Paper presentation

"Middle-tier Database Caching for e-Business"

Presenter: Yiwen Huang

Outline

- Introduction & background
- Design and implementation
- Evaluation methodology
- Experimental Results
- Conclusion
- Q & A

Introduction

- Problem description:
  - increase performance of multi-tier web-based applications
- What is multi-tier configuration?
  - the web server, web application server and the databse server resides on different machines
- Why using multi-tier configuration?
  - scalability:
    - workload balancing
  - availability:
    - fail over support

Introduction (cont'd)

- Problems in multi-tier configuration:
  - Back end database becomes the performance bottleneck
  - Back end databse is also a single point of failure
- Solution:
  - middle-tier database cache

Introduction (cont'd)

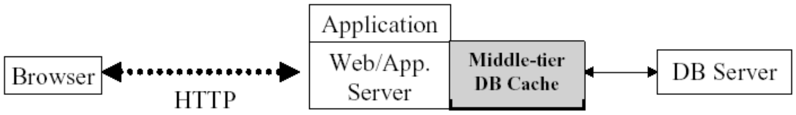

Figure 1: *Single-connection* Approach to Database Caching

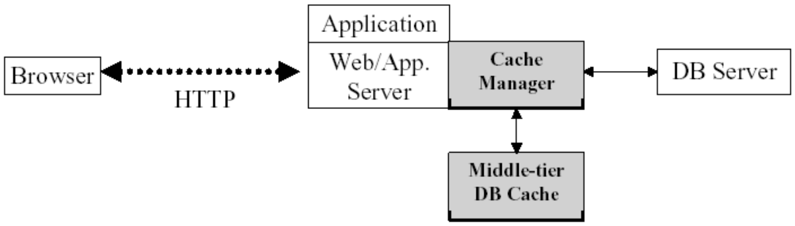

Figure 2: *Double-connection* Approach to Database Caching

Introduction (cont'd)

- Different types of solutions
  - special purpose solutions
    - eg. eBay
  - general-purpose industrial strength DBMS solutions
    - used in this paper
    - the Database Cache of Oracle's 9i IAS
    - TimesTen's Front-tier

Background

- e-Commerce application requirements
  - reliability
  - scalability
  - manageability
- e-Commerce application characteristics
  - mostly OLTP-type queries
  - table accesses are highly skewed on a few read-dominant tables
  - exist a clear separation between write-dominant tables and read-dominant tables

Background (cont'd)

- Rationale behind choosing a general-purpose industrial strength DBMS
  - provide transactional support, multiple consistency levels, and efficient recovery services
  - provide a variety of tools for application development
  - is transparent to the application, no change required in the application code

Background (cont'd)

- Rationale behind choosing DB2
  - leveraging existing DB2 federated (DataJoiner), and DataPropagator features
    - to provide query routing and data replication functions needed in the cache
  - be able to effectively process distribute queries
    - the query optimizer to decide what portion of the query should be processed in the front end and what portion is in the back end
  - reuse of existing technology

Background (cont'd)

- DB2 federated feature:
  - allow access to remote data through a single federated DB2 database
    - *federator* identifies the local database, which accepts user queries
    - *node* identifies a remote host
    - *server* identifies a remote database
    - *nickname* identifies a table or view in the remote database
    - the federator translates a user query over a local "alias" for remote data into a distributed query to remote data sources

Background (cont'd)

- DB2 DataPropagator feature
  - tools for asynchronous data replication for relational databases
  - used in conjunction with DataJoiner, can support non-relational data replication
  - consists of three independent programs:
    - a data change capture program
    - an update apply program
    - an administration program (contains control tables)
  - Uses setup replication requests through subscriptions:
    - specify which tables to replicate
    - specify frequency of update propagation
    - specify min. size of each data transfer

Design of DBCache

- Design Requirements
  - there should be no change in the application code, and the underlying database schema
    - DBCache is transparent to the application
    - DBCache is able to understand any SQL statements the back end database can handle
  - DBCache should support *reasonable* update semantics
    - relaxed condition due to e-Commerce application characteristics
      - high tolerance for slightly out-of-date data

