Measuring and Evaluating Live Content Consistency in a Large-Scale CDN

Guoxin Liu, Haiying Shen, Harrison Chandler
The Department of Electrical and Computer Engineering
Clemson University, Clemson, SC 29631
Email: {guoxinl, shenh, hchandl}@clemson.edu

Jin Li
Microsoft Research
Redmond, WA 98052
Email: jinl@microsoft.com

***Abstract*—Content Delivery Networks (CDNs) play a central role of today's Internet infrastructure, and have seen a sharp increment in scale. More and more internet sites are armed with dynamic (or live) content (such as live sports game statistics, e-commerce and online auction), and there is a need to deliver dynamic content freshly in scale. To achieve high scalability, the consistency maintenance problem for dynamic content (contents with frequent updates) served by CDNs is non-trivial. The large number of widely scattered replicas guarantee the service QoS of end-users, meanwhile largely increase the complexity of consistency maintenance. Current consistency maintenance infrastructures and methods cannot simultaneously satisfy the two requirements: scalability and consistency. In this paper, we first analyze our crawled trace data of a cached sports game content on thousands of content servers of a major CDN. We analyze the content consistency from different perspectives, from which we try to break down the reasons for inconsistency among content servers. Finally, we further evaluate the performance in consistency, scalability and overhead for different infrastructures with different update methods. We itemize the advantages and disadvantages of different methods and infrastructures in different scenarios through the evaluation. We aim to give guidance for appropriate selections of consistency maintenance infrastructures and methods for a CDN, and for choosing a CDN service with different considerations.**

I. Introduction

Over the past decade, Content Delivery Networks (CDNs) have seen a dramatic increase in popularity and use. There were 28 commercial CDNs [1] reported in this crowded market, including Akamai, Limelight, Level 3, and more recent entrants like Turner and ChinaCache. Among them, Akamai [2], as a major CDN, has more than 85,800 servers in about 1,800 districts within a thousand different ISPs in more than 79 countries. The trend of scale is growing rapidly at about 50% per year, due to the 100% increase of traffic per year [3]. The vast growth of traffic and infrastructure illustrates that CDNs serve as a key part of today's Internet, and undertake heavy content delivery load. This promising growth makes CDNs a hot spot for research.

Figure 1 shows the standard architecture of current CDNs [4]. When an end-user tries to visit web content, the request is forwarded to the local DNS server, which returns the IP of the URL if it exists in the cache and is not expired. Otherwise, the local DNS server forwards the request to the CDN's recursive DNS servers, which returns the IP of the content server (server in short) close to this end-user with load-balancing consideration [5]. Then, the user sends its content request to the IP of a content server, which returns content. The content servers periodically poll content updates from the content provider (provider in short).

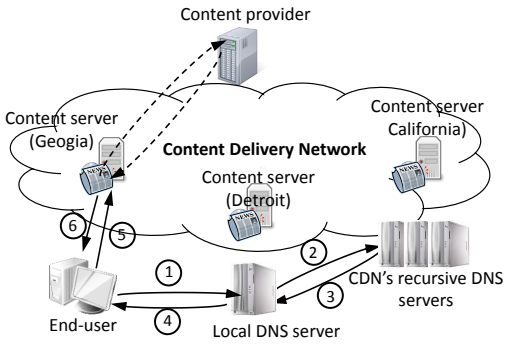


Figure 1: The architecture of CDNs.

CDNs not only serve static contents (contents without updates such as photos and videos), but also dynamic (or live) contents (contents with frequent updates such as live game statistics), which need updates to be delivered from providers to all replicas. Caching/Replicating to surrogate servers near the network edge is widely used in CDNs to optimize the end user experience with short access latency. The large amount of widely scattered replicas make the consistency maintenance methods non-trivial. In addition, this method has two key requirements: scalability and consistency guarantee.

Based on the infrastructure, there are three common infrastructures used to deliver updates for consistency maintenance: i) unicast [6, 7], ii) broadcast [8] and iii) multicast tree [9–16]. However, none of these approaches can satisfy both requirements simultaneously. The unicast approach can guarantee consistency, but since it relies on centralized content providers for updates, it causes congestion at bottleneck links, and cannot promise scalability. Broadcasting can efficiently propagate the updates inside a local network, and guarantee the consistency. However, it generates very high overhead due to an overwhelming number of updating messages. Thus, it cannot support the scalability required for large world-wide CDNs due to a vast number of redundant messages. The multicast approach produce fewer update messages than broadcasting, but node failures break the structure intactness, and hence lead to unsuccessful update propagation. Node failure aside, the structure maintenance

will incur high overhead and complicated management due to the dynamism of servers in the multicast tree.

With each update architecture, there are three basic methods to update replicas: i) Time To Live (TTL) [6, 17], ii) push [10–15] and iii) server-based invalidation [7, 16, 18]. All of these update methods cannot guarantee the aforementioned two requirements. In TTL, servers poll the updates from providers whenever the cached content is outdated, which supports greater scalability. TTL value offers a tradeoff between freshness and CDN efficiency. In push, an update is transmitting to every replicas right after updating time, which guarantees short inconsistency. However, an update will be pushed to all replicas immediately, which depends on the consistency infrastructure to support scalability. Also, push may generate unnecessary update messages to uninterested replicas. In server-based invalidation, whenever there are updates on source providers, an invalidation message is received by each replica, and replicas only fetch the update whenever the needed content is invalid. It can save traffic cost compared to push if the content visit rates on servers in CDNs are smaller than the update rate of this content.

None of current update architectures together with update methods can fully solve both scalability and strong consistency in current CDNs. With the rapid growth of CDNs, consistency maintenance in CDNs needs to be particularly studied. Can the current update method used in the CDN provide high consistency for dynamic contents? If not, what are the reasons for the content inconsistency? What are the advantages and disadvantages of employing previously proposed consistency maintenance approaches in the CDN environment? *The answers to these questions help develop consistency maintenance approaches specifically for CDNs with different considerations.* Thus, in this paper, we focus on measuring the inconsistency of a CDN's servers, and break down the reasons for this inconsistency. Then, we conduct a trace-driven evaluation to measure the performance of consistency maintenance infrastructures and methods, and different parameters' effects on performance. The contributions of this paper are as follows:

- *Measuring the inconsistency of a major CDN.* This paper is the first to measure content consistency for a large amount of globally scattered servers in a major CDN.
- *Breaking down the reasons for the inconsistency of the CDN.* Through our measurement, we break down the reasons for inconsistency to different factors and analyze their effect.
- *Evaluating infrastructures and methods for consistency maintenance through trace-driven experiments.* We further evaluate the performance for different infrastructures and update methods. We itemize the advantages and disadvantages of different methods and infrastructures in different scenarios through our evaluation.

