Throw Away Concurrency Control and Two Phase Commit

Arthur Whitney
KX Systems, a@kx.com

Dennis Shasha
New York University, shasha@cs.nyu.edu
Our Biases

- Lots of experience helping people speed up applications.
  Arthur: design a better programming language.
  Dennis: tune the databases and fix bugs.

- Modern database systems seem to require many escapes, e.g. for Wall Street analytics, graphical user interfaces. We prefer a monolingual environment that embodies a programming language.

- Our customers are impatient and very nervous.
What’s Wrong With Concurrency Control and 2PC

• **Concurrency Control:**
  Nasty bugs: if aborts aren’t handled properly by the application, effects of aborted transactions leave effects on program variables. These bugs are not discovered until production.

  Performance: bottlenecks on shared resources are frequent. They require radical schema changes, e.g. hash the processid into a special field, version fields.

• **Two Phase Commit:**
  Customers don’t like the message overhead of two phase commit, particularly in WAN environments, so they settle for replication servers which give warm, not hot, backups.
An Example Alternative

- Main memory database, logically single threaded, operation logging, full replication. Net result: a replicated state machine whose commits are uncoordinated.

- Operations are logged and acked in batches, so much less than one overhead message per transaction. Each site dumps its state in a round robin fashion.
Large Databases: what is needed

- Since the algorithm executes transactions sequentially, big databases (i.e., too big for RAM) can’t take advantage of disk bandwidth. Like life before operating systems.

- Executing in order of arrival is too slow, but want to appear to do so.

- NB: Serializability at each replicated site is not enough. Do you see why?
Algorithm OBEYORDER

1. Construct a predicate called CONFLICT that takes two transaction instances and determines whether they would conflict.

2. If t CONFLICTS with t’ and t has an earlier arrival number than t’, then form a directed edge (t, t’). This produces a graph $G = (T, E)$ where $T$ is the set of all the transactions in the batch (or batches) and $E$ is the set of directed edges formed as described.

3. Execute $T$ in parallel, respecting the order implied by $E$. 
Issues with OBEYORDER

• Need a CONFLICT predicate. (Can be difficult to write.)

• If there are many conflicts, must do more. Observe: can always prefetch data provided it is globally visible.

• The old standby: if data can be partitioned, buy another processor.
Summary

• In-memory databases can use a different approach from on-disk databases (just ask TimesTen). No concurrency control, operation recovery, and hot backups.

• If data spills over to disk, then you need to invent a new concurrency control scheme.

• You can get transactional guarantees plus hot backup, all with low overhead.
Lessons from Wall Street: case studies in configuration, tuning, and distribution

Dennis Shasha
Courant Institute of Mathematical Sciences
Department of Computer Science
New York University
shasha@cs.nyu.edu
http://cs.nyu.edu/cs/faculty/shasha/index.html
occasional database tuning consultant on Wall Street
Wall Street Social Environment

- Very secretive, but everyone knows everything anyway (because people move and brag).

- Computers are cheap compared to people. e.g., 2 gigs of RAM is a common configuration for a server and will grow once 64 bit addressing comes in.

- Two currencies: money and fury.
Wall Street Technical Environment

- Analytical groups use APL or FAME or object-oriented systems or Excel with extensions to value financial instruments: bonds, derivatives, and so on. These are the “rocket scientists” because they use continuous mathematics and probability (e.g. Wiener processes).

- Mid-office (trading blotter) systems use Sybase. These maintain positions and prices. Must be fast to satisfy highly charged traders and to avoid arbitrage (delays can result in inconsistencies).

- Backoffice databases handle final clearance.
Overview

• Configuration — disaster-proof systems, interoperability among different languages and different databases.

• Global Systems — semantic replication, rotating ownership, chopping batches.

• Tuning — clustering, concurrency, and hashing; forcing plans.

• Complaints, Kudos, and a Request
Preparing for Disaster

- Far from the simple model of stable storage that we sometimes teach, though the principles still apply.

