Edge Cache Offloading and Freshness-Sensitive Caching

Paper Review by David G. Watson

Overview

- Dynamic pages are no longer the exception but rather the rule
- Different techniques have been used to improve responsiveness and lessen server-side load
  - Offload serving of static content to proxies so server can concentrate on dynamic requests
  - Distribute database web application servers so that the server nearest the client handles requests

Evaluation of Edge Caching/Offloading for Dynamic Content Delivery

By Chun Yuan, Yu Chen, and Zheng Zhang of Microsoft Research Asia

Related Work

- Server-side content caching
- Fragment caching at proxy or at CDN stations
  - Fragments are static pieces of dynamic pages that can be cached and reassembled
- Application offloading
  - Presentation and Business Logic layers moved to edge server
  - Origin server only for back-end database

Work has been done in the following areas

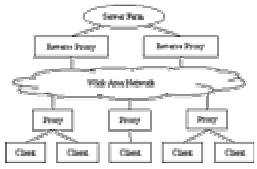
- Server-side content caching
- Fragment caching at proxy or at CDN stations
- Application offloading
  - Presentation and Business Logic layers moved to edge server
  - Origin server only for back-end database

New Information in Paper

- This paper focuses on proxies near clients – these are the edge servers
- First to examine tradeoffs for different partitioning schemes

Evaluation of Edge Caching

- 4 Places offloading can happen
  - Client
  - Proxy
  - Reverse Proxy
  - Server

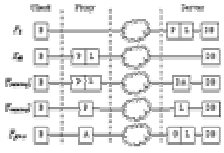


Edge Caching Continued

- Three important factors must be considered in any offloading scheme
  - Security
    - What can we trust to be handled outside of our control?
  - Complexity
    - Is this too difficult to be practical?
  - Performance
    - Does this actually improve speed, or may it result in slowdowns?

Experimental Setup

- Runs on Microsoft .NET
- Uses established benchmark – the Web Pet Shop
- Tried five ways of arranging the components



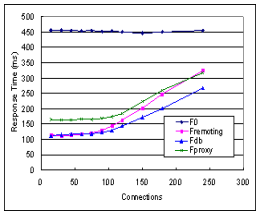
B: Browser
A: Page Assembling and Fragment Caching
G: Fragment Generation
P: Presentation
L: Business Logic except Data Access Layer
DA: Data Access Layer
L: Business Logic
DB: Database
Cloud: Wide Area Network

Offloading Schemes

- F_0 - Front end runs Pet Shop, back end runs DB (No change from original setup)
- $F_{remoting}$ - Presentation tier at front-end, rest at back-end
- F_{db} - Pet Shop runs at front end, accesses DB at back-end
- F_{proxy} - Page assembly & fragment caching at front-end, modified Pet Shop & DB at back-end

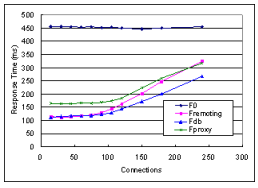
Experiment Results – Response Time

- All the methods tried were faster than no offloading
- Fdb was the fastest, with Fremoting very comparable under 300 connections



Experiment Results – Server Load

- Aggregated server loads shown
- With the offload configurations, front end servers are bottleneck
- Again Fdb is the best, followed by Fremoting and Fproxy.



Conclusions

- If there is end-to-end security, offload all the way up to the proxy
- If this is not the case, augment proxies with fragment caching and page generation

Paper Strengths / Weaknesses

- Overall was quite a strong paper – excellent information on tradeoffs of various schemes
- Methods suggested can involve significant re-engineering if applications were not initially designed to be distributed
- Security is a major consideration for any proxy-side operations except for fragment caching and page generation

Engineering and Hosting Adaptive Freshness-Sensitive Web Applications on Data Centers

Wen-Syan Li, Oliver Po, Wang-Pin Hsiung, K. Selcuk Candan, Divyakant Agrawal

Introduction & Related Work

- Much work has been done applying caching solutions to web database applications
- All prior work has focused on static schemes for assuring freshness, and has not looked at adaptive methods

System Architecture

- Wide area database replication & web application suite
  - Distributed so that the whole server suite can be closer to the users
  - Results in better latency and less workload
- Terminology
  - Origin Site: consists mainly of the master DB
  - Datacenter: mirror DB, Web App. Server (WAS)
  - Edge cache: located near users

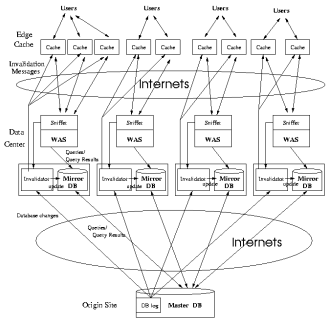
Problem: Freshness

- Freshness is defined as how out-of-date is the information received
  - If latency from server to browser is 8 seconds, we know the information is at least 8 seconds old
  - We want the information received to be as fresh as we can get it
  - This is complicated, because we must synchronize out mirror databases with the master DB at the datacenter

