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

Guoxin Liu, Haiying Shen, Harrison Chandler, Jin Li

Presenter: Haiying Shen
Associate professor
Department of Electrical & Computer Engineering
Clemson University
Outline

• Introduction
• Related work
• Inconsistency analysis
• Performance evaluation
• Conclusion
Introduction

• Content Delivery Networks (CDNs)
  – Cache/replicate contents to surrogate servers near the network edge
  – Optimize the end user experience with short access latency

• Trends:
  – Increased number of enterprises (28 commercial CDNs)
    • Akamai, Limelight, Level 3, Turner, ChinaCache, ...
  – Scale up rapidly, as Akamai:
    • 85,800 servers in 1800 districts over 79 countries
    • Growing scale: 50% servers due to 100% increases of traffic per year
Introduction

- **Architecture of CDNs**
  - (1)-(4) recursively resolve the hostname
    - (2)-(3) for load balancing with Locality awareness
  - (5)-(8) get the requested content
    - Acting as a proxy

- **Dynamic contents: (live game statistics)**
  - Non-trivial for consistency maintenance: Large amount & widely scattered replicas
  - Introduce two requirements: **Scalable and consistency guarantee**
Introduction

• Our contribution:
  – To help develop consistency maintenance approaches for CDNs by answering
    • Can the current update method used in the CDN provide high consistency for dynamic contents?
      – Measuring the inconsistency of a major CDN
    • What are the reasons for the content inconsistency?
      – Breaking down the inconsistency reasons
    • What are the advantages and disadvantages of employing previously proposed consistency maintenance approaches in the CDN environment?
      – Trace-driven experiments show the performance
Outline

• Introduction
• Related work
• Inconsistency analysis
• Performance evaluation
• Conclusion
Related work

• Update infrastructures:
  – Unicast: Low scalability
  – Broadcast: High overload
  – Multicast: Not dynamism resilient

• Update method:
  – Time To Live (TTL): high scalability vs. low consistency
  – Push: high consistency vs. unnecessary traffic
  – Invalidation: traffic saving vs. long access latency

• Problem:
  – None of current update infrastructures together with update methods can achieve both scalability and consistency guarantee with traffic cost minimization.
Outline

• Introduction
• Related work
• Inconsistency analysis
  – Trace crawl
  – Inconsistency breakdown
• Performance evaluation
• Conclusion
Trace crawl

• CDN and content publisher server crawl process:
  – Retrieval all domain names after crawling all webpages of a sport game portal
  – 300 geo-distributed PlanetLab nodes to get IPs through local DNS service
    • Domain -> CNAMEs -> Edge server URL-> IPs
  – 10 IPs of the provider and 50064 IPs of the CDN

• Content crawl process:
  – 200 globally distributed PlanetLab nodes.
    • Towards 3000 random selected IPs
  – Live statistical webpages served by a major CDN
    • 15 day trace between May 15, 2012 and June 4, 2012.
Inconsistency breakdown

- Inconsistency measurement method
  - $C_i$: The $i^{th}$ update
  - $\alpha_{C_i}$: The first time when $C_i$ shows up among all servers
  - $\beta_{S_{C_{i-1}}}$: The last time when $C_{i-1}$ shows up
  - $\Delta_{C_{i-1}} = \text{Max}\{\beta_{S_{C_{i-1}}} - \alpha_{C_i}\}$

Server A  Server B  Server C  Server D

Time

$\alpha_{C_2}$

$\Delta_{C_1}$

$\beta_{S_{C_1}}$

$\beta_{S_{C_2}}$

$\Delta_{C_2}$

$\alpha_{C_3}$
Inconsistency breakdown

• Is there any inconsistency?
  – 10.1% having inconsistency < 10s
  – 20.3% having inconsistency > 50s

• Does a user can observe inconsistency?
  – Inconsistency: Continuous inconsistency time is proportional to TTL of a user’s browser
  – Cause: Switching between CDN edge servers
  – Conclusion: The edge servers have inconsistencies
Inconsistency breakdown

- **Effect of TTL of CDN servers**
  - Measure the inner cluster inconsistency (by location)
    - Exclude the propagation delay effect
    - TTL = 80s = 2 * Average inconsistency = 2 * 40s
  - **TTL refinement**
    - Exclude the other factors’ affection
    - Calculate the standard deviation
      - Expected distribution VS. Actual distribution within expected TTL
    - TTL=60s (with smallest deviation) = 75% of 80s
Inconsistency breakdown