- Memory fails, disks fail (in batches), fires happen (Credit Lyonnais, NY Stock Exchange), and power grids fail. If your system is still alive, you have a big advantage.

- You can even let your competitors use your facilities ... for a price.
Case: Bond Futures

- Server for trading bond futures having to do with home mortgages.

- Application used only a few days per month, but the load is heavy. During a weekend batch run, 11 out of 12 disks from a single vendor-batch failed.
High Availability Servers

- A pair of shared memory multiprocessors attached to RAID disks.

- If the primary multiprocessor fails, the backup does a warm start from the disks.

- If a disk fails, RAID masks it.

- Does not survive disasters or correlated failures.
Writes go to the primary and into the high availability disk subsystem. This subsystem is normally a RAID device, so can survive one or more disk failures. If the primary fails, the secondary works off the same disk image (warm start recovery).

Vulnerability: High availability disk subsystem fails entirely.
Dump and Load

• Full dump at night. Incremental dumps every three minutes.

• Can lose committed transactions, but there is usually a paper trail.

• Backup can be far away.
Replication Server

- Full dump nightly. All operations at the primary are sent to the secondary after commit on the primary.

- May lose a few seconds of committed transactions.

- Slight pain to administer, e.g. schemas, triggers, log size.
Basic architecture of a replication server.
The backup reads operations after they are committed on the primary. Upon failure, the secondary becomes the primary by changing the interface file configuration variables.

Vulnerability: if there is a failure of the primary after commit at the primary but before the data reaches the secondary, we have trouble.
Remote Mirroring

- Writes to local disks are mirrored to disks on a remote site. The commit at the local machine is delayed until the remote disks respond.

- Backup problems may cause primary to halt.

- Reliable buffering can be used (e.g. Qualix), but the net result is rep server without the ability to query the backup.
Two Phase Commit

- Commits are coordinated between the primary and backup.

- Blocking can occur if the transaction monitor fails. Delays occur if backup is slow.

- Wall Street is scared of this.
Two phase commit: transaction manager ensures that updates on the primary and secondary are commit-consistent. This ensures that the two sides are in synchrony.

Vulnerability: blocking or long delays may occur at the primary either due to delays at the secondary (in voting) or failure of the transaction manager.
Quorum Approach (e.g., DEC, HP, IBM, ISIS....)

- Servers are co-equal and are interconnected via a highly redundant wide area cable.

- Clients can be connected to any server. Servers coordinate via a distributed lock manager.

- Disks are connected with the servers at several points and to one another by a second wide area link.
Heartbeats

- Heartbeats monitor the connectivity among the various disks and processors.

- If a break is detected, one partition holding a majority of votes continues to execute.

- Any single failure of a processor, disk, site, or network is invisible to the end users (except for a loss in performance).
Quorum Approach as Used in most Stock and Currency Exchanges.
Survives Processor, Disk, and Site failures.

Quorum approach used in most exchanges.
Which to Use

- Stock exchanges use the quorum approach.

- Midoffice database servers often use dump and load or rep server. Symmetric approaches that may cause the primary to delay are too scary.

- Don’t buy batches from one vendor.
Case: Indicative Data Display

- Indicative data is data that doesn’t change much, e.g. payment schedules of bonds, customer information.

- Must be at a trader’s fingertips.

- Relational connections to personal computers are too slow. So data is held outside the database.
The fashion is for the serves to be stateless, but this implies that clients may have out-of-date data. Stateful Servers are better.

What happens if two clients update concurrently?
How to Handle Updates?

- Ignore updates until the next day (used all too often).

- Program clients to request refresh at certain intervals.

- Have the server hold the state of the clients. Send messages to each client when an update might invalidate a client copy or simply refresh the screens.
Question for Vendors

- Consider data structures that are kept outside the database because it is too computationally or graphically intensive to put in the database.

- How best should you keep the data used by that application up-to-date?

