"Caching Strategies for Data-Intensive Web Sites"

by Khaled Yagoub, Daniela Florescu, Valerie Issarny, Patrick Valduriez

Sara Sprenkle
CPS296
March 5, 2002

Motivation: The Problem

- Dynamic Web services
  - Require processing by database and Web server
  - Difficult to scale
- Reduce response time
  - Decrease processing time
  - Decrease server/database load

Motivation: Problem Context

- Architecture of declarative data-intensive Web services:
  - Data stored in a DBMS
  - HTML code is separate from generation
  - Page structure and content is separate from page layout (XML)
  - Logical model describes structure and data content (graph)
  - Declarative definition language describes how raw data maps to logical model (SQL)

Motivation: The Problem

- Response time includes:
  - Network communication time
  - HTTP connection time
  - Web application startup time
  - DBMS connection time
  - SQL execution time
  - XML generation time
  - HTML generation time

Motivation: Evaluating solutions

- Previous work: pushed Web caching solutions to extreme
  - Cache: query results or Web pages
  - Update strategies: push or pull
  - Choices were not appropriate for all cases
- Analyze real bottlenecks of data-intensive web sites
  - Create an application-appropriate solution

Previous Work

- Materialize HTML pages
  - Dynamic: on the fly
  - Static: before requested
    - good response time
    - coarse materialization granularity—pages contain multiple fragments
    - high space overhead (including duplicate information, template)
    - difficult to propagate updates into materialized pages
    - cannot handle responses to forms

Previous Work

- Cache query results
  - Benefits apps with high query execution cost, high hit ratio
  - Simpler update algorithms
  - Can control cache granularity
  - Increases the load on the DBMS

Previous Work

- Cache XML data representations
  - Intermediate data abstractions
  - Less redundant data stored, if XML fragments divided up appropriately
  - Beneficial when XML fragments are much smaller than the HTML pages
  - Reduces overhead of executing DB queries

Goal

- Flexible caching policies, strategies
  - Appropriate for set of applications running on the server
- Support caching of HTML, XML, and queries

Materialization Strategies

- What kind of data should be materialized?
  - Queries, XML, HTML
- When must materialization be performed?
  - {before, after, predictive} wrt requests
- Where should the materialized intermediate results by placed for effective performance improvement?
  - DB server, Web server, proxy, Web client

Materialization Strategies

- How are updates from the database propagated to the materialized data?
  - Push: guarantee freshness, costly
  - Pull: potential staleness, OK for some apps
- Which particular data items must be materialized and which ones must be computed upon request?
  - Cache items which have high computation costs, do not get updated frequently, are accessed frequently
  - Otherwise, compute-on-request

Answering the questions

- Answers depend on characteristics of the system
  - Size of Web site, usage patterns
  - Freshness, response time constraints
  - Hardware and software environment

Weave

- Web site management system
- Goal: application-appropriate caching strategies

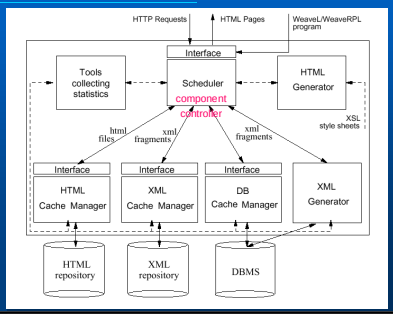
WeaveL

- Declarative language for Web site specification→structure, content
- Site class: describes a class of Web pages
- Define each Web page as an instance of site class
  - Describes content of Web page
  - Instances of class are distinguished by specified parameters

WeaveL

- Specification of site class includes
  - Description, query for parameters
  - SQL query to execute to get content
  - Hyperlinks, forms in page

Site Architecture



Customizing runtime policies

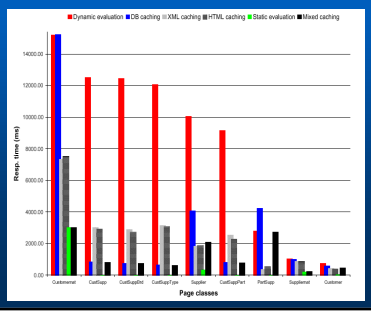
- WeaveRPL – language used to describe the caching policies
  - Rules for what, when, where, how, and how much to cache
    - Includes cache replacement policy
  - Language allows for flexible, application-appropriate policies

Experiments

- WaveBench – test platform
- ⩾1 client on ⩾ 1 machine
- TPC/D benchmark
- Caching strategies
  - Dynamic evaluation – all processing on the fly, worst case
  - Static evaluation – all precomposed
  - Mixed caching – flexible, application-appropriate caching

Summary of results

colors in image courtesy of Kevin Walsh



Analysis

- Dynamic evaluation is usually worst—base case
- Results as expected (?)
- Mixed caching performs the best of practical options
  - Exception: PartSupp, Supplier

Missing analysis

- Experiments
  - No updates
  - Only one client making requests because of high XML overhead
    - Prove scalability?
- Quantify cache hit rates
  - How many requests hit in cache?
  - How many results fit in cache?

Missing analysis

- Quantify usability, complexity
  - How difficult to create caching strategies?
  - How many iterations before find appropriate strategy?
  - How does increased caching effect scheduler performance? (Scheduler becomes bottleneck, single point of failure)
- Quantify overhead
  - Basically have three-tier caching; what is the overhead of visiting each cache?

Missing analysis

- Scaling scheduler load
  - Adding new components which also require scaling
  - May require coordinating schedulers

Future Work

- Ease configuration
  - Difficult to configure site automatically
  - Requires programmer knowledge to create reasonable policies
  - Have tools to ease administrator's analysis
- Prototype - update propagation
  - Only pull model

Conclusions

- Created a generic framework for flexible, application-appropriate caching strategies in data-intensive Web sites
- Need more experiments, results, and analysis to prove that this approach is correct, practical, useful

Web Caching: The Problem

- Problem: Dynamic content changes when parameters and underlying data changes
  - Application-defined consistency, staleness constraints
  - Data updated consistently, completely
  - Content that depends on changed data must be updated within some bounds

Web Caching: Solutions

- WebView
  - Materialization: virtual, in DBMS, in Web server
  - Performance depends on app characteristics
- TriggerMonitor
  - Given data→content dependencies, can apply update algorithm with low overhead
  - Scalable (handles high request rates)
  - Practical: implemented on Olympics Web site
  - Potential for wasted resources

Web Caching: Solutions

- Weave
  - Generic framework for flexible caching policies
  - Usable? Practical?
    - No updates

The Future of Web Caching

- Still a hot research area
  - Haven't found the best model, solution
- Problems left to solve
  - Update policies
  - Consistency
    - Application-appropriate → TACT