

Intelligent Workload Factoring for A Hybrid Cloud Computing Model

Hui Zhang*, Guofei Jiang*, Kenji Yoshihira*, Haifeng Chen* and Akhilesh Saxena*

*NEC Laboratories America

Princeton, NJ 08540

Emails: {huizhang,gfj,kenji,haifeng,saxena@nec-labs.com}

Abstract—Despite the hype, cloud computing still stays on IT managers’ watching list. The hindrances to the adoption of public cloud computing services into enterprise IT include service reliability, security & data privacy, regulation compliant requirements, and so on. To address those concerns, we propose a hybrid cloud computing model which enterprise customers 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.

An intelligent workload factoring service is designed for this hybrid cloud computing model. It enables federation between on- and off-premise infrastructures for hosting an Internet-based application, and the intelligence lies in the explicit segregation of *base* workload and *trespassing* workload, the two naturally different components in 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 *changing* application data popularity.

Through analysis and extensive evaluation with real-trace driven simulations and experiments on a hybrid testbed consisting of local compute platform and Amazon Cloud service platform [1], we showed that the workload factoring technology can enable reliable workload prediction in the base load zone (with simple statistical technique), achieve resource efficiency (e.g., 78% higher server capacity than that in base load zone) and reduce data cache/replication overhead (up to two orders of magnitude) in the trespassing load zone, and react fast (with an X^2 speed-up factor) to the changing application data popularity upon the arrival of load spikes.

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 computing infrastructure hosting multiple applications where IT management complexity is hidden and resource multiplexing leads to efficiency; more computing resources are allocated on demand to an application when its workload incurs more resource demand than it is currently allocated.

Despite the advantages in management simplification and utility billing model, cloud computing remain in doubt for enterprise IT adoption. The concerns on the current cloud computing services include service reliability, lack of Service Level Agreements, security & customer data privacy, government compliance regulation requirements, etc (please refer to [3] for an interesting discussion). 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 Amazon cloud services such as the S3 service outage on February 5, 2008 [4] refreshes 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 aggregate workload of an Internet application: *base* load and *trespassing* load, are under explicitly separate management. *Base* load refers to the smaller and smoother workload experienced by the application platform all the time, while *trespassing* load refers to the much larger but transient load spikes experienced at rare time (e.g., the 5%-percentile heavy load time). The resource platform for base load can be managed in the enterprise data center with economical capacity planning and high resource utilization, while the resource platform for trespassing load can be provisioned on demand through a cloud service by taking advantage of the elastic nature of the cloud infrastructure.

An intelligent workload factoring mechanism 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 planned in amount, and the trespassing load part incurs minimal cache/replication 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 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 showcase of the hybrid cloud computing model. It has a local cluster as the base load resource zone and the Amazon EC2 infrastructure [1] as the trespassing zone; the workload factoring scheme was implemented as a load controller to arbitrate the stream load distribution between the two zones. With extensive analysis, trace-driven simulations, and testbed experiments, we showed the workload factoring technology can enable reliable

workload prediction in the base load zone (with simple statistical technique), achieve resource efficiency (e.g., 78% higher server capacity than that in base load zone) and reduce data cache/replication overhead (up to two orders of magnitude) in the trespassing 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 trespassing 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 by the hybrid cloud computing architecture. Besides, 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 scheme in Section III, and analysis results of the fast frequent data item detection algorithm used by the workload factoring scheme at Section IV. Section V shows the evaluation results. We present the related work in Section VI, and conclude this paper with future work in 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 measured¹ during a 46-days period on Yahoo! Video [8], the 2nd largest U.S. online video sharing website [9]. Applying statistical analysis techniques including autoregressive integrated moving average (ARIMA) and Fourier Transfer analysis, we observed that there were clear periodic components (exemplified by the workload between July 23 and July 27) 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

¹The measured workload was the number of video streams served per hour on Yahoo! Video site. For further details, please refer to [7].