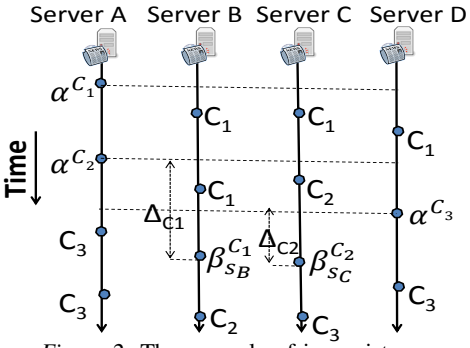


Figure 2: The example of inconsistency.

II. Trace Analysis

A. Measurement Methodology

In order to study the consistency maintenance strategies used in current CDNs, we study our crawled cached content of a popular sports game from a large number of servers in a major CDN. The content we crawled was live game statistics webpages that need to be continuously updated during the live game. To identify the IP addresses of the CDN servers, we retrieved all domain names in all webpages, and used the method in [4] to translate the domain names to IPs. Third, we validated each IP's corporation to derive the IP addresses of the CDN servers and providers using the same method in [4]. Finally, we found 10 IPs of the provider and 50064 IPs of the CDN. Compared to the IPs of the CDN's crawled in [4], we have crawled most (57.2%) of the servers in their trace, which has 59581 IPs in total. There are 26.9% more new servers compared to their trace, which indicates the rapid increase of the CDN. We also use the same method to track the IPs of the source provider's servers.

We randomly polled live game statistics from 3000 randomly chosen CDN servers every 10 seconds during around two and half hours on each day from 200 globally distributed randomly chosen PlanetLab nodes. We collected 15 day trace between May 15, 2012 and June 4, 2012.

To measure data inconsistency, for each poll, we retrieve the snapshot of statistics and current GMT (Greenwich Mean Time) time on that server in order to avoid the interference of network delay. As shown in Figure 2, we identified different snapshots from all polled snapshots and use C_i to denote the i^{th} content snapshot. We find the first time when each snapshot C_i shows up in the trace and denote it as α^{C_i}. For each server s_n, we ordered its content snapshots over time. For each C_i, we found the last time when C_i shows up, which is denoted by $\beta_{s_n}^{C_i}$. The inconsistency length of C_{i-1} (denoted by $\triangle_{C_{i-1}}$) for that server is calculated by $\triangle_{C_{i-1}} = Max\{\beta_{s_n}^{C_{i-1}} - \alpha^{C_i}\}$. The inconsistency length of t seconds means that the content is expired for at least t seconds. While this metric understates the inconsistencies between servers and providers, it fairly evaluates differences between user experience in different locations.

B. Is There Inconsistency in the CDN Servers?

Figure 3 shows the cumulative distribution function (CDF) of inconsistency lengths for all content requests during the 15 days. We see that only 10.1% of requests have inconsistency lengths less than 10 seconds, and 20.3% of requests have inconsistency lengths greater than 50 seconds. The results indicate that content inconsistency exists among the CDN servers in serving the dynamic contents of game statistics that require frequent updates.

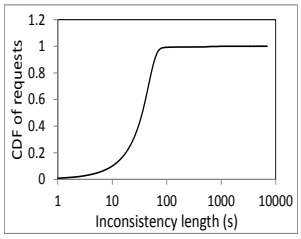


Figure 3: The inconsistencies of data served by the major CDN.

C. Does a User Observe Self Inconsistency?

The game statistics is sequencing over time. A user observes self inconsistency (inconsistency in short) when (s)he sees a statistics (such as game score) prior to the most recently statistics (s)he has seen. For example, a user sees a game score of 2:3 at 1:00pm, and then sees the score changes to 2:2 at 1:01pm. Such user observed inconsistency is caused by the user receiving an older webpage content that is not updated in time from another server.

The local DNS server caches the IP of the CDN's server for each visited domain name. The IP expires in a short time. When the local DNS server receives a request from an end-user, if the IP is not expired, it sends the IP to the end-user; otherwise, it forwards the request to the recursive DNS servers for the IP of a server. The recursive DNS servers consider the load balancing between the servers and send back an IP of a server. To update the dynamic contents in a webpage, the end-user sends out a request every 10 seconds. Therefore, the request may be redirected to another server due to the expired cached IP in the local DNS server and the server reassignment by the recursive DNS servers. If the content in the newly assigned server is not updated, the user may observe the inconsistency.

To investigate the user observed inconsistency, we used 200 world-wide distributed PlanetLab nodes to visit the same game statistics through its URL once every 10 seconds during the time period of a game. We recorded the IPs of serving servers and the received statistics contents for each user. We measure the consistency performance from a single user's perspective. First, we introduce three metrics: the percent of requests redirected to another server, continuous consistency time and continuous inconsistency time.

Suppose a user creates N requests in total, in which M requests are served by redirected servers; then, the percent of requests redirected to another server equals $\frac{M}{N}$. Continuous inconsistency time is the time period from the time the user observes an inconsistency to the time of next consistency record. The continuous consistency time is the time period from the time the user observes an consistency to the time of next inconsistency record.

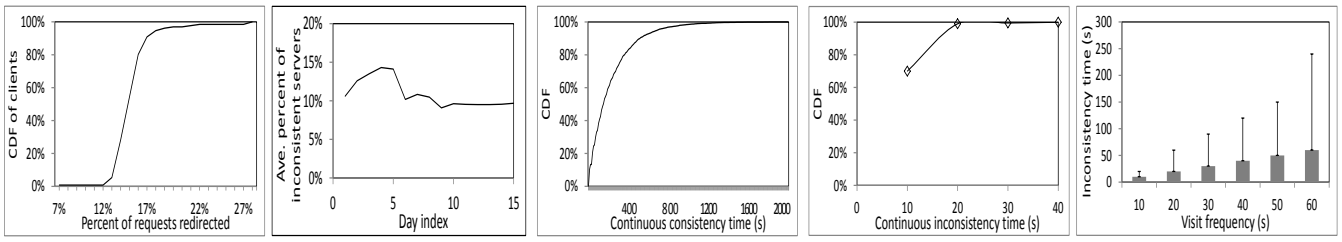
We first measured how many percentage of visits of a single user are redirected to another server different from the current server. Figure 4(a) shows the CDF of users versus the percent of visits redirected to another server. It shows that most of the users have 13%-17% of visits switched to another server. From the trace we observed that with continuous updates of the game statistical result, there are around 11% of servers on average with inconsistent content at each polling time during all 15 days as shown in Figure 4(b), which shows the average of percent of inconsistent servers in every 10 second on each day. This means that a user's request has around 11% probability being redirected to a server that has outdated content. Thus, 1.43% to 1.87% of visits of a single user during a game will be redirected to outdated content.