- What facilities should you give to handle concurrent client updates?
Case: Interdatabase Communication

- As in many industries, financial database systems grow up as islands and then discover — surprise — they must interconnect. Source sends data to destination which then sends some confirmation.

- Replication server is a possible solution to this problem, but
  (i) Commits at the source may not make it.
  (ii) Responses from the destination imply two-way replication. Known to be hazardous.
  (iii) Hard to determine where the data is.
Use Tables as Buffers

• Implement a buffer table on the source system side that holds information in the denormalized form required by the destination side.

• The destination database reads a new tuple $t$ from the buffer table.

• After processing $t$, the destination database flags $t$ as deletable or deletes $t$ itself in the buffer table.
Overcoming Blocking

- Get blocking if source and destination scan the buffer table for update purposes.

- Approach 1: Destination database puts responses in a different table to avoid update-update conflicts on the buffer table.

- Approach 2: Use clustering in the buffer to avoid blocking.
Source system transactions write to buffer tables and back office systems read from them. If back office systems must respond, then either cluster the buffer tables or use a second response table written by the back office system.
Clustering Solution Expanded

- Cluster on some hashing of a key value.

- Source writes new records to one cluster at a time.

- Destination updates records in one cluster at a time in round-robin fashion. No FIFO guarantee, but every sent tuple will be received.
The Need for Globalization

- Stocks, bonds, and currencies are traded nearly 24 hours per day (there is a small window between the time New York closes and Tokyo opens).

- Solution 1: centralized database that traders can access from anywhere in the world via a high-speed interconnect.

- Works well across the Atlantic, but is very expensive across the Pacific. Need local writes everywhere in case of network partition.
Distributed Solution

- Two phase commit worries users because of blocking and delays. Replication can result in race condition/anomalies. (e.g. Gray et al. Sigmod 96).

- Sometimes, application semantics helps.
Case: Options traders

- A trading group has traders in 8 locations accessing 6 Sybase servers. Access is 90% local.

- Exchange rate data, however, is stored centrally in London. Rate data is read frequently but updated seldom (about 100 updates per day).

- For traders outside of London, getting exchange rates is slow. Can we replicate the rates?
Consistency Requirements

- If a trader in city X changes a rate and then runs a calculation, the calculation should reflect the new rate (So, can’t update London and wait for replication.)

- All sites must agree on a new exchange rate after a short time (must converge). (So, can’t use vanilla replication server.)
Clock-based Replication

- Synchronize the clocks at the different sites. (Use a common time server.)

- Attach a timestamp to each update of an exchange rate.

- Put a database of exchange rates at each site. An update will be accepted at a database if and only if the timestamp of the update is greater than the timestamp of the exchange rate in that database.
Clients send rates to local machines where they take immediate effect.
Rates and timestamps flow from one server to the other.
Latest timestamp does the update.
Ensures: convergence and primacy of latest knowledge.

Timestamped Replication
Case: Security Baskets

- Trade data is mostly local, but periodically traders collect baskets of securities from multiple sites.

- The quantity available of each security must be known with precision.

- The current implementation consists of an index that maps each security to its home database. Each site retrieves necessary data from the home site.
Rotating Ownership

- Maintain a full copy of all data at all sites.

- Not all of this data will be up-to-date ("valid") at all times however. Can be used for approximate baskets.

- When a market closes, all its trades for the day will be sent to all other sites. When receiving these updates, a site will apply them to its local database and declare the securities concerned to be "valid."
Rotation Issues

- Receiving ownership must be trigger-driven rather than time-driven.

- Suppose New York assumes it inherits ownership from London at 11 AM New York time. If the connection is down when London loses its ownership, then some updates that London did might be lost.
Ownership travels from east to west as exchanges close. A given exchange should assert ownership only after it is sure that the previous exchange has processed all trades.

Rotating Ownership
Case: Batch and Global Trading

- When the trading day is over, there are many operations that must be done to move trades to the backoffice, to clear out positions that have fallen to zero and so on. Call it “rollover.”

