financial information distribution through 2002 and received the M.S. in CS at New York University in 2004. He is currently Associate Director, Solutions Incubation in NEC Laboratories America, Inc. in NJ.

Haifeng Chen received the BEng and MEng degrees in automation from Southeast University, China, in 1994 and 1997, respectively, and the Ph.D. degree in computer engineering from Rutgers University, New Jersey, in 2004. He has worked as a researcher in the Chinese National Research Institute of Power Automation. He is currently a senior researcher at NEC Laboratory America, Princeton, New Jersey. His research interests include data mining, autonomic computing, pattern recognition, and robust statistics