We then measure the continuous (in)consistency time from a singe user's perspective. We calculated all continuous (in)consistency times of all users. Figure 4(c) shows the CDF of the continuous consistency time. The median continuous consistency time is around 160 seconds, 82.4% of all continuous consistency times are within 400 seconds. The result means that most users can observe inconsistency and receive outdated contents during watching. Figure 4(d) shows the CDF of continuous inconsistency times. In this figure, 70% of all the continuous inconsistency times is 10 seconds, and around 99% of all the continuous inconsistency times is no larger than 20 seconds. There are no inconsistency times longer than 40s. The result indicates that users may observe outdated dynamic web content, but it always lasts no more than 20 seconds, which means that the inconsistency usually lasts no more than two continuous visits.

We varied the polling frequency from 10 seconds per poll to 60 seconds per poll with a 10 second increase in each step. For each polling frequency, we collected all continuous (in)consistency times of all users and calculated the 5th percentile, median and 95th percentile of the continuous inconsistency times. Figure 4(e) shows the results with different polling frequencies. From the figure, we observe that the median value always equals the 5th percentile value, meaning most inconsistency lasts for a short time period. Also, we see that the median and the 95th percentile value of the inconsistency time increase in proportion to the visit frequency due to the slower polling frequency. From Figure 4 and Figure 4(e), we can infer that an individual user can observe inconsistency on dynamic contents in the CDN. This implies that the current update strategy in the CDN can be improved to prevent users from receiving outdated information for dynamic contents.

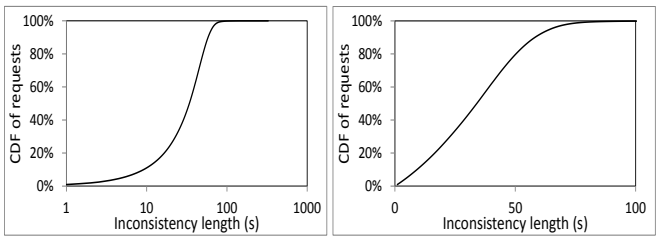
D. What are the Causes of Inconsistency in CDNs?

In the above, we observed that content inconsistency exists in the servers and end-users can also observe the in-



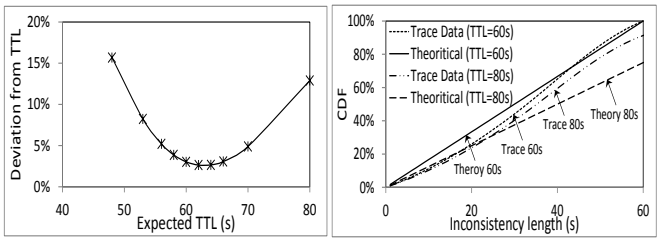
(a) Visits on another server (b) Ave. percent of inconsistent servers (c) Continuous consistency time (d) Continuous inconsistency time (e) Inconsistency time

Figure 4: User perspective consistency.



(a) CDF of all requests (b) CDF of requests with inconsistency less than 100s

Figure 5: CDF of inner-cluster inconsistency.



(a) TTL refinement (b) Inconsistency in trace vs. in theory

Figure 6: The CDN content severs' TTL.

consistency (i.e., receive outdated content). In the following, we identify and explore potential causes of the content inconsistency in the CDN, which will provide guidance for designing consistency maintenance mechanisms. We measured the individual influence on the inconsistency among CDN servers of each potential cause, including the TTL value, source provider's inconsistency, propagation delay, shortage of bandwidth, server/source provider overload/failure.

1) Time-to-live based consistency maintenance: In the TTL-based consistency maintenance method, when a CDN server receives a content request from an end-user, it first checks its cache for the content. If the content exists in the cache and its TTL has not expired, the server serves the content. If the content does not exist in the cache or its TTL has expired, the server retrieves the content from the source provider, sends it to the user, and caches the content. Therefore, if the content changes before the TTL has expired, the CDN server will inadvertently fulfill requests with outdated content. [19] indicates that the CDN uses TTL-based consistency maintenance.

We study the impact of the TTL-based method on content inconsistency. In order to minimize the influence of the provider-server distance and server-user distance on content inconsistency, we clustered geographically close servers, used the same or geographical close PlanetLab nodes to poll the contents from servers in the same cluster. We examined the distribution of inner-cluster inconsistency lengths, which refer to inconsistency lengths calculated only within clusters of collocated nodes rather than all servers in the CDN. To create clusters, we first translated the IPs of the CDN's servers to geographical locations by an online IP geolocating service [20], and grouped the servers with the same longitude and latitude into a cluster.

Figure 5(a) shows the CDF of requests for different inconsistency lengths. The figure shows that only 31.5% of served requests have inconsistency lengths less than 10 seconds. Also, the CDF of requests approximately exhibits a linear increase when the inconsistency length increases from 0 to 60. We can assume that the inner-cluster inconsistency length is evenly distributed in [0, TTL], which will show a linear increasing in CDF within [0,TTL]. Then we can assume that the TTL for cached content is around 60 s.

Below, we attempt to derive TTL using another method. We assume all updates are independent, and all servers independently start to cache the dynamic content. We then derive the average inconsistency lengths of all servers, denoted by $E[I]$. If we split the time into slots, each of which lasts TTL, then a server can poll the content at any time within $[0, TTL]$ in a slot with the same probability. If the first server gets the update C_i at time t, since servers poll the content independently, other servers receive the update at any time t' within $[t, t+TTL]$ with uniform distribution. As the inconsistency length equals $t'-t$, it is then uniformly distributed within $[0, TTL]$. Thus, $E[I] = \frac{TTL}{2}$.