- Straightforward provided no trades are hitting the database at the same time.

- In a global trading situation, however, rollover in New York may interfere with trading in Tokyo.
Chop the batch

- “Chop” the rollover transaction into smaller ones.

- The conditions for chopping are that the ongoing trades should not create cycles with the rollover pieces.

- New trades don’t conflict with rollover. Lock conflicts are due to the fact that rollover uses scans.
Good Candidates for Chopping

- Batch operations that don’t logically conflict with ongoing operations. (Index conflicts are not a problem).

- Chopping means take each batch operation and break it into independent pieces, e.g., delete zero-valued positions, update profit and loss.

- If batch operations are not idempotent, it is necessary to use a “breadcrumb” table that keeps track of which batch operations a process has completed.
Tuning Case: Sequential keys, clustering and blocking

- Sequential keys (i.e., keys whose values are monotonic in time) are used to identify rows in trade and position tables uniquely. Suppose the table is clustered on a sequential key.

- Buffer behavior is good since all inserts hit the same few pages.

- Multiple concurrent inserts will conflict on the last page of a data structure or of a data page. Especially bad for page-level locking systems.
Hash Clusters

- Create a key: 
  concat(hash(process id), sequential key).

- Inserts cluster at as many locations as there are possible hash values.

- Good clustering without concurrency loss.
Different random key, sequential key concatenations will not conflict with one another. They will still however give good buffering behavior since only one page per random key need be in the database cache.
Tuning Case: Interest Rate Clustering

- Bond is clustered on interestRate and has a non-clustered index on dealid. Deal has a clustered index on dealid and a non-clustered index on date.

- Many optimizers will use a clustering index for a selection rather than a non-clustering index for a join. Often good. The trouble is that if a system doesn’t have bit vectors, it can use only one index per table.
Query to be Tuned

```sql
select bond.id
from bond, deal
where bond.interestRate = 5.6
and bond.dealid = deal.dealid
and deal.date = '7/7/1996'
```
What Optimizer Might Do

- Pick the clustered index on interestRate.

- May not be selective because most bonds have the same interest rate.

- This prevents the optimizer from using the index on bond.dealid. That in turn forces the optimizer to use the clustered index on deal.dealid.
Alternative

- Make deal use the non-clustering index on date (it might be more useful to cluster on date in fact) and the non-clustering index on bond.dealid.

- Logical IOs decrease by a factor of 40 (170,000 to 4,000).
Complaints and Kudos

• It’s important to know what your system does badly. For Sybase, the NOT IN subquery is particularly bad. Rewriting queries to get rid of them can reduce the number of logical IOs by a factor of 6 in cases I’ve seen.

• Removing DISTINCTs when they are unnecessary can improve a query’s performance by 25%.
Case: Temporal Table Partitioning

- Position and trade were growing without bound. Management made the decision to split each table by time (recent for the current year and historical for older stuff). Most queries concern the current year so should be run faster.

- What happened: a query involving an equality selection on date goes from 1 second with the old data setup to 35 seconds in the new one. Examining the query plan showed that it was no longer using the non-clustered index on date. Why?
Use of Histogram

- Optimizer uses a histogram to determine usefulness of a non-clustering index.

- Histogram holds 500 cells, each of which stores a range of date values.

- Each cell is associated with the same number of rows (those in the cell’s date range).
Initially, each cell was associated with several days’ worth of rows.

After reducing the size of the table, each cell was associated with less than a day’s worth of rows. So, a single day query spills on several cells.

Non-clustering index is not used if more than one cell contains the searched-for data.
Heuristic Brittleness

- The optimizer’s rule is that a non-clustering index may be used only if the value searched fits entirely in one cell.

- When the tables became small, an equality query on `date` spread across multiple cells. The query optimizer decided to scan.