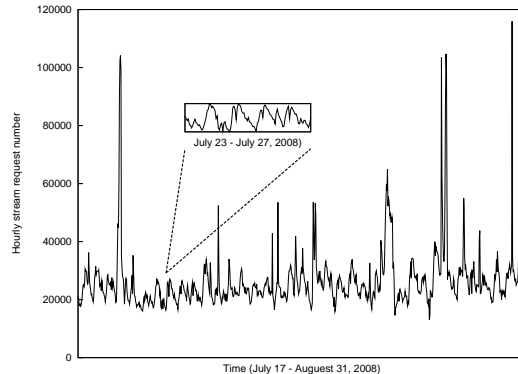


Fig. 1. Video stream workload evolution on Yahoo! Video Site.

is as high as 5.3 (12.9 if workload was measured in half an hour interval), which makes over-provisioning 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 with reliable prediction on the base workload seen most of the time, while reactive management deterministically response to sudden workload surges in rare time with the requirement of agile and responsive performance.

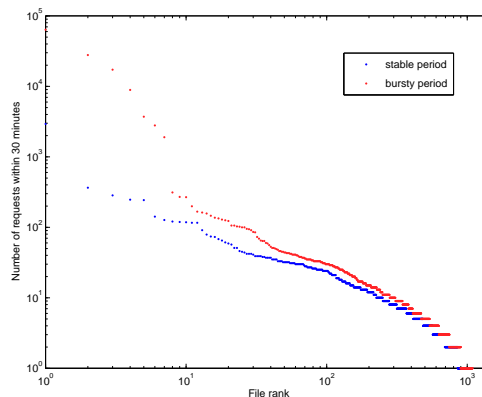


Fig. 2. Popularity comparison of the stable interval and bursty interval on the load in Figure 1.

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 event happens. The comparison of the data popularity distribution during one spike interval and that in the normal interval right before it in Figure 1 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 trespassing load that is caused by several very popular data items (video clips). Actually this phenomenon is not 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 trespassing load part because most requests will be for a small number of unique data items. As the trespassing 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 lie 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 *trespassing 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 trespassing zone is provisioned on demand and expected to be on for transient periods. While the resource in the trespassing zone might have to be overprovisioned at a large factor (e.g., several times larger than the compute resource 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 the details further in this paper.

At the entry 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 trespassing 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 trespassing 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 trespassing zone platform:

- with only a small set of active application data accessed, the data storage component at the trespassing 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 then 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 therefor 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 enterprise 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 on 1% of the time); 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.

When we refer the clouds to the public cloud services, our model can be applied to private clouds as well. When an enterprise data center has ample resources (servers) and a private cloud will be built for resource multiplexing through server consolidation, a hub-and-spoke infrastructure architecture can be designed based on the hybrid model. The base 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 statistical multiplexing effect is maximized since the trespassing loads dispatched by the workload factoring scheme are random in nature.

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 ² is modeled as 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

²The 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, database tables, or even interaction categories.

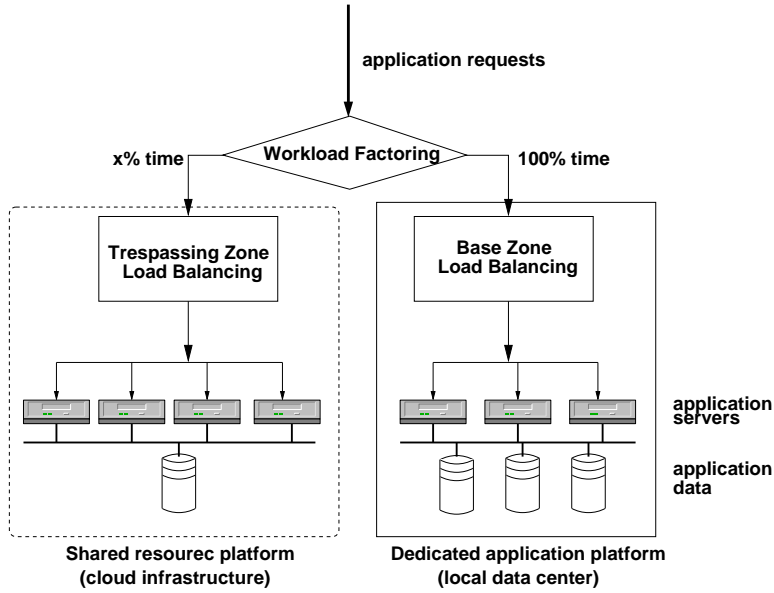


Fig. 3. Application hosting platform architecture in the hybrid Cloud Computing Model

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 vertices (data objects) to K ($K=2$ in our case) disjoint nonempty locations without the expected workload beyond their capacities, and achieve minimal partition cost

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

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); $\sum_{v_i \in V_{trespassing}} s_i$ is the total size of the data objects in the trespassing 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 bi-section 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 vertices from the base zone one by one ranked by their popularity until reaching the trespassing zone's capacity. This is equal to redirect the requests for the most popular data items in a top-k list into the trespassing 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 gives the details of the workload

factoring process.