Since TTL is not the sole factor of inconsistency, the true average inconsistency length from the trace denoted by $E'[I]$ is larger than $E[I]$. If TTL is the sole factor, $2E'[I]$ should not be larger than TTL. Therefore, we use recursive refining to derive the TTL used by the CDN from the trace. We first calculated the average inconsistency length in trace $E'[I]$, and then calculated $TTL' = 2E'[I]$. Then, we calculated $E''[I]$ from the inconsistency lengths in the trace that are no larger than TTL' and derive a new $TTL'' = 2E''[I]$. We then calculated the deviation of the two TTLs as

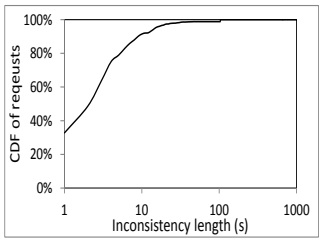


Figure 7: The inconsistencies of data served from the provider.

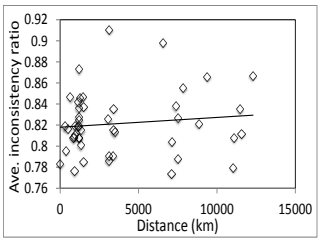


Figure 8: The distance of servers.

$(TTL'' - TTL')/TTL'$. We repeated this procedure to derive a TTL, which is closet to $2 * E[I]$ according to the above equation. Thus, the TTL' with the smallest deviation is the actual TTL used in the CDN. Figure 6(a) shows the deviation distribution versus each derived expected TTL. The smallest deviation is at 60s, which means that the CDN's TTL is approximately equals to 60s.

We then verify if TTL=60s is correct. Using 60s and 80s as the TTL respectively, we calculated the inconsistency distribution based on the uniform distribution. For TTL=60s (and 80s), we removed the inconsistency lengths larger than 60s (and 80s) in the trace (which are inconsistencies caused by reasons other than the TTL) and plotted the inconsistency distribution based on the remaining data. Figure 6(b) shows the CDF of inconsistency lengths. From the figure, we see that the deviation between the trace and theoretical inconsistency with TTL=60s is smaller than that with TTL=80s. The root mean square error of TTL=60s is 0.0462, while that of TTL=80s is 0.0955. We also tested other TTL values, and found that TTL=60s leads to the smallest deviation between trace and theoretical inconsistency. Thus, the actual TTL should be 60s, which introduces an average inconsistency length of 30s. We notice that the average inconsistency is 40s in the trace. Thus, we can conclude that TTL is the main cause for the inconsistency, and other factors such as provider/server inconsistency, provider/server overload, network congestion, server failure and so on introduce a small part of inconsistency, around $\frac{80-60}{80} = 25\%$.

2) Source provider inconsistency: One potential cause of inconsistency between servers is inconsistency at providers that provide contents to the servers. We requested statistics contents for the same game from the providers using the same setup as before. Figure 7 shows the CDF of inconsistency length for requests served by the providers. The figure shows that 90.2% of served requests have inconsistency lengths less than 10 seconds, only 1.2% of requests have inconsistency lengths greater than 50 seconds, and the average inconsistency is 3.43 seconds. The inconsistency length is much lower than that of the CDN-served content as shown in Figure 3. We have checked the geographical locations of all our identified providers and found that they are in the same geographical location. In this case, the content providers can provide higher consistency than the servers that are dispersed worldwide. Even if multiple providers provide the same dynamic content to the servers, the providers have negligible content inconsistency; therefore, their responsibility on the inconsistent contents received by the end-users is negligible.

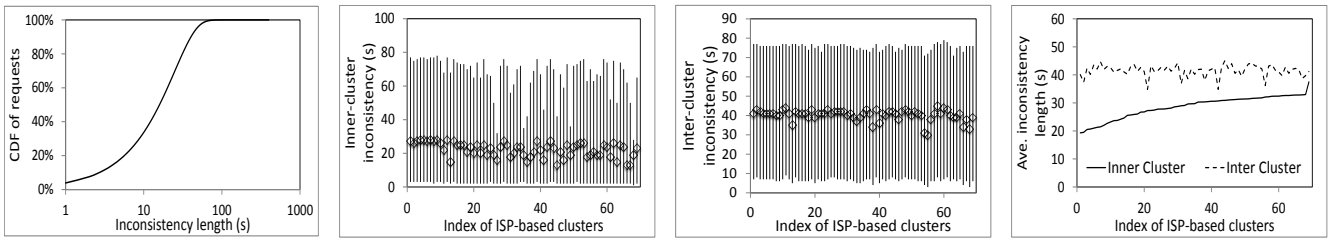
3) Provider-server propagation delay: Content servers are distributed globally so that end-users can be served by their geographically close servers. However, globally dispersed servers face considerable propagation delay for content originating from a central location. Since propagation delay varies for different servers, inconsistency can be introduced. We introduce a metric called *consistency ratio* for a server, which is calculated by $1 - \frac{\sum inconsistency\ lengths}{total\ trace\ time}$, which indicates the capability of a server to maintain consistency. We clustered servers with the same distance to the provider and calculated the average consistency ratio.

Figure 8 shows the average consistency ratio versus the provider-server distance. The figure shows that in that overall, as the provider-server distance increases, the average consistency ratio exhibits a very slight increase. The average consistency ratio and distance have little correlation ($r = 0.11$). As distance is directly related to propagation delay, this result indicates that propagation delay have a little effect on inconsistency.

We further investigate whether inter-ISP traffic has an effect on inconsistency. Traffic transmitting between ISPs is more costly for ISPs, and such traffic competes for the limited transmission capacity [21]. To identify the ISP for each server, we first found the ISP of each server based on its IP using IPLOCATION [20]. We further increase the accuracy of these identified ISPs. We use Traceroute to diagnose the entire path for each request from a PlanetLab node to a CDN server. Since the CDN's servers are close to the backbone routers of ISPs [4], we checked whether the router in the last several routing hops in a route belongs to the identified ISP. If not, we removed the trace record of the server. We successfully verified the ISPs of 99.6% of the servers.

