Web Data Management

Charlie Garrod
25 Apr 2005

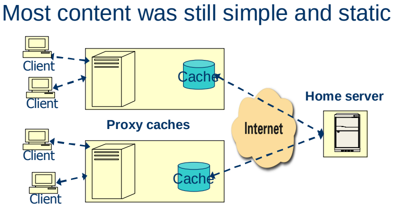
12 years ago…

- Al Gore had just invented the internet
- A (relatively) small number of users put content on the web
- And a (relatively) small number of users downloaded it

Most content was simple!

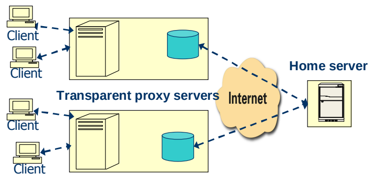
8 years ago... web caches

- Much larger number of users
- Most content was still simple and static



5 years ago... CDNs

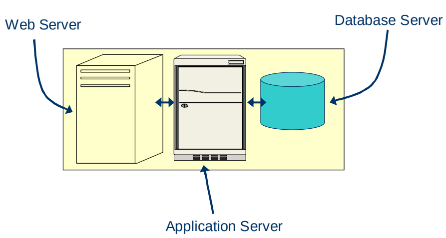
- Content Distribution Networks
- Move web content to the “edge”



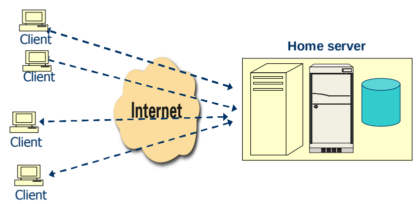
Today…

- Web content is complex and dynamic
  - interactive and personalized
- Amazon, CNN, Google, USAir, LiveJournal, and of course the 15-721 course homepage…

Dynamic content generation



Dynamic content generation



Web database workloads

- Most queries are small and simple (OLTP)
  - Show me the last 25 journal entries by “puuj”
  - Show me non-full flights to LAX next Friday
  - Find all websites about fire-breathing space monkeys
- Few updates
- Other than that, workloads vary greatly between applications

Web database workloads

- Queries and updates are often instantiations of more general templates
  - Q1: SELECT id FROM users WHERE age > ?
  - U2: UPDATE users SET age = ? WHERE id = ?

The $65,536 question:

How do we make dynamic content scalable?

Web Data Management Outline

- Introduction
- Overview of common approaches
- WebView Materialization
- DBProxy: A dynamic data cache for Web applications
- Conclusions

Web Data Management Outline

- Introduction
- Overview of common approaches
- WebView Materialization
- DBProxy: A dynamic data cache for Web applications
- Conclusions

Solution #1: WebView Materialization

Make the content "static"

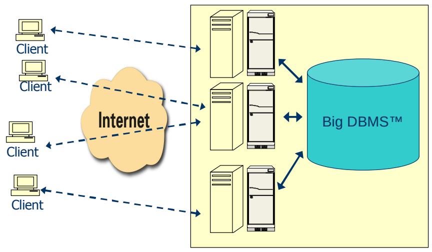
Solution #1: WebView Materialization

- Generate new static version of webpage every time it is updated
- Works great for CNN, Slashdot, etc. where the content is semi-static
  - Does not adapt well to personalized or interactive websites

Solution #2

Build a custom solution

Solution #2: Big DBMS™



Solution #2: Big DBMS™

- Build a custom, semi-centralized DBMS system
- Good for big companies such as Google, Amazon, EBay, etc. with an established user base and significant market investment
- Very expensive to implement!

Solution #3

Try something else!

Solution #3: Dynamic CDN

- Try to apply the principles of caching and content-distribution to dynamic web pages
  - Build a nice, general solution to scale dynamic workloads
  - Adaptable to personalization and interaction
  - Cheaper than a custom, specialized solution

Solution #3: Dynamic CDN

- Try to apply the principles of caching and content-distribution to dynamic web pages
  - Build a nice, general solution to scale dynamic workloads
  - Adaptable to personalization and interaction
  - Cheaper than a custom, specialized solution

This is easier said than done!

Web database workloads revisited

- Most queries are small and simple (OLTP)
  - Show me the last 25 journal entries by “puuj”
  - Show me non-full flights to LAX next Friday
  - Find all websites about fire-breathing space monkeys
- Few updates
- Other than that, workloads vary greatly between applications

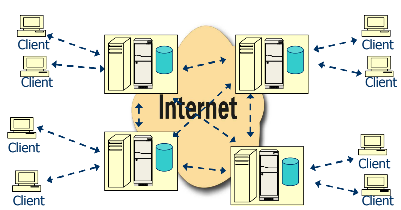
Distributing dynamic content

- Which server components should we distribute?
  - Everything?
  - Just the web server and application server?
  - Partially replicate the database?