Design of DBCache (cont'd)

- Caching Scheme
  - full table level caching
    - only need schema information
    - supports arbitrary queries on cached tables
    - OLTP-type queries does not need complex intermediate result caching
- Update Scheme
  - all update actions (UDI queries) are processed at back end database
  - change are propagated back to the DBCache by the DPropR program

DBCache Implementation

- Cache Initialization: DBCacheInit tool
  - purpose: automatically create the database schema for a cache database and initialize it
  - Steps:
    - gather back end database information
    - choose cacheable table (provided a-priori)
    - create cache database
    - load initial data and set up replication subscription

DBCache Implementation (cont'd)

- DBCache Mode
  - use DBMS instance level for easy implementation
  - support only one remote server per DBCache instance
- Auto-passthru
  - decides where to route the query, to the DBCache, to the back end database, or to both places
  - built on top of DB2's existing "set passthru" mechanism

DBCache Implementation (cont'd)

- Auto-passthru (cont'd)
  - query executed at the back end database if:
    - it is a UDI-type query
    - application need to access most up-to-date data
      - indicated by a special register: REFRESH-AGE
    - any nicknames in the query
      - handled by DB2 federated feature
    - DDL (Data Definition Language) statements
  - query executed at the DBCache if
    - it is a read-only query involves only cached tables
      - handled by DB2 federated feature
    - it is a query targeted to tables in DBCache
      - e.g. DPropR's apply program
      - achieved by "set passthru local" statement

Evaluation Methodology

- Middle-of-the-road approach
  - real e-commerce applications (both sofewere andhardware) to build the test environment
  - use an e-commerce benchmark to simulate workload
    - WCS: an integrated e-commerce solution
    - SilkPerformer: load and performance testing tool
    - ECDW benchmark: measures web applications and web transactions

Evaluation Methodology (cont'd)

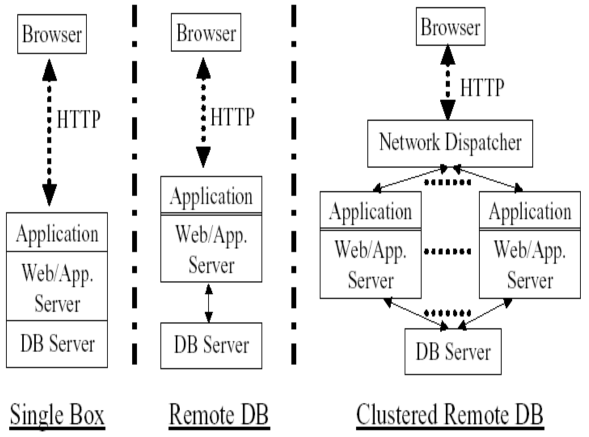

Figure 7: Three Non-Caching Server-side Topologies

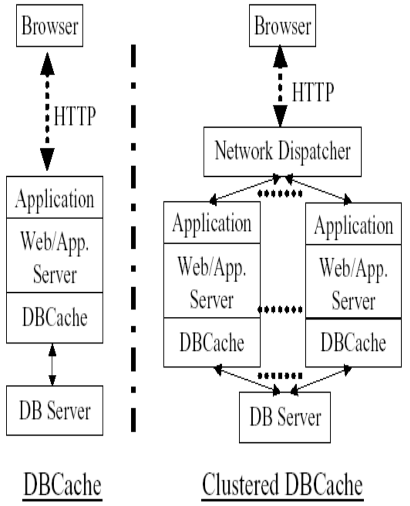

Figure 8: Two Caching Server-side Topologies

Experimental results

- Workload characteristics study
  - short query execution time
  - highly skewed table access
  - clear separation of read-dominant and write-dominant tables
- Most experiments are done on browsing-only scenario
  - browsing represent the majority of the total workload
  - browsing follow the same pattern as the regular shopping scenario

Experimental results (cont'd)