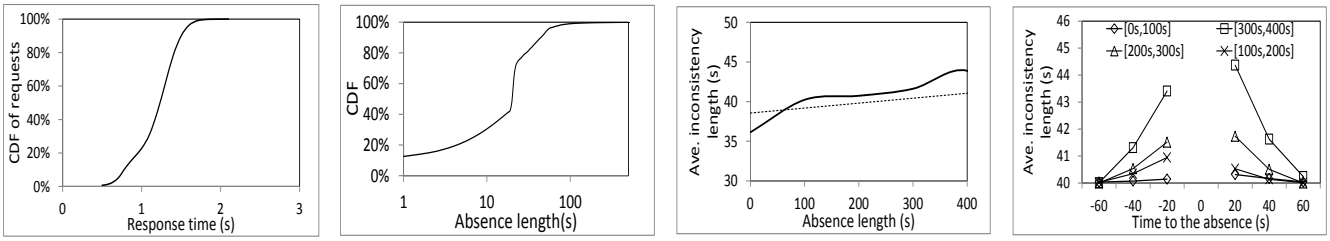
We grouped servers within the same ISP to a cluster. We calculated the inconsistency lengths for all servers in each ISP-based cluster. Figure 9(a) shows the CDF of the intra-ISP inconsistency. We see that 33.7% of requests have inconsistency lengths less than 10 seconds, and only 3.9% of requests have inconsistencies greater than 60 seconds. This inconsistency distribution is only slightly better than that in Figure 3. Thus, we can conclude that although inter-ISP traffic competes for the transmission capacity, it only contributes slightly to inconsistency in servers.

In order to better understand the degree of influence of inter-ISP traffic from providers on the inconsistency, we compare the inner-cluster and inter-cluster inconsistency lengths. The inter-cluster inconsistency lengths are calculated using the same method as the inner-cluster inconsistency lengths except that the α^{C_i} of a cluster is the earliest time of C_i's appearance in all other clusters. Figures 9(b) and 9(c) show the 5th, 50th and 95th percentiles of the



(a) intra-ISP inconsistency. (b) The inner-cluster inconsistency. (c) The inter-cluster inconsistency. (d) Ave. inconsistency of inter-clusters and inner-clusters

Figure 9: The effect on the inconsistency of inter-ISP traffic from the provider.



(a) The response time distribution of provider. (b) The length of absence. (c) The length of absence for Akamai nodes. (d) Inconsistency before and after inconsistency.

Figure 10: The effect of server overload/failure on the inconsistency.

inner-cluster and inter-cluster inconsistency lengths of each ISP-based cluster. We see that the inter-cluster inconsistency lengths are always higher than the inner-cluster inconsistency lengths, meaning the inter-ISP traffic from providers affects the inconsistency. The 50th percentiles of the inner-cluster and inter-cluster inconsistency lengths range from [13,28] and [30,45] respectively, and the 95th percentiles of the inner-cluster and inter-cluster inconsistency lengths range from [36,73] and [71,79] respectively. The 50th and the 95th percentiles of the inter-cluster inconsistency lengths are larger than those of the inner-cluster inconsistency lengths, and the increment indicates the degree of influence of inter-ISP traffic from providers. The increment of the average inconsistency lengths is illustrated in Figure 9(d). We see that on average, the inter-ISP traffic from providers increases the inconsistency lengths by [3.69, 23.2]s.

4) Content provider bandwidth: If the providers are overloaded or have insufficient bandwidth, they will not be able to receive up-to-date content. We measure each request's response time by $t_r - t_i$, where t_i is the time that the PlanetLab node initiates the request and t_r is the time that it receives the content. Figure 10(a) plots the CDF of requests versus the response time. The figure shows that the response time are in the range of [0.5, 2.1]s, and 90% of requests are resolved within 1.5s. Thus, there is no large delay due to congestion or overloaded of the providers. This indicates that the source providers have sufficient computing capabilities and bandwidth to handle all requests. Considering the time interval between the worst case and the best case of the response time, the provider's network resource constraint only introduces less than 1.6s inconsistency in servers. In the trace, the size of statistics content is relatively small, it hardly causes congestion in provider's uplinks. However in some situations, such as live video streaming, the content provider's bandwidth may introduce large inconsistency when overloaded.

5) Content server failure and overload: Content server failure and overload could also be the cause of content inconsistency. When a server has failed or is overloaded, it cannot quickly send out content requests to or receive content from the provider. Suppose that two successive response times of a server upon polling are t_i and t_{i+1}. We calculated the *absence length* of the server as $t_{i+1} - t_i - 10s$, where $10s$ is the time interval of two successive polls. The absences could be due to either node overload, reboot, or failure. Suppose the content responded at t_{i+1} from the server that was absent is C_{i+1}, then the consistency length of C_{i+1} is the consistency length of this absence.

Figure 10(b) shows the CDF of absence lengths of servers. The figure shows that absence lengths range from [1,500] seconds, with 30.4% less than 10 seconds and 93.1% less than 50 seconds. The CDN has a load balancing technique that balances the load between servers, so it is unlikely that an overloaded node would remain in the overloaded status for an extended period. Thus, node failures/reboots are responsible for most of the absences lasting longer than 50 seconds.

We plot the node absence length with the average inconsistency length after node returns in Figure 10(c). We group trace records first by their absence length. Since for a specific absence length, there may not be enough inconsistencies to show its general case of inconsistency with such an absence length, we group absence length by every 50s. In order to show the average inconsistency without absence,

we divide first group with absence length of [0,50]s to [0,0]s and (0,50]s. From the figure, we can see that the inconsistency is increasing slightly from 38.1s to 43.9s while the absence length increases from 0s to 400s. It indicates that the node overloads and failures do not have adverse effects on consistency. They can increase the average inconsistency by 15.22%.

In order to determine the effect of node absences on the inconsistency, we grouped the inconsistency lengths associated with the same absence length and calculated the average. We then group the average absence lengths with inconsistency lengths in the range of (0,50]s, (50,100], $\cdots$, (350,400]. In order to show the average inconsistency without absence, we also plot the average inconsistency lengths for absence length equals 0.

Figure 10(c) plots the average consistency length in different ranges of absence lengths. From the figure, we see that the inconsistency length increases from 38.1s to 43.9s while the absence length increases from 0s to 400s. That is, the absence contributes no more than 6s inconsistency. This indicates that the overload and failure do have influence on consistency and that they can increases the average inconsistency by 15.22%. The figure also shows that a larger absence length leads to a higher average inconsistency length, because servers may not receive updates or send out update requests in time when overloaded or failed.