B. Logic view

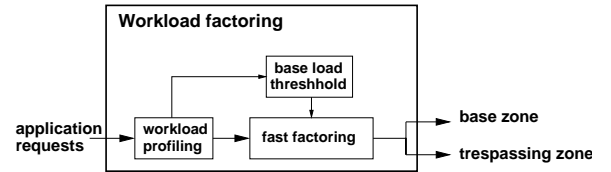


Fig. 4. Logic view of Workload Factoring Component.

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 to the base load threshold to decide if the system is in a *normal* mode or a *panic* 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 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 *panic* 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 trespassing zone; otherwise, it is forwarded to the base zone.

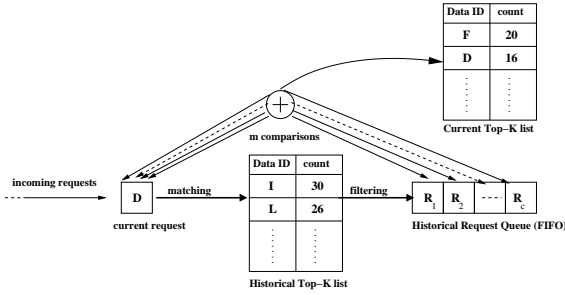


Fig. 5. Schematic Description of fastTopK algorithm

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 items access frequency. Given a request r , the algorithm outputs “base” if r will go to the base zone, “trespassing” otherwise. It works as following:

- 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: 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 “panic” mode; for each item in the current top- k list, the rate is given in Lemma 1.
- 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.

The key ideas in the fastTopK algorithm for speeding up frequent data item detection have 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 distribution.

IV. ANALYSIS

In this section, we present the performance analysis results of the FastTopK algorithm. The proofs are skipped due to page limit and are available in the companion technical report [12].

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 $p(T) = \frac{\text{total requests to } T}{\text{total requests}}$. FastTopK will determine an estimate $p(\hat{T})$ such that $p(\hat{T}) \in \left(p(T)(1 - \frac{\beta}{2}), p(T)(1 + \frac{\beta}{2})\right)$ 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 estimation 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 measured 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.

The following result is used to estimate the proportion $p(T)$ from the frequency counter value of a data item in the current top- K list.

Lemma 1 Let $M_k(N, T)$ represent the frequency counter value for the target data item T after N arrivals for fastTopK with k comparison. Then,

$$\sqrt{Nk} \left[\sqrt{\frac{M_k(N, T)}{Nk}} - p(T) \right] \sim \mathcal{N} [0, \sigma_T^2]$$

where

$$\sigma_T^2 = \frac{(1 - p^2(T))(1 + \frac{2(2k-1)p(T)}{(1+p(T))})}{4} \quad (1)$$

Proof: Follow Theorem 5 in [13]. ■

B. Estimation Time

Note the historical top- k list serves as a filter on the information 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 is a filtering version. For the estimation time of basic FastTopK, we have the following result:

Lemma 2 Given the accuracy requirement described in Section IV-A, and N^C be the number of samples required for basic fastTopK.

$$N_{basic}^C = \frac{3Z_\alpha^2}{k\beta^2}.$$

Proof: Follow the results in [13]. ■

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

$$X = \frac{r_{after}}{r_{before}}$$

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

Theorem 1 *Given the accuracy requirement described in Section IV-A, N_{basic}^C be the number of samples required for basic fastTopK, and $N_{filtering}^C$ be the number of samples required for filtering fastTopK.*

$$N_{filtering}^C = \frac{N_{basic}^C}{X^2}.$$

Therefore, we have a X^2 speedup of the detection process even with a X -factor on rate amplification due to historical information filtering.

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 trespassing 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 used 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.