- A warning might help.
RAID disks

- Raid 5 discs seem to be much faster for load applications than Raid 0, giving approximately a 60% improvement (14 minutes to 5 minutes)

- Raid 5: each disk has a portion of a sector. There are n subsector portion and a parity subsector that make up a sector. A small update will write a single subsector.
RAIDs on online transaction processing

• How Raid 5 works when updating a subsector: it must read the old version of that subsector Sold, read the parity subsector Pold, Pnew := (Sold \text{xor} Snew) \text{xor} Pold

• This is two reads and two writes for one write.

• Savage and Wilkes (Usenix ref) have a nice solution to this problem that involves delaying the write to the parity disk. Ted Johnson and I have a technique for making this safe.
Kudos: sample of new monitoring tools

- Average utilization of packets (if high, then bigger network packets might help).

- Why the log buffer is flushed (if before transaction completes, then perhaps make it bigger).

- Reason for task context switches. (Tells you if there is too much locking.)
My Requests to Industry

- Database systems on Wall Street require (i) an engine, (ii) a user interface, (iii) a third generation language for math functions and interprocess communication. Getting these to work together is hard.

- Most of the updatable data fits in a few gigabytes however.

- Perhaps a programming language approach is better.
Shape of this Programming Language

- Array based for time series, top ten queries etc.

- Integrated GUI, e.g. negative values of some variable turn red, newly updated values blink. This happens by defining an attribute on the variable. (Ravi Krishnamurthy proposed something like this in Sigmod 1996).

- Include interprocess communication.
Transaction Processing with a Programming Language

- Operation logging. Recovery by replaying the log from the last dump.

- Eliminate concurrency control by single threading or run-time conflict detection. Deadlocks and blocking require too much development time.

http://cs.nyu.edu/cs/faculty/shasha/papers/papers.html
Summary: the main challenges

- Wall Street is different from you and me, it has more money… Also, more demands.

- High availability and reliability: hot remote backup with low probability of blocking.

- Global: must worry about distribution across WANs, where delays are significant and breakdowns.
Research and Products: db system issues

• Batch cycle overlaps online activity. This results in significant blocking and requires concurrent maintenance operations (e.g. tear down and build up of indexes).

• Need a science of tuning in the spirit of Schek and Weikum’s Comfort project.

• Would really like a good sizing tool: given a distributed application, what hardware and interconnection bandwidth should I buy?
Research and Products: language issues

- SQL 92 is complicated and too weak. SQL 93 and object-relational systems may fill the void.

- Bulk operations on arrays would be really useful however.

- There is a whole class of applications that would be better off without concurrency control.
References


References — Continued


2Q: a low overhead, high performance, buffer management replacement algorithm

Dennis Shasha, New York University

Ted Johnson, University of Florida
Buffer Management

- Buffer (or buffer cache) is a portion of virtual memory (mostly RAM) used to hold copies of disk pages.

- When an application issues a read of page (or block) \( p \), the operating system (or database operating system) first looks into the buffer to see if \( p \) is there. If so (a hit), it returns \( p \) to the application (1000 times faster than if \( p \) were on disk).

Hit ratio = proportion of read pages that are in buffer.
Least Recently Used Replacement Algorithm

- Ideally, buffer holds all pages ever used, but there is usually insufficient memory for such a utopia.

- Question: which page should buffer replace if room is needed for a new page? Least recently used: replace the page you used least recently.

- Tarjan and Sleator (1985): If we apply LRU to a buffer of size $B$ and perform $R$ page replacements over some time interval, then even a clairvoyant algorithm will perform at least $R/2$ page replacements for a buffer of size $B/2$. 
Problem Closed?

- No way. Such factors of 2 look innocent to a theoretician, but can have devastating practical effects.

- Example from Stonebraker (1981): If we repeatedly scan a file consisting of $B+1$ pages, then LRU will do $B+1$ replacements per scan.

  An optimal algorithm could keep the first $B-1$ pages in the buffer permanently and just exchange the last two pages into the final buffer. Factor of $B/2$ better than LRU on a buffer of size $B$.