Recall that then the inconsistency length of C_{i-1} is calculated by $\triangle_{C_{i-1}} = Max\{\beta_{s_n}^{C_{i-1}} - \alpha^{C_i}\}$. If a server's absence length is larger than 0, we calculated the inconsistency lengths of C_{i-1} polled during $[t_i - x, t_{i+1}]$ and $[t_i, t_{i+1}+x]$, where $x = 20, 40, 60$. For all the inconsistency lengths in each range of all absent servers, we derived those with absence lengths within [0s,400s], classified them to 4 groups with absence lengths in [0s,100s], [100s,200s], [200s,300s] and [300s,400s], and then calculated the average inconsistency length in each group. Figure 10(d) plots the average inconsistency lengths in each group in a certain time period before and after node absence. It shows that in each group, the inconsistency measured closer to the absence is larger and vice versa. We suspect this is because when a server is about to be overloaded or has just recovered from overload or failure, it has a lower probability of sending or receiving update requests. From the figure, we also see that a larger absence length leads to a higher inconsistency length and that sharper inconsistency length increase. These results indicate that server overload and failure affect inconsistency and we need to avoid system failure when there are continuous updates of the dynamic contents, as it causes largely degraded user experience.

E. Summary of Trace Analytical Results

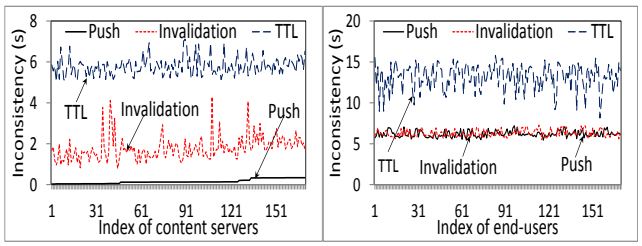
The major CDN's servers indeed have large inconsistencies for their cached dynamic content. Thus, the CDN cannot guarantee consistency for dynamic content by using unicast with TTL method. Also, a user can observe the inconsistency of his/her viewed content due to server redirection. The inconsistency is caused by several factors including TTL, provider-server propagation delay, providers' inconsistency and bandwidth, server overload and failure. The biggest factor is the TTL, which contributes around 75% of average inconsistency. The other factors contribute to the inconsistency significantly less than TTL, and they are not easy or expensive to solve. For example, the server overload problem can be resolved by improving the capabilities of current servers and links. Compared to the other factors, improving the TTL-based consistency maintenance method is the easiest and only way to significantly improve content consistency. Thus, we conduct trace-driven experiments to evaluate the performance of different consistency maintenance approaches in a CDN to provide guidance for selecting or designing optimal consistency maintenance approaches for CDNs.

III. Trace-Driven Performance Evaluation

We conducted trace-driven experiments to evaluate the performance of consistency maintenance and overhead for different consistency maintenance infrastructures and methods in a CDN. The experimental results shed light on the selection or design of optimal consistency maintenance methods based on different needs of a CDN.

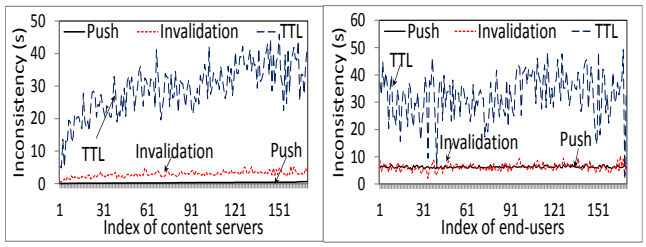
We built our simulated CDN on PlanetLab [22]. According to the distribution of the CDN servers in the CDN [4], we selected 170 PlanetLab nodes with high performance and light load mainly in the U.S., Europe, and Asia. We chose one node in Atlanta as the provider. We randomly selected one-day live game events on Jun. 2nd, 2012 in our trace data as the content. It includes 306 different snapshots lasting 2 hours and 26 minutes. We regard the time of each snapshot's first appearance in the trace as the update time in the provider. In each PlanetLab node, we also created five simulated end-users browsing the content.

We evaluate three different consistency maintenance methods, Push, Invalidation and TTL-based method (TTL in short), on two updating infrastructures, Unicast-tree and Multicast-tree. We do not evaluate broadcast, since CDN is a large network over various local networks, while broadcast is only effective and efficient inside a local network. In the multicast-tree, the provider is the tree root and geographically close nodes (measured by inter-ping latency) are connected to each other to form a binary tree. In the unicast-tree, the provider directly connects to all servers in multiple unicast channel. The size of all consistency maintenance related packages and content request packages was set to 1KB. The end-users' TTL was set to 10s according to the trace. In each experiment, the provider starts to deliver content at time 60s. Each end-user starts requesting the content from a time randomly chosen from [0s,50s].



(a) Content inconsistency of servers (b) Inconsistency of end-users

Figure 11: Inconsistency in the unicast-tree infrastructure.



(a) Content inconsistency of servers (b) Inconsistency of end-users

Figure 12: Inconsistency in the multicast-tree infrastructure.

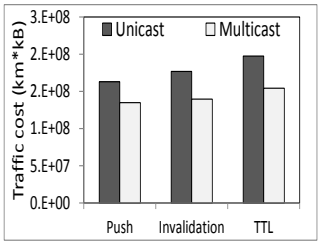


Figure 13: Consistency maintenance cost.

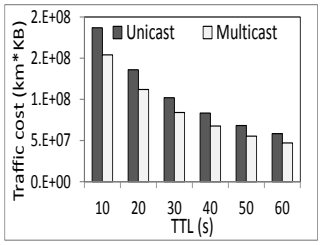


Figure 14: Consistency maintenance cost vs. TTL of content servers.

A. *Inconsistency in the Unicast-Tree Infrastructure*

With the unicast-tree structure, in Push (or Invalidation), the provider directly sends updates (or notifications) to the servers, and in TTL, the servers poll the provider directly. With the multicast-tree structure, in Push and Invalidation, the update (or notification) is pushed along the tree in the top-down manner, and in TTL, the children poll their parents in the tree in the bottom-up manner. There are two levels of content inconsistency: i) the inconsistency between servers and the provider, and ii) the inconsistency between the end-users and the provider.