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

TABLE I

ECONOMICAL COST COMPARISON OF THREE HOSTING SOLUTIONS

2) *Economical cost comparison: three hosting solutions:* We compared three application hosting solutions to host the measured Yahoo! Video stream load ³:

- Local data center solution. In this solution, a local data center is overprovisioned 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 current Amazon EC2 pricing policy [1].
- Our hybrid cloud computing model based solution. In this solution, a local data center is overprovisioned 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 only include the server cost even though there are many other cost factors such as bandwidth, storage, power, cooling, physical plant, and operations 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.

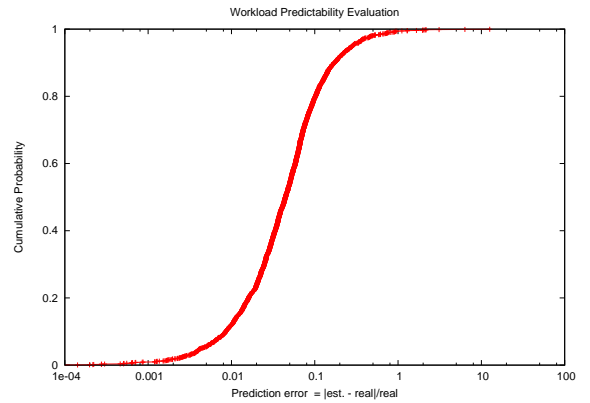


Fig. 6. Prediction error with a simple heuristic on the base load

3) *Workload smoothing effect in the base zone:* We defined the load spikes as the top 5-percentile data points in terms of request rate 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

³In all hosting solutions, we assume the capacity of a video streaming server is 300 (i.e., supports 300 concurrent streaming threads).

a reasonable choice. We also tried 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 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

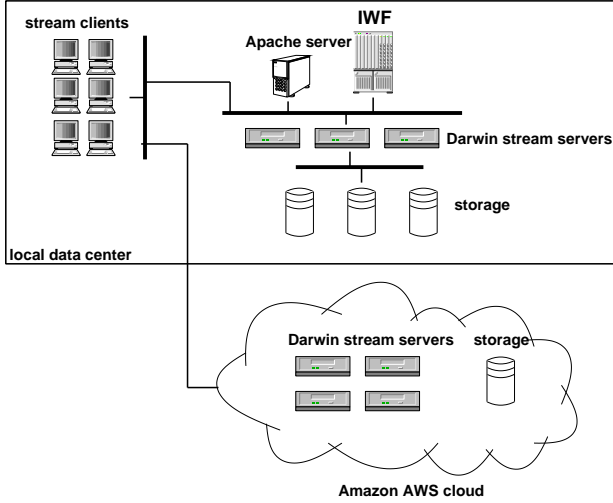


Fig. 7. The video streaming service testbed with a hybrid platform

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 [14] 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 stream 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 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.

The load controller also contains the implementation of two well-known load balancing algorithms [15]: the Least Connections balancing algorithm for the local server farm and the Round-Robin balancing algorithm for the server farm on Amazon Cloud.

We developed a distributed workload generator based on openRTSP [16] to generate real video streaming load. Depend-

ing 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 $\frac{D_3}{D_1+D_3}$.

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 [17]. 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 IWF except that it uses the RATE scheme [18] to detect the top-K data items.

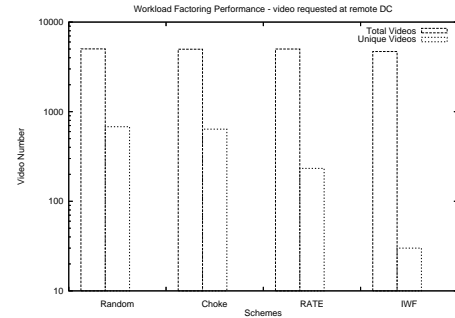


Fig. 8. IWF performance: D_1 - Zipf distribution, D_3 - uniform distribution

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 the graph represents one stream 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 8 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 trespassing zone, IWF

outperformed the other three significantly in terms of unique video files requested (two orders of magnitudes compared to random dispatching); in the trespassing zone totally 5000 streams were served on only 30 unique video clips.