- Overhead of adding a front end cache
  - insignificant when the server is fully loaded

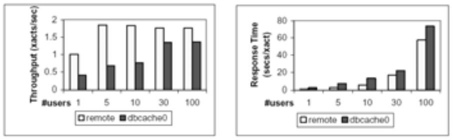

Figure 10: Overhead of Adding a Front End Cache with a 0% Cache Hit Rate

- Server workload sharing
  - DBCache outperforms when server is fully loaded

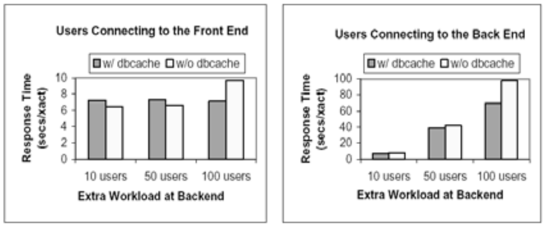

Figure 12: Caching Effect With Varying Server Workload

Experimental results (cont'd)

- Update propagation cost
  - insignificant at front end when the server is fully loaded
  - 20% overhead cost at back end when the server is fully loaded

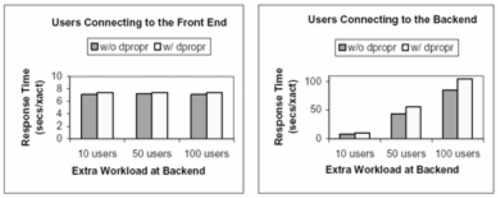

Figure 14: Update Propagation Cost with Varying Server Workload

- Web application server clustering
  - scale up throughput with the help of DBCache

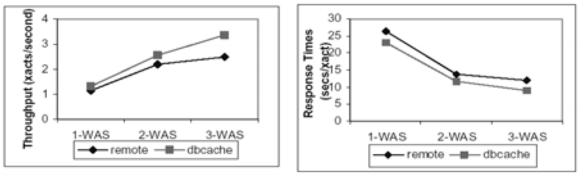

Figure 15: Varying Number of WAS machines

Conclusion

- Solve the performance problem in e-Business
  - scale up back end database
- Present the prototype implementation of a middle-tier database cache
  - re-use of existing technology
  - is able to handle distributed query

Conclusion (cont'd)

- How does this paper fit into the big picture of web caching?
  - there are two groups of latency in web-based application
    - network latency
    - server latency
- Middle-tier database caching improves on cross-tier communication and interaction bottleneck in server latency

A slightly different version of the paper can be found here:

http://www.almaden.ibm.com
/u/mohan/Middle-tier%20Database%20Caching%20for%20e-Business.pdf

This version contains more details about the "auto-passthru" mechanism in the DBCache prototype

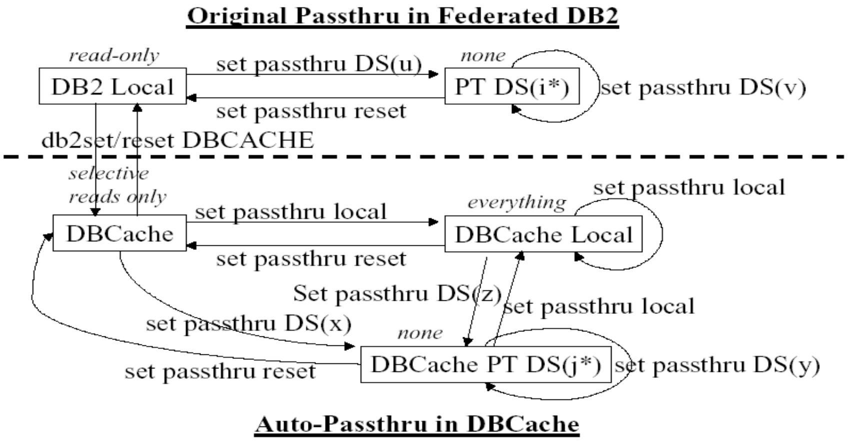

Figure 5: Comparison of Passthru State Transition