Distribute everything!

- All proxy servers contain a web server, app server, and database
- The perfect solution for scaling queries!
- Updates are practically impossible
  - Distributed databases are fundamentally hard to build and are usually intended only for LANs

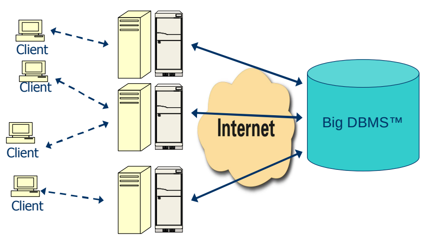
Distribute everything!



Distribute the web and app server!

- Efficiently off-loads the web server and application execution to remote proxy servers
  - Reduces bandwidth usage
- Still relies on a centralized database
- Interactions with the database become high-latency

Distribute the web and app server!

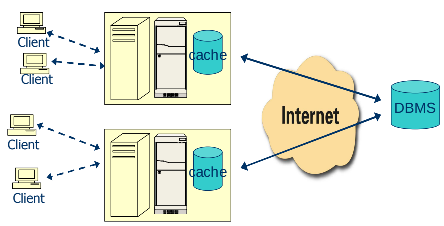


And finally…

Partial Replication (and Caching)

- Distributes web and app server load as before
  - Reduces bandwidth, etc.
- Updates are potentially less expensive than with full replication
  - But still non-trivial

Partial Replication (and Caching)



Intermission

Web Data Management Outline

- Introduction
- Overview of common approaches
- WebView Materialization
- DBProxy: A dynamic data cache for Web applications
- Conclusions

WebView Materialization

- Strategy #1: make the content static
- Labrinidis and Roussopoulos, University of Maryland, circa 2000.
- Introduced a formal cost model for evaluating materialization of “WebViews” at the web server, within the DBMS, or not at all
- Experimentally evaluated the different strategies

Strategy #1: “Virtual” materialization

- Query is re-executed at database and webpage is regenerated
- Updates are cheap since only the “standard” update must be executed at the DBMS
- Queries are expensive since all work must be re-done every time

Strategy #2: Materialization at DBMS

- The query result is saved at the database, but the resultant webpage itself is regenerated
- Updates are more expensive since the materialized view at the DBMS must be regenerated as well
- Queries are slightly cheaper since only the webpage must be regenerated

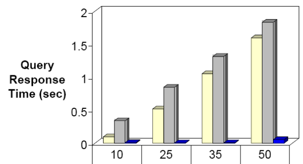
Strategy #3: Materialization at web server

- The full materialized webpage is stored at the web server
- Updates are very expensive, essentially the cost of a standard update plus a query plus the cost of generating the resultant webpage
- Queries are very cheap since the page is just retrieved as if it were static content

Experimental Methodology

- Used a single Sun system as a server (running Apache and Informix), 22 Sun systems as clients, all within a single LAN
- Measured query response time for each strategy for various access rates, update rates, number and size of views, and view selectivity

Results, yada, yada



	10	25	35	50
virt	0.09604	0.51774	1.05175	1.59493
mat-db	0.33903	0.84658	1.3145	1.83115
mat-web	0.00921	0.00459	0.00576	0.05372

Access Rate (requests/sec)

Problems with their methodology

- Relatively small number of views (100-2000)
- Results are indicative of an open system under low load
  - For “materialization at web server” updates are executed as a separate background process
  - Only query response time is measured
  - Cheaters!

WebView Materialization Conclusions

- Still show that materialization at web server can effectively reduce overall load for a relatively small number of views, which can greatly improve performance for some loads
- Somewhat surprising that materialization at DBMS often hurts!
- A nice mix of theoretical and experimental methodology!

Web Data Management Outline

- Introduction
- Overview of common approaches
- WebView Materialization
- DBProxy: A dynamic data cache for Web applications
- Conclusions

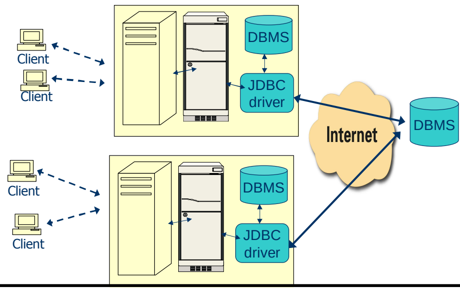
DBProxy: A dynamic data cache…

- Amiri et al., IBM T.J. Watson, circa 2002.
- Based on partial replication
  - Queries are processed locally at a proxy server if possible
  - All updates forwarded to a central database, which periodically propagates the updates to the proxy servers