  Moral: treat scans specially (many DBMS’s use most recently used for scanned pages).
Random Access Example

- Online transaction processing (and any other application making use of an index) traverses a relatively small data structure to get to many data pages. Typically 1 index leaf page to 100 data pages.

- Ideal algorithm would keep index pages in buffer instead of data pages. LRU fills about half the buffer with the most recently accessed data pages.
LRU/2

• Pat and Betty O’Neil and Gerhard Weikum in ACM SIGMOD 93 proposed a variant of LRU called LRU/k.

• Basic idea: give priority to a page p based on its kth most recent access. In practice, they suggest k=2.

• Plausibility argument: does right thing for scans and for on-line transaction processing, giving low priority to scanned pages and random data pages. Experiments on traces from a Swiss Bank back up the plausibility argument.
LRU/2 Characteristics

• Under LRU, when a page is accessed, it is moved to the front of a list (pages are removed from the tail of that list). Constant time operation.

• Under LRU/2, when a page is accessed, its priority must be adjusted based on its penultimate (second-to-last) access. This will not normally move the page to the front, but to some random place in the queue.

   Best implementation: priority queue. Requires log B time, where B is buffer size in numbers of pages.
2Q Motivation

- Can we get the benefits of LRU/2 without the costs and complexity of a priority queue?

- Answer: yes.
2Q Algorithm — basic idea

- First time a page is accessed put it on the front of an “accessed once” queue (A1). Subsequent accesses put the page on the front of an “accessed many” queue (Am).

- Replace pages from the tail of A1 as long as A1 contains a certain minimum number of pages. Then replace from the tail of the Am queue.

- Plausibility: Like LRU/2, the 2Q algorithm will give low priority to scanned pages and to randomly accessed data pages (they will get to A1 only and then will be thrown out).
2Q Algorithm — some details

One can save space by keeping only pointers to some of the pages of A1.

This suggests dividing A1 into the portion where pages are maintained in the buffer (A1in) and the portion where the pages are outside the buffer (A1out) though the system remembers those pages so if they are accessed again, then they are promoted to Am.

A1out takes very little space (around 10 bytes per page).
2Q Algorithm — full picture

FIRST ACCESS TO p: put p in front of A1in.


IF SECOND ACCESS TO p WHILE ON A1out: promote to front of Am

WHILE ON Am: move to front of Am on each access (like LRU).
Fixing the Parameters

Across a variety of applications, the following parameter setting seems good.

A1in — pages kept in buffer (takes 1/4 of buffer space).

A1out — pages on disk, but pointers in buffer (to handle as many pages as would take up half of the buffer space, less than 1/100 of space).

Am — pages kept in buffer (approximately 3/4 of the buffer space).
Experiments

- Independent Random Variables with various hotness (80/20 and 45/20 rules).

- 100 index leaf pages plus 10,000 data pages, each selected uniformly at random.

- Access to a DB2 commercial application (from Ted Messinger of IBM). 500,000 references to 75514 unique pages.

- Access to program of a windowing application (from Steve Pendergrast of Novell). Contains 427,618 references to 290 pages each of length 256 bytes.
80/20 Independent Random Variables

<table>
<thead>
<tr>
<th>Page slots</th>
<th>LRU/2</th>
<th>LRU</th>
<th>2Q Kin=30%</th>
<th>2Q Kin=20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>.443</td>
<td>.428</td>
<td>.457</td>
<td>.457</td>
</tr>
<tr>
<td>8</td>
<td>.551</td>
<td>.536</td>
<td>.598</td>
<td>.598</td>
</tr>
<tr>
<td>16</td>
<td>.762</td>
<td>.730</td>
<td>.785</td>
<td>.792</td>
</tr>
<tr>
<td>24</td>
<td>.852</td>
<td>.825</td>
<td>.845</td>
<td>.849</td>
</tr>
<tr>
<td>32</td>
<td>.893</td>
<td>.876</td>
<td>.911</td>
<td>.913</td>
</tr>
<tr>
<td>40</td>
<td>.943</td>
<td>.946</td>
<td>.946</td>
<td>.946</td>
</tr>
</tbody>
</table>
## DB2