Figure 11(a) shows the average of all inconsistencies of each content server with different update methods in the unicast-tree infrastructure, where all servers are sorted by their inconsistency in Push. We see the inconsistency results follow Push<Invalidation<TTL. In Push, the provider pushes an update to the servers upon an update occurrence, leading to the smallest inconsistency. Its inconsistency is due to the traffic latency including the transmission delay, the propagation delay and the queuing delay at the output ports of the provider. In Invalidation, a server receives notifications for outdated content but does not request the new update until it receives a request from an end-user. Therefore, its inconsistencies are higher than Push. Since there are no user requests during the inconsistency time period, these inconsistencies do not affect the consistency of the contents received by users. In TTL, the content in a server is considered fresh during a TTL. This is why TTL generates the largest inconsistency, the average of which equals 5.7s, around $TTL/2$.

Figure 11(b) shows the largest average inconsistency of the end-users on each PlanetLab node. We see that Push and Invalidation produce similar inconsistencies that are lower than that of TTL. In Push and Invalidation, the servers always supply updated content. However, since end-users request the content periodically, they may send requests a certain time period after the content update, thus generating inconsistencies. This result implies that "pushing to end-users" should be a better method to improve the consistency of contents viewed by end-users. In TTL, the first-level servers have certain inconsistencies. The second-level content servers periodical polling amplifies inconsistencies as they may not poll right after the servers' polling. Therefore, TTL generates larger inconsistencies than other methods. Also, TTL's inconsistencies of end-users are higher than those of the servers in Figure 11(b).

B. *Inconsistency in the Multicast-Tree Infrastructure*

In this section, we measure the inconsistency of different methods in the multicast-tree infrastructure. Figure 12(a) shows the average of all inconsistencies of each server with different update methods in the multicast-tree infrastructure, where all servers are sorted by their inconsistencies with the Push method. It shows that the inconsistency follows Push<Invalidation<TTL due to the same reason as in Figure 11(a). Compared to Figure 11(a), nodes in the lower-level of the multicast tree with the TTL method have higher inconsistencies, since higher-level nodes are closer to the provider and expected to receive the update earlier. For example, an update can reach nodes in level 1 with a longest delay as TTL, but for nodes in level 2, the longest delay will be $2 * TTL$. In general, a node in level m has around $m - 1$ times the expected inconsistency compared to a node in level 1.

Figure 12(b) shows the average of all inconsistencies of each end-user with different update methods in the multicast-tree infrastructure, where all servers are sorted by their inconsistencies with the Push method. We see that the inconsistencies of end-users for TTL increase compared to those in the unicast tree because of the increased inconsistencies in the servers in the multicast tree. The other two methods in the multicast tree have the same performance as unicast tree.

C. *Efficiency of Consistency Maintenance*

As in [23], we measure the traffic cost as $km * KB$ for all packets for consistency maintenance. Figure 13 shows the

total traffic cost of all update methods in both the unicast-tree and the multicast-tree. It shows that multicast can save at least $2.8 * 10^7 km * KB$ in traffic cost of unicast for all methods. That is due to the proximity-awareness multicast tree, so updates are transmitted between proximity close nodes with short latency. In unicast, the provider needs to communicate with all servers distributed worldwide. The figure also shows that in both unicast and multicast, the traffic cost follows Push<Invalidation<TTL. In the trace, the update frequency is low. Thus, TTL wastes traffic in probing unchanged content. Invalidation has additional notification and polling packets compared to Push. As a result, Push generates lower traffic cost than Invalidation and TTL, producing the lowest traffic cost.

As shown in Figure 14, the overhead of consistency maintenance decreases as the time-to-live in TTL method increases in both unicast and multicast. There are two reasons for this decrease: (a) the traffic overhead querying messages can be saved due to the larger time interval between queries; (b) larger time-to-live has a higher probability to skip an update. Since increment of TTL has the same effect while decreasing the update frequency, it indicates that with frequent updates, TTL can be used for applications with weak consistency requirement to save consistency maintenance cost.

D. Summary of Trace-Driven Experimental Results

We summarize our experimental results below, which may provide guidance for selecting or designing a CDN update approach.

- Push provides better consistency on content servers than other methods in a small-scale network with either unicast or multicast. However its performance deteriorates rapidly in a large-scale network with heavy traffic burden.
- From the end user perspective, Invalidation can supply similar consistency guarantee as Push. It can also reduce traffic costs with infrequent visits from end users on frequently updated content. However, for frequently updated contents, it introduces heavy network burden by transmitting the additional invalidation notifications.
- The TTL-based method can supply a weak consistency with inconsistency no larger than its TTL. It should have better scalability than the other two methods even by releasing update transmission load of the content provider. However, it may waste unnecessary traffic costs on contents with infrequent updates.
- The proximity-aware multicast tree infrastructure can save more traffic costs and support better scalability than the unicast-tree infrastructure. However, it introduces much more inconsistency into the TTL-based method.

According to measurements, not a single update method or an infrastructure supports both scalability and consistency in all scenarios. However a combination of different methods with different infrastructures could work. New APIs may be needed to probe visit and update frequency of live contents, with which we can infer the changes of the scale of interested users. Additionally, considering customized requirements such as consistency, a self-adapting strategy could switch between update methods and infrastructures to find an optimal combination.

IV. Related Work

Commercial CDNs enable efficient delivery for many kinds of Internet traffic, such as e-commerce and live sports. Serving dynamic contents not only requires a scalable CDN, but also requires consistency guarantees, either strong or weak. Recent studies of consistency maintenance have been applied to different applications, such as P2P networks, web caches, and CDNs. Based on the infrastructure, these studies can be categorized into three classes.

One class of methods is based on unicast. In [7], an invalidation method is recommended, since it is better at saving traffic costs and reducing end-user query times. In [6], an adaptive TTL is proposed to predict the update time interval based on a historical record of updates. Compared to a fixed TTL, it may reduce traffic costs as well as support stronger consistency.

Another class of methods is based on broadcasting. Lan *et al.* [8] proposed to use flooding-based push for near-perfect fidelity or a push/pull hybrid method for high fidelity. Broadcasting is widely used in local computer networks but fails to be sufficiently scalable for use in large scale networks such as CDNs due to a large number of redundant messages.