• Effect of content provider’s inconsistency
  – 90.2% of served requests have inconsistency < 10s
  – Average inconsistency = 3.43s = **4.3%** * 80s

• Effect of provider-server propagation delay
  – Average consistency ratio VS. provider-server distance
  – Correlation between two factors = 0.11
    • Little effect on inconsistency
Inconsistency breakdown

- **Effect of content provider’s bandwidth**
  - Measure the response time for querying contents
  - $[0.5, 2.1]$s and 90% requests < 1.5
  - Inconsistency effect: $0.5s = 0.6\% \times 80s$ (negligible)

- **Effect of CDN server failure and overload**
  - Measure the inconsistency after the absence with certain length
  - Effect < 8s = **10%** *80s
Inconsistency breakdown

• Possible causes:
  – TTL of CDN servers
  – Provider-server propagation delay
  – Content provider servers’ inconsistency
  – Content providers’ bandwidth
  – CDN server overload and failure

• Influence:
  – TTL contributes around 75% of average inconsistency
    • Easy to solve by changing update methods
  – Other factors contribute significantly less than TTL
    • Expensive to solve compared to TTL
Outline

• Introduction
• Related work
• Inconsistency analysis
• Performance evaluation
• Conclusion
Performance evaluation

• Experimental settings:
  – CDN servers: 170 PlanetLab nodes with high performance and light load in the U.S., Europe, and Asia.
  – Content provider server: One PlanetLab node in Atlanta
  – Trace: Live game events on Jun. 2nd, 2012
    • 306 different snapshots
    • 2 hours and 26 minutes long
  – Users: Each PlanetLab node simulates five browsers
Performance evaluation

• Inconsistency under unicast
  – Inconsistency among CDN servers
    • Push < Invalidation < TTL
    • Push needs a long time to update (central server bottleneck)
  – Inconsistency among users
    • Push \approx \text{Invalidation} < \text{TTL}
Performance evaluation

- **Inconsistency under multicast**
  - **Inconsistency among CDN servers**
    - Push < Invalidation < TTL (larger inconsistency for nodes at lower level in the tree)
    - Push needs a small time to update (scalable)
  - **Inconsistency among users**
    - Push $\approx$ Invalidation < TTL
Performance evaluation

• Traffic cost
  – Multicast vs. Unicast
    • Unicast > Multicast (locality-aware)
  – Traffic cost for different methods
    • Scenario: Frequent & Rare updates and frequent visits
    • Push < invalidation < TTL
    • Different scenarios lead to different results -> Provide guidance for selecting or designing a CDN’s consistency maintenance methods

![Traffic cost comparison](image1)

![Traffic cost over TTL](image2)
Performance evaluation summary

• Update methods
  – Push:
    • Better consistency in a small-scale network
    • Lacks of scalability
  – Invalidation:
    • Similar consistency guarantee as Push to users with reduced traffic cost
    • Has heavy network burden for invalidation notification for frequently updated contents
  – TTL
    • Weak consistency
    • Better scalability
    • Waste cost by rarely updated contents

• Update infrastructures
  – Unicast
    • Little effect on inconsistency
    • Lacks of scalability
  – Multicast
    • Scalable (needs dynamism resilience)
    • Large effect on inconsistency when using TTL
Outline

• Introduction
• Related work
• Inconsistency analysis
• Performance evaluation
• Conclusion
Conclusion

• Trace on thousands of CDN servers
  – Inconsistency does exist and users can observe it
  – Possible causes:
    • 1) TTL of CDN servers (major effect), 2) Provider-server propagation delay, 3) Content provider servers’ inconsistency, 4) Content providers’ bandwidth 5) CDN server overload and failure (large effect)

• Experiments (thousands of CDN servers)
  – Different infrastructures and methods
    • Effectiveness: consistency performance
    • Overhead: scalability

• Future work:
  – A hybrid and self-adapted consistency maintenance method
    • Scalable & Consistency & Cost minimization
Thank you!

Questions & Comments?

Haiying Shen

shenh@clemson.edu

Associate Professor of Electrical and Computer Engineering

Clemson University