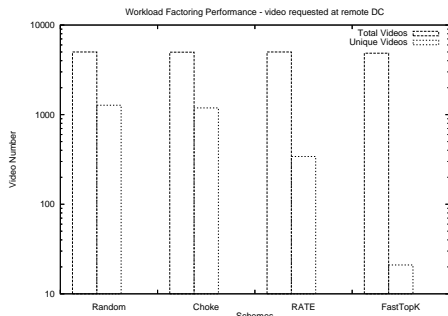


Fig. 9. IWF performance: D_1 - uniform distribution, D_3 - uniform distribution

In Figure 9 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.

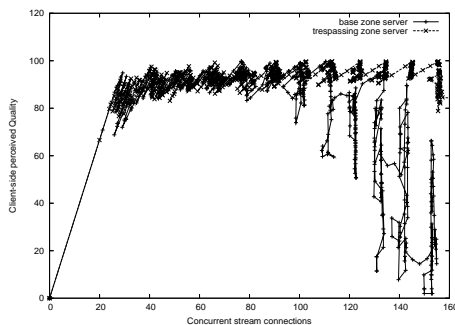


Fig. 10. IWF performance: client-side perceived stream quality at two zones

Figure 10 shows the client-side perceived stream quality from a base zone server and a trespassing 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 stream 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 trespassing zone server delivered more reliable streaming quality than the base 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 steadily at above 80. In the testbed configuration, we set the base zone server capacity at 90 concurrent connections and that for trespassing zone servers at 160 (78% higher than that in the base zone), and enforce it during dispatching.

VI. RELATED WORK

In November 2008, Amazon launches CloudFront [19] 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 [20], 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 our workload factoring technology is proposed as one answer for it.

Berkeley researchers [21] 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 happen 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 [22] [23] [24] exist for cache hit ratio improvement through locality exploration, and their focus is on compact data structure design to exchange data item access information.

For fast frequent data item detection in data streams, many schemes have proposed for fast rate estimation in traffic monitoring [18] [25] [26], and fast data item counting in CDN [27]; their focus is on compact data structure design to memorize request historical information at a static distribution.

VII. CONCLUSIONS & FUTURE WORK

In this paper, we present the design of a hybrid cloud computing model. With the proposed workload factoring technology, the hybrid cloud computing model 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. Given the elastic nature of the cloud infrastructure, It create a situation where cloud resources are used as an extension of existing infrastructure. It’s not an all or nothing decision; companies can ease into the cloud without abandoning established infrastructure and applications.

For the future work, extending the hybrid cloud computing model to stateful applications such as n-tier 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 [28] into our mechanism.

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. [Online]. Available: <http://developer.amazonwebservices.com/connect/message.jspa?messageID=7%9978#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 *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2008, 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. [Online]. 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 *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*. New York, NY, USA: ACM, 1999, pp. 343–348.
- [12] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, and A. Saxena, "Intelligent workload factoring for a hybrid cloud computing model," *NEC Labs America Technical Report 2009-L036*, Feb 2009.
- [13] F. Hao, M. S. Kodialam, T. V. Lakshman, and H. Z. 0002, "Fast, memory-efficient traffic estimation by coincidence counting," in *INFO-COM*, 2005, pp. 2080–2090.
- [14] "Darwin streaming server," <http://developer.apple.com/darwin/projects/streaming/>.
- [15] M. Arregoces and M. Portolani, *Data Center Fundamentals*. Cisco Press, 2003.
- [16] "openrtsp," <http://www.live555.com/openRTSP/>.
- [17] 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*, vol. 2, 2000, pp. 942–951 vol.2. [Online]. Available: <http://dx.doi.org/10.1109/INFCOM.2000.832269>
- [18] 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*. New York, NY, USA: ACM, 2005, pp. 211–220.
- [19] "Amazon," <http://aws.amazon.com/cloudfront/>.
- [20] "Vmware cloud vservices," <http://www.vmware.com/technology/virtual-datacenter-os/cloud-vserves/>.
- [21] 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. [Online]. Available: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>
- [22] 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.
- [23] 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*, 2000, pp. 174–183.
- [24] 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.
- [25] 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*, Jun. 2004, to appear.
- [26] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 271–282, 2000.
- [27] 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*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 767–778.
- [28] 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 iB₂ Best Student Paper Award iB₂*, May 2003.