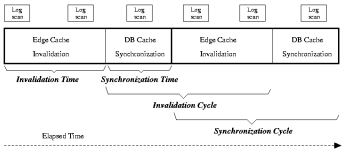
Proposed New Architecture

- Two new software components added to suite
  - Sniffer installed at Web App Server (WAS) and master DB
    - At WAS, creates mappings between URLS and query statements issued for requested pages
    - At master DB, Tracks DB content changes
  - Invalidator installed at DB mirrors
    - Propagates content change log from master to mirrors
    - Performs invalidations checks for caches based on DB changelog, URL/query mapping, content in mirror DB

CachePortal System

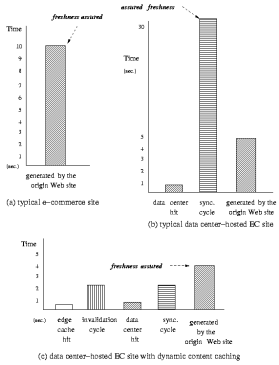


System Parameters

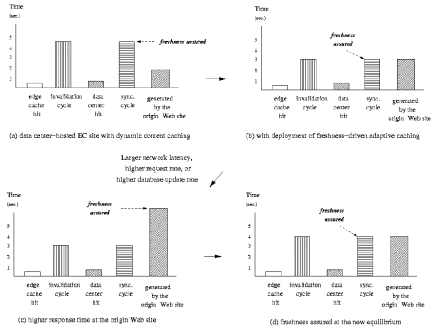


Invalidation and Synchronization are interleaved.
Invalidation Time: time required to process invalidation checks on all page in edge caches
Synchronization time: time required to propagate database updates from master DB log to DB caches

Freshness in current systems



Freshness with New System



Determination of Query Types

- One example query type might be 'SELECT firstname, lastname FROM people WHERE idnumber=$var'
  - An instance of query might be 'SELECT firstname, lastname FROM people WHERE idnumber=234'
- Invalidator tracks query types that have been invalidated by examining DB change log

Adaptive Caching

- We want our response time to be close to the invalidation cycle, so system is at equilibrium
- We do this by adjusting the number of cached query types
  - If response time is too high, increase number of cached types
  - If response time is too low (relative to invalidation cycle), reduce number of cached types

Experimental Setup

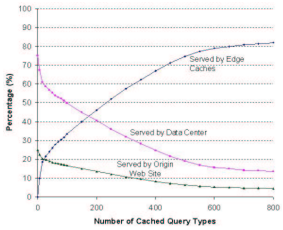
- 2 heterogenous networks
  - Negligible local latency
  - 250 ms latency between the two
- Server Setup
  - All machines PIII 700MHz, 1 GB RAM, RedHat 7.2

Experimental Setup

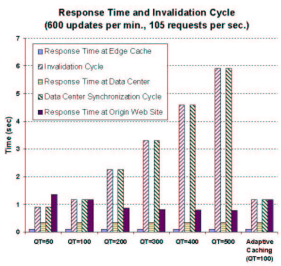
- Server Software
  - WAS: BEA WebLogic 7.0
  - DBMS: Oracle 9i
    - 7 tables, 1M rows, 600 row updates per table per minute
  - Cache: Modified version of Squid
    - 1M cached pages of 1000 query types, 1000 instances of each
    - 20% of queries non-cacheable due to security, authentication, dynamic content, etc.

Experiment Results

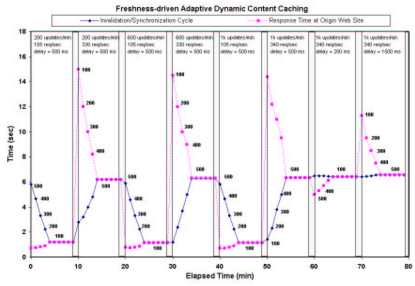
- Number of cached query types affects which pages are served by caches
- This in turn affects response time



Results continued

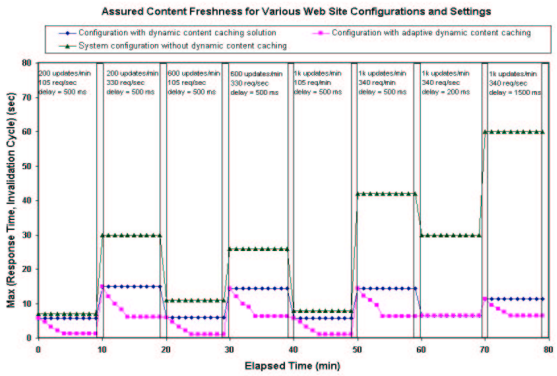


More Experiment Results



- Here we see effects of adaptive freshness-sensitive system
- System adjusts number of cached query types until it reaches an equilibrium point as the number of updates changes

Results Continued



Advantages

- Under heavy loads, freshness-driven adaptive caching support content freshness up to 20x better than without it
- Freshness 7x better than datacenter-hosted applications with dynamic content caching
- Provides faster response times and better scalability

Advantages / Disadvantages

- On the positive side, there were many experiments performed that tested performance in many different situations
- On the negative side, no clear side-by-side comparisons of the various alternatives
- Actually engineering the solutions looks to be very complex

Comparison of Papers

- Caching/Offloading paper had excellent information, but security issues may pose difficulties
- Freshness-sensitivity paper introduces innovative new technique, which can improve functioning of co-located web database apps, but it is difficult to know how the adaptive caching will actually work – few details were given on how it was implemented