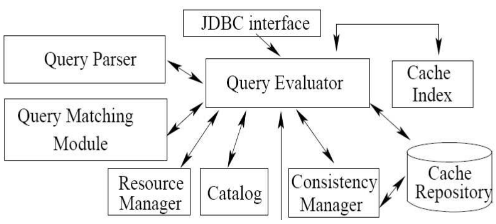
Overall goals

- Database independence
  - Any back-end database could be used
- Self-management
  - Cache dynamically adapts to a changing workload without administrator intervention
- Consistency
  - Must be efficient even with a large cache and heavy update traffic

DBProxy architecture



DBProxy JDBC driver architecture



DBProxy local database

- Stores subsets of tables from the central database (both horizontally and vertically partitioned)
- And catalog information from the central database…
- And information about the queries that are currently cached…

DBProxy query matching

- Uses the SELECT and WHERE clauses to determine if the query is a subset of the union of queries already in the cache
- Can potentially answer queries that have not yet been issued before
  - Q1: SELECT id FROM users WHERE age < 25
  - Q2: SELECT id FROM users WHERE age > 18
  - Q3: SELECT id FROM users WHERE age > 21

DBProxy update mechanism

- All updates are forwarded to the central database
- All proxies subscribe to a stream which contains all updates at the database
  - Not just the updates they care about

DBProxy consistency guarantees

- Lag consistency
  - The proxy server is not too outdated
- Monotonic state transitions
  - The view of the database at the proxy moves only forward with time
- Immediate visibility of updates
  - An application observes the effects of its own updates

DBProxy consistency guarantees

- Lag consistency
  - The proxy server is not too outdated
- Monotonic state transitions

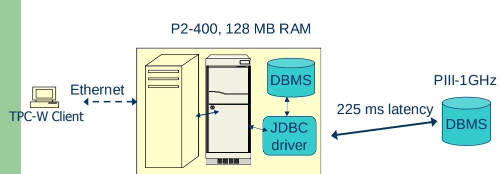
No transactional consistency! only

- Immediate visibility of updates
  - An application observes the effects of its own updates

DBProxy cache replacement

- Runs as a background process, garbage collecting results that are not used by any cached query and occasionally evicting cached queries to reclaim space
  - General replacement algorithm, taking into account recency and frequency of use, space used, miss cost vs. hit cost, etc.

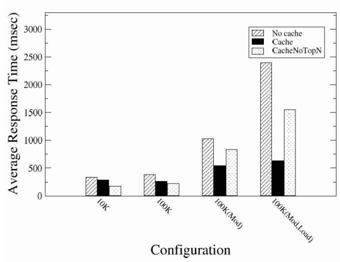
Experimental methodology



Experimental methodology

- Modified TPC-W (which simulates a simple web bookstore workload) to introduce some additional complexity
- Measured proxy response time and hit rate with several database sizes, several cache configurations, and various loads on the back-end database
- Started with a warm cache

Proxy response time



Proxy cache hit rates

	Baseline TPC-W			Modified TPC-W		
Query Category	**Response time**	**Hit rate**	**Query frequency**	**Response time**	**Hit rate**	**Query frequency**
Simple	51	91 %	23 %	317	37 %	47 %
Top-N	935	68 %	12 %	852	66 %	37 %
Exact-match	211	76 %	65 %	458	54 %	15 %
Total	263	73 %	100 %	540	50 %	100 %
No Cache	385	–	100 %	1024	–	100 %

Using 100K database, 80K users

Problems with their methodology

- No comparison to centralized-only configuration
- No mention of throughput, an important performance metric
- Used TPC-W browsing mix only
  - Did not measure the effect of various update loads on the system

Harsh (and slightly unfair) DBProxy conclusions

- Great cache configuration and query-matching
- Poor update-handling and consistency management
- While initially impressive, performance results do not support the use of DBProxy compared to a centralized architecture or for any workloads with a non-trivial update component

Web Data Management Outline

- Introduction
- Overview of common approaches
- WebView Materialization
- DBProxy: A dynamic data cache for Web applications
- Conclusions

Many similar projects

- DBCache (IBM Almaden), DBProxy (IBM Watson), GlobeDB (ETH Zurich)
- Similar projects that focus on file system workloads (UT Austin)
- And…

Shameless plug: S-3 (CMU)

- Ailamaki, Garrod, Maggs, Manjhi, Mowry, and Olston (among others)
- Efficient transactional consistency
- Theoretical framework for the effect of data secrecy on scalability
- Exploiting knowledge of query and update templates

Conclusions

- Overall, this is still very much an area of on-going research!
- Lots of people working on this problem, and nobody yet has come up with a satisfactory solution
- And it’s really a $64 billion question