<table>
<thead>
<tr>
<th>Page slots</th>
<th>LRU/2</th>
<th>LRU</th>
<th>2Q Kin=30%</th>
<th>2Q Kin=20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>.345</td>
<td>.317</td>
<td>.349</td>
<td>.349</td>
</tr>
<tr>
<td>100</td>
<td>.432</td>
<td>.413</td>
<td>.444</td>
<td>.430</td>
</tr>
<tr>
<td>200</td>
<td>.516</td>
<td>.512</td>
<td>.521</td>
<td>.510</td>
</tr>
<tr>
<td>500</td>
<td>.610</td>
<td>.606</td>
<td>.624</td>
<td>.611</td>
</tr>
<tr>
<td>1000</td>
<td>.665</td>
<td>.654</td>
<td>.686</td>
<td>.681</td>
</tr>
<tr>
<td>2000</td>
<td>.712</td>
<td>.704</td>
<td>.729</td>
<td>.730</td>
</tr>
<tr>
<td>3000</td>
<td>.738</td>
<td>.730</td>
<td>.750</td>
<td>.753</td>
</tr>
<tr>
<td>4000</td>
<td>.755</td>
<td>.748</td>
<td>.767</td>
<td>.766</td>
</tr>
<tr>
<td>5000</td>
<td>.768</td>
<td>.762</td>
<td>.778</td>
<td>.777</td>
</tr>
<tr>
<td>6000</td>
<td>.778</td>
<td>.773</td>
<td>.785</td>
<td>.786</td>
</tr>
<tr>
<td>7000</td>
<td>.786</td>
<td>.781</td>
<td>.791</td>
<td>.791</td>
</tr>
<tr>
<td>8000</td>
<td>.792</td>
<td>.789</td>
<td>.795</td>
<td>.795</td>
</tr>
<tr>
<td>9000</td>
<td>.798</td>
<td>.795</td>
<td>.798</td>
<td>.799</td>
</tr>
<tr>
<td>10000</td>
<td>.802</td>
<td>.801</td>
<td>.801</td>
<td>.801</td>
</tr>
</tbody>
</table>
Conclusion

• LRU/2 and 2Q both improve on LRU over a wide variety of applications by 5-10%.

• 2Q is about as efficient as LRU (constant time).
Hierarchically Split Cube Forests

Prof. Dennis Shasha
Computer Science Department
Courant Institute of Mathematical Sciences
New York University
Joint Work with Ted Johnson
The Decision Support Problem

- You are a retailer of hunting and fishing gear in North America. (LOCATION)

- The equipment you sell is separated into categories like fishing poles, waders, and rifles. Categories split into subcategories. (PRODUCT)

- Some of your items are seasonal. (TIME)
Possible Queries

- Select/Insert/Update a single sale record into the database. (OLTP)

- Find the sales of fly fishing equipment: (i) in the Northeast on October 17, 1995. (ii) nationally since the film “A River Runs Through It.” (DECISION SUPPORT/OLAP/MULTIDIMENSIONAL)

- Find promising customers for hunting boots. (DATA MINING)
What it Takes to Find Answers

Just as you find it faster to find a phone number in an alphabetized directory than in an ordered list, so do indexes help find information in databases.

- **OLTP** — B tree. A kind of compressed telephone directory.

- **DECISION SUPPORT** — bit vectors, sparse matrices, multidimensional indexes with B tree nodes (MDI) to compute summary data.

- **DATA MINING** — multidimensional indexes or massive hardware.
Comparison of Index Structures: Bottom Line

- Bit vector plus: long history, good for multi-condition SQL queries.

- Sparse matrix plus: good for spreadsheet layout, less space than Cube Forests, hard to update.

- Cube Forests pluses:
  (i) decision support/OLAP.
  (ii) data mining.