The last class of methods is based on multicast. Li *et al.* [15] presented a scheme that builds replica nodes into a proximity-aware hierarchical structure (UMPT) in which the upper layer form a DHT and nodes in the lower layer attach to physically close nodes in the upper layer. SCOPE [14] builds a replica-partition-tree for each key based on its original P2P system. It keeps track of the locations of replicas and then propagates updates. CUP [12] and DUP [13] propagate updates along routing paths. In FreeNet [10], updates are routed to other nodes based on key closeness. In a hybrid push/poll algorithm [9], flooding is replaced by rumor spreading to reduce communication overhead. When a new node joins or a node reconnects, it contacts online replica nodes to poll updated content. This hybrid push/poll scheme only offers probabilistic guarantee of replica consistency. GeWave [16] builds a poll-based multicast tree for consistency maintenance, in which the replica of a parent node have higher visit frequency than the replicas of their child nodes. Tang and Zhou [24] studied inconsistency in distributed virtual environments, which create a common and consistent presentation of a set of networked computers. Peluso *et al.* [25] introduced a distributed multi-version concurrency control algorithm for transactional systems that rely on a multicast-based distributed consensus scheme. S2PC-

MP [26] is a metadata consistency scheme for distributed file systems based on multicast. Benoit *et al.* [27] studied replica placement in tree networks subject to server capacity and distance constraints, and proposed efficient approximation algorithms for the placement problem without considering the consistency maintenance.

So far, there has been no consistency maintenance method specifically proposed for large-scale CDNs. Also, there has been no study that investigates the content inconsistency in current CDNs based on real trace. This is the first work that analyzes the consistency performance and causes in a major CDN based on the real trace, and extensively evaluates the consistency and overhead performance in trace-driven experiments in different scenarios.

V. Conclusions

In this paper, we analyzed our crawled trace data of a cached sports game content on thousands of servers in a major CDN. From our analysis, we pointed out that the inconsistency problem does exist in the CDN. We analyzed the consistency from different perspectives, such as inner- or inter-clustering, geographical distance to the content publisher and absent time of a server and so on. From the analysis, we not only comprehensively evaluated the inconsistency of dynamic content among the CDN's servers, but also broke down the reasons for inconsistency among end-users. Finally we further evaluated the performance in consistency and overhead for different infrastructures with different update methods and itemized the advantages and disadvantages. We aim to give guidance of appropriate selections of consistency maintenance infrastructures and methods when building a CDN or choosing a CDN service. In our future work, we will study a hybrid and self-adapted consistency maintenance method that can change the update method and infrastructure by considering the varying update and visit frequencies of the live content as well as the consistency requirements from the customer.

Acknowledgements

This research was supported in part by U.S. NSF grants IIS-1354123, CNS-1254006, CNS-1249603, CNS-1049947, CNS-0917056 and CNS-1025652, Microsoft Research Faculty Fellowship 8300751.

References

[1] D. Rayburn. CDN Market Getting Crowded: Now Tracking 28 Providers In The Industry. *Business of Online Video Blog*, 2007.

[2] Akamai. http://www.akamai.com/.

[3] M. Zhao, P. Aditya, Y. Lin, A. Harberlen, P. Druschel, W. Wishon, and B. Maggs. A First Look at a Commercial Hybrid Content Delivery System. http://research.microsoft.com/apps/video/default.aspx?id=154911.

[4] C. Huang, A. Wang, J. Li, and K. W. Ross. Measuring and Evaluating Large-Scale CDNs. In *Proc. of IMC*, 2008.

[5] G. Pallis and A. Vakali. Insight and Perspectives for Content Delivery Networks. *Communications of the ACM*, 2006.

[6] Z. Fei. A novel approach to managing consistency in content distribution networks. In *Proc. of WCW*, 2001.

[7] P. Cao and C. Liu. Maintaining Strong Cache Consistency in the World-Wide Web. In *Proc. of the Seventeenth International Conference on Distributed Computing Systems*, 1997.

[8] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham. Consistency Maintenance in Peer-to-Peer File Sharing Networks. In *Proc. of WIAPP*, 2003.

[9] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *Proc. of ICDCS*, 2003.

[10] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proc. of the International Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[11] A. Ninan, P. KulkarniI, P. Shenoy, K. Ramamritham, and R. Tewari. Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks. In *Proc. of WWW*, 2002.

[12] M. Roussopoulos and M. Baker. CUP: Controlled Update Propagation in Peer to Peer Networks. In *Proc. of USENIX*, 2003.

[13] L. Yin and G. Cao. Ynamic-Tree Based Update Propagation in Peer-to-Peer Networks. In *Proc. of ICDE*, 2005.

[14] X. Chen, S. Ren, H. Wang, and X. Zhangi. SCOPE: Scalable Consistency Maintenance in Structured P2P Systems. In *Proc. of INFOCOM*, 2005.

[15] Z. Li, G. Xie, and Z. Li. Locality-Aware Consistency Maintenance for Heterogeneous P2P Systems. In *Proc. of IPDPS*, 2007.

[16] H. Shen. GeWave: Geographically-Aware Wave for File Consistency Maintenance in P2P Systems. In *Proc. of ICPP*, 2008.

[17] V. Cate. Alex: A Global File System. In *Proc. of the USENIX File System Workshop*, 1992.

[18] H. Shen. IRM: Integrated File Replication and Consistency Maintenance in P2P Systems. *TPDS*, 2010.

[19] Best Practices when Developing with Akamai. http://http://seabourneinc.com/2011/04/28/best-practices-when-developing-with-akamai/.

[20] Iplocation. http://www.iplocation.net/.

[21] V. Valancius, C. Lumezanu, N. Feamster, R. Johari, and V. V. Vazirani. How Many Tiers?: Pricing in the Internet Transit Market. In *Proc. of SIGCOMM*, 2011.

[22] PlanetLab. http://www.planet-lab.org/.

[23] M. P. Wittie, V. Pejovic, L. B. Deek, K. C. Almeroth, and Y. B. Zhao. Exploiting Locality of Interest in Online Social Networks. In *Proc. of ACM CoNEXT*, 2010.

[24] X. Tang and S. Zhou. Update Scheduling for Improving Consistency in Distributed Virtual Environments. *TPDS*, 2010.

[25] S. Peluso, P. Ruivo, P. Romano, F. Quaglia, and L. Rodrigues. When Scalability Meets Consistency: Genuine Multiversion Update-Serializable Partial Data Replication. In *Proc. of ICDCS*, 2012.

[26] J. Xiong, Y. Hu, G. Li, R. Tang, and Z. Fan. Metadata Distribution and Consistency Techniques for Large-Scale Cluster File Systems. *TPDS*, 2011.

[27] T. Distler and R. Kapitza. Optimal Algorithms and Approximation Algorithms for Replica Placement with Distance Constraints in Tree Networks. In *Proc. of EuroSys*, 2011.